Driving Wireless Convergence®

RS-WC-201/301
Software Programming Reference Manual

Version 2.54

May, 2014

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

Revision History

Document Version	**Changes**
2.50	Max webpage size info added. Modifications done for power save mode flow charts and explanation. Explanations for new features added under feature select bitmap.. New stats are added in get stats response structure. Rejoin section is modified.
2.51	IGMP error added. Note about rejoin, under Disassoc section in SPI and UART is modified. Webserver bypass explanation is modified.
2.54	Feature bit added to allow error codes for send data.Added comments on Wi-Fi direct appendix section.

Table of Contents

List of Figures

List of Tables

1 Introduction

This document describes the commands to operate the RS-WC-201 and RS-WC-301 modules. The parameters in the commands and their valid values; and 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 RS-WC-201/301 in UART Mode describes commands to operate the module using the UART interface. Section RS-WC-201/301 in USB Mode describes commands to operate the module using the USB interface. Section RS-WC-201/301 in SPI Mode describes commands and processes to operate the module using the SPI interface. Section Driver Porting Guide for SPI describes how to port a sample driver for SPI that is provided with the software release.

2 Architecture Overview

The following figure depicts the software architecture of the RS-WC-201/301 module.

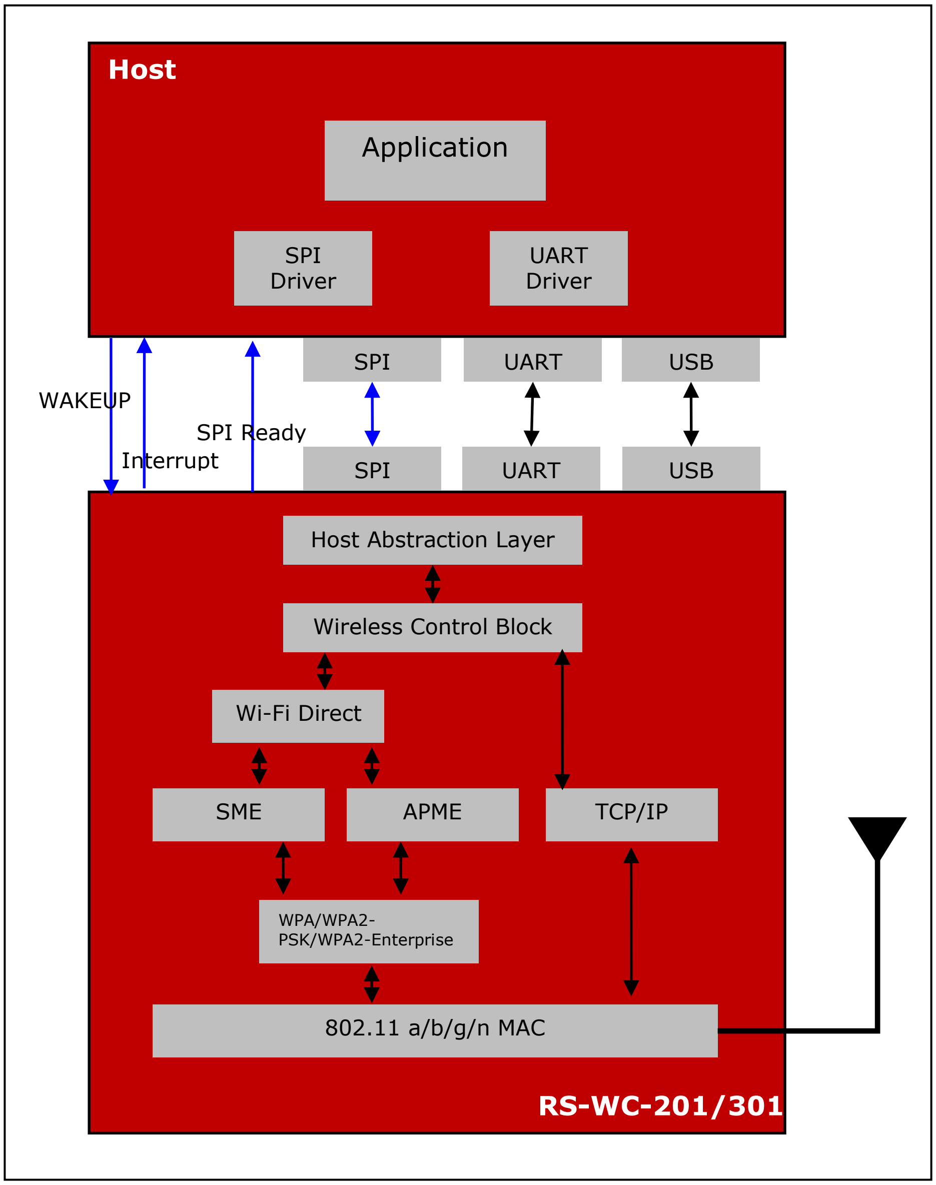


Figure 1:RS-WC-201/301 Software Architecture

The RS-WC-201/301 module is integrated with the Host using either UART, SPI or USB interfaces.

2.1 Components of RS-WC-201/301

The following sections describe the components of the RS-WC-201/301 module in brief.

2.1.1 SPI

The SPI on the RS-WC-201/301 acts the SPI slave. It is a 4-wire SPI interface. Along with the SPI interface, an Interrupt output (active high, level triggered) and an SPI_READY output signal are used to handshake with the Host. An input signal to the module, WAKEUP, is used in one of the power save modes while using the SPI interface, as described in the section Power save Operation. The Interrupt, WAKEUP and SPI_READY signals are not used in UART mode.

The interrupt pin is used by the module in SPI mode in the below cases:

1. When the module has data in its output buffers that needs to be read by the Host, through the SPI interface.
2. When the module wakes up from sleep in Power Save Mode, while using SPI interface.

The interrupt is active high, level triggered. The SPI_READY signal is an output from the module to be connected to a GPIO of the Host MCU. It is used as a handshake signal in SPI mode.

2.1.2 UART

The UART on the RS-WC-201/301 module is the physical interface which transmits/receives data from the Host in UART mode.

2.1.3 USB

The modules support USB interface from firmware version 2.0.0.x.x.x. The USB interface in the module corresponds to the CDC-ACM class and presents itself as a virtual COM port to the Host. The USB interface of the module supports full speed USB mode (12 Mbps physical data rate)

2.1.4 Hardware Abstraction Layer (HAL)

The HAL abstracts the lower layers in the Host interface with which the RS-WC-201/301 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.

2.1.5 Wireless Control Block (WCB)

The data from/to the Host is classified as Wi-Fi specific frames and TCP/IP specific frames. The WCB layer processes the frame obtained and acts accordingly. The functionality of the WCB module depends on the type and direction of the frame.

2.1.6 Wi-Fi Control frames

The WCB interprets the Wi-Fi control information from the Host and interacts with the SME (Station Management Entity). Configuration of the RS-WC-201/301 module from the Host for Wi-Fi access is through AT commands or SPI commands.

2.1.7 TCP/IP Control frames

If the frames from the Host are interpreted as TCP/IP specific frames then the WCB interacts with the TCP/IP stack.

2.1.8 Station Management Entity (SME)

The SME is the core layer which manages the Wi-Fi connectivity. 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.

2.1.9 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.

2.1.10WPA Supplicant

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.

3 RS-WC-201/301 in UART Mode

The following figure illustrates a general flow for operating a UART module.

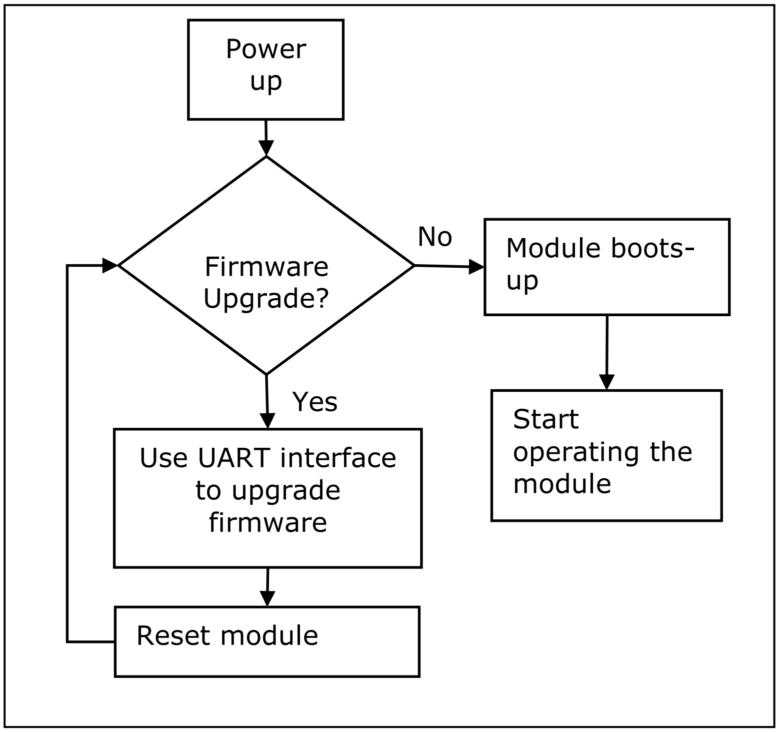

Power up
Firmware Upgrade?
No
Module boots-up
Yes
Use UART interface to upgrade firmware
Start operating the module
Reset module

Figure 2: Firmware Upgrade and General Operation in UART modules

RS-WC-201 and RS-WC-301 modules use 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, 200000 bps, 230400 bps, 460800 bps 921600 bps

Data bits: 8

Parity: None

Stop bits: 2

Flow control: None

3.1 Messages on Power-up

For users using firmware version 2.0.0.1.2.4 and below:

When the module is powered up, the following sequence is executed

1. The module sends five 0xFC bytes out in the UART interface
2. The module sends the message “Welcome to WiSeConnect” to the Host and then starts boot-up:

[0xFC 0xFC 0xFC 0xFC 0xFC Welcome to WiSeConnect\r\n]
……..
0xFC 0xFC 0xFC 0xFC 0xFC 0x57 0x65 0x6C 0x63 0x6F 0x6D 0x65 0x20 0x74 0x6F 0x20 0x57 0x69 0x53 0x65 0x43 0x6F 0x6E 0x6E 0x65 0x63 0x74 0x0D 0x0A

NOTE:

Host should ignore any byte/bytes coming from module (A byte '0x00' is observed before the first 0xFC in some systems) until it receives the byte 0xFC. Among the five 0xFC's last 0xFC may come as 0xFF or any other random byte.

3. After boot-up is complete, the module issues a message

   READY\r\n>

   ……
   0x52 0x45 0x41 0x44 0x59 0x0D 0x0A

4. The module is now ready to accept commands from the Host.

115200 bps is the baud rate supported in firmware version 2.0.0.1.2.4 and below.

<u>For users using firmware version above 2.0.0.1.2.4:</u>

When the module is powered up, the following sequence is executed

1. The module sends four 0xFC bytes and one 0xFF byte out in the UART interface at a baud rate 115200 bps. If the Host is configured at a different baud-rate, it might not receive these bytes correctly, but this should not be an issue
2. The Host must send 0x00 to the module at the Host's baud rate. The module, after power up, waits for a maximum of 10 secs to receive this byte from the Host. If the module times out beyond 10 secs, it configures its baud rate with a default of 115200 bps.
3. The module, after receiving 0x00, sends 0x55 to the Host at the Host's baud rate
4. The Host, after receiving 0x55 must again send 0x55 to the module.
5. The module now sends the message "Welcome to WiSeConnect" to the Host and then starts boot-up:
6. After boot-up is complete, the module issues a message

   READY\r\n>

   ……
   0x52 0x45 0x41 0x44 0x59 0x0D 0x0A

7. The module is now ready to accept commands from the Host.

Steps #1 to #4 are used as part of the Auto-baud rate detection (ABRD) sequence, by which the module recognizes the UART baud rate of the Host. The module waits for a maximum of 10 secs for ABRD to be successfully completed. If the byte exchanges are not as described above, the module, after a maximum effort of 10 secs configures its baud rate with a default value of 115200 bps. The following baud rates are supported: 9600 bps, 19200 bps, 38400 bps, 57600 bps, 115200 bps, 200000 bps, 230400 bps, 460800 bps 921600 bps.

3.2 UART Commands

The Wi-Fi AT command set represents the frames that are sent from the Host to operate the RS-WC-201/301 module. The command set resembles the standard AT command interface used for modems.

AT commands start with "AT" and are terminated with a carriage return and a new line character. The AT command set for the RS-WC-201/301 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 certain values inside the module.

APPENDIX A: Sample Flow of Commands in UART captures sample flow of commands to configure the module in various functional modes.

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.

3. Each command has a fixed timeout of 10 minutes. If the host fails to issue the complete command with in this time, then module will throw an error and reset on it its own.

Host will receive ERROR < error code 50 - Hex >\r\n and then OK\r\n after successful reset.

3.2.1 Set Operating Mode

Description

This is the first command that should be sent from the Host. This command configures the module in different operating modes.

Command

at+rsi_opermode

Usage

at+rsi_opermode=*mode_val*\r\n

Parameters

mode_val(1 byte, ASCII) : Sets the mode of operation

0– **Operating Mode 0**: Normal Client Mode. Wi-Fi Direct and Access Point modes are disabled in this mode. The module works as a client that can connect to an Access Point with WPA/WPA2-PSK in CCMP and TKIP modes of security and in open mode.

1– **Operating Mode 1: Wi-Fi Direct™** mode. In Wi-Fi Direct Group Owner mode, a maximum of 4 client devices are supported. Wi-Fi Direct Group Owner mode is described in the following sections.

2– **Operating Mode 2:** Enterprise Client Mode. Wi-Fi Direct and Access Point modes are disabled in this mode. The module works as a client that can connect to an Access Point with WPA/WPA2-Enterprise security.

6– **Operating Mode 6: Access Point** mode. In this mode, a maximum of 4 client devices are supported.

> Note: This format of command and the corresponding byte stream under the dotted line is followed in all examples. Operating Mode 3, 4, 5 and 7 are used to enable TCP/IP by pass mode, which are not valid in UART mode.

Response

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

The string ERROR is transmitted in its ASCII form and the error code in its two's complement form.

If there is an Error, an ERROR message with a corresponding 2-byte code in two's complement format will be returned.

For example,

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

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

If an ERROR is returned by the module, the least significant byte of the Error code is returned first. For example, if the error code is -4

```
ERROR -4\r\n
………………………..

0x45   0x52   0x52   0x4F   0x52  0xFC  0xFF  0x0D  0x0A
```

3.2.2 Band

Description

This command configures the band in which the module should operate. RS-WC-201 is a single band module (2.4 GHz only) and RS-WC-301 is a dual band module (2.4 GHz and 5 GHz).

Command

```
at+rsi_band
```

Usage

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

Parameters

When Operating Mode =0 or 2

band_val (1 byte, ASCII):

0– 2.4 GHz

1– 5 GHz. Applicable only for RS-WC-301 module.

When Operating Mode =1

Wi-Fi Direct Mode: If the module is configured as a Wi-Fi Direct node within Operating Mode 1, then the below description should be used.

0– 2.4 GHz is used both during Group Owner (GO) negotiation and general operation

1– 2.4 GHz is used during GO Negotiation but module will operate on 5GHz if it becomes the GO after the GO negotiation process is over.

Access Point Mode: If the module is configured as an AP within Operating Mode 1, then the below description should be used.

0– AP is configured to operate in 2.4 GHz

1– AP is configured to operate in 5 GHz.

For example, band command is given as

at+rsi_band=1\r\n

……………………………..

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x62 0x61 0x6E 0x64 0x3D 0x31 0x0D 0x0A

Response

Result Code	Description
OK	Successful execution of the command
ERROR<Error code>	Failure, Possible error codes are 0x0005, 0x0021, 0x0025, 0x002C, 0xFFF8

Relevance

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

3.2.3 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+rsi_init

Usage

at+rsi_init\r\n

Parameters

No parameters

Response

Result Code	Description
OK<*MAC_Address*>	*MAC_ Address* (6 bytes, Hex): The MAC Address of the module
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFF8

For example, the init command is given as

at+rsi_init\r\n

……………………

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

OK 0x00 0x23 0x12 0x13 0x14 0x15\r\n

……………………

0x4F 0x4B 0x00 0x23 0x12 0x13 0x14 0x15 0x0D 0x0A

Relevance

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

3.2.4 Antenna Selection

Description

This command configures the antenna to be used. RS-WC-201/301 provides two options – an inbuilt chip 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 chip antenna is selected.

Command

at+rsi_ antenna

Usage

at+rsi_antenna=antenna_val\r\n

Parameters

antenna_val (1 byte, hex):

1– Chip antenna selected

2– UFL connector selected

For example,

at+rsi_antenna=1\r\n

…………………………..

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x61 0x6E 0x74 0x65 0x6E 0x6E 0x61 0x3D 0x31 0x0D 0x0A

Response

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

Relevance

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

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

Description

This command is used to set the configuration information for Wi-Fi Direct mode. After issuing this command, the module scans for Wi-Fi Direct nodes. If any Wi-Fi Direct node is found, then it will send the information to the Host using the message **AT+RSI_WFDDEV.** If there is any remote Wi-Fi Direct node sends a connect request for the module, then module will send the information to the host using the message **AT+RSI_CONNREQ**.

Command

From Host to Module

at+rsi_wfd

Asynchronous Message from Module to Host

AT+RSI_WFDDEV (returned in upper case characters)

Another asynchronous message from Module to Host

AT+RSI_CONNREQ (returned in upper case characters)

Usage

From Host to Module

at+rsi_wfd
=Group_Owner_intent,device_name,channel_num,ssid_postfix,psk\r\n

Parameters

From Host to Module

Group_Owner_intent (maximum of 2 bytes, ASCII):

Wi-Fi Direct Mode: 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 in this mode 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.

Device_name (maximum of 32 bytes, ASCII): 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.

Channel_num (maximum of 2 bytes, ASCII): Operating channel to be used in Group Owner (GO) or Access Point mode. The specified channel is used if the device becomes a GO. The supported channels can be any valid channel in 2.4GHz or 5GHz. If *at+rsi_band=0* is used, then a channel in 2.4 GHz should be supplied to this parameter. If *at+rsi_band=1* is used, then a channel in 5GHz should be supplied to this parameter. The valid values for this parameter are listed in tables Channels in 2.4 GHz and Channels in 5 GHz. '0' is not a valid value for this parameter.

Ssid_postfix(maximum of 23 characters, ASCII): This parameter is used to add a postfix to the SSID in WiFi Direct GO mode.

If the module becomes a Wi-Fi Direct Group Owner, it would have an SSID with "DIRECT-xy" prefixed to the *ssid_postfix* 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[1].

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

psk (maximum of 63 bytes, ASCII): Passphrase. 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.

Response

Result Code	Description
OK	Successful execution of the command
AT+RSI_WFDDEV=<device_state> <device_name><device_mac>	Asynchronous Message AT+RSI_WFDDEV from

[1] After the module becomes a GO in WiFi direct mode, it appears as an Access Point to client devices.

Result Code	Description
<device_type>	module to host device state (1-byte, hex). 0x01– A new remote Wi-Fi Direct node has been found device_name (32 bytes, ASCII): Device name of the remote Wi-Fi Direct node. If the device name of the remote node is less than 32 bytes, 0x00's are padded by the module to make the length 32 bytes device_mac (6 bytes, hex): MAC ID of the remote Wi-Fi Direct node. Device_type (2 bytes, hex): Type of the device. The first byte returned is the primary ID, and the second byte is the sub-category ID. Refer to Wi-Fi Direct Device Type 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. device_state (1-byte, hex). 0x00– The remote Wi-Fi Direct node was found in the previous scan iteration device_name (32 bytes, ASCII): All are 0x00's device_mac (6 bytes, hex): MAC ID of the remote Wi-Fi Direct node which is moved out of range.

Result Code	Description
	Device_type (2 bytes, hex)
AT+RSI_CONNREQ<device_name>	Another asynchronous message from Module to Host, sent when module receives a connection request from any remote Wi-Fi Direct node.
ERROR<Error code>	Failure, Possible error codes are 0x001D, 0x0021, 0x0025, 0x002C, 0xFFF8

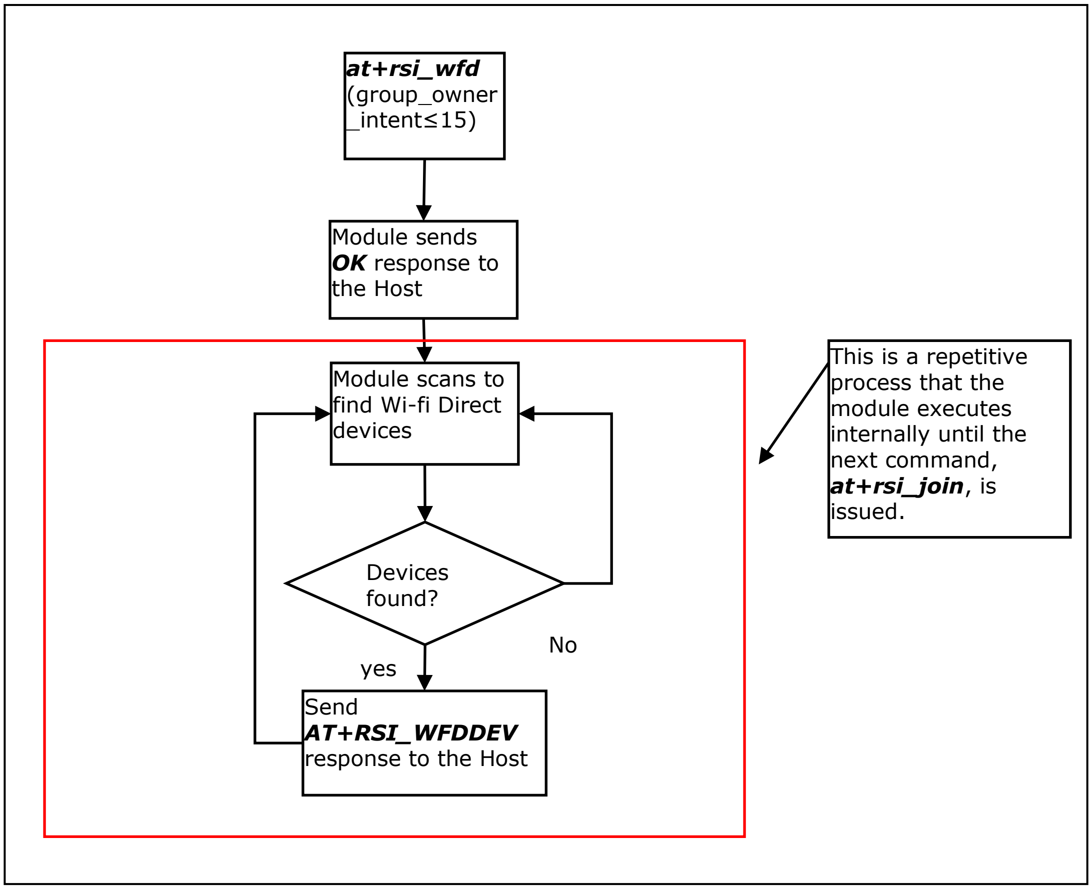

at+rsi_wfd (group_owner _intent≤15)
Module sends OK response to the Host
Module scans to find Wi-fi Direct devices
Devices found?
No
yes
Send AT+RSI_WFDDEV response to the Host
This is a repetitive process that the module executes internally until the next command, at+rsi_join, is issued.

Figure 3: Operation after issuing at+rsi_wfd command

Category	Primary ID	Sub Category	Sub ID
Computer	0x01	PC	0x01

		Server	0x02
		Media Center	0x03
		Ultra-mobile PC	0x04
		Notebook	0x05
		Desktop	0x06
		Mobile Internet Device	0x07
		Netbook	0x08
Input Device	0x02	Keyboard	0x01
		Mouse	0x02
		Joystick	0x03
		Trackball	0x04
		Gaming controller	0x05
		Remote	0x06
		Touchscreen	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
		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
		Playstation	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
Others	0xFF		

Table 1: Wi-Fi Direct Device Type

For example,

Command:

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_phone 0x00 0x23 0x12 0x13 0x14 0x16 0x0A 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 0x00 0x01 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 0x01 0x02 0x03 0x04 0x05 0x06 0x00 0x00 0x0D 0x0A

Relevance

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

Note: After getting the connect request from remote device, host need to issue the "join" command with the remote device name that sent the request. User need to make sure that the remote device is scanned by us too (AT+RSI_WFD=<remote device name>…). If the user issues join before remote device get scanned by us, will get join response with error "0x19".

3.2.6 Configure AP Mode

Description

If the module is to be used as an AP, this command is used to set the parameters of the AP.

Command

at+rsi_apconf

Usage

```
at+rsi_apconf
=channel_number,ssid,sec_type,enc_type,psk,beacon_interval,dtim_count,m
ax_sta_support\r\n
```

Parameters

channel_number (maximum of 2 bytes, ASCII): The channel in which the AP would operate. Refer tables Channels in 2.4 GHz and Channels in 5 GHz. A value of '0' is not allowed.

ssid (maximum of 32 bytes, ASCII): SSID of the AP to be created

sec_type(1 byte, ASCII): Security type.

0-Open

1-WPA

2-WPA2

enc_type (1 byte, ASCII): Encryption type.

0-Open

1-TKIP

2-CCMP

psk (maximum of 63 bytes, ASCII): PSK of the AP in security mode. If the AP is in Open mode, this parameter can be set to '0'.

beacon_interval (maximum of 4 bytes, ASCII): Beacon interval of the AP in milliseconds. Allowed values are integers from 100 to 1000 which are multiples of 100.

*dtim_count (*maximum of 3 bytes, ASCII*):* DTIM count of the AP in number. Allowed values are from 1 to 255.

max_sta_support (1 byte, ASCII): Number of clients supported. The maximum value allowed is 4. For example, if this value is 3, not more than 3 clients can associate to the client.

Response

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

For example,

Command:

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
```

Relevance

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

Note: In WiSeConnect^{TM} AP mode, only mixed 802.11g and 802.11b is supported.

3.2.7 Feature Select

Description

This command is used to control behaviours with respect to specific commands.

Command

```
at+rsi_featsel
```

Usage

at+rsi_featsel=*bit_map*\r\n

Parameters

bit_map: This is a 32-bit unsigned word that is supplied in this decimal form.

bit_map[0] :

‘1’ - Add Cisco AP name in the “Scan” command’s response

‘0’ – Don’t add Cisco AP name in the “Scan” command’s response.

bit_map[1] :

‘1’ - Add SNR value in the “Scan” command’s response

‘0’ – Don’t add SNR value in the “Scan” command’s response.

bit_map[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_map[3]: If this bit is set to '1', HTTP web server in the module is disabled completely. The default value of this bit is '0'.

bit_map[4]: If this bit is set to '1', UART hard ware flow control is enabled. The default value of this bit is '0'.

bit_map[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_map[6] : 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, sends out a DNS address with assigned IP and Subnet values to the client. The default value of this bit is '0'.

bit_map[7] : If this bit is set to '1', the DHCP client behavior, when the module is in STA mode, changes. The DHCP client sends DHCP discover and DHCP request with unicast flag. The default value of this bit is '0'.

bit_map[8] : If this bit is set to '1', then BT co-existence is enabled. Once the WLAN connection is successful, the module honours BT priority data. The default value of this bit is '0'.

bit_map[9] : This is not applicable in case of UART.

Note: After UART hardware flow control is enabled with feature select command, user need to close the UART on the host side and has to open it again with hardware flow control enabled.

BT priority and WLAN active pins from module should be connected to BT device for co-existence to work. BT-Priority is an input to the module and WLAN active is an output from the module. When there is BT priority data, BT device should set BT priority pin. Please refer to data sheets for pin numbers of BT priority and WLAN active.

bit_map[31:9]: Reserved, should be set to all '0'.

Response

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

For example,

if bit_map[1:0] = '11', then the 32-bit format is [0 0 0 ….1 1], and the decimal value is 3.

Note: This is not a mandatory command. It is advised to NOT use this command, unless specific behavior is expected of the "Scan" command, as described in the section below. If this command is used, it should be issued as the first command to the module, before the at+rsi_opermode command.

Command:

```
at+rsi_featsel=3\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x66 0x65 0x61 0x74  0x73
0x65  0x6C  0x3D  0x33  0x0D 0x0A
```

Response:

```
OK\r\n
```

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

Relevance

This command is relevant to the operating modes 0, 1, 2 or 6.

3.2.8 Scan

Description

This command makes the module scan 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 list.

Command

at+rsi_scan

Usage

at+rsi_scan=*chan_num*,*SSID*\r\n

Parameters

chan_num (maximum of 2 bytes, ASCII): 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 *at+rsi_band* command.

Parameters for 2.4 GHz

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

Table 2: Channels in 2.4 GHz

Parameters for 5 GHz[1]

Channel Number	chan_num parameter
All channels	0
36	1
40	2
44	3
48	4
149	20
153	21
157	22
161	23
165	24

Table 3: Channels in 5 GHz

SSID (maximum of 32 bytes, ASCII): Optional Input. For scanning a specific AP or a hidden AP, its SSID can be provided as part of the SCAN command.

The maximum number of scanned networks reported to the host is 11.

[1] DFS is currently not supported, it is advised to not use channels from 52 to 140 if the environment is expected to co-exist with Radars.

Notes:

1. If the number of APs is more than 11 around the area, all the APs may not get scanned. In this case you need to explicitly set the channel to scan that particular AP.

2. Scan requests with channel numbers 12, 13 or 14 is not supported. If user gives scan requests with these scan channels, error 36 will be returned.

3. When the scan is given for all channels, by default channels from1 to 11 will be scanned. Maximum channels to be scanned will be adjusted based on the country IE found in the beacons/probe responses of the APs scanned in channels from 1 to 11.

Response

The response frame for the scan command is as shown below. The fields from "Channel" through "Reserved" are repeated according to the number of access points found.

Result Code	Description
OK<ScanCount><Reserved><Channel1><SecurityMode1><RSSIVal1><NetworkType1><DeviceName1><BSSID1><Reserved1><CISCO_AP_Name1><Channel2><SecurityMode2><RSSIVal2><Network type2><Device Name2><BSSID2><Reserved2><CISCO_AP_Name2>………up to the number of scanned nodes	*ScanCount* (4 bytes, hex): Number of scanned access points. The least significant byte is sent first. For example, if the ScanCount is 10(Decimal), then the sequence of bytes is 0x0A, 0x00, 0x00, 0x00 *Reserved* (4 bytes, hex): All '0' *Channel* (1 byte, hex): Channel number of the Access Point. *SecurityMode* (1 byte, hex): 0x00– open 0x01- WPA 0x02- WPA2 0x03- WEP 0x04- WPA Enterprise, 0x05- WPA2 Enterprise *RSSIVal* (1 byte, hex): Absolute value of the RSSI information. It indicates the signal strength of the P2P node/Access Point. For example, if the RSSI is -20

Result Code	Description
	dBm, then the value reported is 0x14. *NetworkType* (1 byte, hex): 0x01 – Infrastructure *DeviceName* (32 bytes, ASCII) : SSID of the Access Point that the module scanned. 34 byte stream, filler bytes (0x00) are put to complete 34 bytes, if actual length is not 34 bytes. *BSSID* (6 bytes, hex): BSSID of the scanned access point *Reserved or SNR* (2 bytes, hex): The module should ignore this field if *bit_map[1]=0* in the command Feature Select. If *bit_map[1]=1,* the least significant byte is the value of the SNR. For example, if the SNR is 20dBm, the value reported is 0x14. The most significant byte should be ignored. CISCO_AP_NAME (16 bytes, ASCII): This field is present ONLY if bit_map[0]=1 in the command Feature Select. It contains the Cisco AP Name, and valid for Cisco Aironet Series of devices. It contains all '0' for other devices.
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFF8

For example, to scan all 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
```

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

```
O K <ScanCount=2> <Reserved>  <Channel1=0x06>
<SecurityMode1=0x00> <RSSIVal1=20> <Network type1=0x01> <Device
Name1=Redpine_net1> <BSSID1=0x00 0x23 0xA7 0x1F 0x1F 0x14>
<Reserved> <Channel2=0x0A> <SecurityMode2=0x02> <RSSIVal2=14>
<Network type2=0x01> <Device Name2=Redpine_net2> <BSSID2=0x00
0x23 0xA7 0x1F 0x1F 0x15> <Reserved> \r\n
```

```
0x4F    0x4B    0x02  0x00 0x00 0x00    0x00    0x00    0x00    0x00    0x06
0x00    0x14 0x01    0x52    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  0x00  0x00
0x23  0xA7  0x1F  0x1F  0x14  0x00 0x00  0x0A    0x02    0x0D    0x01
0x52    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  0x00  0x00  0x23  0xA7  0x1F
0x1F  0x15  0x00 0x00 0x0D  0x0A
```

Relevance

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

3.2.9 Join

Description

This command is used for following:

Associate to an access point (operating mode = 0 or 2)

Associate to a remote device in Wi-Fi Direct mode (operating mode 1)

Create an Access Point (operating mode 6)

Allow a third party to associate to a Wi-Fi Direct group created by the module

Command

at+rsi_join

Usage

at+rsi_join=device_name,TxRate,TxPower\r\n

Parameters

device_name (maximum of 32 bytes, ASCII):

When the module is in Operating modes 0 and 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 and 2, and wants to connect to an access point in WPS mode then the value of this parameter is a constant ASCII string WPS_SSID. (Refer "Join to a WPS enabled Access Point APPENDIX A: Sample Flow of Commands in UART).

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 *at+rsi_apconf*.

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.

TxRate (maximum of 2 bytes, ASCII): Rate at which the data has to be transmitted. Refer to the table below for the various data rates and the corresponding values. Set to 0 if *Group_Owner_intent* in "Configure Wi-Fi P2P" command is 16.

Data Rate (Mbps)	**Value of uTxDataRate**
Auto-rate	0
1	1
2	2
5.5	3
11	4
6	5
9	6
12	7
18	8
24	9
36	10
48	11
54	12

Data Rate (Mbps)	Value of uTxDataRate
MCS0	13
MCS1	14
MCS2	15
MCS3	16
MCS4	17
MCS5	18
MCS6	19
MCS7	20

Table 4: Data Rate Parameter

TxPower (1 byte, ASCII): 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 (15 +/- 2) dBm

At 5 GHz

0– Low power (5+/-1) dBm

1– Medium power (7 +/-1) dBm

2– High power (12 +/- 2) dBm

Response

Result Code	Description
OK<*Go_Status*>	Successful execution of the command. *GO_Status* (1 byte, hex): The value of this parameter varies with the firmware version used. Firmware version 1.1.0.1.0.0 or below: 0x00 – if the module becomes a Group Owner (GO) after the GO negotiation stage. 0x01 – if the module does not become a GO after the GO negotiation stage. Firmware version 1.2.1.1.1.0 or above:

Result Code	Description
	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. Note: The module gets a default IP of 192.168.100.76 if it becomes a Group Owner or Access Point.
ERROR<Error code>	Failure, Possible error codes are 0x0004, 0x0008, 0x0009, 0x000E, 0x0016, 0x0018, 0x0019, 0x001E, 0x0020, 0x0021, 0x0023, 0x0025, 0x0026, 0x002A, 0x002B, 0x002C, 0xFFF8

For example,

To associate to an Access Point named Test_AP, the following command is used.

at+rsi_join=Test_AP,0,2\r\n

………

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6A 0x6F 0x69 0x6E 0x3D 0x54 0x65 0x73 0x74 0x5F 0x41 0x50 0x2C 0x30 0x2C 0x32 0x0D 0x0A

To associate to a WPS enabled Access Point.

at+rsi_join=WPS_SSID,0,2\r\n

……….

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6A 0x6F 0x69 0x6E 0x3D 0x57 0x50 0x53 0x5F 0x53 0x53 0x49 0x44 0x2C 0x30 0x2C 0x32 0x0D 0x0A

Response:

After successful join(with Access Point or WiFi Direct Group owner) in client mode.

OKC\r\n

0x4F 0x4B 0x43 0x0D 0x0A

After successfully became Access Point or WiFi Direct Group owner

OKG\r\n

0x4F 0x4B 0x47 0x0D   0x0A

Relevance

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

When the module is in Wi-Fi Direct Mode in Operating Mode 1, this command initiates a Group Owner (GO) negotiation and subsequent association to a Wi-Fi Direct node using the WPS push button method. A Wi-Fi Direct node need not supply a separate password to join to the module. In Operating Mode 0 and 2, it initiates an authentication and association process with an Access Point.

3.2.10 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 issues ERROR 37\r\n. The module aborts the re-join after a fixed number of re-tries. If this happens, an asynchronous response "ERROR 8\r\n" or ""ERROR 25\r\n" is sent from the module to the Host. This is not a response to a command, but an asynchronous message sent to the Host if re-tries fail.

Command

N/A

Usage

N/A

Parameters

N/A

Response

Result Code	**Description**
ERROR	Asynchronous Message with error value

For example,

Command:

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 messages to indicate rejoin failure to host.
ERROR<Error code=8>\r\n
……………………..
0x45 0x52 0x52 0x4F 0x52 0x08 0x00 0x0D 0x0A

Or

ERROR<Error code=25>\r\n
……………………..
0x45 0x52 0x52 0x4F 0x52 0x19 0x00 0x0D 0x0A

Relevance

This command is relevant when the module is configured in Operating Modes 0, 2, 3 and 5.

3.2.11 Set Sleep Timer

Description

This command configures the timer for power save operation.

Command

at+rsi_sleeptimer

Usage

at+rsi_sleeptimer=timer_val\r\n

Parameters

timer_val (maximum of 5 bytes, ASCII): Value of the timer in seconds. Maximum value is 65000 (decimal). Default value is 1 second.

Response

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

Result Code	Description
	Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFF8

Relevance

This command is relevant when the module is configured in Operating Mode 0, 2, 3 or 5.

3.2.12 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 *Init* command.

Command

at+rsi_pwmode

Usage

at+rsi_pwmode=power_val\r\n

Parameters

power_val (1 byte, ASCII):

0–Mode 0: Disable Power save mode

1–Power Save Mode 1

2–Power Save Mode 2

Response

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

Relevance

This command is relevant when the module is configured in Operating Mode 0, 2, 3 or 5.

3.2.12.1 Power save Operation

The behavior of the module differs according to the power save mode it is put in.

3.2.12.1.1 Power save Mode 1

Once the module is put to power save mode 1, it wakes itself up whenever the sleep timer expires (*at+rsi_sleeptimer*). After waking up, the module sends an ASCII string WKP\r\n (0x57 0x4B 0x50 0x0D 0x0A) to the host. After giving WKP message, if it doesn't have any other operations to do, it will give SLP\r\n (0x53 0x4C 0x50 0x0D 0x0A) to the Host immediately. This SLP message is to indicate host that the module is ready to go to sleep. If the Host has any data to transmit, it can execute corresponding commands or data. Once the module processed the commands or data given from host, it will give a SLP message to host again.Now, the host can put the module back to sleep by sending the string ACK\r\n.

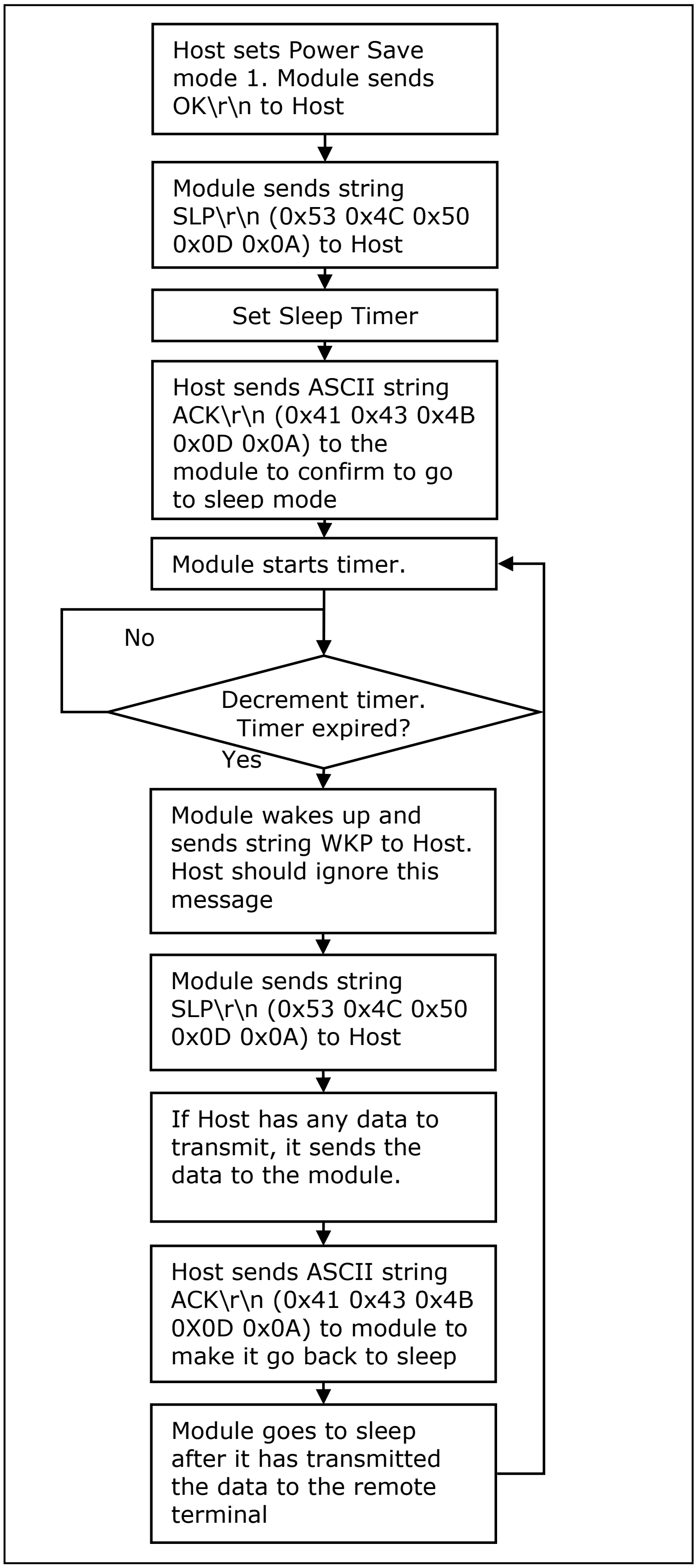

Host sets Power Save mode 1. Module sends OK\r\n to Host
Module sends string SLP\r\n (0x53 0x4C 0x50 0x0D 0x0A) to Host
Set Sleep Timer
Host sends ASCII string ACK\r\n (0x41 0x43 0x4B 0x0D 0x0A) to the module to confirm to go to sleep mode
Module starts timer.
No
Decrement timer. Timer expired?
Yes
Module wakes up and sends string WKP to Host. Host should ignore this message
Module sends string SLP\r\n (0x53 0x4C 0x50 0x0D 0x0A) to Host
If Host has any data to transmit, it sends the data to the module.
Host sends ASCII string ACK\r\n (0x41 0x43 0x4B 0X0D 0x0A) to module to make it go back to sleep
Module goes to sleep after it has transmitted the data to the remote terminal

Figure 4: Setting Power Save Mode 1

After having put the module to power save mode, the Host can issue subsequent commands only after the module has indicated to the Host that it has woken up. The module can however always receive data from the remote terminal at any point of time and can send the data to the Host.

3.2.12.1.2 Power save Mode 2

Once the module is put to power save mode 2, it can be woken up by the Host. The Host needs to send a dummy character in its UART TX line till the module wakes up. For example, a character that can be sent from the Host is A\r\n. Once the module wakes up, it sends an ASCII string WKP\r\n to the Host to indicate that it has woken up. . After giving WKP message, if it doesn't have any other operations to do, it will give SLP\r\n (0x53 0x4C 0x50 0x0D 0x0A) to the Host immediately. This SLP message is to indicate host that the module is ready to go to sleep. If the Host has any data to transmit, it can execute corresponding commands or data. Once the module processed the commands or data given from host, it will give a SLP message to host again. Now, the host can put the module back to sleep by sending the string ACK\r\n.

The Host should give commands to operate the module only when it is awake. The module can however always receive data from the remote terminal at any point of time and can send the data to the Host.

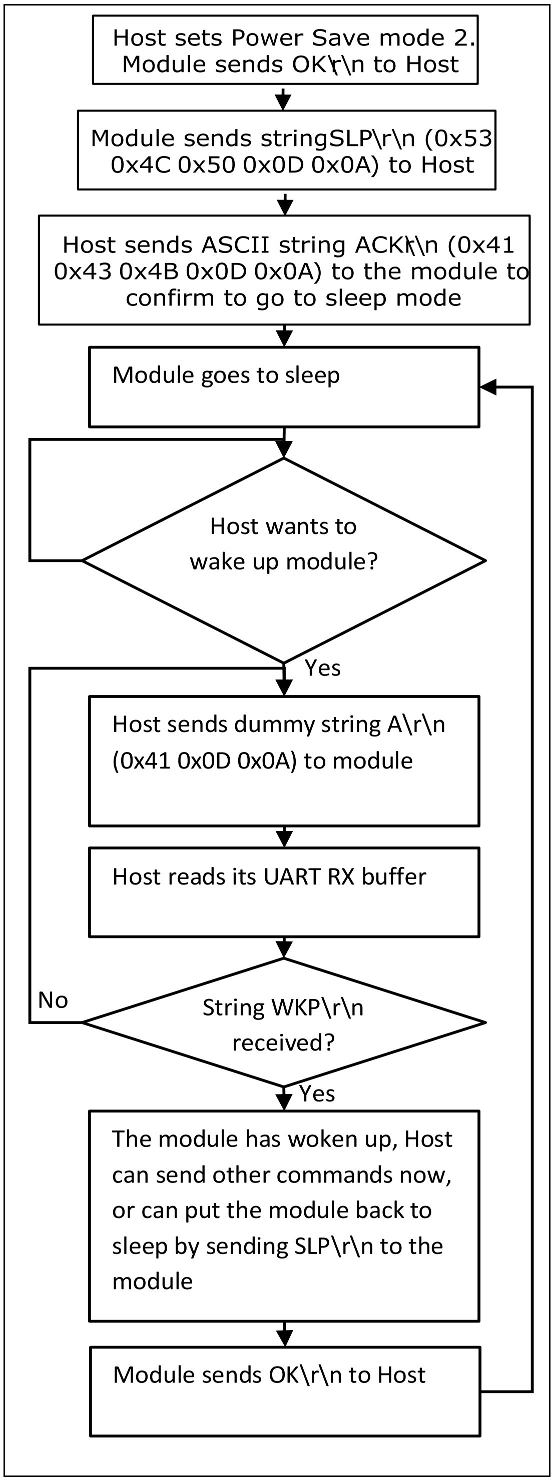

Host sets Power Save mode 2.
Module sends OK\r\n to Host
Module sends stringSLP\r\n (0x53 0x4C 0x50 0x0D 0x0A) to Host
Host sends ASCII string ACK\r\n (0x41 0x43 0x4B 0x0D 0x0A) to the module to confirm to go to sleep mode
Module goes to sleep
Host wants to wake up module?
Yes
Host sends dummy string A\r\n (0x41 0x0D 0x0A) to module
Host reads its UART RX buffer
No
String WKP\r\n received?
Yes
The module has woken up, Host can send other commands now, or can put the module back to sleep by sending SLP\r\n to the module
Module sends OK\r\n to Host

Figure 5: Power Save Mode 2

Note: WiSeConnect doesn't support power save modes while operating in AP or group owner mode.

3.2.13 Pre Shared Key

Description

The command is used to set the PSK (Pre shared key) to join to WPA/WPA2-PSK enabled APs. This command should be issued to the module before the *Join* command if the AP is in secure mode. It can be ignored if the AP is in Open mode.

Command

at+rsi_psk

Usage

at+rsi_psk=pre_shared _key\r\n

Parameters

pre_shared_key (maximum of 63 bytes, ASCII): Pre shared key of the AP to which the module wants to associate.

Response

Result Code	Description
OK	Successful execution of the command
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x0028, 0x002C, 0xFFF8

Example:

Command:

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

at+rsi_psk=12345ABCDE\r\n

…………………………………………….

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

Response:

OK\r\n

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

Relevance

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

3.2.14 Set WEP Key

Description

This command configures the WEP key in the module to connect to an AP with WEP security.

Command

```
at+rsi_wepkey
```

Usage

```
at+rsi_wepkey=key_index,key1,key2,key3,key4\r\n
```

Parameters

Key_index(1 byte, ASCII): 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.

Key1, key2, key3, key4: 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.

<u>WEP (Hex Mode):</u> In this mode, the key to be supplied to the AP should be 10 digits (for 64 bit WEP mode) or 26 digits (for 128 bit WEP mode), and only the following digits are allowed for the key: A,B,C,D,E,F,a,b,c,d,e,f,0,1,2,3,4,5,67,8,9.

Following are the examples,

Command:

Example 1:

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

Example 2:

If the user wants to enter only one valid key

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

Example 3:

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
```

Response

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

Relevance

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

3.2.15Set WEP Authentication Mode

Description

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

Command

```
at+rsi_authmode
```

Usage

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

Parameters

auth_mode (1 byte, ASCII):

0-Open WEP authentication

1-Shared WEP authentication

Response

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

For example,

Command:

```
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
```

Relevance

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

3.2.16 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.

Command

```
at+rsi_eap
```

Usage

```
at+rsi_eap =eap_method, inner_method,user_identity,password\r\n
```

Parameters

eap_method (maximum of 4 bytes, ASCII): EAP authentication method. Valid values are TLS, TTLS, PEAP and FAST sent as an ASCII string.

Inner_method: Inner method used in TTLS, PEAP or FAST. This parameter is not used in case of TLS. The value can be set to MSCHAPV2 in all cases, including TLS, where it will not be used.

User identity (maximum of 64 bytes, ASCII): User identity. This is present in the user configuration file in the radius sever.

Password (maximum of 128 bytes, ASCII): password in ASCII format. This should be same as the password in the user configuration file in the Radius Server for that User Identity.

Response

Result Code	Description
OK	Successful execution of the command
ERROR<Error code>	Failure, Possible error codes are 0x001C, 0x0021, 0x0025, 0x002C, 0xFFF8

Example: at+rsi_eap=TTLS,MSCHAPV2,user1,user1pass\r\n

…………………………………………….

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x65 0x61 0x70 0x3D
0x54 0x54 0x4C 0x53 0x2C 0x4D 0x53 0x43 0x48 0x41 0x50
0x56 0x032 0x2C 0x75 0x73 0x65 0x72 0x31 0x2C 0x75 0x73
0x65 0x72 0x31 0x70 0x61 0x73 0x73 0x0D  0x0A
```

Relevance

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

3.2.17 Set Certificate

Description

This command is used to load the certificate or PAC file, after issuing the *at+rsi_eap* command. This command should be issued if the security mode is EAP-TLS or EAP-FAST

Command

at+rsi_cert

Usage

at+rsi_cert =cert_type,cert_len,key_password,certificate\r\n

Parameters

cert_type: Type of the certificate.

1–TLS client certificate

2–FAST PAC file

cert_len (variable size, ASCII): Length of the certificate in number of bytes, sent in ASCII format. Maximum length of certificate is 6522.

If this value is put to ‘0[1]’, the following are applicable:

There are two modes of using EAP-TTLS or PEAP: Password based authentication and server based authentication. If the user is using password based authentication, then this parameter should be set to ‘0’. If the user is using server based authentication, then the correct length of the parameter *certificate* should be supplied in this command and the appropriate CA (Certification Authority) Root file shou*ld be suppl*ied to the *certificate* parameter.

In general a value of ‘0’ can be used to clear the current certificate in the module’s memory. The remaining parameters *key_password, certificate* need not be supplied in such a case.

key_password: Private key password, used to generate the certificate

certificate: TLS certificate in TLS mode, PAC file in ASCII format in EAP-FAST mode. In EAP- TTLS and PEAP modes, this *cert_len* is not ‘0’, then parameter should contain the file with the CA Root information.

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 name 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)
```

Response

[1] Value of ‘0’ is supported from firmware version 2.0.0.1.2.4 onwards

Result Code	Description
OK	Successful execution of the command
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x0029, 0x002C, 0xFFF8

Relevance

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

3.2.18 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, 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

at+rsi_disassoc

Usage

at+rsi_disassoc=mode,mac_addr\r\n

Parameters

mode:

0-Module is in client mode. The second parameter *mac_addr* need not be supplied if mode is 0.

1-Module is in AP mode

mac_addr: MAC address of the client to disconnect.

Example 1: 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

Example 2: 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
```

Response

Result Code	Description
OK	Successful execution of the command
ERROR<Error code>	Failure, Possible error codes are 0x0006, 0x0013, 0x0021, 0x0025, 0x002C, 0xFFF8

Relevance

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

> Note:
>
> During rejoin, if user wants to connect to another AP, diassoc has to be issued although an error 0x0006 will be given and after handling this error user has to start from init command.
>
> 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.
>
> When the module is operating in AP mode, station may get disconnected from AP because of some other reasons other than disconnect (Station is idle and so). No asynchronous response will be given to host upon a station removal. Host has to issue “GO params” command to know the connected stations information.

3.2.19 Set IP Parameters

Description

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

Command

```
at+rsi_ipconf
```

Usage

```
at+rsi_ipconf=DHCP_MODE,IP address,Subnet,Gateway\r\n
```

Parameters

DHCP_MODE (1 byte, ASCII): Used to configure TCP/IP stack in manual or DHCP modes.

0– Manual

1– DHCP enabled

IP address (variable length, ASCII): IP address in dotted decimal format. This can be 0's if DHCP is enabled.

Subnet (variable length, ASCII): Subnet mask in dotted decimal format. This can be 0's if DHCP is enabled.

Gateway (variable length, ASCII): Gateway in the dotted decimal format. This can be 0's if DHCP is enabled.

Example 1: 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  0x31    0x2E  0x31  0x36  0x38  0x2E  0x31  0x2E  0x31  0x0D
0x0A
```

Example 2: 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

Result Code	Description

OK<MAC_Address><IP_Address><
Subnet_Mask><Gateway>
	MAC Address (6 Bytes, hex):
	MAC address of the module
	IP Address (4 Bytes, hex):
	IP address of the module
	Subnet_Mask (4 Bytes, hex):
	Subnet mask
	Gateway(4 Bytes, hex):
	Subnet mask

ERROR<Error code>	Failure, Possible error codes are

Result Code	Description
	0x0021, 0x0025, 0x002C, 0xFFFC, 0xFF9C, 0xFF9D, 0xFFF8

Relevance

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

Note: When WiSeConnect operating in client mode and its IP is assigned using DHCP, an asynchronous error will come to host in case of DHCP renewal failure. When this error comes, user need to issue “disassoc” or “reset” command to join network and get an IP again.

3.2.20 Open a TCP Socket

Description

This command opens a TCP client socket and attempts to connect it to corresponding server TCP socket. A server TCP socket should be created in the remote terminal before issuing this command.

Command

```
at+rsi_tcp
```

Usage

```
at+rsi_tcp=dipaddr,dport,lport\r\n
```

Parameters

dipaddr (variable length, ASCII)– IP Address of the Target server

dport (variable length, ASCII)– destination port number. Value ranges from 1024 to 49151

lport (variable length, ASCII)– local port number in the module. Value ranges from 1024 to 49151

Response

Result Code	Description
OK<socket_type><socket_handle><lport><module_ipaddr>	*socket_type* (2 bytes, hex): Indicates the type of socket. The least significant byte is returned first. 0x0000 –Indicates TCP client socket *socket_handle* (2 bytes, hex): Upon

Result Code	Description
	successfully opening and connecting the TCP socket to the Host port, a socket handle is returned. The least significant byte is returned first. In operating mode 0 and 2: *socket_handle* ranges from from 1 to 8. The first socket opened will have a socket handle of 1, the second socket will have a handle of 2 and so on. In operating mode 1: If the module is GO or Access Point, then *socket_handle* ranges from from 2 to 8. The first socket opened will have a socket handle of 2, the second socket will have a handle of 3 and so on. *Lport* (2 bytes, hex): Port number of the socket in the module. The least significant byte is returned first. *Module_ipaddr* (4 bytes, hex): Module's IP address. The most significant byte is returned first. For example,
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFFE, 0xFF80, 0xFF81, 0xFF85, 0xFF87, 0xFFA1, 0xFFF8

Example 1

at+rsi_tcp=192.168.40.10,8000,1234\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 0x30 0x2C 0x31 0x32 0x33 0x34 0x0D 0x0A

OK <socket_type> <socket_handle> <dipaddr> <dport>\r\n

…………………….

0x4F 0x4B 0x00 0x00 0x01 0x00 C4 09 C0 A8 64 67 0x0D 0x0A

Relevance

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

3.2.21 Open a Listening TCP Socket

Description

This command opens a server/listening TCP socket in the module. Once the listening socket is open, it accepts remote connect requests from client sockets. Only one connection can be established on a single invocation of this command.

If multiple connections on a port have to be established, then the same command has to be invoked another time.

a. Open the first LTCP socket in module (for example port no. 8001)
b. Socket handle returned for this socket would be 1 (if module is in operating mode 0 or 2) or 2 (if module is in operating mode 1 or 6).
c. Connect this socket to the remote peer socket
d. You can now open the second socket in module with the same port no. 8001
e. Socket handle returned for the new socket would be 2 (if module is in operating mode 0 or 2) or 3 (if module is in operating mode 1 or 6)
f. Connect this socket to another remote peer socket

Command

at+rsi_ltcp

Usage

at+rsi_ltcp=lport\r\n

Parameters

lport (variable length, ASCII)– Port number of the listening socket in the module. Value ranges from 1024 to 49151

Response

Result Code	Description
OK<socket_type> <socket_handle><lport> <module_ipaddr>	*socket_type* (2 bytes, hex): Indicates the type of socket. The least significant byte is returned first. 0x0002 –Indcates listening/server TCP socket *socket_handle* (2 bytes, hex): Upon successfully opening the socket, a socket handle is returned. The least significant byte is returned first. Operating mode 0 and 2: *socket_handle* ranges from from 1 to 8. The first socket opened will have a socket handle of 1, the second socket will have a handle of 2 and so on.

Result Code	Description
	Operating mode 1: If the module is GO or Access Point, then *socket_handle* ranges from from 2 to 8. The first socket opened will have a socket handle of 2, the second socket will have a handle of 3 and so on. *Lport* (2 bytes, hex): Port number of the socket in the module. *Module_ipaddr* (4 bytes, hex): Module's IP address.
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFFE, 0xFF80, 0xFF81, 0xFF85, 0xFFA1, 0xFF2D, 0xFFF8

```
at+rsi_ltcp=8000\r\n

…………………………………………………………………….

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6C 0x74 0x63 0x70 0x3D 0x38
0x30  0x30 0x30  0x0D 0x0A

OK <socket_type=0x0002> <socket_handle=0x0001> \r\n
…………………….
0x4F 0x4B 0x02 0x00 0x01  0x00 0x0D 0x0A
```

Relevance

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

> NOTE: A maximum of 8 sockets can be opened in operating mode 0 and 2, with any combination of TCP or UDP sockets can be operational at a time. The first socket opened has a socket handle of 1, while the last socket has a socket handle of 8.
>
> A maximum of 7 sockets can be opened in operating mode 1. The first socket opened has a socket handle of 2, while the last socket has a socket handle of 8.

3.2.22 Open a Listening UDP Socket

Description

This command opens a UDP socket and binds to a specified port. The UDP socket waits for the data from the peer. This socket is not connected to any peer and is used if the user wants to receive/send data from/to any peer.

Command

at+rsi_ludp

Usage

at+rsi_ludp=lport\r\n

Parameters

lport (variable length, ASCII)– Port number of the listening socket in the module. Value ranges from 1024 to 49151

Response

Result Code	**Description**
OK<socket_type> <socket_handle><lport> <module_ipaddr>	*socket_type*(2 bytes, hex): Indicates the type of socket. The least significant byte is returned first. 0x0004 –Indicates listening UDP socket *socket_handle* (2 bytes, hex): Upon successfully opening UDP socket, a socket handle is returned. The least significant byte is returned first. Operating mode 0,2: *socket_handle* ranges from from 1 to 8. The first socket opened will have a socket handle of 1, the second socket will have a handle of 2 and so on. Operating mode 1: If the module is GO or Access Point, then *socket_handle* ranges from from 2 to 8. The first socket opened will have a socket handle of 2, the second socket will have a handle of 3 and so on. *Lport*(2 bytes, hex): Port number of the socket in the module. *Module_ipaddr* (4 bytes, hex): Module's IP address.
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFFE,

Result Code	Description
	0xFF80, 0xFF81, 0xFF85, 0xFFA1, 0xFFF8

Example:

at+rsi_ludp=8000\r\n

…………………………….

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6C 0x75 0x64 0x70 0x3D 0x38 0x30 0x30 0x30 0x0D 0x0A

OK <socket_type=0x0004> <socket_handle=0x0001> \r\n
…………………….
0x4F 0x4B 0x04 0x00 0x01 0x00 0x0D 0x0A

Relevance

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

3.2.23Open a UDP Socket

Description

This command opens a UDP socket and links to the remote system‗s specific host and port address. The UDP socket is virtually connected to the peer specified by the IP and the port.

Command

at+rsi_udp

Usage

at+rsi_udp=dipaddr,dport,lport\r\n

Parameters

dipaddr(variable length, ASCII) – IP Address of the Target server

dport (variable length, ASCII)– destination port. Value ranges from 1024 to 49151

lport – Local port on the module. Value ranges from 1024 to 49151

Response

Result Code	Description
OK<socket_type><socket_handle><lport><module_ipaddr>	*socket_type* (2 bytes, hex): Indicates the type of socket. The least significant byte is returned first.

Result Code	Description
	0x0001 –Indicates UDP socket. *Socket_handle* (2 bytes, hex): Upon successfully opening UDP socket, a socket handle is returned. The least significant byte is returned first. Operating mode 0,2: *socket_handle* ranges from 1 to 8. The first socket opened will have a socket handle of 1, the second socket will have a handle of 2 and so on. Operating mode 1: If the module is GO or Access Point, then *socket_handle* ranges from from 2 to 8. The first socket opened will have a socket handle of 2, the second socket will have a handle of 3 and so on. *Lport*: Port number of the socket in the module. Returned in hex, 2 bytes. *Module_ipaddr* : Module's IP address. Returned in hex, 4 bytes
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFFE, 0xFF80, 0xFF81, 0xFF85, 0xFFA1, 0xFFF8

Example:

at+rsi_udp=192.168.40.10,8000,1234\r\n

…………………………………………………………………….

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x75 0x64 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 0x0D 0x0A

OK <socket_type> <socket_handle> <dipaddr> <dport>\r\n
…………………….
0x4F 0x4B 0x02 0x00 0x01 0x00 C4 09 C0 A8 64 67 0x0D 0x0A

Relevance

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

3.2.24 Query a Listening Socket's Active Connection Status

Description

This command is issued when a listening/server TCP socket has been opened in the module, to know whether the socket got connected to a client socket.

Command

at+rsi_ctcp

Usage

at+rsi_ctcp=socket_handle\r\n

Parameters

socket_handle (1 byte, ASCII)– Socket handle for an already open listening TCP socket (LTCP) in the module.

Response

Result Code	**Description**
OK<*socket_handle*> *<IP_Address><Port>*	*socket_handle* (2 bytes, hex): Socket handle of an active LTCP socket in the module. The least significant byte is sent first.
ERROR -1	Listening TCP socket in module is not connected to the remote peer, or the LTCP socket is not yet opened in the module Other possible error codes are 0x0021, 0x0025, 0x002C, 0xFFF8

Example 1:

at+rsi_ctcp=1\r\n

…………………………………………………………………….

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x63 0x74 0x63 0x70 0x3D 0x31 0x0D 0x0A

Example 2:

OK<socket_handle=7><IP_Address=192.168.40.10><Port=8001> \r\n

…………………….

0x4F 0x4B 0x07 0xC0 0xA8 0x28 0x0A 0x41 0x1F 0x0D 0x0A

Relevance

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

3.2.25 Close a Socket

Description

This command closes a TCP/UDP socket in the module.

Command

at+rsi_cls

Usage

at+rsi_cls=socket_handle\r\n

Parameters

socket_handle (1 byte, ASCII): Socket handle of an already open socket.

For example, to close the socket with handle 1, the command is

at+rsi_cls=1\r\n

……………………..

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x62 0x6C 0x73 0x3D 0x31 0x0D 0x0A

Response

Result Code	Description
OK<socket_handle>	*socket_handle* (2 bytes, hex): socket handle of the socket that is closed.
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFF40, 0xFFF8

Relevance

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

NOTE:

In the case of TCP socket, when a remote peer closes the socket connection, the module sends the "AT+RSI_CLOSE<socket_handle>\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, hex. The least significant byte is returned first. AT+RSI_CLOSE is returned in uppercase and ASCII format.

When TCP keep alive time out happens, socket will be closed and AT+RSI_CLOSE indication will be given to host. But, no error will be returned with this asynchronous response.

When module operating in AP mode, If a station is disconnected or removed from AP, all the sockets connected to that particular station will be closed and asynchronous socket close (AT+RSI_CLOSE) will be given to host. But, no error will be returned with this asynchronous response.

3.2.26 Send Data to a Socket

Description

This command sends a byte stream of a certain size to the socket specified by the socket handle.

Command

at+rsi_snd

Usage

at+rsi_snd=socket_handle,data_len,dipaddr,dport, data_stream\r\n

Parameters

socket_handle (1 byte, ASCII)– Socket handle of the socket over which data is to be sent.

data_len(variable length, ASCII) – Length of the data that is getting transmitted, wrong parameter may cause module hang in some cases.

Dipaddr (variable length, ASCII)– Destination IP Address. Should be ‘0’ if transacting on a TCP socket

dport (variable length, ASCII)– Destination Port. Should be ‘0’ if transacting on a TCP socket

data_stream (maximum length of 1400 bytes, hex)– Actual data to be sent to be sent to the specified socket.

Response

Result Code	**Description**
OK<length>	2 bytes length, length of data sent

Result Code	Description
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 0x0021, 0x0025, 0x002C, 0xFFF7, 0xFFF8

For example 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
```

For example 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 0x2C 0x2C 0x61 0x62 0x63 0x64 0x65 0x66 0x67
0x68 0x69 0x6A 0x0D 0x0A
```

For example, for 250 bytes sent, the response is

```
OK 250\r\n
```

………………….

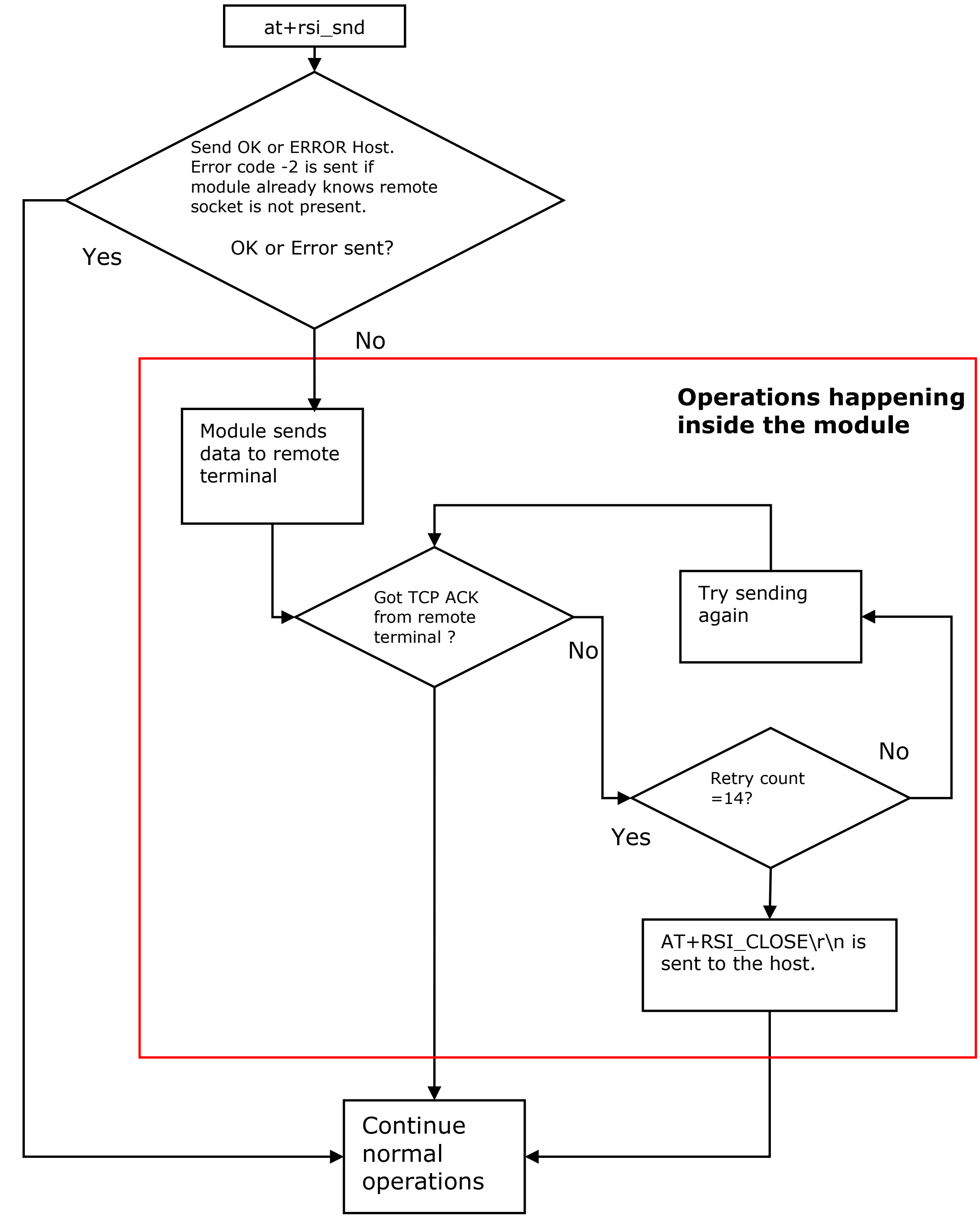
0x4F 0x4B 0xFA 0x00 0x0D 0x0A

NOTE: The parameter *data_stream* contains the actual data and not the ASCII representations of the data.

IMPORTANT : User need to consider following for “snd” command in case of UART and USB mode.

User will get an immediate “OK\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\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.



Some common reasons for not getting a TCP ACK from the remote terminal:

- Abnormal destruction of the remote network, plugging out network cables etc.

Figure 6: 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 is the only command that requires 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 5: Byte Stuffing

3.2.27 Receive Data on a Socket

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.

Command

N/A

Usage

N/A

Parameters

N/A

Response

Result Code	Description
AT+RSI_READ<socket_handle><size><sipaddr><Sport><stream>	Asychronous message AT+RSI_READ(returned in upper case) socket_handle (2 bytes, hex) – socket handle of the socket over which the data is received. The least significant byte is returned first. size (2 bytes, hex) – Number of bytes received. Size = 0 indicates remote termination for a TCP socket. The least significant byte is sent first. For example, 900 bytes (0x0384) would be sent as <0x84> <0x03> sipaddr (4 bytes, hex) – Source IP address. This field is not present in the message if the data is received over a TCP socket. Sport (2 bytes, hex) – Source port. This field is not present in the message if the data is received over

Result Code	Description
	a TCP socket. Stream – 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.

Example 1, if 'abcd' is sent from remote terminal to module, on an UDP socket with handle 1, from source ip 192.168.1.20 and source port 8001, the module sends the following response to the host.

AT+RSI_READ 1 4 192 168 1 1 8001 abcd \r\n

………………………..

0x41 0x54 0x2B 0x52 0x53 0x49 0x5F 0x52 0x45 0x41 0x44 0x01 0x00 0x04 0x00 0xC0 0xA8 0x01 0x01 0x41 0x1F 0x61 0x62 0x63 0x64 0x0D 0x0A

Example 2, 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 1 4 abcd \r\n

………………………..

0x41 0x54 0x2B 0x52 0x53 0x49 0x5F 0x52 0x45 0x41 0x44 0x01 0x04 0x00 0x61 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).

Relevance

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

3.2.28 Load Web Page in Module

Description

The module has an embedded Web Server and can respond to HTTP Get and Post requests from a remote terminal. This command is used to load a user defined web page on the module. If a new webpage is loaded, it overwrites the old webpage previously present in the module's memory.

Command

at+rsi_webpage=total_webpage_len,current_chunk_len,more_chunks,webpage_data\r\n

Parameters

total_webpage_len– The total length of the characters in the source code of the webpage. The maximum value of this parameter is 3 KB.

current_chunk_len– Total number of characters in the current segment.

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_data- This is the actual source code of the current segment .

Segments are created when the overall length of the source code of the web page is more than 1400 characters.

Example 1: The source code of a reference page (91 characters in all).

<html><head><title>Untitled Document</title></head><body><h1>Hello World</h1></body></html>

This can be sent in the command as

at+rsi_webpage=91,91,0,<source code>

…………………………………..

```
0x61  0x74  0x2B  0x72  0x73      0x69   0x5F  0x77  0x65  0x62  0x70
0x61  0x67  0x65  0x3D  0x39  0x31 0x02C  0x39  0x31  0x2C  0x30 0x2C
0x3C  0x68  0x74  0x6D  0x6C  0x3E …….0x0D  0x0A
```

Example 2:

If the web page source code is of 3000 characters, the Host should send it through 3 segments, the first two of 1024 bytes, and the last one of 952 bytes as shown:

at+rsi_webpage=<total_webpage_len=3000>,<current_chunk_len=1024>,<more_chunks=1>,<webpage_data=data of the 1st segment>\r\n

Receive OK and send next command

at+rsi_webpage=<total_webpage_len=3000>,<current_chunk_len=1024>,<more_chunks=1>,<webpage_data=data of the 2nd segment>\r\n

Receive OK and send next command

at+rsi_webpage=<total_webpage_len=3000>,<current_chunk_len=952>,<more_chunks=0>,<webpage_data=data of the 3rd segment>\r\n

Response

Result Code	Description
OK	Success
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFF8

Relevance

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

The web-server in the module can be bypassed. The user can implement a web server at the Host and communicate with the module through port number 80. User has to set the corresponding feature bit in feature select command.

3.2.28.1 Web Server Functionality with Multiple pages

The section Load Web Page in Module describes how to load a single web page into the module. There might be use cases in which more than a single page may be required to service. In such a case, these pages can be stored in the Host memory and can be sent to the module one at a time.The below process shows the mechanism.

1. Remote terminal (Laptop) connects to the module.
2. Remote terminal queries for a particular web page by typing the URL in its browser
3. Module receives the query and checks if it already has the page in its memory. If yes, it sends out the page to the remote terminal and the query is serviced. If the page is currently not in the module's memory, it sends the following asynchronous message to Host.

   AT+RSI_URLREQ<url_length><url_name>\r\n

   url_length(2 bytes, hex)- This is the number of characters in the requested URL. The least significant byte is sent first. For example, if the total length is 10 characters, the values sent are 0x0A 0x00.

   url_name (ASCII)- This is the actual url name.

   Note that the message AT+RSI_URLREQ is in uppercase letters.

4. The Host , after receiving this asynchronous message from the module, should fetch the page from its memory and give it back to the module with the message

   AT+RSI_URLRSP=total_len,chunk_len,more_chunks,webpage\r\n

   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, and the remaining parameters need not be sent.

   chunk_len- Total number of characters in the current segment.

   more_chunks(1 byte)-

   '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

   Segments are created when the overall length is more than 1400 characters.

   Example 1: If the web page source code is of 3000 characters, the Host should send it through 3 segments, the first two of 1024 bytes, and the last one of 952 bytes as shown:

   AT+RSI_URLRSP=<total_len=3000>,<chunk_len=1024>,<more_chunks=1>,<webpage=code of the 1st segment>\r\n.

   AT+RSI_URLRSP=<total_len=3000>,<chunk_len=1024>,<more_chunks=1>,<webpage=code of the 2nd segment>\r\n

   AT+RSI_URLRSP=<total_len=3000>,<chunk_len=952>,<more_chunks=0>,<webpage=code of the 3rd segment>\r\n

   Example 2: If the queried web page is not found in the Host, it sends

   AT+RSI_URLRSP=0\r\n

5. After all the segments are sent, the module aggregates them, stores in the internal memory and dispatches the page to the remote terminal.

   Note that only the page for the requested URL should be supplied by the Host to the module. The maximum allowed size of such page is 3 Kilo bytes. Only one such page is allowed at a time. If remote requests for one more host webpage, before the completion of previous request then the new URL request will not be indicated to host and "Not Found" will be the response for remote peer request. It applies the same for the SPI case also.

3.2.29 Load Web Fields in Module

Description

The command provides an incremental way for the Host to update data in designated fields of an already loaded webpage.

Command

```
at+rsi_webfields=1;data1,2;data2,….upto 10;data10\r\n
```

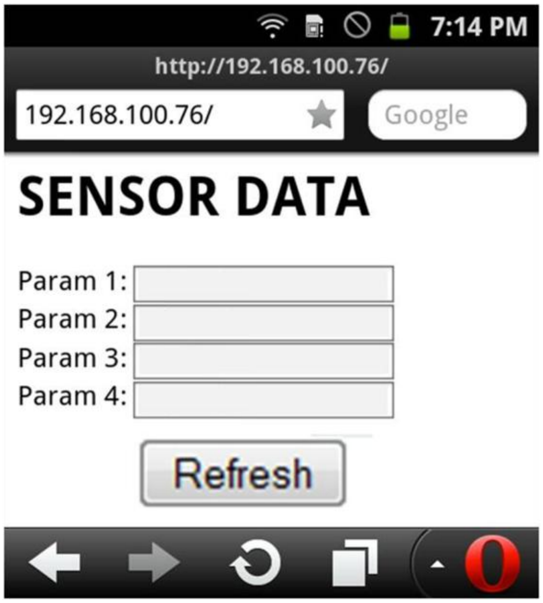
Parameters

data(n): Dynamic data

For example, below is the source code of a page with configurable fields, that is loaded into the module.

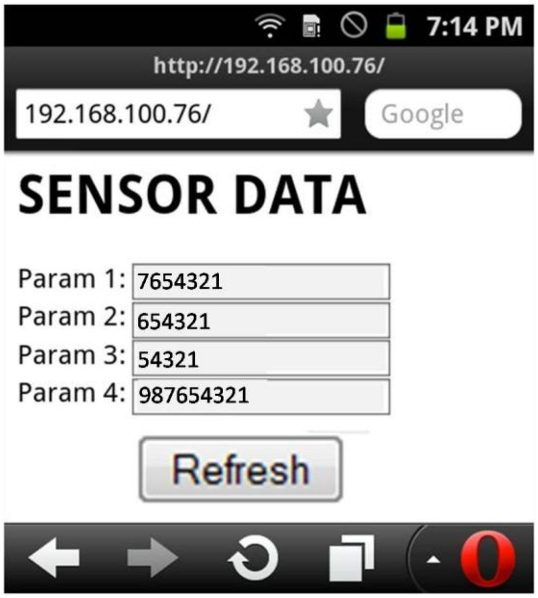
```
at+rsi_webpage=487,487,0,<html><body><script
type="text/javascript">function
reloadPage(){window.location.reload()}</script></head><body><C><h1>
<B> SENSOR DATA</b></h1><form >Param 1: <input type="text"
name="param1" value="%#1--#%" /><br />Param 2: <input type="text"
name="param2" value="%#2-#%" /><br/>Param 3: <input type="text"
name="param3" value="%#3#%" /><br />Param 4: <input type="text"
name="param4" value="%#4----#%" /><br /><input type="button"
value="Refresh" onclick="reloadPage()" /><C></body></html>\r\n
```

When the page is opened in a remote terminal, it would show the below



New values for the variables Param1 to Param4 can be loaded from the Host as at+rsi_webfields=1;87654321,2;8654321,3;654321,4;54321. The numbers 1,2, 3 and 4 correspond to the parameter index highlighted in green in the source code of the web page. If the user now refreshes the page in the remote terminal, the following will appear:

at+rsi_webfields=1;7654321,2;654321,3;54321,4;987654321



SENSOR DATA

Param 1: 7654321
Param 2: 654321
Param 3: 54321
Param 4: 987654321

Notes:

1. The identifier for the parameters (highlighted in green in the source code of the web page above) should range from "%#1 #%" to "%#10 #%". Other characters are not allowed. The module parses for these identifiers, they should not be present in any other part of the HTML code.
2. The length of the field is determined from the first '%' to the last '%'. For example, for the 4th parameter to be 7 characters long, %#4--#%. Similarly, for the parameter to be 15 characters long, %#4----------#% should be used. A maximum of 10 configurable parameters are allowed, the maximum length of each parameter is 64 characters and the minimum length is 5 characters.
3. To update the values of the parameters, a new value only with the designed length should be sent. For example, if the fourth parameter was configured as 7 characters by putting name="param4" value="%#4--#%" then

   at+rsi_webfields=4;1234567\r\n can be sent. The new value 1234567 is of a length of 7 characters, same as the length configured for Param4.

Response

Result Code	Description
OK	Success
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C,

Result Code	Description
	0xFFF8

Relevance

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

NOTE: In any operating mode, a remote terminal can access the webpage stored in the module by typing http://xxx.xxx.xxx.xxx (this is the IP address assigned to the module) in its browser.

3.2.30 DNS Server

Description

This command is used to provide to the module the DNS server's IP address. This command should be issued before the "DNS Resolution" command and after the "Set IP Parameters" command. Refer "Associate to an Access Point (with WPA2-PSK security) as a client" in APPENDIX A: Sample Flow of Commands in UART.

Command

at+rsi_dnsserver

Usage

at+rsi_dnsserver=dnsmode, ipaddress_primary_DNS_server, ipaddress_Secondary_DNS_server\r\n

Parameters

dnsmode:

1-The module can obtain a 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;

ipaddress_Primary_DNS_server: This is the IP address of the Primary DNS server to which the DNS Resolution query is sent. Should be set to '0' if *dnsmode* =1.

Ipaddress_Secondary_DNS_server: This is the IP address of the Secondary DNS server to which the DNS Resolution query is 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

Result Code	Description
OK<DNS_Server_Primary_IPAddr><DNS_Server_SecondaryIPAddr>	*DNS_Server_Primary_IPAddr* (4 bytes, hex): IP address of the primary DNS server *DNS_Server_Secondary_IPAddr* (4 bytes, hex): IP address of the secondary DNS server If mode=0, then the addresses supplied by the user are returned in the above parameters. If any of the parameters is supplied as '0' in this mode, the module will return 4 bytes of 0.
ERROR	Failure. Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFF8

Relevance

This command is relevant in Operating Modes 0 and 2.

Example 1:

at+rsi_dnsserver=1,0,0\t\n

....

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x64 0x6E 0x73 0x65 0x72 0x76 0x65 0x72 0x3D 0x31 0x2C 0x30 0x2C 0x30 0x0D 0x0A

OK<primary=1.2.3.4><secondary=5.6.7.8>

....

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

Example 2:

at+rsi_dnsserver=0,8.8.8.8,0

....

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x64 0x6E 0x73 0x65 0x72 0x76 0x65 0x72 0x3D 0x30 0x2C 0x38 0x2E 0x38 0x2E 0x38 0x2E 0x38 0x2E 0x2C 0x30 0x0D 0x0A

OK< primary=1.2.3.4><secondary=0>

0x4F 0x4B 0x01 0x02 0x03 0x04 0x00 0x00 0x00 0x00 0x0D 0x0A

3.2.31 DNS Resolution

Description

This command is issued by the Host to obtain the IP address of the specified domain name.

Command

at+rsi_dnsget

Usage

at+rsi_dnsget=domain_name, Primary or Secondary DNS server\r\n

Parameters

domain_name- This is the domain name of the target website. A maximum of 150 characters is allowed.

Primary or Secondary DNS server– Used to indicate the DNS server to resolve the Query.

1-Primary DNS server
2-Secondary DNS server

Response

Result Code	Description

OK<num_IPAddr><IPAddr1><IPAddr1>…<IPAddr10>	*num_IPAddr*(2 bytes, hex):
	Number of IP addresses resolved
	IPAddr(4 bytes, hex):
	Individual IP addresses, up to a maximum of 10

ERROR	Failure. Possible error codes are 0x0021, 0x0025, 0x002C, 0xFF42, 0xFFAB, 0xFFB5, 0xFFB6, 0xFFB7, 0xFFB8, 0xFFBA, 0xFFBB, 0xFFF8

Relevance

This command is relevant in Operating Modes 0, 2 or 6.

3.2.32 HTTP Get

Description

This command is used to transmit an 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 a previously issued HTTP GET request. The Host connected to the module acts as a HTTP client when this command is used.

Message

at+rsi_httpget

Usage

at+rsi_httpget=ipaddr_len,url_len,header_len,reserved,Buffer\r\n

Parameters

ipaddr_len – The length of the IP Address (including the digits and dots). For example, if the IP address of www.website.com is 192.168.40.86, *ipaddr_len* = 13

url_len – The length of the URL. For example, if www.website.com/index.html is the webpage, then *url_len* = 11 for "/index.html", www.website.com is not included as it is specified in the IP address

header_len – The length of the header of the HTTP GET request.

Reserved – Set this value to 0.

Buffer – Buffer contains actual values in the order of <IP Address>,<URL>, <Header> and <Data>. *Data* is the actual data involved in the HTTP GET request.

IP Address: 192.168.40.86 (IP address of the domain such as www.website.com)

URL: /index.html

HEADER: User-Agent: HTMLGET 1.00\r\n\r\n

Data = <data>

The contents of the *Buffer* field may require byte stuffing. Refer table for Byte Stuffing further details. The total length of characters from *ipaddr_len* to *Buffer* should be a maximum of 1400 bytes.

Response

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>
```

Response Parameters

More (4 bytes, hex): This indicates whether more HTTP data for the HTTP GET response is pending. The least significant byte is sent first.

0x00000000 – More data pending. Additional AT+RSI_HTTPRSP messages may be sent by the module till all the data received is given to the Host.

0x00000001 – End of HTTP data

Data Offset (4 bytes, hex): This indicates the offset value from where the actual HTTP response data is starting in the response.

Data Length (4 bytes, hex): This indicates the data length value.

Data: Actual data in the HTTP response. This is not byte stuffed by the module when sent to the Host. This will start after 'Data Offset' number of bytes from start of 'more' field.

The string AT+RSI_HTTPRSP is in uppercase ASCII.

Example: To send the following information

IP Address: 192.168.40.86

URL: /index.html

HEADER: User-Agent: HTMLGET 1.00\r\n\r\n

Data = <data>, the below command is used

at+rsi_httpget=13,11,24,0,192.168.40.86/index.htmlUser-Agent: HTMLGET 1.00\r\n\r\n<data>\r\n"

Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFFF8.

Relevance

This command is relevant in Operating Modes 0, 1, 2 or 6.

3.2.33 HTTP Post

Description

This command is used to transmit an 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. The Host connected to the module acts as a HTTP client when this command is used.

Command

at+rsi_httppost

Usage

at+rsi_httppost=ipaddr_len,url_len,header_len,data_len,buffer\r\n

Parameters

ipaddr_len – The length of the IP Address (including the digits and dots). For example, if the IP address of www.website.com is 192.168.40.86, *ipaddr_len* = 13

url_len – The length of the URL. For example, if www.website.com/index.html is the webpage, then *url_len* = 11 for "/index.html", www.website.com is not included as it is specified in the IP address

header_len – The length of the header of the HTTP POST request *data_length* – This is the length of the data field in the *Buffer* parameter.

Buffer – Buffer contains actual values in the order of <IP Address>, <URL>, <Header> and <Data>. *Data* is the actual data involved in the HTTP POST request.

The contents of the *Buffer* field require byte stuffing. Refer table for Byte Stuffing for further details.

The total length of characters from *ipaddr_len* to *Buffer* should be a maximum of 1400 bytes.

Response

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 as shown below.

```
AT+RSI_HTTPRSP=<More><Data Offset><Data Length><Data>
```

More (4 bytes, hex): This indicates whether more HTTP data for the HTTP GET response is pending. The least significant byte is sent first.

0x00000000 – More data pending. Additional AT+RSI_HTTPRSP messages may be sent by the module till all the data received is given to the Host.

0x00000001 – End of HTTP data

Data Offset (4 bytes, hex): This indicates the offset value from where the actual HTTP response data is starting in the response.

Data Length (4 bytes, hex): This indicates the data length value.

Data: Actual data in the HTTP response. This is not byte stuffed by the module when sent to the Host. This will start after 'Data Offset' number of bytes from start of 'more' field.

The string AT+RSI_HTTPRSP is in uppercase ASCII.

Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFFF8.

Relevance

This command is relevant in Operating Modes 0, 1, 2 or 6.

Note: There is no timeout programmed for HTTP Get/HTTP Post requests. The module will wait for infinite time for the response to come.

3.2.34 Query Firmware Version

Description

This command is used to retrieve the firmware version in the module.

Command

at+rsi_fwversion

Usage

at+rsi_fwversion?\r\n

Parameters

None

Response

Result Code	Description
OKMajor11.Minor12.Minor13,Major21.Minor22.Minor23	The firmware version follows after OK. Each byte is separated by a dot and there is a comma after the first 3 bytes. All vales returned in ASCII.
ERROR<Error code>	Failure. Possible error codes for this command are 0xFFF8

For example,

OK 1.2.3,0.0.1\r\n

………………….

0x4F 0x4B 0x31 0x2E 0x32 0x2E 0x33 0x2C 0x30 0x2E 0x30 0x2E 0x31 0x0D 0x0A

Relevance

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

3.2.35 Query RSSI value

Description

This command is used to get the signal strength of the Access Point or network that the module is connected to.

Command

at+rsi_rssi

Usage

at+rsi_rssi?\r\n

Parameters

N/A

Response

Result Code	Description
OK<RSSI>	*RSSI* (1 Byte, hex) : Absolute value of RSSI. For example, if RSSI is -20dBm, then the return value is 0x14
ERROR<Error code>	Failure. Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFFF8

For example, for a RSSI of -20dBm, the return string is

```
OK <RSSI=-20> \r\n
………………….
0x4F 0x4B 0x14 0x0D 0x0A
```

Relevance

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

Note: The RSSI values ranges from -100 dB to -15 dB. Closer the RSSI value to '0', stronger the signal strength.

3.2.36 Query SNR value

Description

This command is used to get the signal to noise ratio of the signal received from the Access Point that the module is connected to. Closer the AP, higher is the SNR.

Command

at+rsi_snr

Usage

at+rsi_snr?\r\n

Parameters

N/A

Response

Result Code	Description
OK<SNR>	*SNR* (1 Byte, hex) : Value of SNR. For example, if SNR is 40dBm, then the return value is 0x28
ERROR<Error code>	Failure. Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFFF8

For example, for a SNR of 40dBm, the return string is

```
OK <SNR=40> \r\n
………………….
0x4F 0x4B 0x28 0x0D 0x0A
```

Relevance

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

> Note: The SNR values ranges from 5 dB to 80 dB. Greater the SNR value, stronger the signal strength.

3.2.37 Query MAC Address of Module

Description

This command is used to retrieve the MAC address of the module.

Command

```
at+rsi_mac
```

Usage

```
at+rsi_mac?\r\n
```

Parameters

N/A

Response

Result Code	Description

Result Code	Description
OK<*MAC_Address*>	*MAC_Address* (6 bytes, hex): MAC address of the module
ERROR<Error code>	Failure. Possible error codes for this command are 0xFFF8

Example:

at+rsi_mac?\r\n

………………………………….

0x61   0x74   0x2B   0x72   0x73   0x69   0x5F 0x6D 0x61 0x63   0x3F 0x0D   0x0A

If the MAC ID is 0x00 0x23 0xA7 0x1B 0x8D 0x31, then the response is

OK 0x00 0x23 0xA7 0x1B 0x8D 0x31 0x0D 0x0A \r\n

……………………………………………………

0x4F 0x4B 0x00 0x23 0xA7 0x1B 0x8D 0x31 0x0D   0x0A

Relevance

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

3.2.38 Query Network Parameters

Description

This command is used to retrieve the WLAN connection and IP parameters. This command should be sent only after the connection to the Access Point is successful.

Command

at+rsi_nwparams

Usage

at+rsi_nwparams?\r\n

Parameters

N/A

Response

Result Code	Description
OK<*wlan_state*><*Chn_no*><*Psk*><*Mac_Addr*><*SSID*><*Network_type*><*Sec_type*><*DHCP_mode*><*Ipaddr*><*Subnet_mask*><*Gateway*><*Num_open_socket*><*reserved*>*[<socket_handle1><socket_type1><source_port1><destination_port1><destination_ip1>][<socket_handle2><socket_type><source_port2><destination_port2><destination_ip2>]…*up to the number of sockets*.*	*Wlan_state* (1 byte, hex): This indicates whether the module is connected to an Access Point or not. 0x00 – Not connected 0x01 – Connected *Chn_no* (1 byte, hex): Channel number of the AP to which the module joined or channel number in which AP is created when module is operating in AP mode. *Psk* (64 bytes, ASCII)*:* Pre-shared key used. If the actual length is less than 64, filler bytes 0x00 are used to make it 64 bytes. *Mac_Addr* (6 bytes, hex)*:* MAC address of the module. *SSID* (34 bytes, ASCII)*:* This value is the SSID of the Access Point to which the module is connected. If the actual length is less than 34, filler bytes 0x00 are used to make it 34 bytes *Network_type* (2 bytes, hex)*:* 0x0003 – AP mode 0x0001 – Infrastructure 0x0000 – Ad-hoc Currently only Infrastructure mode and AP modes are supported *Sec_type* (1 byte, hex): Security mode of the AP. 0x00 – Open mode 0x01 – WPA security 0x02 – WPA2 security 0x03 – WEP 0x04 – WPA-Enterprise 0x05 – WPA2-Enterprise *DHCP_mode* (1 byte, hex): This value indicates whether the module is configured for DHCP or Manual IP configuration. 0x00 – Manual IP configuration

Result Code	Description
	0x01 – DHCP *Ipaddr* (4 bytes, hex): This is the IP Address of the module. If the Set IP Params command was not sent to the module before Query Network Parameters command, the module returns a default value which should be ignored. *Subnet_mask*(4 bytes, hex): This is the Subnet Mask of the module. If the Set IP Params command was not sent to the module before Query Network Parameters command, the module returns a default value which should be ignored. *Gateway*(4 bytes,hex): This is the Gateway Address of the module. If the Set IP Params command was not sent to the module before Query Network Parameters command, the module returns a default value which should be ignored. *Num_open_socket* (2bytes, hex): This value indicates the number of sockets currently open. The least significant byte is returned first. The below parameters are for each open socket. *Reserved* (12 bytes, hex): Host should ignore these bytes. *Socket_handle* (2 bytes, hex): This indicates the socket handle. *Socket_type*(2 bytes, hex): 0x0000 – TCP client 0x0001 – UDP 0x0002 – TCP server (Listening TCP) 0x0004 – Listening UDP The least significant byte is sent first *Source_port* (2 bytes, hex): Port number of the socket in the module. The least significant byte is returned first. *Destination_Port* (2 bytes, hex): Port number of the socket in the remote terminal. The least significant byte is returned first.

Result Code	Description
	Destination_ip (4 bytes, hex): IP of the remote terminal.
ERROR<Error Code>	Failure. Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFFF8

Relevance

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

3.2.39 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 is issued 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

at+rsi_goparams

Usage

at+rsi_goparams?\r\n

Parameters

N/A

Response

Result Code	Description
OK<*SSID*><*BSSID*><*Channel_num*><*PSK*><*reserved*><*IPAddr*><*Station_count*><*MAC1*><*IP1*><*MAC2*><*IP2*>….<*MAC4*><*IP4*>	*SSID* (34 bytes, ASCII)*:* SSID of the Group Owner. If the SSID is less than 34 characters, then filler bytes 0x00 are added to make the length 34 bytes *BSSID* (6 bytes, hex)*:* MAC address of the module *Channel_num* (2 bytes, hex): Channel number of the group owner. The least significant byte is returned first. *PSK* (63 bytes, ASCII): PSK that was

Result Code	Description
	supplied in the command *at+rsi_wfd.* Third party clients should use this PSK while associating to the Group Owner (the Group Owner appears as an Access Point to third party clients). *Reserved* (1 byte):): reserved, host should ignore *IPAddr* (4 bytes, hex): IP Address of the module. Most significant byte is returned first. For example, if the IP is 192.168.40.10, 192 is returned first, then 168 and so on *Station_count* (2 bytes, hex): Number of clients associated to the Group Owner. The least significant byte is returned first. A maximum of 4 clients is supported *MAC*: MAC address of the connected client *IP*: IP address of the connected client
ERROR<Error Code>	Failure. Possible error codes for this command are 0x0021, 0x0022, 0x0025, 0x002C, 0xFFF8

Relevance

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

3.2.40 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.

Command

```
at+rsi_reset
```

Usage

```
at+rsi_reset\r\n
```

Parameters

None

Response

Result Code	Description
OK	Success
ERROR<Error Code>	Failure. Possible error codes for this command are 0xFFF8

Relevance

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

3.2.41 Open a Multicast socket

Description

This command opens a multicast socket.

Command

at+rsi_multicast

Usage

at+rsi_ multicast=Iphost,rport,lport\r\n

Parameters

Iphost – Multicast IP address

Rport – Target port (0 to 65535)[1]

Lport – Local port on the RS9110-N-11-2X module

Response

Result Code	Description
OK<socket_type> <socket_handle><lport> <module_ipaddr>	*socket_type*(2 bytes, hex): Indicates the type of socket. The least significant byte is returned first. 0x0004 –Indicates listening UDP socket *socket_handle* (2 bytes, hex): Upon successfully opening UDP socket, a

[1] To Receive Multicast packets from any peer in multicast group the Rport of the module should be always 00.

Result Code	Description
	socket handle is returned. The least significant byte is returned first. Operating mode 0,2: *socket_handle* ranges from from 1 to 8. The first socket opened will have a socket handle of 1, the second socket will have a handle of 2 and so on. Operating mode 1: If the module is GO or Access Point, then *socket_handle* ranges from from 2 to 8. The first socket opened will have a socket handle of 2, the second socket will have a handle of 3 and so on. *Lport*(2 bytes, hex): Port number of the socket in the module. *Module_ipaddr* (4 bytes, hex): Module's IP address.
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFFE, 0xFF80, 0xFF81, 0xFF85, 0xFFA1, 0xFFF8

Example:

at+rsi_multicast=239.0.0.0,8000,8001\r\n

…………………………………………………………………….

0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6D 0x75 0x6C 0x74 0x69 0x63 0x61 0x73 0x74 0x3D 0x32 0x33 0x39 0x2E 0x30 0x2E 0x30 0x2E 0x30 0x2C 0x38 0x30 0x30 0x30 0x2C 0x38 0x30 0x30 0x31 0x0D 0x0A

For a socket handle 1, the response is

OK <socket_type=0x0003> <socket_handle=0x0001> <local port = 0x1F41><module ip = 0xC0 0xA8 0x01 0x05>\r\n
…………………….
0x4F 0x4B 0x04 0x00 0x01 0x00 0x41 0x1F 0xC0 0xA8 0x01 0x05 0x0D 0x0A

Relevance

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

3.2.42Configure GPIOs

Description

This command configures the GPIO pins that are coming out of WiSeConnect™ module as output pins.

Command

at+rsi_gpioconf

Usage

at+rsi_gpioconf=pin_no,pin_direction,pin_val\r\n

Parameters

pin_no – pin number[1] to be configured, either '1' or '2'

pin_direction – it should be set to '1'

pin_val – value on the pin, '0' for making it low and '1' for making it high.

Response

Result Code	Description
OK	Success
ERROR<Error code>	Failure, Possible error codes are 0x0025, 0x0026, 0xFFF8

Relevance

This command is relevant to the operating modes 0, 1, 2 or 6.

3.2.43 Ping from module

Description

This command sends the ping request from WiSeConnect module to target IP address.

Command

at+rsi_ping

Usage

[1] In WiSeConnect™ 201/301 module, user can configure two pins with this command. For 201, these 1 and 2 are pin-25 and pin-26 respectively. For 301, these 1 and 2 are pin-30 and pin-31 respectively.

at+rsi_ping=targetIP, ping_size\r\n

Parameters

targetIP – target IP address

ping_size – Data size in ping request[1]

Response

Result Code	Description
OK<ping_rsp_ len><IPaddress>	*ping_rsp_len*(2 bytes, hex): Ping response length from the target IP address *IPaddress* (4 bytes, hex): Target IP address
ERROR<Error code>	Failure, Possible error codes are 0x0025, 0x002E, 0x002F, 0x0031, 0xFFF8

Relevance

This command is relevant to operating modes 0, 1, 2 or 6.

Note: Module will give the error for the ping requests until the ARP entry for the destination IP is successful.

3.2.44 Get socket information

Description

This command is used to query the information of the TCP socket with the socket handle requested.

Command

at+rsi_sock_info

Usage

at+rsi_sock_info=sock_handle\r\n

Parameters

sock_handle – handle for the socket

Response

[1] WiSeConnect supports up to ping of data size from 1 to 54 bytes.

Result Code	Description
OK<socket_descriptor><mss><reserved>	*socket_desc*(2 bytes, hex): Socket handle or descriptor *mss* (2 bytes, hex): Maximum segment size reserved(8 bytes, hex) – reserved for future
ERROR<Error code>	Failure, Possible error codes are 0x0025, 0x0026, 0xFFF8

Relevance

This command is relevant to the operating modes 0, 1, 2 or 6.

3.2.45 Get statistics

Description

This command is used to query the statistics of transmitted and received packets from/to WiSeConnect module.

Command

at+rsi_stats

Usage

at+rsi_stats?\r\n

Parameters

None

Response

Result Code	Description
OK<tx_mgmt_rate><tx_retries><rx_retries><signal_rssi><snr_value><reserved>	tx_mgmt_rate (2bytes, hex) – Tx data rate for mgmt packets tx_retries (2bytes, hex)– Number of Tx retries rx_retries (2bytes, hex)– Number of Rx retries signal_rssi (2bytes, hex)– Signal strength of connected AP

Result Code	Description
	snr_value (2bytes, hex)– SNR value reserved (10bytes, hex)– reserved for future
ERROR<Error code>	Failure, Possible error codes are 0x0025, 0x0026, 0xFFF8

Relevance

This command is relevant to the operating modes 0 or 2.

Note: The stats values (tx_retries, rx_retries) wrap around to zero after reaching the maximum value (0xffff) or for every time "get stats" command is issued from host.

3.3 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, 1 (AP mode) and 2.

3.3.1 Storing Configuration Parameters in Client mode

3.3.1.1 Store Configuration in Flash Memory

Description

This command is used to save in internal memory the parameters of an access point to connect to (in auto-join mode) or that of the Access Point to create when the module is powered up (in auto-create mode).

Command

at+rsi_cfgsave

Usage

at+rsi_cfgsave\r\n

Parameters

None

Response

Result Code	Description
OK	Successful execution
ERROR<Error Code>	Failure. Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFFF8

3.3.1.2 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

at+rsi_cfgenable

Usage

at+rsi_cfgenable=0\r\n disables the feature

at+rsi_cfgenable=1\r\n enables the feature

Response

Result Code	Description
OK	Successful execution
ERROR	Failure. Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFFF8

3.3.1.3 Get Information about Stored Configuration

Description

This command is used to get the configuration values that have been stored in the module’s memory and that are used in auto-join or auto-create modes.

Command

at+rsi_cfgget

Usage

at+rsi_cfgget?

Response

Result code	Description
OK <cfg_enable><opermode><band><reserved><join_ssid><uRate><uTxPower><client_psk><scan_cnum><dhcp_enable><IP_addr><snmask><dgq<user_identity><passwd><sec_type><encryption_type><beacon_interval><dtim_period>	*cfg_enable* (1 byte, hex): 0x00- auto-join or auto-create modes are disabled 0x01- auto-join or auto-create modes are enabled *opermode* (1 byte, hex): 0x00- Auto-join mode enabled Client mode with personal security (WPA/WPA2-PSK) 0x01- Auto-create mode enabled 0x02- Auto-join mode enabled with Enterprise Security *Band* (1 byte, hex) 0x00- Module configured to operate in 2.4 GHz 0x01- Module configured to operate in 5 GHz *Reserved* (1 byte): Host should ignore this value *ssid* (34 bytes, ASCII): 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, hex): Data rate configured in the module. Refer table Data Rate Parameter *uTXPower* (1 byte, hex): Tx power configured in the module. *psk* (64 bytes, ASCII): PSK configured in the module in auto-join or auto-create mode. Filler bytes of 0x00 are added to make it 64 bytes if the

Result code	Description
	original PSK is less than 64 bytes. *cnum* (1byte, hex): Channel number of the module in auto-join or auto-create mode *dhcp_enable* (1 byte, hex): 0x00- DHCP client is disabled in module (auto-join mode) 0x01- DHCP client is enabled in module (auto-join mode) *IP_addr* (4 bytes, hex): 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, hex):* Subnet mask *dgw(4 bytes, hex):* Default gateway *Eap_method (*1 byte, hex*):* 0x01- TLS, 0x02- TTLS, 0x03- PEAP, 0x04- FAST *Reserved (*1 byte, hex*):* The Host should ignore this. *user_identity* (64 bytes, hex): User ID in enterprise security. Refer to the parameter *user_identity* in the command *at+rsi_eap*. *Passwd* (128 bytes, ASCII): 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. sec_type(1 byte, hex): Security type of the AP. 0-Open 1-WPA 2-WPA2 encryption_type(1 byte, hex):

Result code	Description
	Encryption type of the AP. 0-Open 1-TKIP 2-AES beacon_interval(2 bytes, hex): Beacon interval dtim_period(2 bytes, hex): DTIM period
ERROR	Failure. Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFFF8

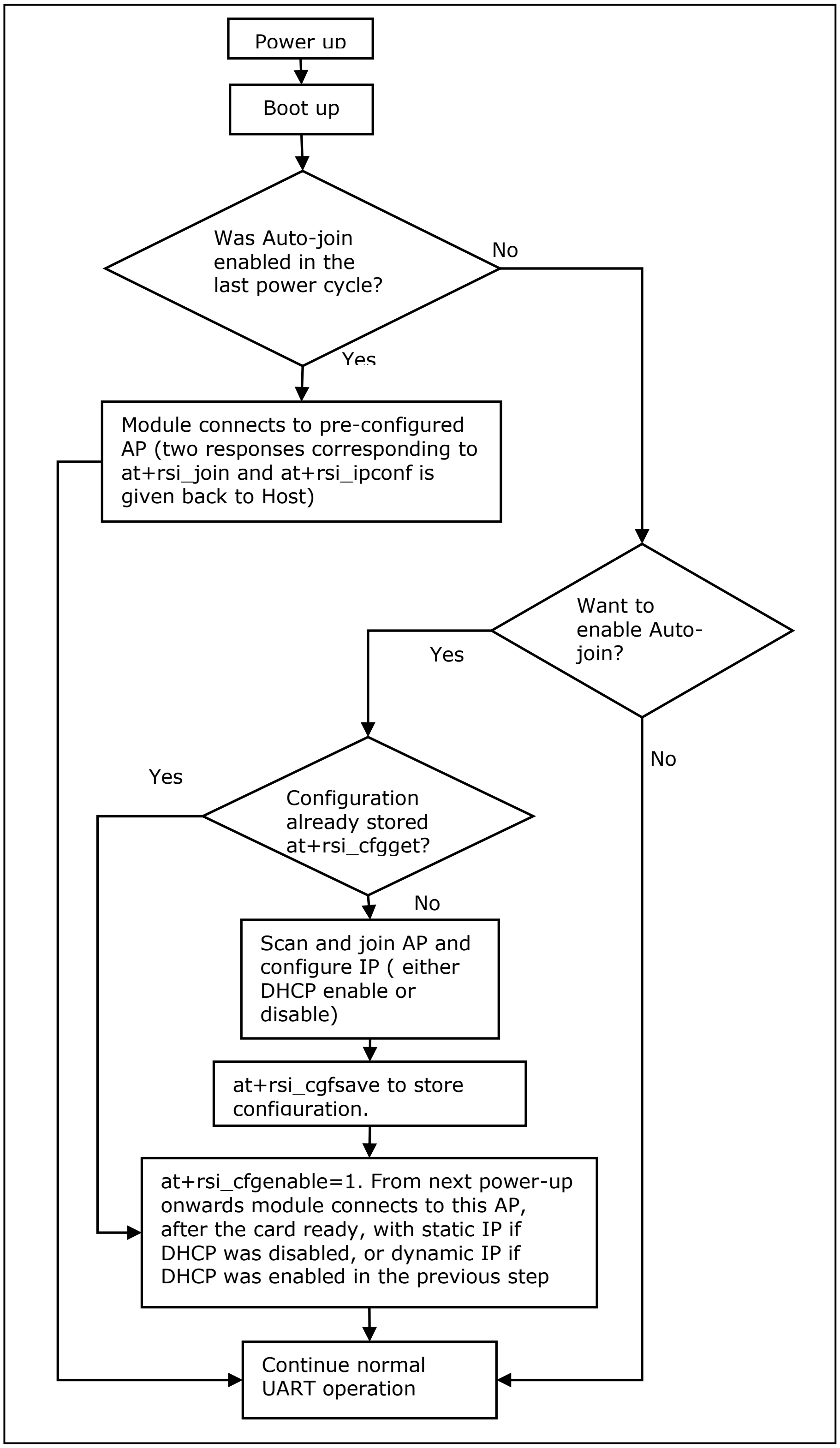

Power up
Boot up
Was Auto-join enabled in the last power cycle?
No
Yes
Module connects to pre-configured AP (two responses corresponding to at+rsi_join and at+rsi_ipconf is given back to Host)
Want to enable Auto-join?
Yes
No
Yes
Configuration already stored at+rsi_cfgget?
No
Scan and join AP and configure IP (either DHCP enable or disable)
at+rsi_cgfsave to store configuration.
at+rsi_cfgenable=1. From next power-up onwards module connects to this AP, after the card ready, with static IP if DHCP was disabled, or dynamic IP if DHCP was enabled in the previous step
Continue normal UART operation

Figure 7: Connecting to pre-configured AP

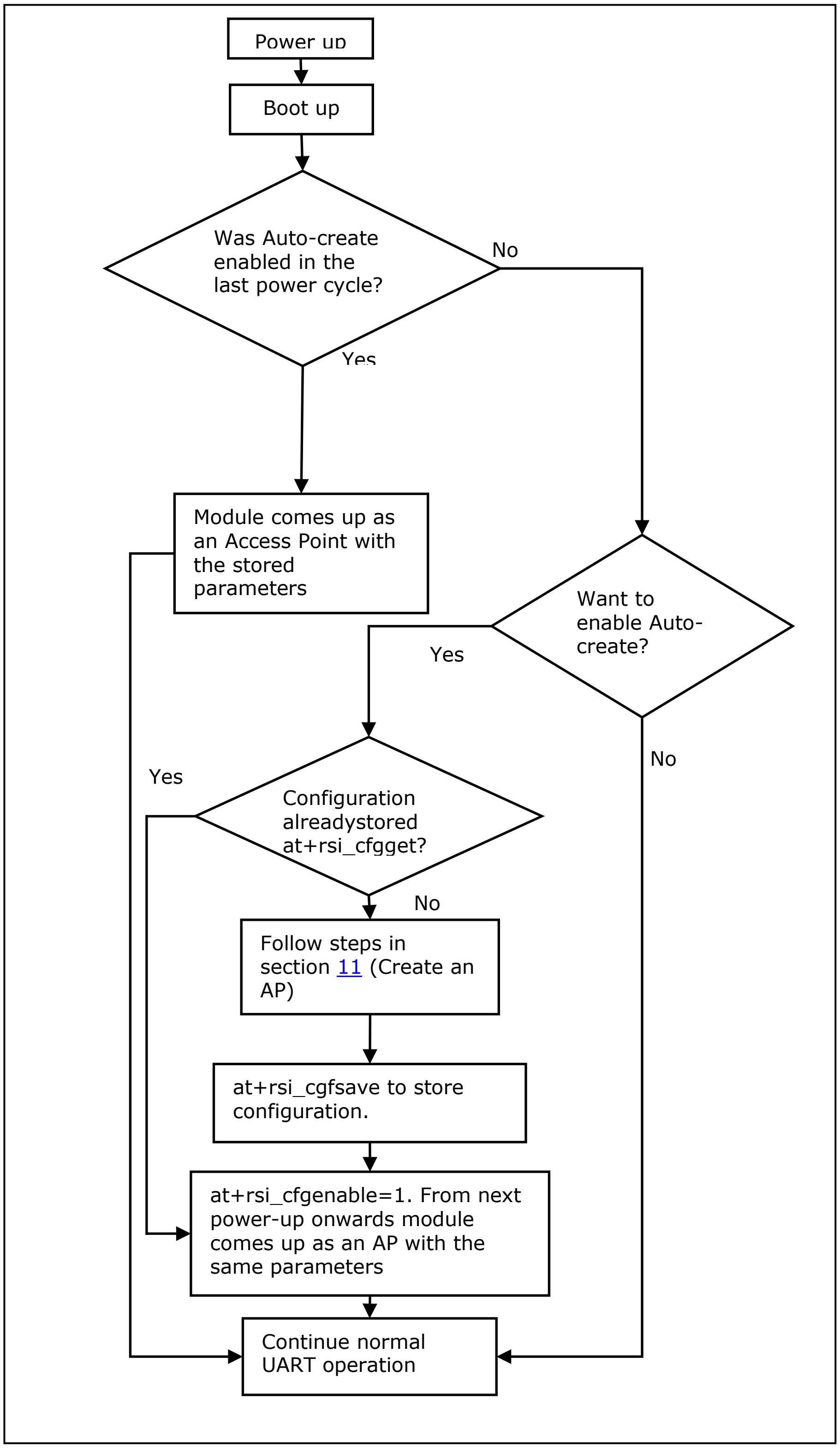

Power up
Boot up
Was Auto-create enabled in the last power cycle?
No
Yes
Module comes up as an Access Point with the stored parameters
Want to enable Auto-create?
Yes
No
Yes
Configuration alreadystored at+rsi_cfgget?
No
Follow steps in section 11 (Create an AP)
at+rsi_cgfsave to store configuration.
at+rsi_cfgenable=1. From next power-up onwards module comes up as an AP with the same parameters
Continue normal UART operation

Figure 8: Creating a Pre-configured AP

3.4 Error Codes

The following are the valid error codes, along with their two's complement values in hexadecimal.

Error Code	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	Module failed to associate to Access Point
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 *at+rsi_disassoc* command
0x0014	Wi-Fi Direct or EAP configuration is not done
0x0015	Memory allocation 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.Re-join failure
0x001A	Frequency not supported
0x001C	EAP configuration failed
0x001D	P2P configuration failed
0x001E	Unable to start Group Owner negotiation
0x0020	Unable to join

Error Code	Description
0x0021	Command given in incorrect state
0x0022	Query GO parameters issued in incorrect operating mode
0x0023	Unable to form Access Point
0x0024	Wrong Scan input parameters supplied to “Scan” command
0x0025	Command issued during rejoin in progress
0x0026	Wrong parameters passed in command (e.g. SSID given is greater than 32 bytes, webpage length is given wrong in the command, more web fields are given, wrong values passed for GPIO configuration command)
0x0028	PSK length less than 8 bytes or more than 63 bytes
0x0029	Failed to clear or to set the Enterprise Certificate (*at+rsi_cert*)
0x002A	Group Owner negotiation failed in Wi-Fi Direct mode
0x002B	Association between nodes failed in Wi-Fi Direct mode
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 timed out
0x002F	Ping size given is beyond the maximum ping size supported
0x0030	Send data packet exceeded the limit or length that is mentioned
0x0031	ARP Cache entry not found
0x0032	UART command timeout happened
0xFFFF	Listening TCP socket in module is not connected to the remote peer, or the LTCP socket is not yet opened in the module
0xFFFE	Sockets not available. The error comes if the Host tries to open more than 8 sockets or If the host tries to send data over socket which is already closed
0xFFF8	1. Invalid command (e.g. parameters insufficient or invalid in the command).

Error Code	Description
	2. Invalid operation (e.g. power save command with the same mode given twice, accessing wrong socket, creating more than allowed sockets)
0xFFF7	Byte stuffing error
0xFF80	Attempt to open more than the maximum allowed number of sockets
0xFF7E	Data length is beyond maximum segment size (mss)
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
0xFF41	HTTP socket creation failed
0xFF42	DNS response was timed out
0xFFA1	ARP request failure
0xFF9D	DHCP lease time expired
0xFF9C	DHCP handshake failure/ DHCP renewal failure
0xFF87	This error is issued when module tried to connect to a non-existent TCP server socket on the remote side
0xFF85	Invalid socket parameters
0xFF81	Socket already open
0xFF40	TCP socket close command is issued before getting the response of the previous close command
0xFF2D	TCP ACK failed for TCP SYN-ACK
0xFF33	TCP keep alive timed out
0xFFA5	IGMP error
APP	While executing the commands, user may get

Error Code	Description
ASSERT:XYZ	asynchronous assert messages. These messages are given to the host with the assert number occurred internally. The assert number ranges from 119 to 185. These are added for providing the debug information. User should note down the assert number and need to provide this debug info along with the sequence of commands executed. Example: APP ASSERT:125<CR><LF>

Table 6: Error Codes for UART

The least significant byte of the Error code is returned first. For example, if the error code is -4

ERROR -4

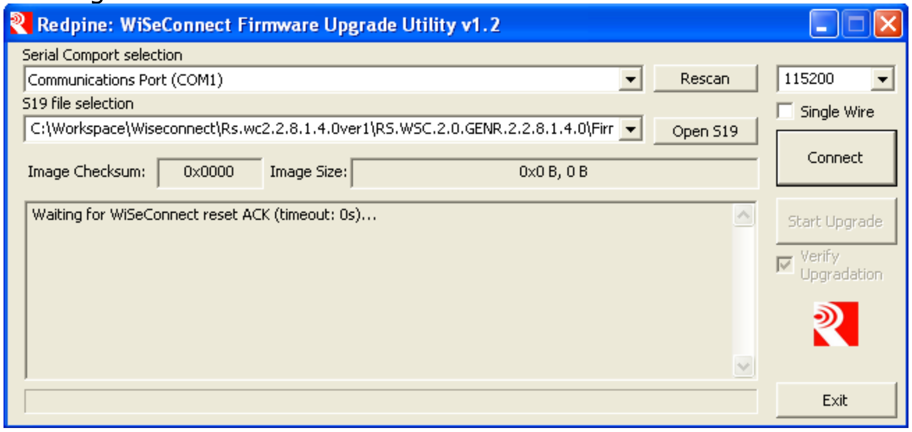
……………………………..

0x45 0x52 0x52 0x4F 0x52 0xFC 0xFF 0x0D 0x0A

4 Upgrading Firmware Through the UART Interface

This section describes the process of upgrading module firmware through the UART interface. There are four firmware files to be loaded for the complete firmware upgrade. The procedure is described below.

1. Connect a PC to the Module through the UART interface, using a UART cable.
2. Run the application
   RS.WSC.x.x.GENR.x.y.z.a.b.c\Resources\UART\Firmware_upgrade\WSC_FW_Upgrade_Util.exe
   on the PC.
   This application can be found in the software release package. The application will automatically scan for UART ports in the PC and display the appropriate port.
   Note: Please use the firmware upgrade utility that comes with the same package.
3. From the drop-down box, select the COM port that is connected to the module's UART interface and select the baud rate as 115200. Please do not use other baud rates.
4. Click "Open S19" button and Select the file *RS.WSC.x.x.GENR.x.y.z.a.b.c\Firmware\WFU\WFU_Control.S19*. Now press the "Connect" button.
5. Once you press the connect button the GUI will wait for the reset message from the module. Please find the Screen shot below.



Serial Comport selection: Communications Port (COM1)

115200

☐ Single Wire

S19 file selection: C:\Workspace\Wiseconnect\Rs.wc2.2.8.1.4.0ver1\RS.WSC.2.0.GENR.2.2.8.1.4.0\Firr

Image Checksum: 0x0000 Image Size: 0x0 B, 0 B

Waiting for WiSeConnect reset ACK (timeout: 0s)...

☑ Verify Upgradation

6. Within 10 seconds of pressing the "Connect" button, give a hard-reset to the module and then observe the "received 0xfc (good)" message on the GUI. If you have observed "received 0xfc (good)" message **without** pressing the hard reset, then please do not reset the module again – proceed directly to the next step. If you have pressed the button by mistake then go back to step one and start again – otherwise there might be a false upgrade and module may not boot up properly.

Note: Before hard reset the module, wait for a character 0xfc (This character may vary depending on the host baud rate) from module.

7. After step 6 the message window of the GUI prompts to start the upgrade. Click on “Start Upgrade” button.
8. After successful execution of step 7 the following message is indicated on the GUI

   ```
   Memory is erased.
   Memory programmed:         100%
   Memory verified:           OK                     ‟
   Upgradation Completed ‟
   ```

   Note: If you observe the message “Memory is not erased” then redo the same step again
9. For the second file, click “Open S19” button. Select the file *RS.WSC.x.x.GENR x.y.z.a.b.c\Firmware\WFU\WLAN_Config.S19*. Now press the “Connect” button. Within 10 seconds of pressing the “Connect” button, give a hard-reset to the module or power cycle the module.
10. The message window of the GUI will prompt to start the upgrade. Click on “Start Upgrade” button.
11. The Message window would now say “Up gradation Completed”. Wait until Card_Ready pin is asserted (goes low) – i.e., LED2 glows. This may take up to two minutes.
    Note: This is important – wait until LED2 glows before proceeding.
12. For the third file, click “Open S19” button. Select the file *RS.WSC.x.x.GENR x.y.z.a.b.c \Firmware\WiSe_WLAN.S19.*
    Now press the “Connect” button.
13. Within 10 seconds of pressing the “Connect” button, give a hard-reset to the module.
14. The message window of the GUI will prompt to start the upgrade. Click on “Start Upgrade” button.
15. The Message window would now say “Upgradation Completed”. Wait until Card_Ready pin goes “Low” i.e LED2 Glows. This may take up to two minutes.
16. For the fourth file, click “Open S19” button. Select the file *RS.WSC.x.x.GENR.x.y.z.a.b.c\Firmware\WiSe_Control.S19.*
    Now press “Connect” button. Within 10 seconds of pressing the “Connect” button, give a hard-reset to the module or power cycle the module.
17. The message window of the GUI will prompt to start the upgrade. Click on “Start Upgrade” button.
18. The Message window would now say “Upgradation Completed”. This completes the Firmware Upgrade process. Close the application in the PC and power cycle the module.

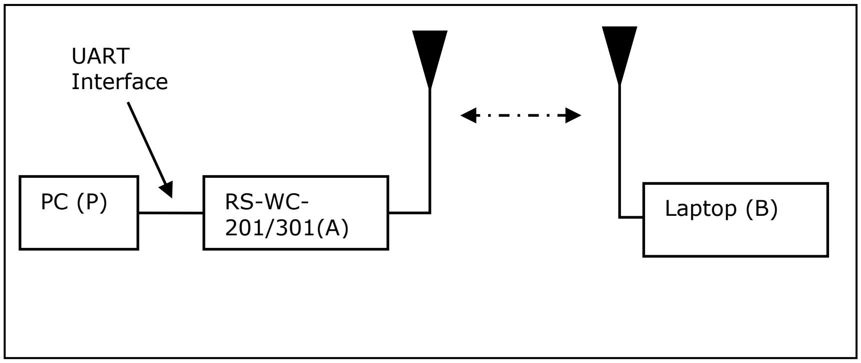
Note: If user wants to upgrade a release with version x, where x is lower to 2.1.0.1.2.5 release, then there is no need to load the first two files. As those releases didn’t support wireless firmware upgrade feature, user won’t find those files (*WFU_Control.S19*, *WLAN_Config.S19*) in the release package.

5 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").

5.1 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.



1. Connect a PC or Host to the module through the UART 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 in UART).
3. Connect a Laptop (B) to the created AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.1, then the URL is http://192.168.100.1/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
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.

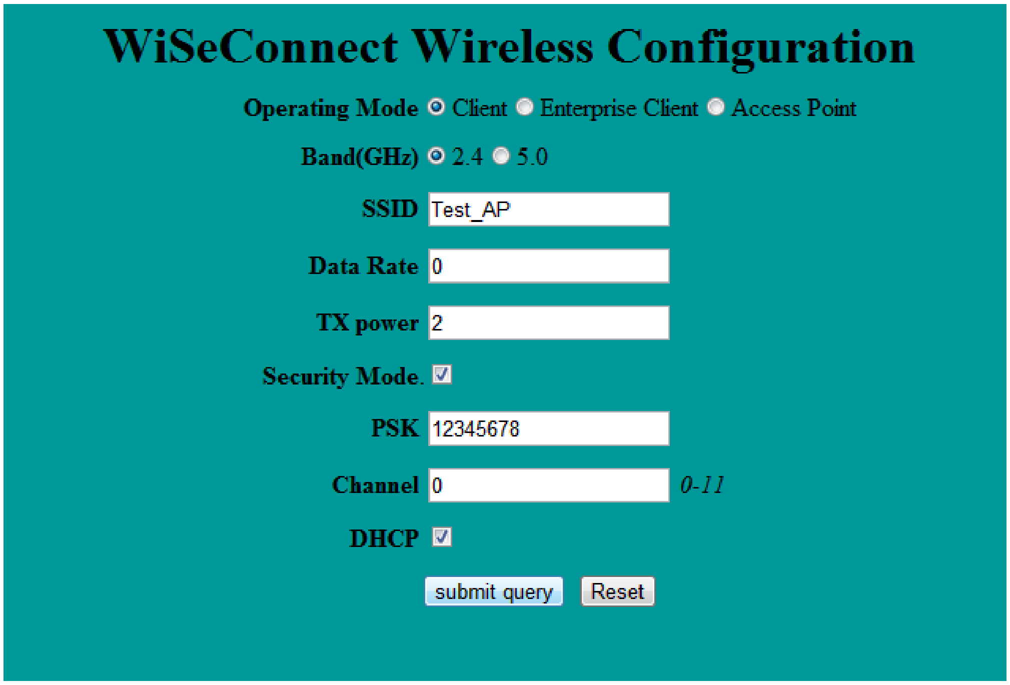
   Data rate: Physical data rate (refer Data Rate Parameter).

   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 Channels in 2.4 GHz and Channels in 5 GHz.



WiSeConnect Wireless Configuration

Operating Mode ● Client ○ Enterprise Client ○ Access Point

Band(GHz) ● 2.4 ○ 5.0

SSID Test_AP

Data Rate 0

TX power 2

Security Mode. ☑

PSK 12345678

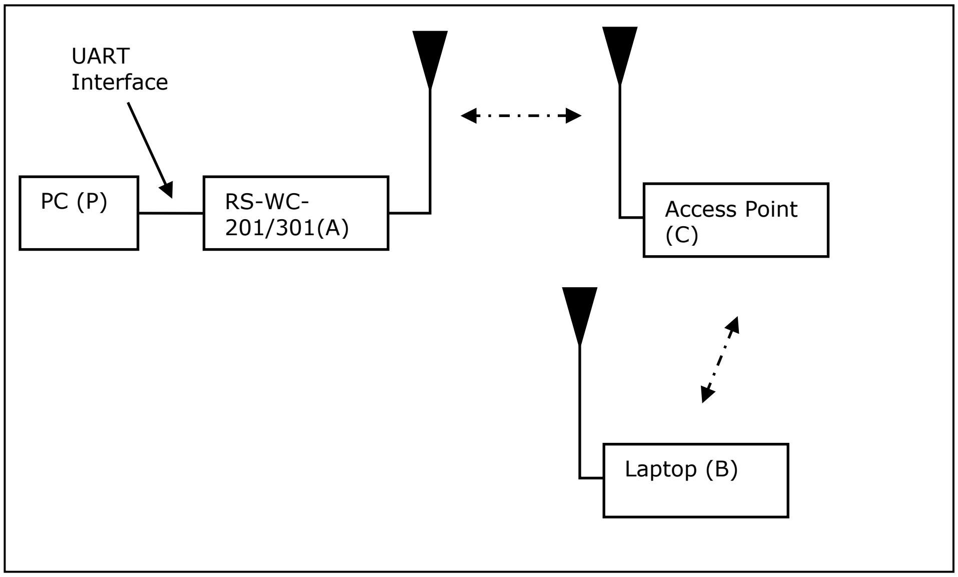
Channel 0 *0-11*

DHCP ☑

Click on "Submit Query" button. The information is sent to the module and stored in its internal flash.

5. 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 internally given "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 UART 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 in UART).
3. Connect a Laptop (B) to the same AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.20, then the URL is http://192.168.100.20/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
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.

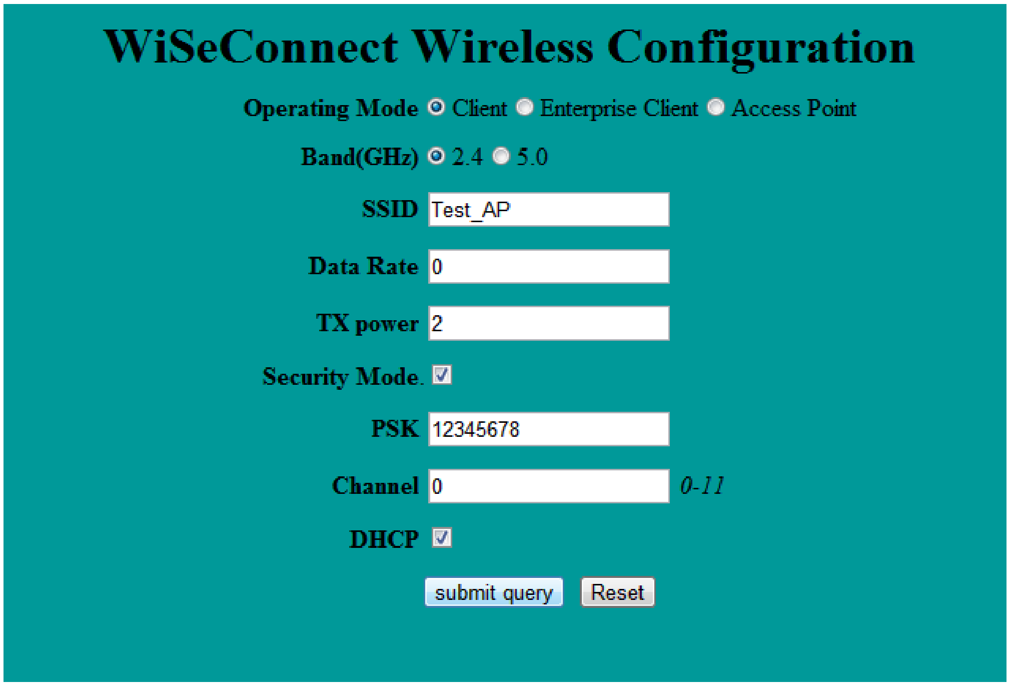
   Data rate: Physical data rate (refer Data Rate Parameter).

   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 Channels in 2.4 GHz and Channels in 5 GHz.



WiSeConnect Wireless Configuration

Operating Mode ● Client ○ Enterprise Client ○ Access Point

Band(GHz) ● 2.4 ○ 5.0

SSID Test_AP

Data Rate 0

TX power 2

Security Mode. ☑

PSK 12345678

Channel 0 *0-11*

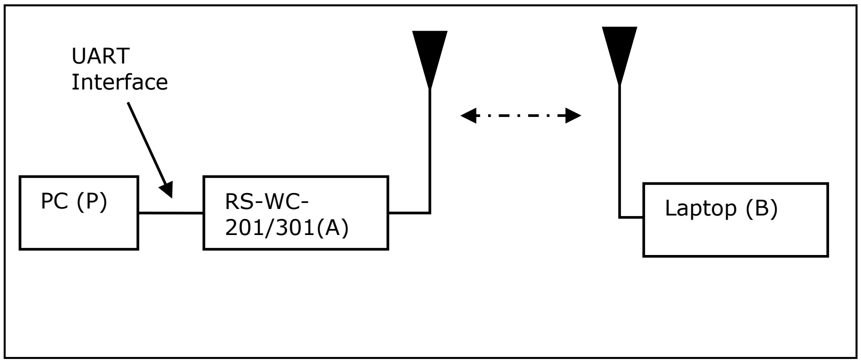
DHCP ☑

Click on “Submit Query” button. The information is sent to the module and stored in its internal flash.

5. 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 internally given “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.

5.2 Configuration 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.



1. Connect a PC or Host to the module through the UART 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 in UART).
3. Connect a Laptop (B) to the created AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.1, then the URL is http://192.168.100.1/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
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.

   Data rate: Set the data rate to '0'.

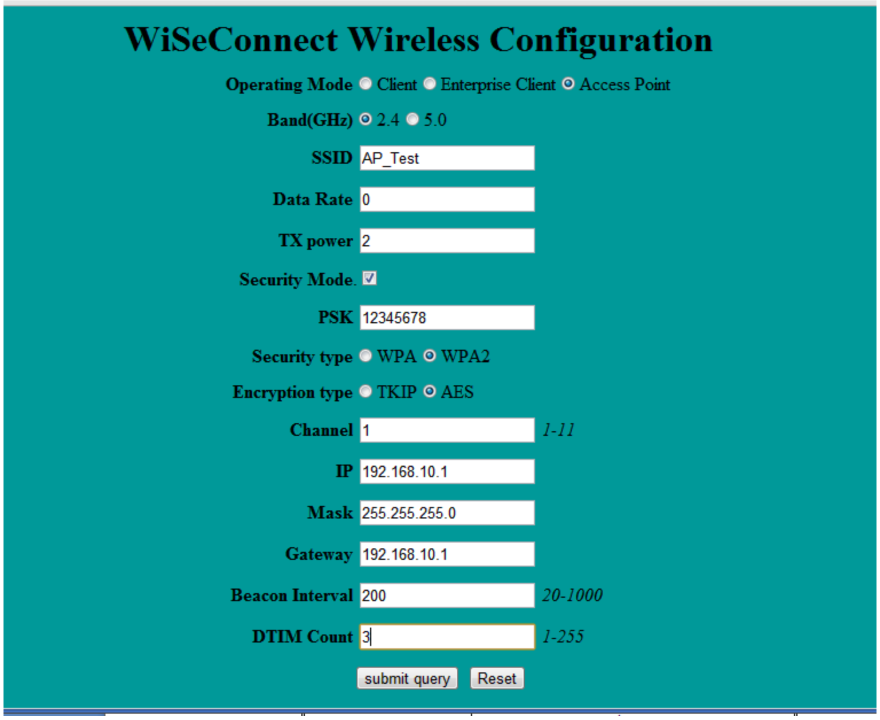
   Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.

   Security mode , 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 Channels in 2.4 GHz and Channels in 5 GHz. Value of '0' is not allowed.

   IP, Mask, 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.



WiSeConnect Wireless Configuration

Operating Mode ○ Client ○ Enterprise Client ● Access Point

Band(GHz) ● 2.4 ○ 5.0

SSID AP_Test

Data Rate 0

TX power 2

Security Mode ☑

PSK 12345678

Security type ○ WPA ● WPA2

Encryption type ○ TKIP ● AES

Channel 1 *1-11*

IP 192.168.10.1

Mask 255.255.255.0

Gateway 192.168.10.1

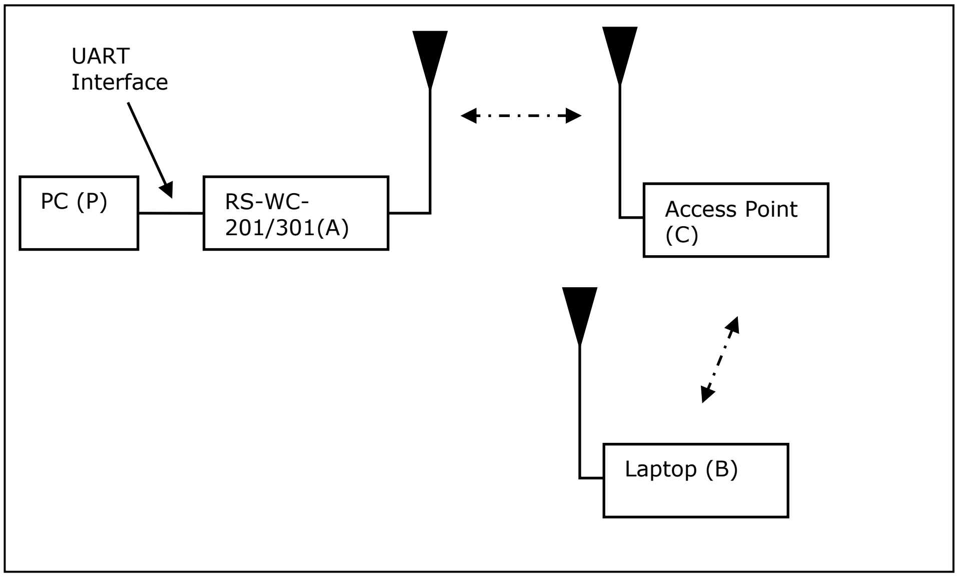
Beacon Interval 200 *20-1000*

DTIM Count 3 *1-255*

Click on "Submit Query" button. The information is sent to the module and stored in its internal flash.

5. The module should now be power cycled or hard reset. It boots up and then automatically creates and AP with the configured parameters. The module will send out two responses to the Host, the first corresponds to the internally given "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.



1. Connect a PC or Host to the module through the UART 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 in UART).
3. Connect a Laptop (B) to the created AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.20, then the URL is http://192.168.100.20/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
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.

   Data rate: Set the data rate to '0'.

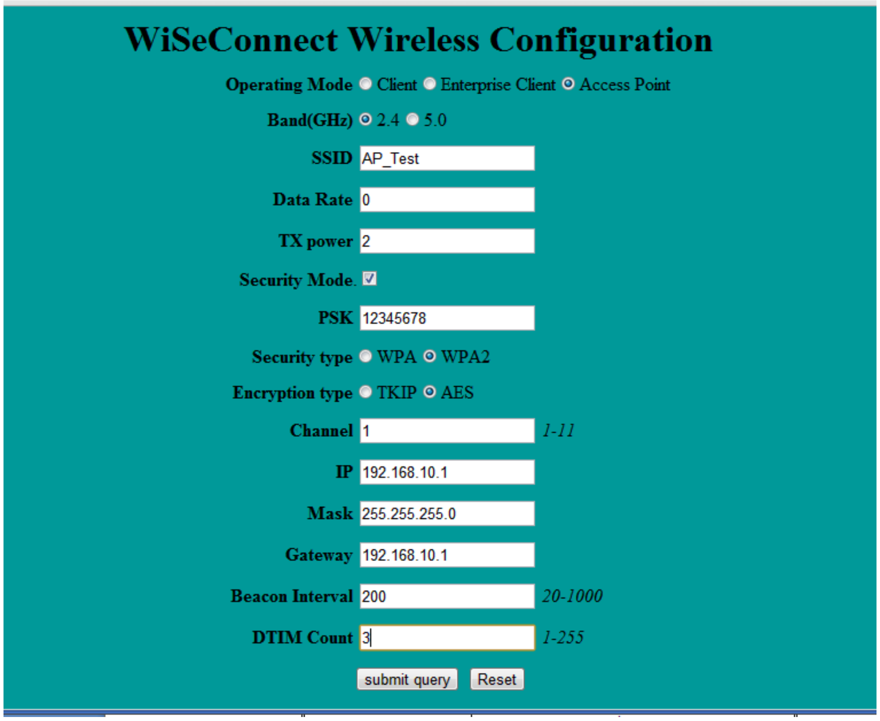
   Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.

   Security mode , 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 Channels in 2.4 GHz and Channels in 5 GHz. Value of '0' is not allowed.

   IP, Mask, 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.



WiSeConnect Wireless Configuration

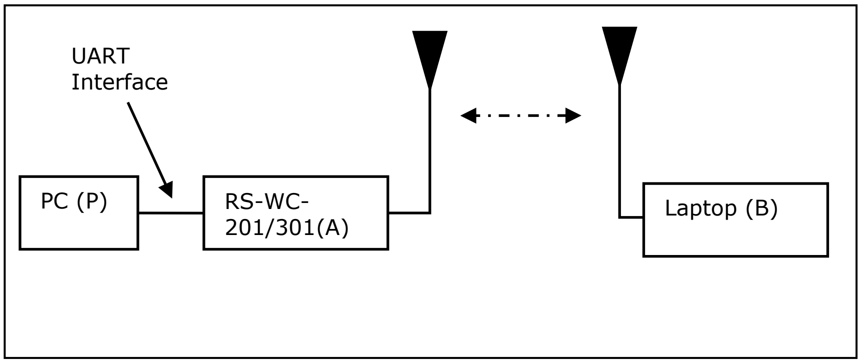
Operating Mode	○ Client ○ Enterprise Client ● Access Point	
Band(GHz)	● 2.4 ○ 5.0	
SSID	AP_Test	
Data Rate	0	
TX power	2	
Security Mode	☑	
PSK	12345678	
Security type	○ WPA ● WPA2	
Encryption type	○ TKIP ● AES	
Channel	1	1-11
IP	192.168.10.1	
Mask	255.255.255.0	
Gateway	192.168.10.1	
Beacon Interval	200	20-1000
DTIM Count	3	1-255

Click on “Submit Query” button. The information is sent to the module and stored in its internal flash.

5. The module should now be power cycled or hard reset. It boots up and then automatically creates and AP with the configured parameters. The module will send out two responses to the Host, the first corresponds to the internally given “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.

5.3 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.



6. Connect a PC or Host to the module through the UART interface and power up the module.
7. Configure the module to become an AP by issuing commands from PC (P) (refer APPENDIX A: Sample Flow of Commands in UART).
8. Connect a Laptop (B) to the created AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.1, then the URL is http://192.168.100.1/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
9. In the web page that opens, select "Enterprise 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 Data Rate Parameter).

   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.

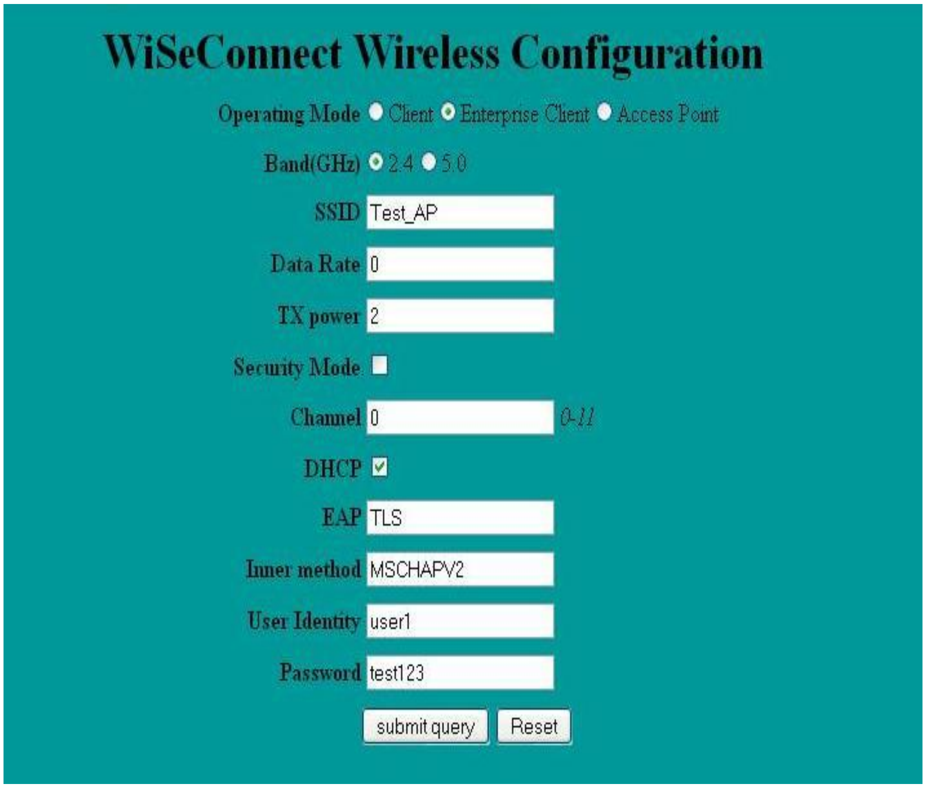
   Channel: Channel number at which the target AP is present. Refer Channels in 2.4 GHz and Channels in 5 GHz.

   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.



WiSeConnect Wireless Configuration

Operating Mode ○ Client ● Enterprise Client ○ Access Point

Band(GHz) ● 2.4 ○ 5.0

SSID Test_AP

Data Rate 0

TX power 2

Security Mode □

Channel 0 *0-11*

DHCP ☑

EAP TLS

Inner method MSCHAPV2

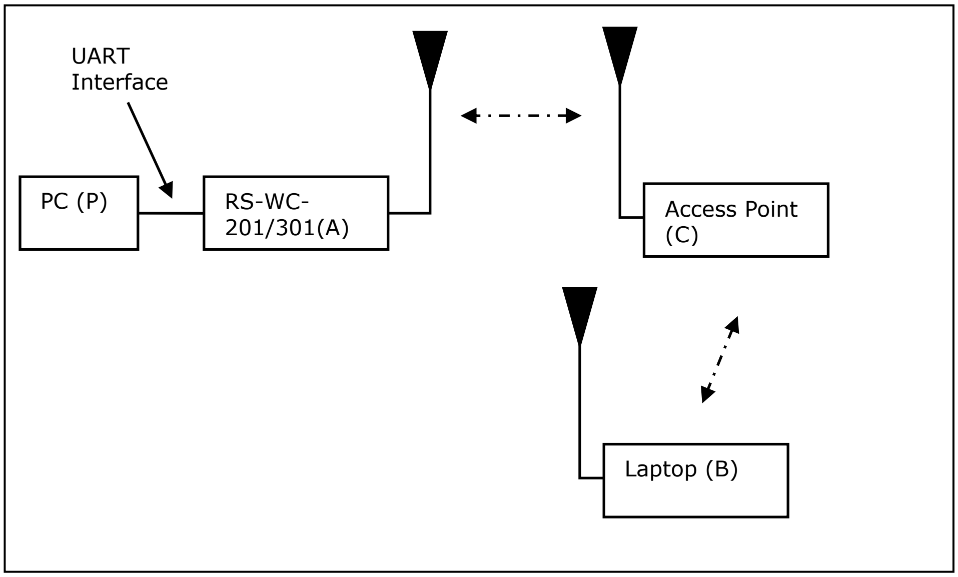
User Identity user1

Password test123

Click on "Submit Query" button. The information is sent to the module and stored in its internal flash.

10. 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 internally given "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.



6. Connect a PC or Host to the module through the UART interface and power up the module.
7. 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 in UART).
8. Connect a Laptop (B) to the same AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.20, then the URL is http://192.168.100.20/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
9. 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 Data Rate Parameter).

   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.

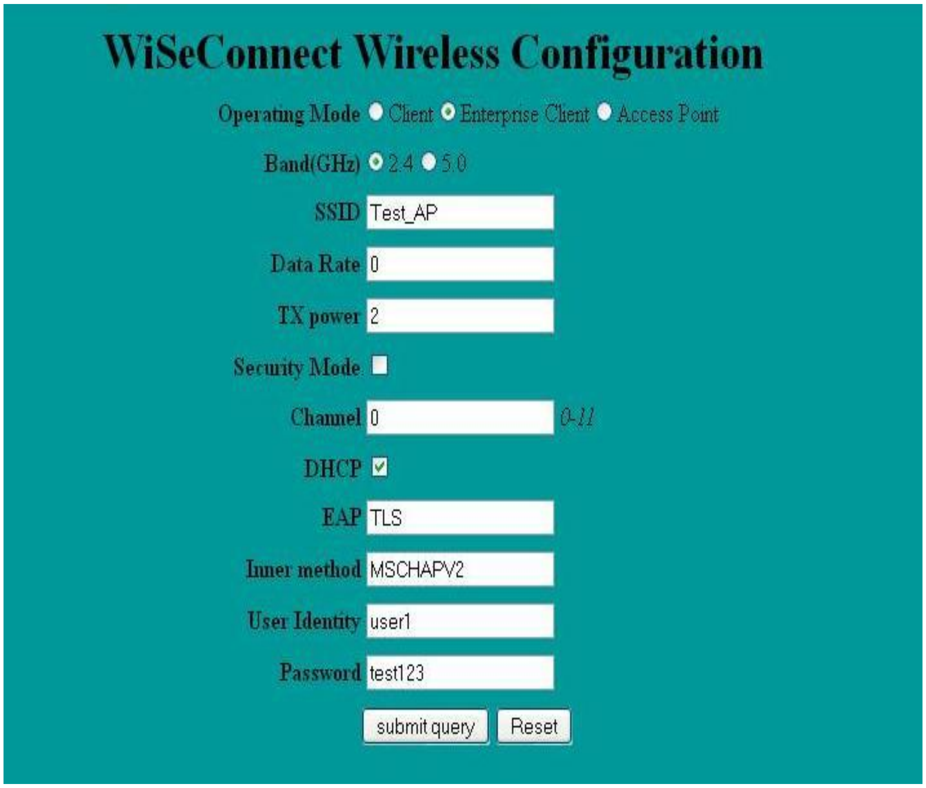
   Channel: Channel number at which the target AP is present. Refer Channels in 2.4 GHz and Channels in 5 GHz.

   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.



WiSeConnect Wireless Configuration

Operating Mode ○ Client ● Enterprise Client ○ Access Point

Band(GHz) ● 2.4 ○ 5.0

SSID Test_AP

Data Rate 0

TX power 2

Security Mode □

Channel 0 *0-11*

DHCP ☑

EAP TLS

Inner method MSCHAPV2

User Identity user1

Password test123

Click on "Submit Query" button. The information is sent to the module and stored in its internal flash.

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 internally given "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.

6 RS-WC-201/301 in USB Mode

The modules support USB interface from firmware version 2.0.0.x.x.x. The USB interface in the module corresponds to the CDC-ACM class and presents itself as a USB Device to the Host USB. A file is provided with the software package that the user should install in the Host platform, in order to communicate with the module.

The parameters corresponding to the device after the USB is detected are:

Device Descriptor:

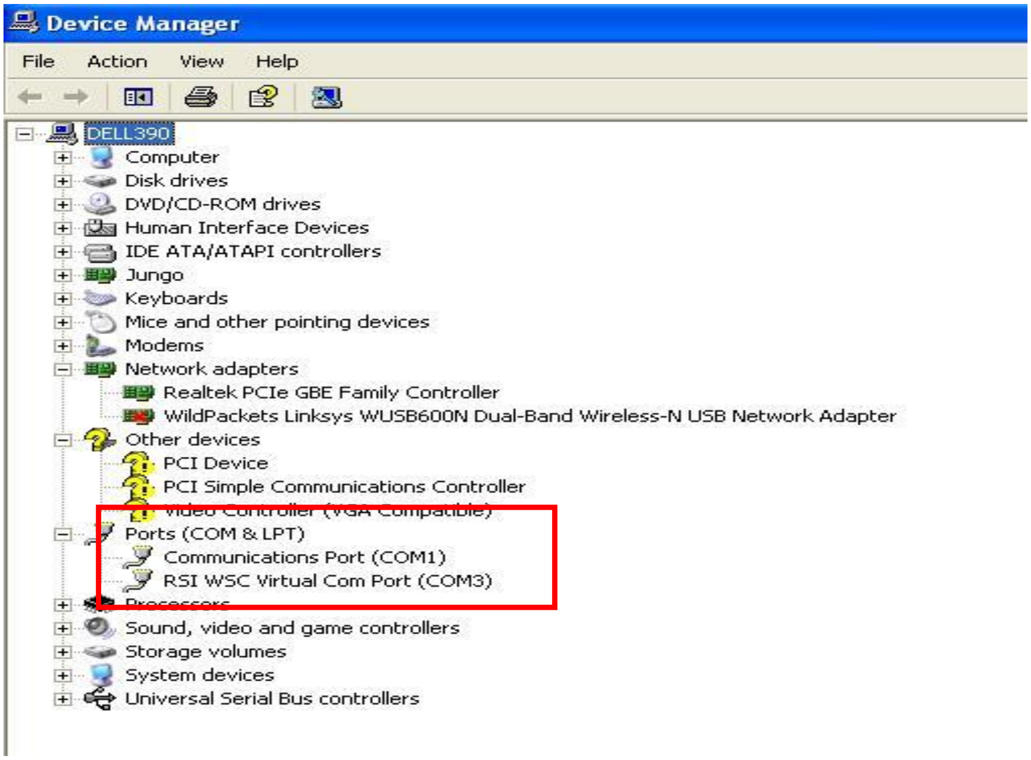
bcdUSB:	0x0200
bDeviceClass:	0x02
bDeviceSubClass:	0x00
bDeviceProtocol:	0x00
bMaxPacketSize0:	0x10 (16)
idVendor:	0x041B
idProduct:	0x0102
bcdDevice:	0x0002
iManufacturer:	0x01

6.1 Using the USB Interface

A sample flow is provided below to use the module with a PC’s USB interface.

1. Connect the USB interface of the PC with that of the module. The PC prompts for installing the driver. Install the file from RS.WSC.2.0.GENR.x.x.x.x.x.x\Resources\USB\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.


Device Manager
File
Action
View
Help
DELL390
Computer
Disk drives
DVD/CD-ROM drives
Human Interface Devices
IDE ATA/ATAPI controllers
Jungo
Keyboards
Mice and other pointing devices
Modems
Network adapters
Realtek PCIe GBE Family Controller
WildPackets Linksys WUSB600N Dual-Band Wireless-N USB Network Adapter
Other devices
PCI Device
PCI Simple Communications Controller
Video Controller (VGA Compatible)
Ports (COM & LPT)
Communications Port (COM1)
RSI WSC Virtual Com Port (COM3)
Processors
Sound, video and game controllers
Storage volumes
System devices
Universal Serial Bus controllers

3. Open Hyperterminal and set Flow Control to “None”. Baudrate, Data bits, 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 except for the exceptions mentioned in the following section. After the module is power cycled, it sends the messages “Welcome to WiSeConnect” and “READY” as in UART mode, after which AT commands from *at+rsi_opermode* onwards can be issued from the hyper-terminal. The USB interface of the module supports the full speed USB mode (12 Mbps physical data rate).

6.2 USB Command Exceptions

Commands for operating the module in USB are exactly the same as AT based commands used in UART mode, except for a few exceptions. The exceptions are described below.

6.2.1 Set certificate

Description

This command is used to load the certificate or PAC file, after issuing the *at+rsi_eap* command. This command should be issued if the security mode is EAP-TLS or EAP-FAST

Command

at+rsi_cert

Usage

at+rsi_cert =cert_type,cert_len,key_password,more_chunks,current_length,certificate\r\n

Parameters

cert_type: Type of the certificate.

1–TLS client certificate

2–FAST PAC file

cert_len (variable size, ASCII): Length of the certificate in number of bytes, sent in ASCII format. If this value is put to '0', then the command clears the current certificate from the module's memory[1]. If this value is '0', the following parameters need not be supplied.

Key_password: Private key password, used to generate the certificate

more_chunks: A maximum of 1400 bytes of the certificate can be sent to the module from the Host. If the certificate length is more than 1400 bytes, then the certificate need to be sent over multiple segments. 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

current_length: Length of the current segment

certificate: Data of the current segment of the certificate (TLS certificate / PAC file). TLS certificate is applicable in EAP-TLS mode, PAC file is applicable in EAP-FAST 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 name 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)
```

[1] Value of '0' is supported from firmware version 2.0.0.1.2.4 onwards

Response

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

Relevance

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

6.2.2 Send Data to a Socket

Description

This command sends a byte stream of a certain size to the socket specified by the socket handle. The command differs between the USB and UART modes in the fact that byte stuffing should not be done in the USB mode.

Command

```
at+rsi_snd
```

Usage

```
at+rsi_snd=socket_handle,length,dipaddr,dport, data_stream\r\n
```

Parameters

socket_handle (1 byte, ASCII)– Socket handle of the socket over which data is to be sent.

length – Exact length of the packet in number of bytes

Dipaddr (variable length, ASCII)– Destination IP Address. Should be '0' if transacting on a TCP socket

dport (variable length, ASCII)– Destination Port. Should be '0' if transacting on a TCP socket

data_stream (maximum length of 1400 bytes, hex)– Actual data to be sent to be sent to the specified socket.

Response

Result Code	Description
OK<dummy>	2 bytes dummy, should be ignored

Result Code	Description
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.

For example 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
```

NOTE: The parameter *data_stream* contains the actual data and not the ASCII representations of the data.

Relevance

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

6.2.3 Other Exceptions

The "Soft Reset" command and "Power Save Modes" are not supported in USB mode.

7 RS-WC-201/301 in SPI Mode

This section describes the commands and processes to operate the module using the SPI interface. The section Driver Porting Guide for SPI, that follows after all the commands and their input and output parameters are described, details the usage of a sample SPI driver and application that is is provided with the software release.

7.1 Communicating using the SPI Interface

The RS-WC-201/301 module can be configured and operated by sending commands from the Host to the module through the SPI interface.

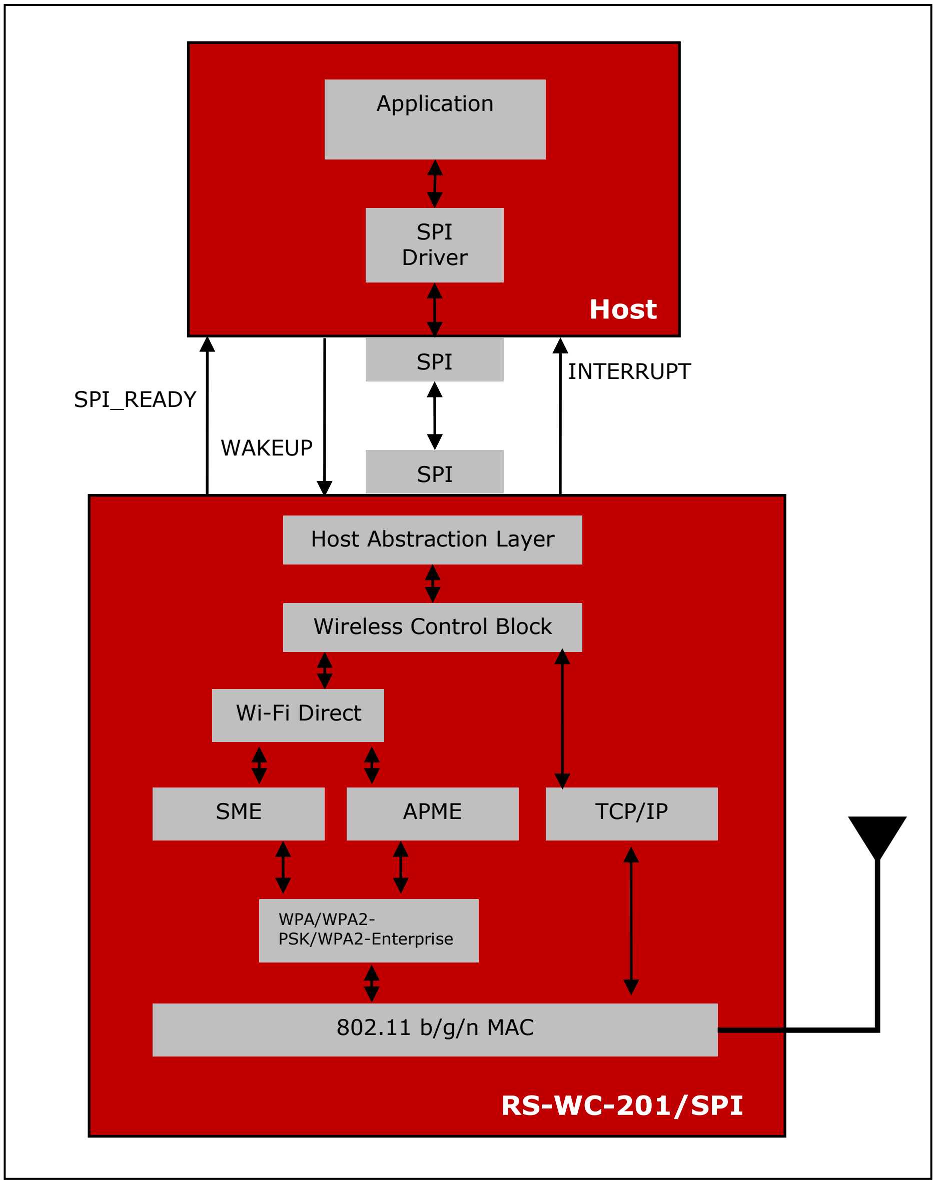


Figure 9: System Architecture with SPI Interface

7.1.1 SPI settings

The SPI Interface between the Host MCU and the module involves the following signals:

SPI_CLK – SPI Clock, driven from the SPI master.

SPI_MISO – Data output from the Module

SPI_MOSI – Data input into the Module

SPI_CS – Slave select input into the Module

INTERRUPT – Interrupt output signal from the module to the Host

SPI_READY – Handshake signal, output from the module to the Host

WAKEUP – Used in power save mode 2. Refer Power save Mode 2.

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.When the module needs to indicate to the Host that it has woken up from sleep.

4.To indicate to the Host that it should initiate a CARD READY operation. This operation is described in the subsequent sections.

The SPI_READY is a handshake signal used in SPI mode and should be connected to a GPIO pin of the Host MCU.

The SPI interface should be configured with the following parameters:

CPOL= 0

CPHASE= 0

The MCU should be configured to correspond to the endianness shown in Endianess in Data Transfer.

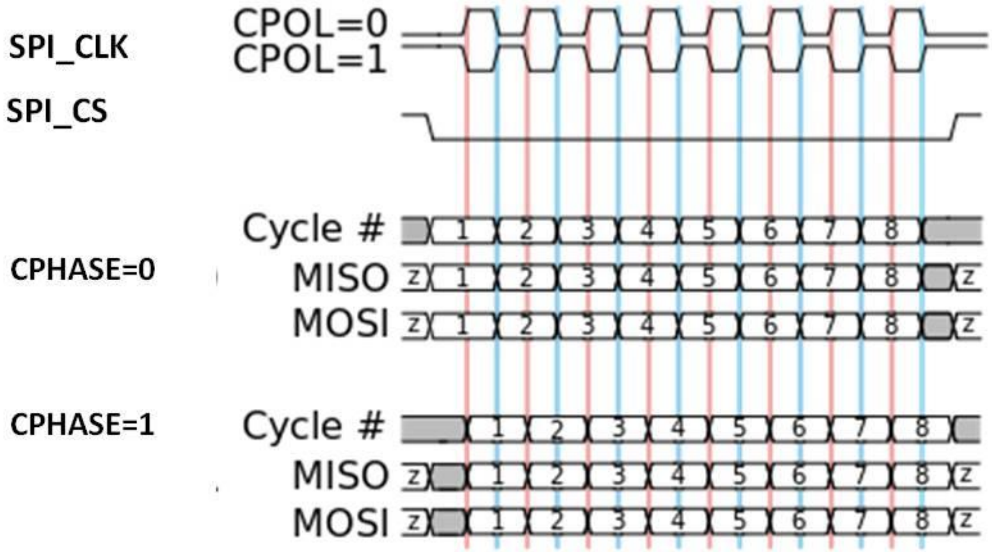


Figure 10: Clock Polarity and Clock Phase

When data is sent from the Host to the module, the MSB should be sent first. The same format is followed when the module sends data to the Host.

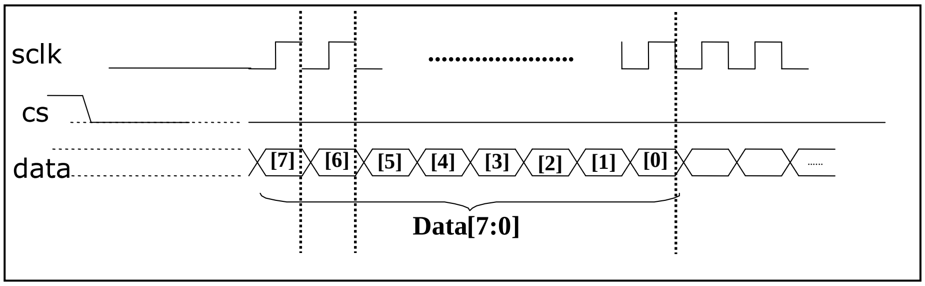


Figure 11: Endianess in Data Transfer

7.2 Configuring and Operating the Module

The main steps to operate the module are named as **Card Ready Operation**, **Tx Operations** and **Rx Operations**. **Tx Operation** and **Rx Operations** are specific activity names defined and used in this document, and should be considered as encompassing more activities than just sending and receiving on-air data. For example, Tx and Rx operations are used to send commands and receive responses from the module as well.

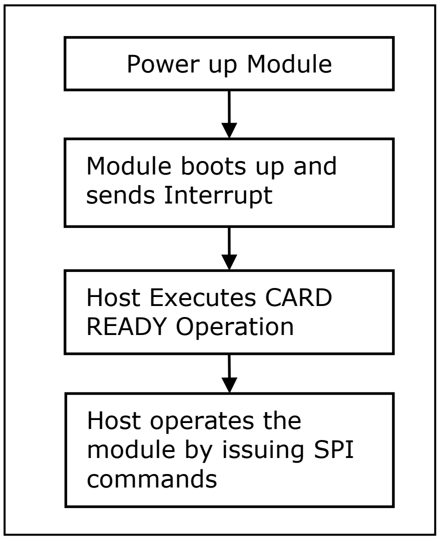

Power up Module
Module boots up and sends Interrupt
Host Executes CARD READY Operation
Host operates the module by issuing SPI commands

Figure 12: Module Operation

1. **Tx Operation** – The Host uses Tx operations:
   a. To send configuration commands to the module from the Host
   b. To send actual data to the module, to be transmitted into air
2. **Rx Operation –** The Host uses this operation:
   a. To receive module's responses to commands issued to the module
   b. To read data received by the module from the remote terminal.
3. **Card Read Operation** – The Card Ready Operation ensures that the module has booted up successfully. It is described in detail in section Card Ready Operation. The Host should proceed with issuing commands for general operation of the module only after the Card Read Operation is successfully executed.

7.2.1 Tx Operation

The below flowchart shows the sequence of steps for Tx operation.

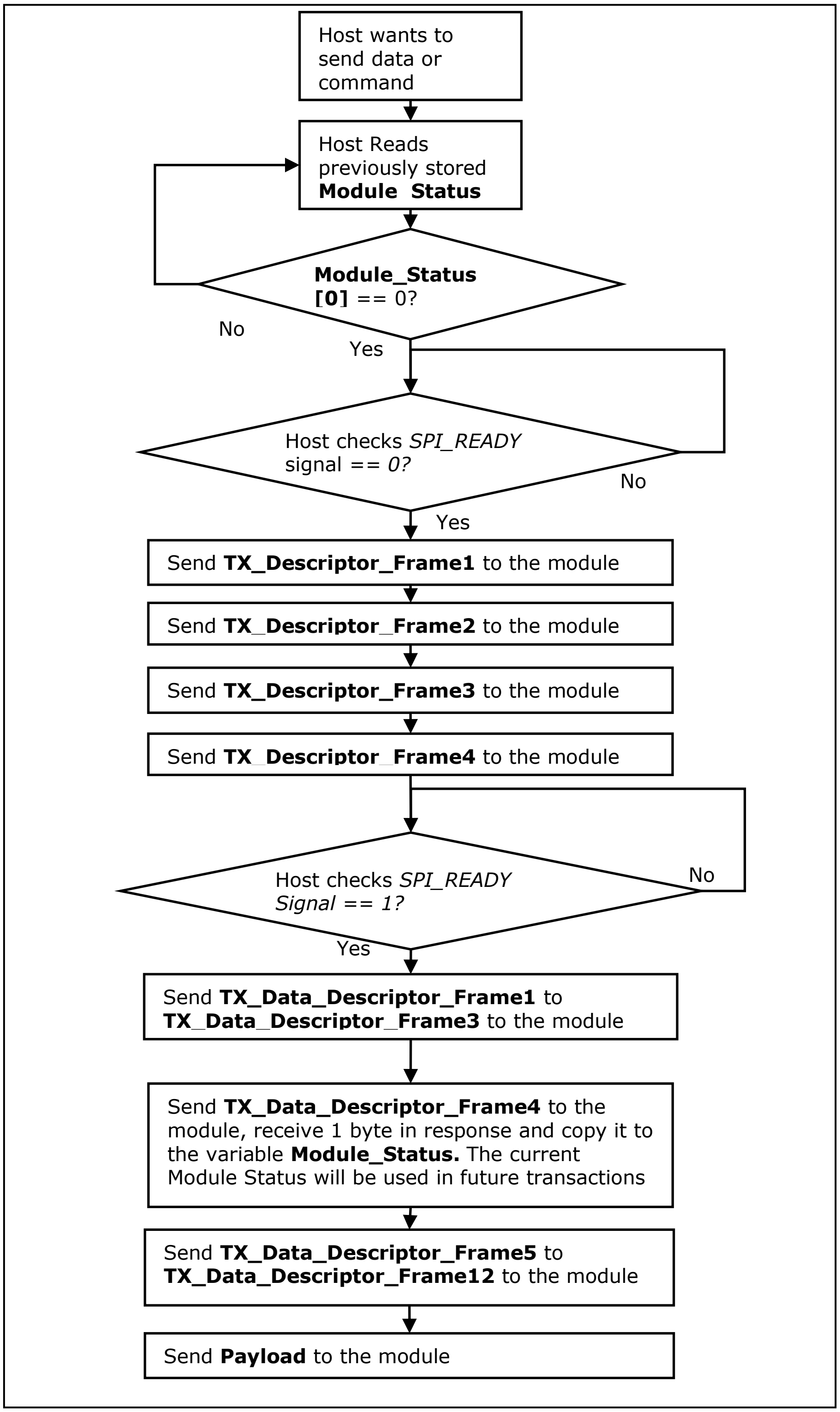

Host wants to send data or command
Host Reads previously stored **Module Status**
Module_Status [0] == 0?
No
Yes
Host checks *SPI_READY* signal == *0?*
No
Yes
Send **TX_Descriptor_Frame1** to the module
Send **TX_Descriptor_Frame2** to the module
Send **TX_Descriptor_Frame3** to the module
Send **TX_Descriptor_Frame4** to the module
Host checks *SPI_READY Signal == 1?*
No
Yes
Send **TX_Data_Descriptor_Frame1** to **TX_Data_Descriptor_Frame3** to the module
Send **TX_Data_Descriptor_Frame4** to the module, receive 1 byte in response and copy it to the variable **Module_Status.** The current Module Status will be used in future transactions
Send **TX_Data_Descriptor_Frame5** to **TX_Data_Descriptor_Frame12** to the module
Send **Payload** to the module

Figure 13: Tx Operation[1]

[1] The operations in the figure are executed by the Host.

7.2.1.1 TX_Descriptor_Frames

TX_Descriptor_Frames are 1-byte frames. Individual bits are described below.

Frame	Description
Tx_Descriptor_Frame1	Bit[7:0] – Length of the payload in number of bytes
Tx_Descriptor_Frame2	Bit[3:0] – Length of the payload in number of bytes Bit[7:4] – ‘0000’ – For sending SPI commands ‘0010’ – For sending actual data to be transmitted
Tx_Descriptor_Frame3	Reserved. Set to all ‘0’
Tx_Descriptor_Frame4	Reserved. Set to all ‘0’

Table 7: Tx_Descriptor_Frames

The length of the payload is a 12-bit field. Tx_Descriptor_Frame2[3] is the MSB of this parameter, and Tx_Descriptor_Frame1[0] is the LSB.

7.2.1.2 TX_Data_Descriptor_Frames

The **TX_Data_Descriptor_Frame**s are 1-byte frames.

Bit Number	Description
TX_Data_Descriptor_Frame1 to TX_Data_Descriptor_Frame10	Set to all ‘0’
TX_Data_Descriptor_Frame11	Command ID
TX_Data_Descriptor_Frame12	Set to all ‘0’

Table 8: Tx_Data_Descriptor Frames

The **Command ID** in Tx_Data_Descriptor_Frame11 is a unique identifier to indicate to Host what kind of command is being issued from the Host to the module.

Command	Command ID
Send Data	0x00
Set Operating Mode	0x10
Band	0x11
Init	0x12
Scan	0x13
Join	0x14
Set Power Mode	0x15
Set Sleep Timer	0x16
Query Network Parameters	0x18
Disconnect	0x19
RSSI Query	0x1A
Select Antenna	0x1B
Soft Reset	0x1C
Feature Select	0x1E
Query SNR	0x1F
Store Configuration	0x20
Enable Auto-join or Auto-create	0x21
Get Configuration Information	0x22
Configure AP Mode	0x24
Set WEP Keys	0x25
GPIO configuration	0x28
Ping request	0x29
Query stats	0x38
Set IP Parameters	0x41
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 Parameters	0x4E
Load Web Fields	0x4F
Load Webpage	0x50
HTTP GET	0x51
HTTP POST	0x52
DNS Server	0x55
Get socket info	0x57

Table 9: Command IDs for Tx Data Operation

7.2.1.3 Module_Status

Module_Status is a 1-byte variable. The variable is used and updated with the latest values both during Tx and Rx Operations as shown in figures Tx Operation and Rx Operation.

Frame	Description
Module_Status	0x01 – Module buffers full 0x02 – Module buffers empty 0x04 – Receive data pending to be read from the module by Host 0x08 – Module ready to go to power save.

Table 10: Module Status

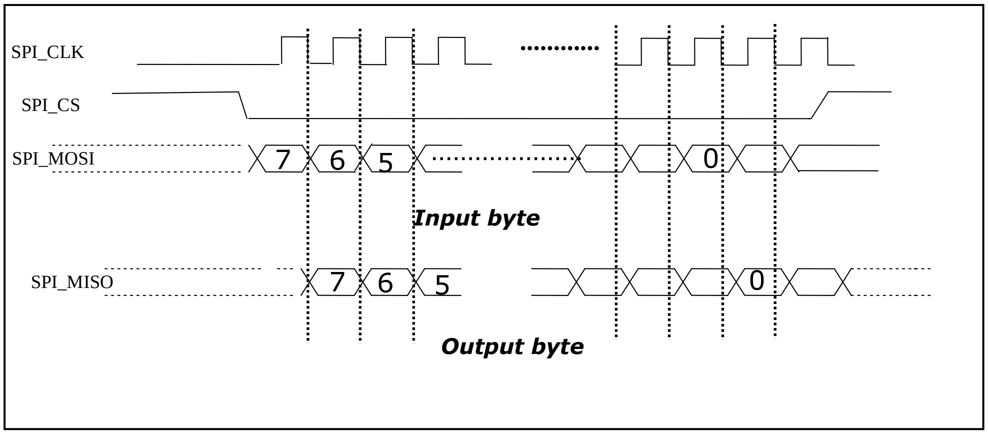


Figure 14: SPI Transactions

Transaction over the SPI interface happens in units of bytes. With every bit of a byte sent from the Host to the module into the SPI_MOSI line, a bit of the corresponding output byte is sent by the module in the SPI_MISO line.

7.2.1.4 Payload

The **Payload** contains parameters for SPI commands from Host to module or actual data to be sent to the remote terminal. The number of bytes in the Payload is given in **Tx_Descriptor_Frame1**[3:0] and **Tx_Descriptor_Frame2**[7:0]. The content of the Payload is given in the individual descriptions of the commands in the sub-section "Payload Structure" in SPI Commands .

7.2.2 Rx Operation

The Host uses this operation to receive responses to commands issued to the module, or to read data received by the module from the remote terminal. The module sends an interrupt to indicate the Host that an Rx Operation should be initiated.

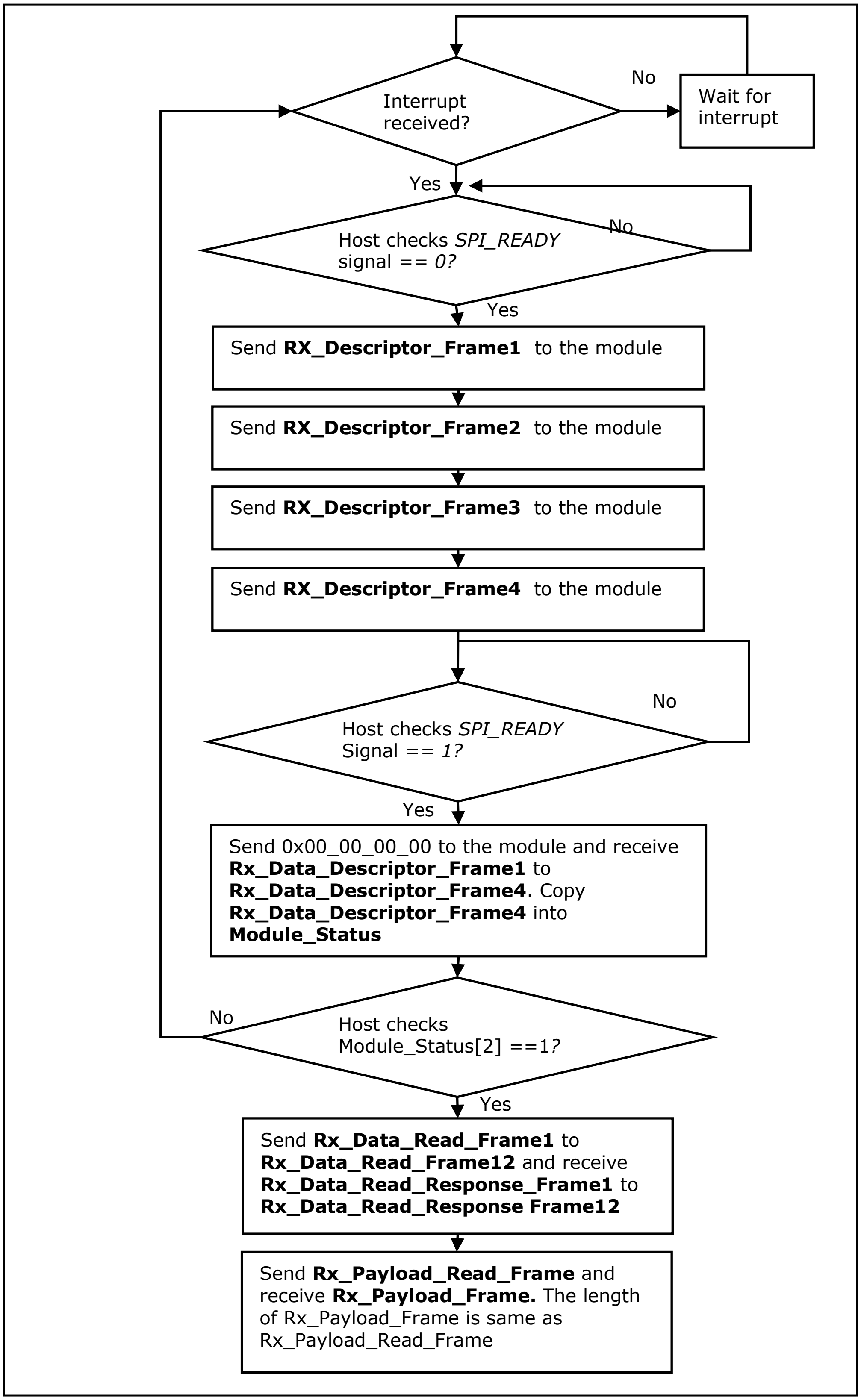

Interrupt received?
No
Wait for interrupt
Yes
Host checks *SPI_READY* signal == *0?*
No
Yes
Send **RX_Descriptor_Frame1** to the module
Send **RX_Descriptor_Frame2** to the module
Send **RX_Descriptor_Frame3** to the module
Send **RX_Descriptor_Frame4** to the module
No
Host checks *SPI_READY* Signal == *1?*
Yes
Send 0x00_00_00_00 to the module and receive **Rx_Data_Descriptor_Frame1** to **Rx_Data_Descriptor_Frame4**. Copy **Rx_Data_Descriptor_Frame4** into **Module_Status**
No
Host checks Module_Status[2] ==1*?*
Yes
Send **Rx_Data_Read_Frame1** to **Rx_Data_Read_Frame12** and receive **Rx_Data_Read_Response_Frame1** to **Rx_Data_Read_Response Frame12**
Send **Rx_Payload_Read_Frame** and receive **Rx_Payload_Frame.** The length of Rx_Payload_Frame is same as Rx_Payload_Read_Frame

Figure 15: Rx Operation

7.2.2.1 RX_Descriptor_Frames and Rx_Data_Descriptor_Frames

RX_Descriptor_Frames and **Rx_Data_Descriptor_Frames** are 1-byte frames. Individual bits are described below.

Frame	Description
Rx_Descriptor_Frame1	Set to all '0'
Rx_Descriptor_Frame2	Set to all '0'
Rx_Descriptor_Frame3	Set to all '0'
Rx_Descriptor_Frame4	Set to 0x01

Table 11: Rx_Descriptor_Frames

Frame	Description
Rx_Data_Descriptor _Frame1	Length of the Payload to be read from the module
Rx_Data_Descriptor _Frame2	Bits[3:0] – Length of the Payload to be read from the module Bits[7:4] '0000' - Interrupt was raised by module to indicate the Host should now read the response to a command '0010' - Interrupt was raised by module to indicate the Host should now read the data that was received by the module from a remote terminal
Rx_Data_Descriptor _Frame3	Error Code for the command as described in table Error Codes for SPI. In case of negative error codes which will be reported to host in two bytes, it is 0xFF. If this byte is 0xFF, the two byte error code can be retrieved from the frames Rx_Data_Read_Response_Frame9 and Rx_Data_Read_Response_Frame10.
Rx_Data_Descriptor _Frame4	Copied to **Module_Status**: Bits[7:0] Bit[0] is set – Module buffers full

Frame	Description
	Bit[1] is set – Module buffers empty Bit[2] is set – Receive data pending to be read from the module by Host Bit[3;] is set – Module ready to go to power save.

Table 12: Rx_Data_Descriptor Frames

The length of the payload to be read from the module is a 12-bit field. Rx_Descriptor_Frame2 [3] is the MSB of this parameter, and Rx_Descriptor_Frame1 [0] is the LSB.

7.2.2.2 RX_Data_Read_Frames and Rx_Data_Read_Response_Frame

Rx_Data_Read_Frame1 to **Rx_Data_Read_Frame12** are 1 byte frames. The contents of this frame are all '0'. In response to sending these frames, **Rx_Data_Read_Response_Frame1** to **Rx_Data_Read_Response_Frame12** are received.

Bit Number	Description
Rx_Data_Read_Response_Frame1 to Rx_Data_Read_Response_Frame8	Reserved, should be ignored
Rx_Data_Read_Response_Frame9 to Rx_Data_Read_Response_Frame 10	Negative error codes for the command as described in table Error Codes for SPI will be reported in two bytes. For example, If error code is '0xFEFF', this represents error code '-1' which is 2's complement of '0xFFFE'.
Rx_Data_Read_Response_Frame11	Response ID. Refer table Response IDs for Rx Operation Command IDs for Tx Data Operation
Rx_Data_Read_Response_Frame12	Reserved, should be ignored

Table 13: Rx_Data_Read_Response_Frame

Command	Response ID
Set Operating Mode	0x10
Band	0x11

Init	0x12
Scan	0x13
Join	0x14
Set Power Mode	0x15
Set Sleep Timer	0x16
Query Network Parameters	0x18
Disconnect	0x19
RSSI Query	0x1A
Select Antenna	0x1B
Feature Select	0x1E
Query SNR	0x1F
Store Configuration	0x20
Enable Auto-join or Auto-create	0x21
Get Configuration Information	0x22
Configure AP Mode	0x24
Set WEP Keys	0x25
GPIO configuration	0x28
Ping from module	0x29
Asynch connection accept request from remote wfd device	0x30
Get stats	0x38
IP Parameters Configure	0x41
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 Parameters	0x4E

Load Web Fields	0x4F
Load Webpage	0x50
HTTP GET	0x51
HTTP POST	0x52
Async WFD	0x54
DNS Server	0x55
Host sends queried web page to the module	0x56
Get socket info	0x57
Async TCP Socket Connection Established	0x61
Async Socket Remote Terminate	0x62
Module indicates to Host that a web page is being queried by a remote terminal	0x64
Card Ready	0x89
Soft Reset	0x89
Receive data	Ignore the response ID field while receiving data from a remote terminal
Error for send data	0x71

Table 14: Response IDs for Rx Operation

The **Response ID** in Rx_Data_Read_Response_Frame11 is a unique identifier to indicate to Host the command for which the module is issuing a response.

Note: The error for send command, in case data length exceeds mss will be reported as mgmt type 0x71.

7.2.2.3 Rx_Payload_Read_Frame and Rx_Payload_Frame

The **Rx_Payload_Read_Frame** is a frame whose length (in number of bytes) is calculated from Rx_Data_Descriptor_Frame1 and Rx_Data_Descriptor_Frame2 (table Rx_Data_Descriptor Frames). The contents of the frame are all '0'.

Rx_Payload_Frame is received by the Host while sending the Rx_Payload_Read_Frame. It is the actual payload (referred to as "Response Payload" in the description for individual commands) received from the module. Its length is same as the length of Rx_Payload_Read_Frame.

IMPORTANT: During a Tx or Rx operation, If host waits for a time greater than SPI ready timeout provided with driver, corresponding API will return SPI ready timeout error. In this case, Host has to reset the module and start from beginning.

7.3 Card Ready Operation

The **Card Ready Operation** is executed as shown below

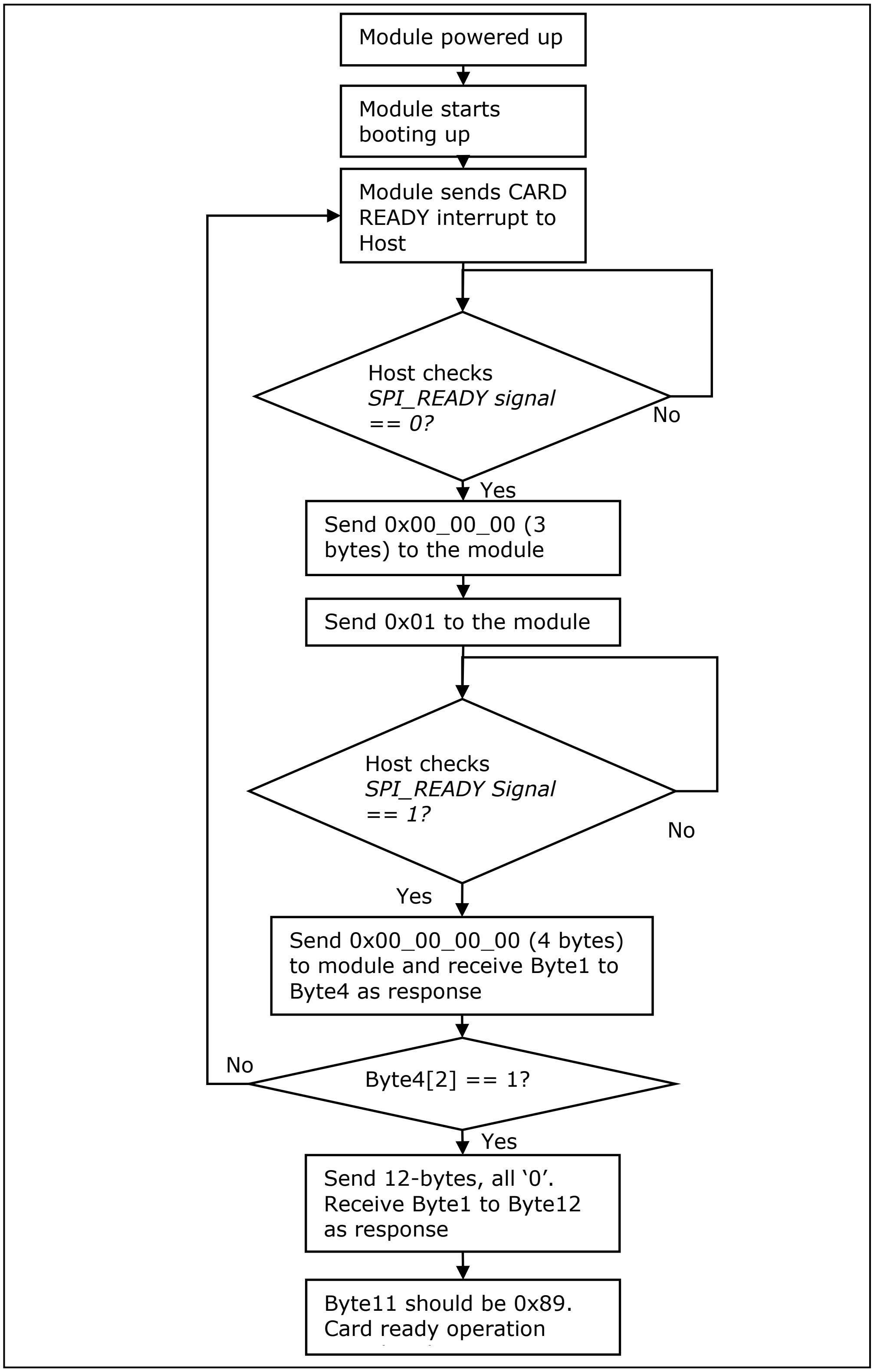
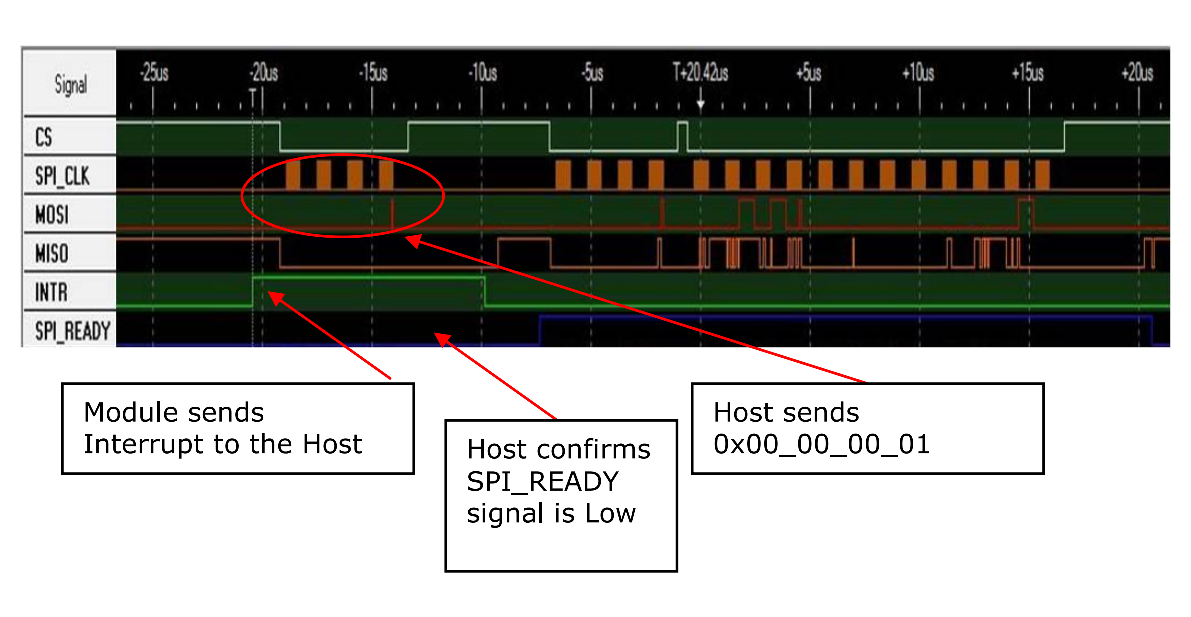

Module powered up
Module starts booting up
Module sends CARD READY interrupt to Host
Host checks *SPI_READY signal == 0?*
No
Yes
Send 0x00_00_00 (3 bytes) to the module
Send 0x01 to the module
Host checks *SPI_READY Signal == 1?*
No
Yes
Send 0x00_00_00_00 (4 bytes) to module and receive Byte1 to Byte4 as response
No
Byte4[2] == 1?
Yes
Send 12-bytes, all '0'. Receive Byte1 to Byte12 as response
Byte11 should be 0x89. Card ready operation

Figure 16: Card Ready Operation



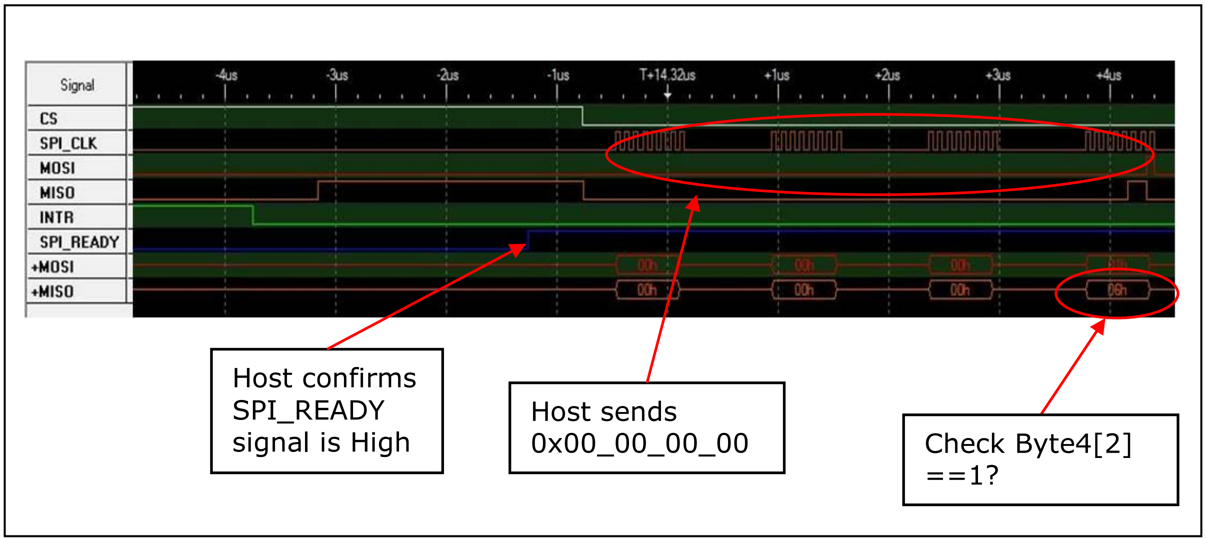
Module sends Interrupt to the Host

Host confirms SPI_READY signal is Low

Host sends 0x00_00_00_01

Figure 17: Example Signal Sequencing -1

The figure above shows the first 4 bytes 0x00_00_00_01 sent when the SPI_READY signal is low. Similarly, 0x00_00_00_00 (Actually these are dummy bytes, no matter what these bytes are) is sent when the SPI_READY is High, as shown in the below figure. Note that the SPI_CS (active low signal) is active when the SPI transactions are taking place.

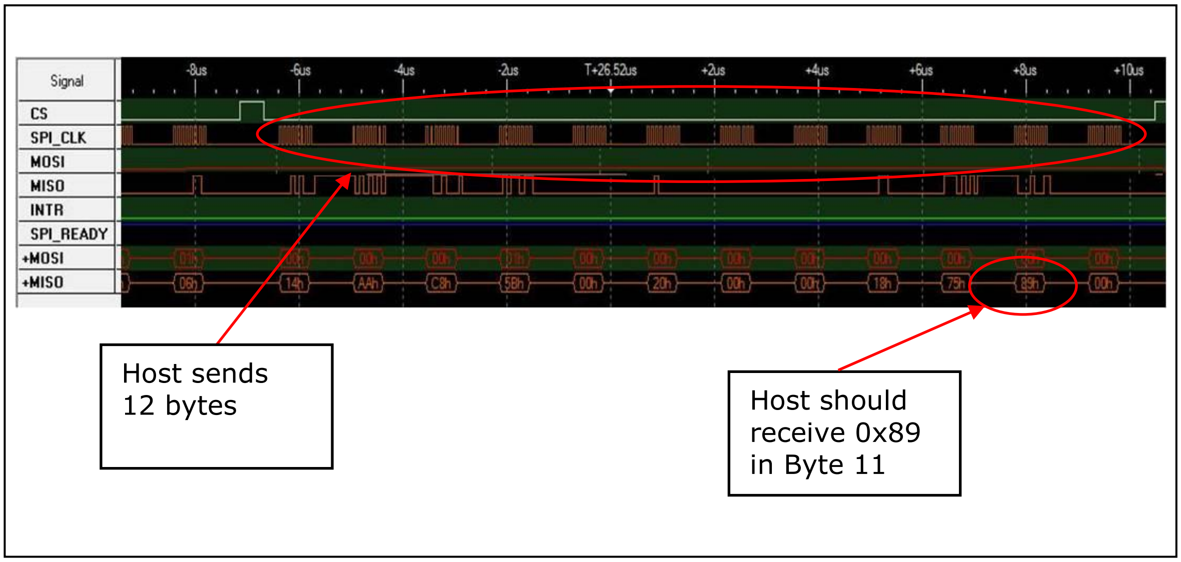


Host confirms SPI_READY signal is High

Host sends 0x00_00_00_00

Check Byte4[2] ==1?

Figure 18: Example Signal Sequencing- 2



Host sends 12 bytes

Host should receive 0x89 in Byte 11

Figure 19: Example Signal Sequencing-3

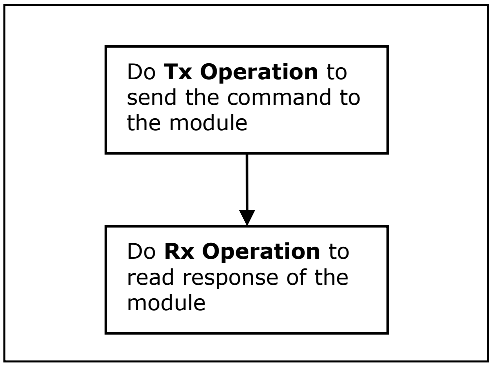
Interrupt Troubleshooting Tip: The CARD READY interrupt is the first valid interrupt sent by the module after power up. Any spurious activity on the interrupt line before the actual valid interrupt should be ignored by the Host, or else CARD READY operation may be wrongly initiated and valid response (0x89) may not be read. This can be done by:

1. Not enabling the interrupt in the Host till the passage of at least 1750 msecs after the Wi-Fi module is powered on. The valid CARD READY interrupt from the module is not sent earlier than 1750 msecs

2. Configuring the interrupt reception of the Host to Level Triggered rather than Edge Triggered so that spurious activity in the interrupt line is ignored.

7.4 SPI Commands

The following commands are used to configure and operate the module. The Host first does a Tx Operation to send the command to the module, and then a Rx Operation to receive the response to the command.


Do **Tx Operation** to send the command to the module
Do **Rx Operation** to read response of the module

7.4.1 Set Operating Mode

Description

This is the first command that needs to be sent from the Host. This command configures the module in different functional modes.

Payload Structure

The structure of the payload is give below

```
struct {
        uint8   operMode;
   } operModeFrameSnd;
```

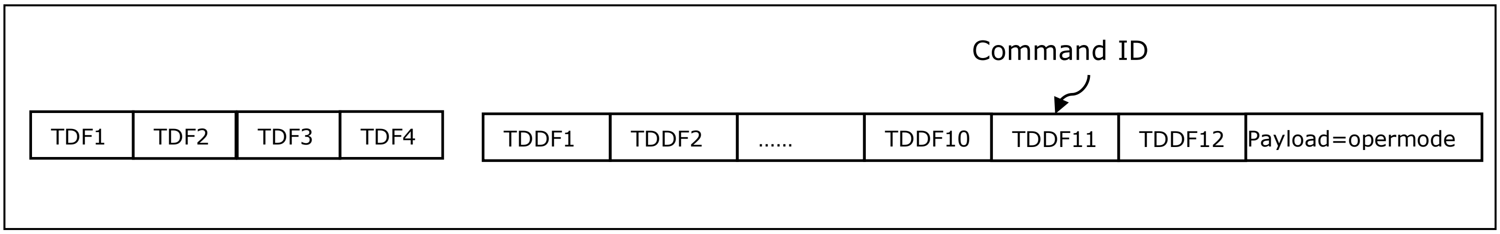


Figure 20: Sending a Command to the Module

The above figure shows the sequence of frames to help the reader correlate the operation in figure Tx Operation to the process of sending a command ("Set Operating mode" in this case). This structure is same for all commands that follow. There are a few commands where the Input Payload will not be present.
Abbreviations used are:
TDF – Tx Descriptor Frame
TDDF- Tx Data Descriptor Frame

Parameters

operMode: Sets the mode of operation

0– **Operating Mode 0**: Normal Client Mode. Wi-Fi Direct and Access Points are disabled in this mode. The module works as a normal client that can connect to an Access Point with WPA/WPA2-PSK in CCMP and TKIP modes of security and in open mode.

1– **Operating Mode 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**, depending on the inputs supplied for the command "Configure Wi-Fi Direct Peer-to-Peer Mode". In Autonomous GO and in Wi-Fi Direct Group Owner mode, a maximum of 4 client devices are supported. Wi-Fi Direct Group Owner mode is described in the following sections.

2– **Operating Mode 2:** Enterprise Client Mode. Wi-Fi Direct and Access Point modes are disabled in this mode. The module works as a client that can connect to an Access Point with WPA/WPA2-Enterprise security.

3– **Operating Mode 3**: Normal Client Mode. Wi-Fi Direct and Access Points are disabled in this mode. The module works as a normal client that can connect to an Access Point with WPA/WPA2-PSK in CCMP and TKIP modes of security and in open mode. The TCP/IP stack is bypassed in this mode. The Host can use its own TCP/IP stack.

4– **Operating Mode 4: Wi-Fi Direct™** or **Access Point** mode. In this mode, the module either acts as a Wi-Fi Direct node or as an Access Point, depending on the inputs supplied for the command "Configure Wi-Fi Direct Peer-to-Peer Mode". In Access Point mode and in Wi-Fi Direct Group Owner mode, a maximum of 4 client devices are supported. Wi-Fi Direct Group Owner mode is described in the following sections. The TCP/IP stack is bypassed in this mode. The Host can use its own TCP/IP stack.

5–**Operating Mode 5**: Enterprise Client Mode. Wi-Fi Direct and Access Point modes are disabled in this mode. The module works as a client that can connect to an Access Point with WPA/WPA2-Enterprise security. The TCP/IP stack is bypassed in this mode. The Host can use its own TCP/IP stack

6– **Operating Mode 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 4 client devices are supported.

7– **Operating Mode 7: 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 4 client devices are supported. The TCP/IP stack is bypassed in this mode. The Host can use its own TCP/IP stack.

NOTE: When the TCP/IP stack is bypassed inside the Wi-Fi module, it presents itself as a standard network interface, which would talk to the Host TCP/IP stack through the network driver in the Host. For transmitting data to the air interface, the data should be framed by the TCP/IP stack in the Host as an Ethernet frame, and the Wi-Fi module does only the WLAN framing on top of it. The interface is according to standard networking protocols. For example : http://www.makelinux.com/ldd3/chp-17-sect-3 describes the structure for Linux. For successful sending and receiving of data, the MAC ID of the module should be assigned to a field inside the net_device structure dev_addr of the network driver on the Host OS. Once this is stored after association with AP, the user can bring the wlan interface up

In Linux, this is usually done by:

ifconfig wlan0 <ipaddr>

ifconfig wlan0 up

The user can query wlan0 interface status using:

ifconfig wlan0

The Redpine Module's MAC address should be listed in the HW Addr (00:23:A7:xx:xx:xx).

Response Payload

There is no response payload for this command.

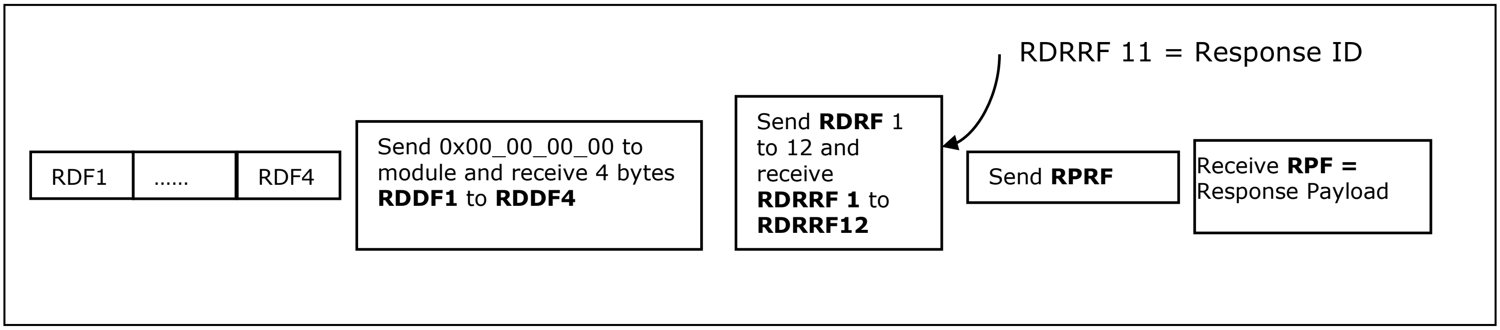

RDRRF 11 = Response ID
RDF1
……
RDF4
Send 0x00_00_00_00 to module and receive 4 bytes **RDDF1** to **RDDF4**
Send **RDRF** 1 to 12 and receive **RDRRF 1** to **RDRRF12**
Send **RPRF**
Receive **RPF =** Response Payload

Figure 21: Receiving Response from the Module

The above figure shows the sequence of frames to help the reader correlate the operation in figure Rx Operation to the process of receiving a response to a command. This structure is same for all commands that follow. There are a few commands where the Response Payload will not be present (for example the current command "Set Operating Mode"). In such cases the Host can stop after completion of "Send RDRF1 to RDRF12 and receive RDRRF1 to RDRRF12". In commands where there is a response payload (for example the Scan command), the entire sequence should be executed.

Abbreviations used are:

RDF – Rx Descriptor Frame
RDDF- Rx Data Descriptor Frame
RDRF- Rx Data Read Frame
RDRRF- Rx Data Read Response Frame
RPRF- Rx Payload Read Frame
RPF- Rx Payload Frame

Possible error codes

Possible error codes are 33, 37 and 44.

7.4.2 Band

Description

This command configures the band in which the module has to be configured. RS-WC-201/301 is a single band module (2.4 GHz only) and RS-WC-301 is a dual band module(2.4 GHz and 5 GHz).

Payload Structure

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

Parameters

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

bandVal:

When Operating Mode =0 or 2

0–2.4 GHz

1–5 GHz. Applicable only for RS-WC-301 module.

When Operating Mode =1

Wi-Fi Direct Mode: If the module is configured as a Wi-Fi Direct node.

0–2.4 GHz is used both during Group Owner (GO) negotiation and general operation

1–2.4 GHz is used during GO Negotiation, but module will operate on 5GHz if it becomes the GO after the GO negotiation process is over.

Access Point Mode: If the module is configured as an AP

0–AP is configured to operate in 2.4 GHz

1–AP is configured to operate in 5 GHz.

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 5, 33, 37 and 44.

Relevance

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 3, 4, 5, 6 and 7.

7.4.3 Init

Description

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

Payload Structure

No Payload required.

Parameters

No parameters

Response Payload

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

Response Parameters

macAddress: The MAC ID of the module.

Possible error codes

Possible error codes are 33, 37 and 44.

Relevance

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 3, 4, 5, 6 and 7.

7.4.4 Antenna Selection

Description

This command configures the antenna to be used. RS-WC-201/301 provides two options – an inbuilt chip 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 chip antenna is selected.

Payload Structure

```
struct {
             uint8 AntennaVal;
          } AntennaSelFrameSnd;
```

Parameters

AntennaVal:

1–Internal Antenna is selected

2–uFL path is selected. The user can plug in an external antenna with this option.

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 33, 37 and 44.

Relevance

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 3, 4, 5, 6 and 7.

7.4.5 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 by raising an interrupt.

Payload Structure

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

Parameters

GOIntent:

Wi-Fi Direct Mode: 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.

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

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

operChannel: Operating channel to be used in Group Owner or Access Point mode. The specified channel is used if the device becomes a GO or Access Point. 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* 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 tables Channels in 2.4 GHz and Channels in 5 GHz. '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 Access Point mode.

If the module becomes a Wi-Fi Direct Group Owner, it would have an SSID with "DIRECT-xy" prefixed to the *ssid_postfix* 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[1].

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

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

Response Payload

There is no response payload for this command.

After the command is received by the module, it scans for WiFi Direct devices if it was configured for WiFi Direct mode. If 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** (table Response IDs for Rx Operation) for the Response ID and the below structure as the *Payload*.

Structure:

```
typedef struct {
        uint8   devState;
        uint8   devName[34];
        uint8   macAddress[6];
        uint8   devtype[2];
}rsi_wfdDevInfo;
```

[1] After the module becomes a GO in WiFi direct mode, it appears as an Access Point to client devices.

After the command is received, the device is scanned by the other Wi-Fi Direct devices too. If any of those devices sends 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** (table Response IDs for Rx Operation) for the Response ID and the below structure as the *Payload*.

```
typedef struct {
          uint8   devName[32];
          /* If the devName is of 32 bytes, then all the bytes are valid
  characters only */
}rsi_ConnAcceptRcv;
```

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 Wi-Fi Direct Device Type.

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

device_state.

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

device_name(32 bytes): All are 0x00's

device_mac (6 bytes): MAC ID of the remote Wi-Fi Direct node which is moved out of range.

Device_type (2 bytes)

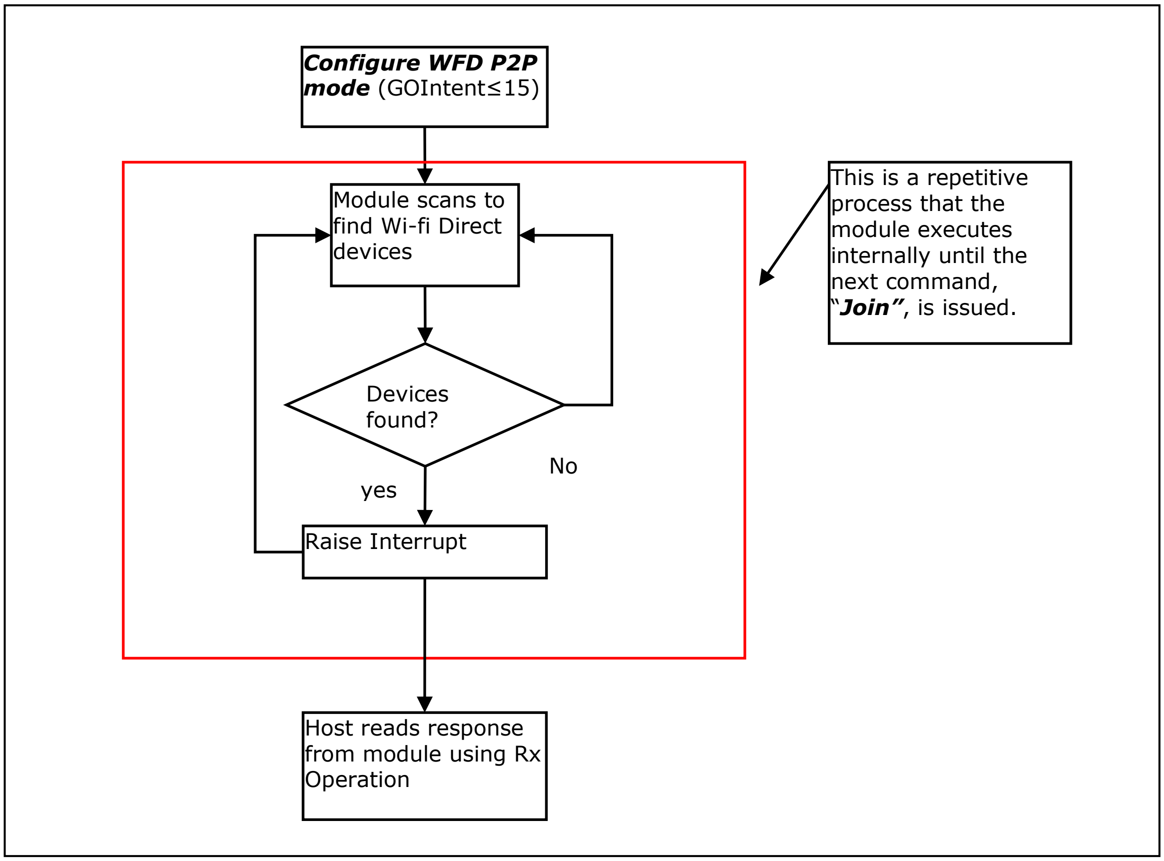

Configure WFD P2P mode (GOIntent≤15)
Module scans to find Wi-fi Direct devices
Devices found?
No
yes
Raise Interrupt
Host reads response from module using Rx Operation
This is a repetitive process that the module executes internally until the next command, "Join", is issued.

Figure 22: Operation after issuing "Configure WFD P2P Mode" command

Possible error codes

Possible error codes are 29, 33, 37 and 44.

Relevance

This command is relevant when the module is configured in Operating Mode 1 and 4.

Note: After getting the connect request from remote device, host need to issue the "join" command with the remote device name that sent the request. User need to make sure that the remote device is scanned by us too (Async WFD with remote device name). If the user issues join before remote device get scanned by us, will get join response with error "25".

7.4.6 Configure AP Mode

Description

If the module is to be used as an AP, this command is used to set the parameters of the AP.

Payload Structure

```
struct ap_conf_s
  {
   UINT8 channel_no[2];
   UINT8  ssid[34];
   UINT8  sec_type;
   UINT8  enc_type;
   UINT8   psk[64];
   UINT8 beacon_interval[2];
   UINT8 dtim_period[2];
   UINT8 reserved[2];
   UINT8 max_sta_support[2];
  };
```

Parameters

channel_number: The channel in which the AP would operate. Refer tables Channels in 2.4 GHz and Channels in 5 GHz. A value of '0' is not allowed.

ssid: SSID of the AP to be created

sec_type: Security type.

0-Open

1-WPA

2-WPA2

enc_type: 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'.

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

max_sta_support: Number of clients supported. The maximumvalue of this parameter is 4. For example, if this parameter is 3, not more than 3 clients can associate to the AP.

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 33, 37 and 44.

Relevance

This command is relevant when the module is configured in Operating Mode 6 and 7.

> Note: In WiSeConnect^{TM} AP mode, only mixed 802.11g and 802.11b is supported.

7.4.7 Feature Select

Description

This command is used to control behaviours with respect to specific commands.

Payload Structure

```
struct {
            uint8   bitmap[4];
} featselFrameSnd;
```

Parameters

bitmap:

bit_map[0] :

'1' – Add Cisco AP name in the "Scan" command's response

'0' – Default behavior of Scan command's response. The default value of this bit is '0'.

bit_map[1] :

'1' – Add SNR value in the "Scan" command's response

'0' – Default behavior of Scan command's response. The default value of this bit is '0'.

bit_map[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_map[3]: If this bit is set to '1', HTTP web server in the module is disabled completely. The default value of this bit is '0'.

bit_map[4]: If this bit is set to '1', UART hard ware flow control is enabled. The default value of this bit is '0'.

bit_map[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_map[6] : 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, sends out a DNS address with assigned IP and Subnet values to the client. The default value of this bit is '0'.

bit_map[7] : If this bit is set to '1', the DHCP client behavior, when the module is in STA mode, changes. The DHCP client sends DHCP discover and DHCP request with unicast flag. The default value of this bit is '0'.

bit_map[8] : If this bit is set to '1', then BT co-existence is enabled. Once the WLAN connection is successful, the module honours BT priority data. The default value of this bit is '0'.

bit_map[9] : If this bit is set to '1', then module reports errors for send data packets (response code used in this case for reporting errors is 0x71). -2 error code is reported, if the socket is not available with the given socket descriptor. -130 error code is reported, if the MSS of the socket is less than the given data packet's payload.

bit_map[31:10]: Reserved, should be set to all '0'

Note: This is not a mandatory command. It is advised to NOT use this command, unless specific behavior is expected of the "Scan" command, as described in the section below. If this command is used, it should be issued as the first command to the module, before the "Set Operating Mode" command.

BT priority and WLAN active pins from module should be connected to BT device for co-existence to work. BT-Priority is an input to the module and WLAN active is an output from the module. When there is BT priority data, BT device should set BT priority pin. Please refer to data sheets for pin numbers of BT priority and WLAN active.

Possible error codes

Possible error codes are 33, 37 and 44.

Relevance

This command is relevant to any of the operating modes, based on the features selected with the bitmap.

7.4.8 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 list.

Payload Structure

```
struct {
        uint8           chan_num[4];
         uint8          ssid[34];     /* Optional field fill with null characters if
                                                                    not used */
} scanFrameSnd;
```

Parameters

chan_num: 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 tables Channels in 2.4 GHz and Channels in 5 GHz.

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 10. If not used, null characters should be supplied to fill the structure.

Response Payload

```
struct{
        uint8                          rfChannel;
        uint8                          securityMode;
        uint8                          rssiVal;
        uint8                          uNetworkType;
        uint8                          ssid[34];
        uint8                          bssid[6];
        uint8                          snr_value;
        /* This field will be present only if the feature is enabled through
        the Feature Select command */
        uint8                          reserved;
        uint8                          ap_name[16];
        /* This field will be present only if the feature is enabled through
        the Feature Select command */
}rsi_scanInfo;


typedef struct {
        uint8                          scanCount[4];
         rsi_scanInfo                  strScanInfo[11 ;
} rsi_scanResponse;
```

Response Parameters

rfChannel: Channel Number of the scanned Access Point

securityMode:

0–Open
1–WPA
2–WPA2
3-WEP
4–WPA Enterprise,
5–WPA2 Enterprise

rssival: RSSI of the scanned Access Point

uNetworkType: Network type of the scanned Access Point

1– Infrastructure mode

ssid: SSID of the scanned Access Point

bssid: MAC address of the scanned Access Point

snr_value: The Host should ignore this field if bit_map[1]=0 in the command Feature Select. If bit_map[1]=1, then this field is the value of the SNR. For example, if the SNR is 20dBm, the value reported is 0x14.

Reserved (1 byte): this should be ignored by the Host.

Ap_name (16 bytes): This field is present ONLY if bit_map[0]=1 in the command Feature Select. It contains the Cisco AP Name, and valid for Cisco Aironet Series of devices. It contains all '0' for other devices

scancount: Number of Access Points scanned

Possible error codes

Possible error codes are 2, 3, 10, 33, 36, 37, 37, 38 and 44.

Relevance

This command is relevant when the module is configured in Operating Mode 0, 2, 3 and 5.

Notes:

1. If the number of APs is more than 11 around the area, all the APs may not get scanned. In this case you need to explicitly set the channel to scan that particular AP.

2. Scan requests with channel numbers 12, 13 or 14 is not supported. If user gives scan requests with these scan channels, error 36 will be returned.

3. When the scan is given for all channels, by default channels from1 to 11 will be scanned. Maximum channels to be scanned will be adjusted based on the country IE found in the beacons/probe responses of the APs scanned in channels from 1 to 11.

7.4.9 Set WEP Key

Description

This command configures the WEP key in the module to connect to an AP with WEP security.

Payload Structure

```
struct wepkey_s
{
  uint8 key_index[2];
  uint8  key[4][32];
};
```

Parameters

key_index: 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: 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.

WEP (Hex Mode): In this mode, the key to be supplied to the AP should be 10 digits (for 64 bit WEP mode) or 26 digits (for 128 bit WEP mode), and only the following digits are allowed for the key: A,B,C,D,E,F,a,b,c,d,e,f,0,1,2,3,4,5,67,8,9

Possible error codes

Possible error codes are 33, 37 and 44.

Relevance

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

7.4.10 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 WiFi Direct mode (operating mode 1)

3. Create an Access Point (operating mode 6 and 7)

4. Allow a third party to associate to a WiFi Direct group created by the module

Payload Structure

```
struct {
        uint8                           reserved1;
        uint8                           wep_shared;
        uint8                           dataRate;
        uint8                           powerLevel;
        uint8                           psk[64];
        uint8                           ssid[34];
        uint8                           reserved3;
        uint8                           reserved4;
         uint8                          reserved5;
        uint8                           ssid_len ;
} joinFrameSnd;
```

Parameters

reserved1: Reserved. Set to '0'

wep_shared: Set to '1' in case of WEP SHARED mode. '0' in remaining all modes.

dataRate: Transmission data rate.Physical rate at which data has to be transmitted. Set to 0 if *GOIntent* in Configure Wi-Fi P2P command is 16.

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
48	11
54	12
MCS0	13
MCS1	14
MCS2	15

Data Rate (Mbps)	Value of dataRate
MCS3	16
MCS4	17
MCS5	18
MCS6	19
MCS7	20

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 (15 +/- 2) dBm

At 5 GHz

0– Low power (5+/-1) dBm

1– Medium power (7 +/-1) dBm

2– High power (12 +/- 2) 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 and 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 and 2, and wants to connect to an access point in WPS mode then the value of this parameter is a constant ASCII string WPS_SSID.

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.

Reserved3: Reserved, set to ‘0’

reserved4: Reserved, to set ‘0’

reserved5: Reserved, set to ‘0’

ssid_len: Actual length of the SSID

Response Payload

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

Response Parameters

operState: The value of this parameter varies with the firmware version used.

Firmware version 1.1.0.1.0.0 or below:

0x00 – if the module becomes a Group Owner (GO) after the GO negotiation stage.

0x01 – if the module does not become a GO after the GO negotiation stage.

Firmware version 1.2.1.1.1.0 or above:

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 Wi-Fi Direct Peer-to-Peer Mode).

Note: The module gets a default IP of 192.168.100.76 if it becomes a Group Owner or Access Point.

Possible error codes

Possible error codes are 4, 8, 9, 14, 22, 24, 25, 30, 32, 33, 35, 37, 38, 42, 43 and 44.

Relevance

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 3, 4, 5, 6 and 7. 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.

7.4.11 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 issues an interrupt. If the Host does an Rx operation after receiving the interrupt (the way it does for all other command responses), it will receive error code 37 in Rx_Data_Descriptor_Frame3 and the Join Response ID in

Rx_Data_Read_Response_Frame11. The module aborts the re-join after a fixed number of re-tries. If this happens, an interrupt is sent to the Host. If the Host does an Rx operation after receiving the interrupt (the way it does for all other command responses), it will receive error codes 8 or 25 in Rx_Data_Descriptor_Frame3 and the Join Response ID in Rx_Data_Read_Response_Frame1.

Relevance

This command is relevant when the module is configured in Operating Modes 0, 2, 3 and 5.

7.4.12Set 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.

Payload Structure

```
struct {
        uint8                          eapMethod[32];
        uint8                          innerMethod[32];
        uint8                          userIdentity[64];
        uint8                          password[128] ;
}setEapFrameSnd ;
```

Parameters

eapMethod: Should be one of TLS, TTLS, FAST or MSCHAPV2, ASCII character string

innerMethod: Should be fixed to MSCHAPV2, ASCII character string

userIdentity: 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.

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 28, 33, 37, 44.

Relevance

This command is relevant when the module is configured in Operating Mode 2 and 5.

7.4.13Set Certificate

Description

This command is used to load the certificate or PAC file, after issuing the Set EAP command. This command should be issued if the security mode is EAP-TLS or EAP-FAST

Payload Structure

```
#define MAX_CERT_SEND_SIZE 1400
#define MAX_CERT_LEN       6522

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];
};
```

Parameters

total_length: Certificate’s total length in bytes.

If this value is put to ‘0[1]’, the following are applicable.

a.There are two modes of using EAP-TTLS or PEAP: Password based authentication and server based authentication. If the user is using password based authentication, then this parameter should be set to ‘0’. If the user is using server based authentication, then the correct length of the parameter certificate should be supplied in this command and the appropriate CA (Certification Authority) Root file should be supplied to the certificate parameter.

b.In general a value of ‘0’ can be used to clear the current certificate in the module’s memory.

cert_type: Type of certificate.

1– TLS client certificate

2– FAST PAC file

[1] Value of ‘0’ is supported from firmware version 2.0.0.1.2.4 onwards

more_chunks: A maximum of 1400 bytes of the certificate can be sent to the module from the Host. If the certificate length is more than 1400 bytes, then the certificate need to be sent over multiple segments. 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[1].Set this parameter to all '0' if *total_length* is '0'.

cert_length: Length of the current segment. Set this parameter to all '0' if *total_length* is '0'

keyPwd: Private key password, used to generate the certificate. Set this parameter to all '0' if *total_length* is '0'

certificate: This is the data of the actual certificate.

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

[1] Check the file RS.WSC.x.x.GENR.x.x.x.x.x.x\Resources\SPI\Driver\API_Lib for reference

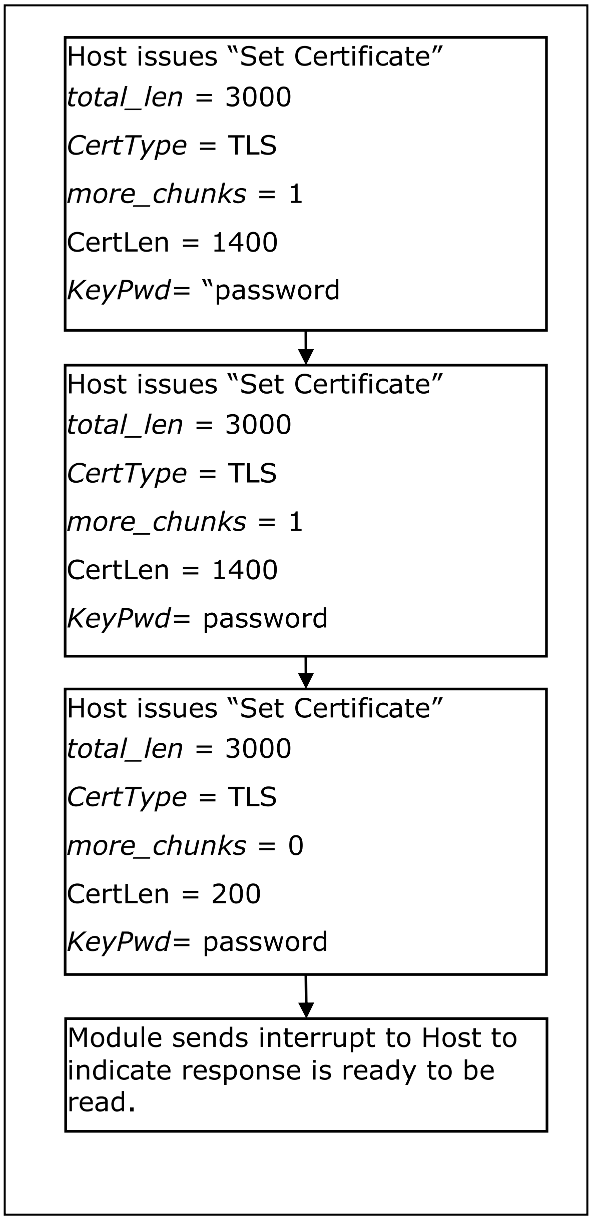


Figure 23: Loading Certificate in SPI mode

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 33, 37, 41 and 44.

Relevance

This command is relevant when the module is configured in Operating Mode 2 and 5.

7.4.14 Set IP Parameters

Description

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

Payload Structure

```
struct {
        uint8                   dhcpMode;
        uint8                   ipaddr[4];
        uint8                   netmask[4];
        uint8                   gateway[4];
        } ipparamFrameSnd;
```

Parameters

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

0–Manual

1–DHCP

ipAddr: IP address in dotted decimal format. This can be 0's in the case of DHCP.

Netmask: Subnet mask in dotted decimal format. This can be 0's in the case of DHCP.

Gateway: Gateway in the dotted decimal format. This can be 0's in the case of DHCP.

Response Payload

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

Response Payload

macAddr: MAC Address

ipAddr: Assigned IP address

netmask: Assigned subnet address

gateway: Assigned gateway address

Possible error codes

Possible error codes are 33, 37, 44, -4, -99 and -100.

Relevance

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

> Note: When WiSeConnect operating in client mode and its IP is assigned using DHCP, an asynchronous error will come to host in case of DHCP renewal failure. When this error comes, user need to issue “disassoc” or “reset” command to join network and get an IP again.

7.4.15 Open a Socket

Description

This command opens a TCP/UDP client socket, a Listening TCP/UDP socket or a multicast socket. This command enables the TCP/IP stack to perform the corresponding action on opening a socket.

Payload Structure

```
struct {
        uint8                   socketType[2];
        uint8                   moduleSocket[2];
        uint8                   destSocket[2];
        uint8                   destIpaddr[4];
} socketFrameSnd;
```

Parameters

socketType: Type of the socket

0– TCP Client

1– UDP Client

2– TCP Server (Listening TCP)

3– Multicast socket

4– Listening UDP

moduleSocket: Port number of the socket in the module. Value ranges from 1024 to 49151, for multicast sockets it can range from 0 to 65535.

destSocket: destination port. Value ranges from 1024 to 49151. Ignored when TCP server or Listening UDP sockets are to be opened.

destIpaddr: IP Address of the Target server. Ignored when TCP server or Listening UDP sockets are to be opened.

Response Payload

```
typedef struct {
        uint8                   socketType[2];
        uint8                   socketDescriptor[2];
        uint8                   moduleSocket[2];
        uint8                   moduleIpaddr[4];
} rsi_socketFrameRcv;
```

Response Parameters

socketType: Type of the created socket.

0–TCP Client

1-UDP Client

2–TCP Server (Listening TCP)

4–Listening UDP

socketDescriptor: Created socket's descriptor or handle, starts from 1. If the module is a WiFi Direct GO or Access Point, then *sockeDescriptor* ranges from from 2 to 8. The first socket opened will have a socket handle of 2, the second socket will have a handle of 3 and so on. When the module is a client, *sockeDescriptor* ranges from from 2 to 8. The first socket opened will have a socket descriptor of 1, the second socket will have 2 and so on.

moduleSocket: Port number of the socket in the module.

moduleIpaddr: The IP address of the module.

Possible error codes

Possible error codes are 33, 37, 44, -2, -95, -121, -123, -127, -128 and -211.

Relevance

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

7.4.16 Close a Socket

Description

This command closes a TCP/UDP socket in the module.

Payload Structure

```
struct {
        uint8                           socketDescriptor[2];
} socketCloseFrameSnd;
```

Parameters

socketDescriptor: Socket descriptor of the socket to be closed

Response Payload

```
typedef struct {
        uint8                           socketDsc[2];
} rsi_socketCloseFrameRcv;
```

Response Parameters

socketDsc: Socket descriptor of the socket closed

Possible error codes

Possible error codes are 33, 37, 44 and -192.

Relevance

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

7.4.17 Query a Listening Socket's Active Connection Status

Description

This command is issued when a listening/server TCP socket has been opened in the module, to know whether the socket got connected to a client socket.

Payload Structure

```
struct {
        uint8   socketDescriptor[2];
} queryLTCPFrameSnd;
```

Response Payload

```
struct {
        uint8   socketDescriptor[2];
        uint8 destIP[4];
        uint8 destPort[2];
} rsi_LTCPConnStatusFrameRcv;
```

Response Parameters

socketDescriptor (2 bytes, hex)– Socket handle for an already open listening TCP socket in the module.

destIP (4 bytes, hex)- Destination IP of the remote peer whose socket is connected

destPort (2 byte, hex)- Port number of the socket at the remote peer

Possible error codes

Possible error codes are 33, 37 and 44

Relevance

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

7.4.18 Query WLAN Connection Status

Description

This command queries the WLAN connection status of the Wi-Fi module after getting associated to an access point.

Payload Structure

No Payload required.

Parameters

No parameters

Response Payload

```
typedef struct {
        uint8 state[2];
} rsi_conStatusFrameRcv;
```

Response Parameters

state: 1-Connected to AP
0-Not connected to AP

Possible error codes

Possible error codes are 33, 37 and 44

Relevance

This command is relevant when the module is configured in Operating Mode 0, 2, 3 and 5.

7.4.19 Load Web Page in Module

Description

The module has an embedded Web Server and can respond to HTTP Get and Post requests from a remote terminal. This command is used to load a user defined web page on the module. The module’s web server supports the following:

1. HTTP 1.0 standard
2. Static and dynamic pages

Payload Structure

```
struct {
        uint8  total_len[2];
        uint8 current_len[2];
        uint8 more_chunks;
        uint8 webpage[MAX_WEBPAGE_SEND_SIZE];
}webpageSnd_t;
```

Parameters

total_len: The total length of the characters in the source code of the webpage. The maximum value of this parameter is 3 KB.

current_len – – Total number of characters in the current segment.

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 .

Segments are created when the overall length of the source code of the web page is more than 1024 characters.

Example 1: Below is given the source code of a reference page that can be sent to load the corresponding web page, 99 (*total_len*) characters in all including newline character.

```
<html>
<head>
<title>Untitled Document</title>
</head>
<body>
<h1>Hello World</h1>
</body>
</html>
```

Example 2: If the web page source code is of 3000 characters, the Host can send it through 3 segments in 3 commands, the first two of 1024 bytes, and the last one of 952 bytes, with *more_chunks*=1 in the first two commands and 0 in the last command .

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 33, 37 and 44

Relevance

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

> NOTE: A remote terminal can access the webpage stored in the module by typing http://xxx.xxx.xxx.xxx (this is the IP address assigned to the module) in its browser. The web page can be accessed in any of the operating modes.

The webserver in the module can be bypassed. The user can implement a web server at the Host and communicate with the module through port number 80. User has to set the corresponding feature bit in feature select command.

7.4.19.1 Web Server Functionality with Multiple pages

The section Load Web Page in Module describes how to load a single web page into the module. There might be use cases in which more than a single page may be required to be serviced. In such a case, these pages can be stored in the Host memory and can be sent to the module one at a time. The below process shows the mechanism.

1. Remote terminal (Laptop) connects to the module.
2. Remote terminal queries for a particular web page by typing the URL in its browser
3. Module receives the query and checks if it already has the page in its memory. If yes, it sends out the page to the remote terminal and the query is serviced. If the page is currently not in the module's memory, it sends out an interrupt to the Host. The Host should execute an Rx Operation, the way it does to responses to normal commands. The read would have an ID of 0x64 (refer table Response IDs for Rx Operation) and a payload as shown below

   ```
   typedef struct
   {
      uint8 url_len[2];
      uint8 url_name[MAX_URL_SIZE];
   }
   ```

   url_length- This is the number of characters in the requested URL.

   url_name (ASCII)- This is the actual url name. The maximum size of the URL allowed is 256 characters.

4. The Host , after reading the information from the module, should fetch the page from its memory and give it back to the module with the command ID 0x56 (refer table Response IDs for Rx Operation) and payload as shown

   ```
   typedef struct
   {
      uint8 total_len[2];
      uint8 current_len[2];
      uint8 more_chunks;
      uint8 webpage[MAX_WEBPAGE_SEND_SIZE];
   }WebpageSnd_t;
   ```

   MAX_WEBPAGE_SEND_SIZE is 1024 bytes.

   total_len- This is the total number of characters in the page

   current_len- Total number of characters in the current segment. If the queried web page is not present in the Host, it should send '0' for this parameter.

   more_chunks(1 byte)-

‘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

Segments are created when the overall length is more than 1024 characters. For example, if the web page code is of 3000 characters, the Host should send it through 3 segments, the first two of 1024 bytes, and the last one of 952 bytes, in a set of three commands

5. After all the segments are sent, the module aggregates them, stores in the internal memory and dispatches the page to the remote terminal.

   Note that only the page for the requested URL should be supplied by the Host to the module. The maximum allowed size of such a page is 3 KB.

7.4.20 Load Web Fields in Module

Description

The command provides an incremental way for the Host to update data in designated fields of an already loaded webpage.

Payload Structure

```
#define MAX_NO_OF_FIELDS  10
 typedef struct field_st{
                uint8 field_index;
        uint8 field_val[64];
 }field_st;
 typedef union {
        struct {
                uint8 field_cnt;
                struct field_st field_st[MAX_NO_OF_FIELDS];
        }webFieldsFrameSnd;
        uint8   uWebFieldBuf[680];
 }rsi_uWebFields;
```

Parameters

field_index: Index of the individual field. For example, the code in the below example, the index of param1 is 1.

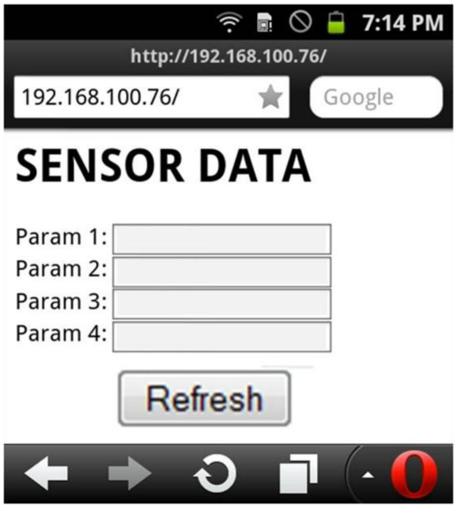
Field_val: Actual value in the field. This is a character array.

Field_count: Number of fields the user wants to update. In the example below, there are 4 parameters. If the user wants to update 3 of those, the field_cnt is 3.

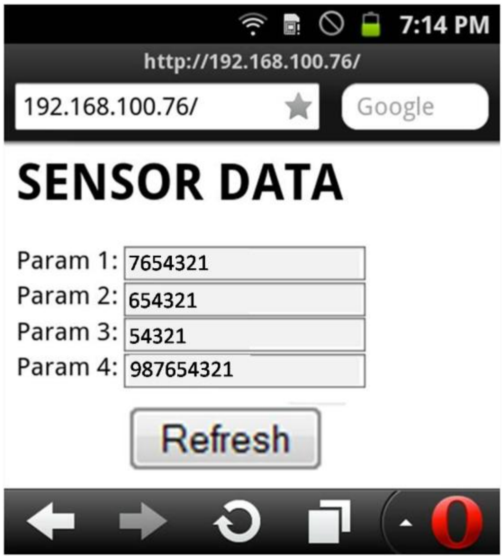
Below is the source code of a page with configurable fields.

```
<html><body><script type=”text/javascript”>function
reloadPage(){window.location.reload()}</script></head><body><C><h1>
<B> SENSOR DATA</b></h1><form >Param 1: <input type=”text”
name=”param1” value=”%#1--#%” /><br />Param 2: <input type=”text”
name=”param2” value=”%#2-#%” /><br/>Param 3: <input type=”text”
name=”param3” value=”%#3#%” /><br />Param 4: <input type=”text”
name=”param4” value=”%#4----#%” /><br /><input type=”button”
value=”Refresh” onclick=”reloadPage()” /><C></body></html>
```

This webpage can be first loaded with the command “Load Web Page”. When the page is opened in a remote terminal, it would show the below



New values for the variables Param1 to Param4 can be loaded from the Host. For example, if field_val for param1, param2, param3 and param4 is provided as 7654321,654321,54321,987654321 respectively, and the user refreshes the page in the remote terminal, the following will appear:



SENSOR DATA

Param 1: 7654321
Param 2: 654321
Param 3: 54321
Param 4: 987654321

Notes:

1. The identifier for the parameters (highlighted in green in the source code of the web page above) should range from "%#1… #%" to "%#10… #%". Other characters are not allowed. The module parses for these identifiers, they should not be present in any other part of the HTML code.
2. The length of the field is determined from the first '%' to the last '%'. For example, for the 4th parameter to be 7 characters long, %#4--#%. Similarly, for the parameter to be 15 characters long, %#4----------#% should be used. A maximum of 10 configurable parameters are allowed, the maximum length of each parameter is 64 characters.
3. To update the values of the parameters, a new value only with the designed length should be sent. For example, if the fourth parameter was configured as 7 characters by putting name="param4" value="%#4--#%" then only 7 characters should be sent to update it.

Possible error codes

Possible error codes are 33, 37, 44

Relevance

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

7.4.21 Query Firmware Version

Description

This command is used to query the version of the firmware loaded in the module.

Payload Structure

No payload required for this command.

Response Payload

```
struct {
        uint8                     firmwareVersion[20];
}queryFirmVersionResp;
```

Response Parameters

firmwareVersion: Firmware version. Each byte is separated by a dot and there is a comma after the first 3 bytes in the following format < OKMajor11.Minor12.Minor13,Major21.Minor22.Minor23>.

Relevance

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 3, 4, 5, 6 and 7.

7.4.22 Query MAC Address

Description

This command is used to query the MAC address of the module. The Host may query the MAC address of the module at any time after the band and init commands.

Payload Structure

No payload required for this command

Response Payload

```
struct {
        uint8                     macAddr[6];
}queryFirmVersionResp;
```

Response Parameters

macAddr: MAC address

Relevance

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 3, 4, 5, 6 and 7.

7.4.23 Send data

Operating Mode 0, 1 and 2

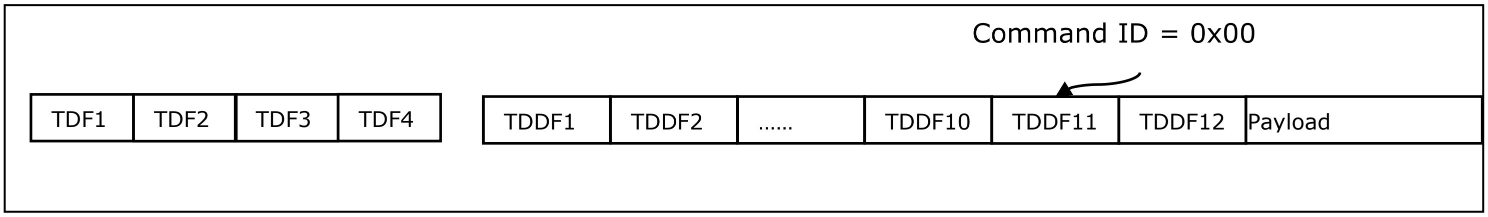
Description

This command sends data from the host to the module, to be transmitted over a wireless media.

Payload Structure

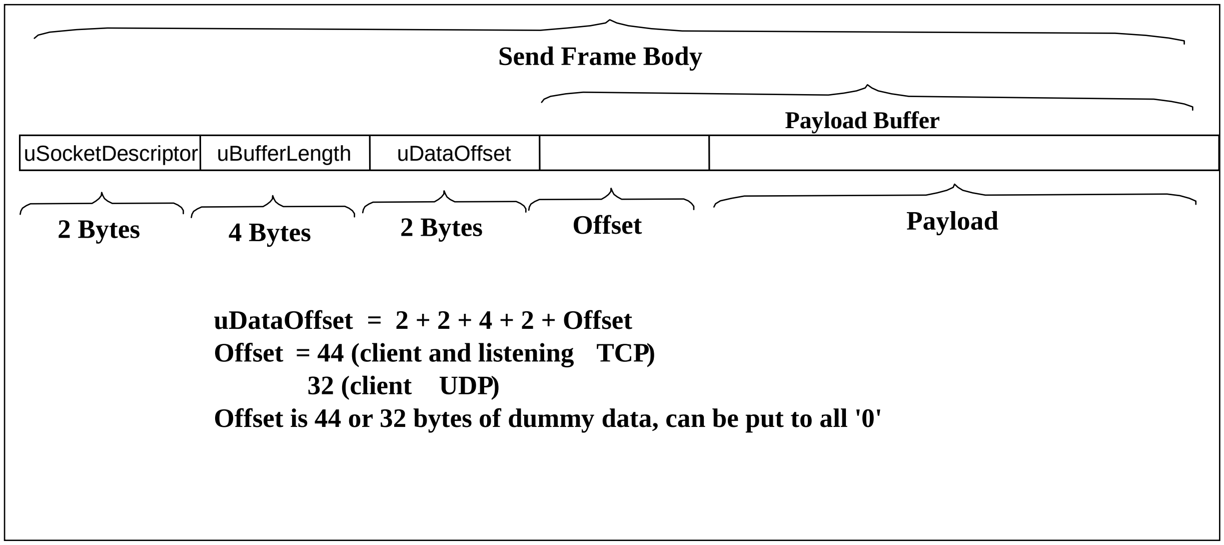
```
struct {
uint8           socketDescriptor[2];
uint8           SendBufLen[4];
uint8           SendDataOffsetSize[2];
uint8           PayloadBuff[PROTOCOL_OFFSET + RSI_MAX_PAYLOAD_SIZE +
                HEAD_ROOM];
} sendFrameSnd;
```

Note: HEAD ROOM is not mandatory. It is optional.

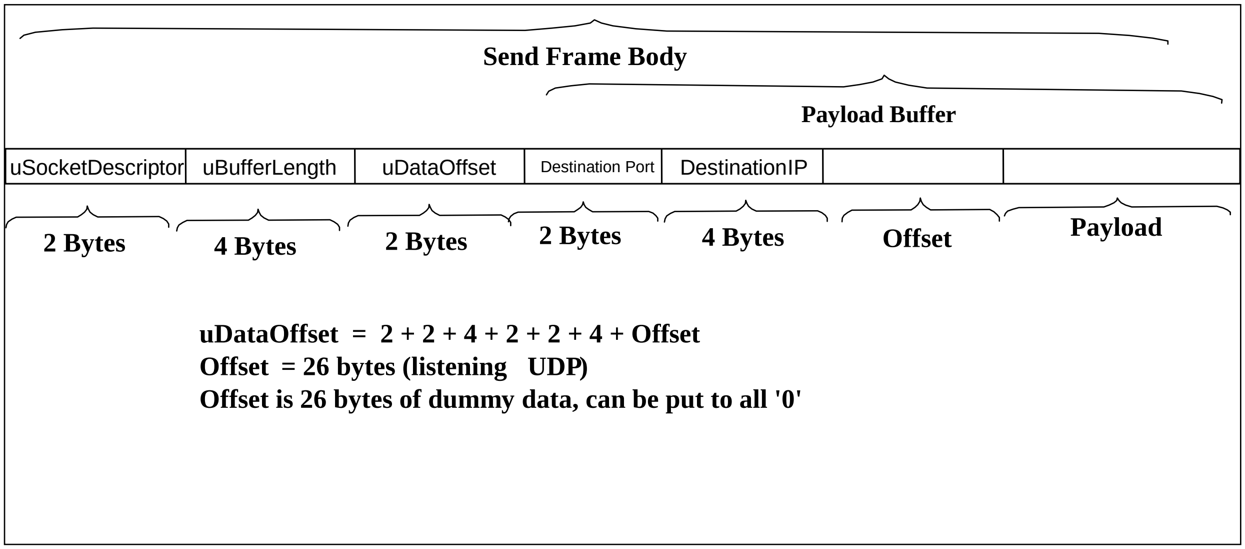


Parameters

Structure Member Name	**Description**
socketDescriptor	Descriptor of the socket over which data is to be sent
SendBufLen	Length of the data to be sent
SendDataOffsetSize	Offset of the data in the buffer. To avoid redundant "memcpy". The format of the send frame body is shown in the figure below.
PayloadBuff	Buffer comprising the data payload. The payload starts from the offset as shown in the figure below.


Send Frame Body
Payload Buffer
uSocketDescriptor
uBufferLength
uDataOffset
2 Bytes
4 Bytes
2 Bytes
Offset
Payload
uDataOffset = 2 + 2 + 4 + 2 + Offset
Offset = 44 (client and listening TCP)
32 (client UDP)
Offset is 44 or 32 bytes of dummy data, can be put to all '0'

Send Frame Body (TCP and client UDP)



Send Frame Body (Listening UDP)

The Host driver should take care of buffering the data, thereby isolating the application from the packet size limitation with respect to the module/firmware.

For example, for 3000 bytes of the data the flow will be as follows:

There has to be a wrapper in the driver that buffers the data from the application layer. The maximum size of the buffer that the modules can handle is 1400 bytes. Thus the wrapper takes care of sending multiple frames to the module.

For the 3000 bytes of the data the wrapper has to send 3 frames to the module. The uDataoffset (refer to the figure above) shall depend on the socket type.

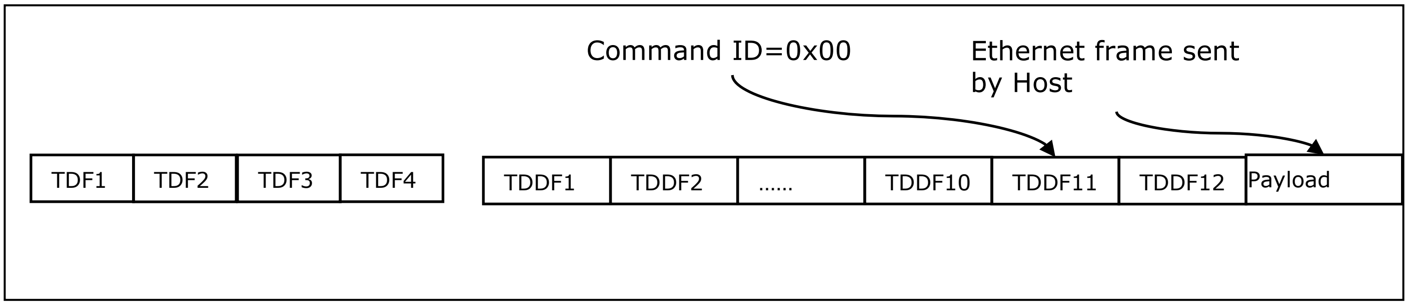
Frame1: SendBufLen = 1400 bytes

Frame2: SendBufLen = 1400 bytes

Frame3: SendBufLen = 200 bytes

Operating Mode 3, 4 and 5

When the TCP/IP stack is bypassed, the Host uses its own TCP/IP stack and sends Ethernet frames to the module. The maximum size of the payload is 1400 bytes. The user needs to use rsi_send_raw_data.c API to send the data in TCP/IP bypass mode.



Response Payload

There is no response payload/response interrupt for this command. For other commands, the module sends an interrupt after receiving the command, to indicate that the response is ready to be read. In case of this command, no interrupt is sent from the module and the Host need not read any response.

Possible error codes

Possible error codes are 33, 37 and 44.

7.4.24Receive data

Operating Mode 0, 1, 2 and 6

Description

This process is used to receive data by the Host from the module. When the module receives data from a remote client, it raises an asynchronous interrupt to indicate to the Host that new data has arrived and needs to be read. The Rx Operation in figure Rx Operation should be executed by the Host after receiving the interrupt.

Response Payload

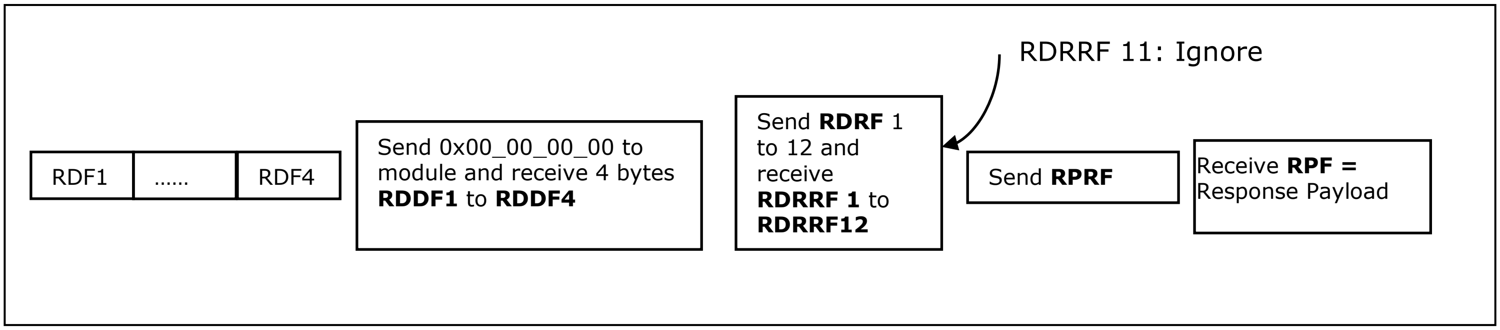
For TCP data:

```
typedef struct {
   uint8              recvSocket[2];
   uint8              recvBufLen[4];
```

```
    uint8                recvDataOffsetSize[2];
    uint8                fromPortNum[2];
    uint8                fromIpaddr[4];
    uint8                reserved[40];
    uint8                recvDataBuf[1400];
   }rsi_recvFrameTcp;
```

For UDP data:

```
typedef struct {
    uint8                recvSocket[2];
    uint8                recvBufLen[4];
    uint8                recvDataOffsetSize[2];
    uint8                fromPortNum[2];
    uint8                fromIpaddr[4];
    uint8                reserved[28];
    uint8                recvDataBuf[1400];
   } rsi_recvFrameUdp;
```


RDRRF 11: Ignore
RDF1
......
RDF4
Send 0x00_00_00_00 to module and receive 4 bytes **RDDF1** to **RDDF4**
Send **RDRF** 1 to 12 and receive **RDRRF 1** to **RDRRF12**
Send **RPRF**
Receive **RPF =** Response Payload

Response Parameters

recvSocket: Handle of the socket in which data is received

recvBufLen: The size of the data to be received

recvDataOffsetSize: This is the offset in the received payload, after which actual data begins.

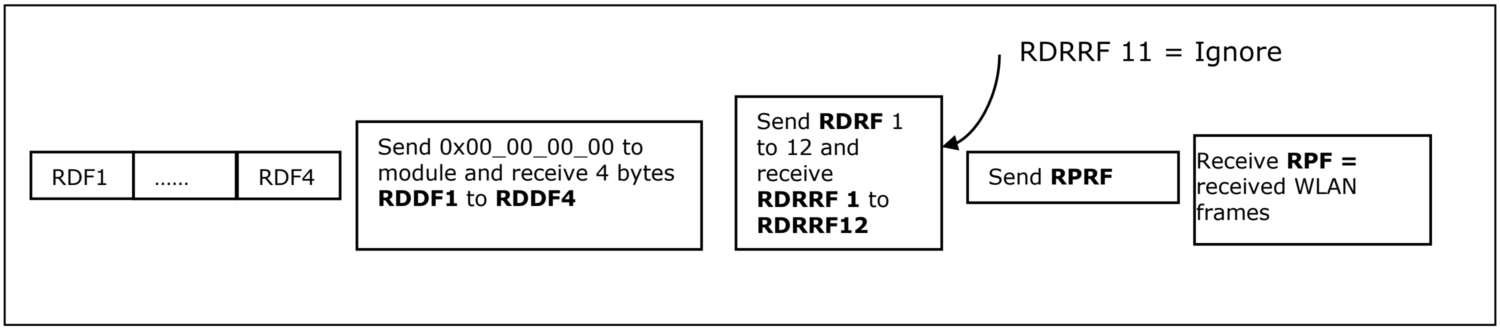
fromPortNum: Port number of the source port

fromIpaddr: The IP address of the source terminal

recvDataBuf: Actual data sent from remote terminal

Operating Mode 3, 4, 5 and 7

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. The Rx Operation in figure Rx Operation should be executed by the Host after receiving the interrupt.


RDRRF 11 = Ignore
RDF1
......
RDF4
Send 0x00_00_00_00 to module and receive 4 bytes RDDF1 to RDDF4
Send RDRF 1 to 12 and receive RDRRF 1 to RDRRF12
Send RPRF
Receive RPF = received WLAN frames

7.4.25 Remote Socket Closure

Description

If after a TCP connection is established between the module and the remote terminal, the remote socket is closed, the module sends an interrupt to the Host indicating the event and closes the corresponding socket in the module. The Host should do an Rx operation after getting the interrupt. The Response ID is **Async Socket Remote Terminate** (table Response IDs for Rx Operation)

Response Payload

```
struct {
        uint16 socketDescriptor;
}
```

Response Parameter

socketDescriptor: Socket descriptor of the socket that was closed in the module.

Relevance

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

NOTE:

1. When TCP keep alive time out happens, socket will be closed and remote close indication will be given to host along with "keep alive timed out" status.

2. When module operating in AP mode, If a station is disconnected or removed from AP, all the sockets connected to that particular station will be closed and asynchronous remote socket close indications will be given to host along with "keep alive timed out" status.

7.4.26 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 interrupt to the Host to indicate that its server socket is now connected to a client socket. The Host should do an Rx operation after getting the interrupt. The Response ID is **Async TCP Socket Connection Established** (table Response IDs for Rx Operation)

Response Payload

```
struct {
    uint8               socket[2];
    uint8               fromPortNum[2];
    uint8               fromIpaddr[4];
} rsi_recvLtcpEst;
```

Response Parameter

socket: Socket descriptor of the server TCP socket.

fromPortNum[2]: Port number of the remote socket

fromIpaddr[4]: Remote IP address

Possible error codes

Possible error codes are 33, 37, 44, -192.

Relevance

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

7.4.27 Set Sleep Timer

Description

This command configures the timer for power save operation.

Payload Structure

```
 struct {
             uint16 sleepTimer;

 } sleepTimerFrameSnd;
```

Parameters

sleepTimer: Value of the timer in seconds. Maximum value is 65000 (decimal). Default value is 1 second.

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 33, 37 and 44

Relevance

This command is relevant when the module is configured in Operating Mode 0, 2, 3 and 5.

7.4.28 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 *Init* command.

Payload Structure

```
struct {
            uint8 powerVal;

} powerFrameSnd;
```

Parameters

powerVal:

0–Mode 0: Disable power save mode

1–Power save Mode 1

2–Power save Mode 2

To set any power save mode in the module, the below steps should be executed.

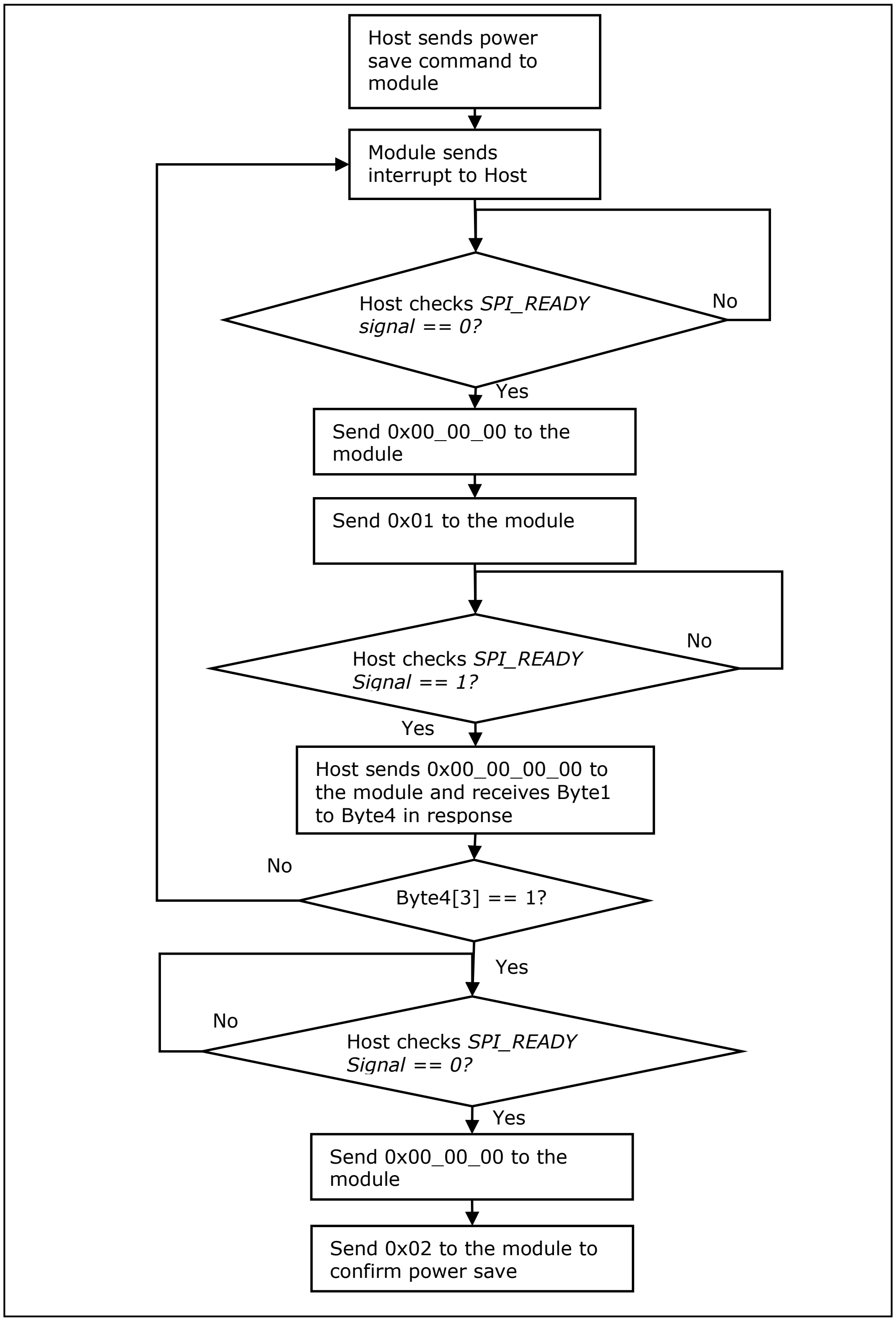

Host sends power save command to module
Module sends interrupt to Host
Host checks *SPI_READY signal == 0?*
No
Yes
Send 0x00_00_00 to the module
Send 0x01 to the module
Host checks *SPI_READY Signal == 1?*
No
Yes
Host sends 0x00_00_00_00 to the module and receives Byte1 to Byte4 in response
No
Byte4[3] == 1?
Yes
No
Host checks *SPI_READY Signal == 0?*
Yes
Send 0x00_00_00 to the module
Send 0x02 to the module to confirm power save

Figure 24: Setting Power Save Modes

If power mode 1 is set, as there is no way to wakeup module from host, sleep timer should be configured before giving power save command.

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 33, 37 and 44

Relevance

This command is relevant when the module is configured in Operating Mode 0, 2, 3 and 5.

7.4.28.1 Power save Operation

The behavior of the module differs according to the power save mode it is put in.

7.4.28.1.1 Power save Mode 1

Once the module is put to power save mode 1, it wakes itself up whenever the Sleep Timer expires. After waking up, the module sends an interrupt to the Host. If the Host has any data to transmit to the module, it should execute corresponding Tx and Rx Operations. Once that is done, the module can be put back to sleep by giving a sleep confirm.

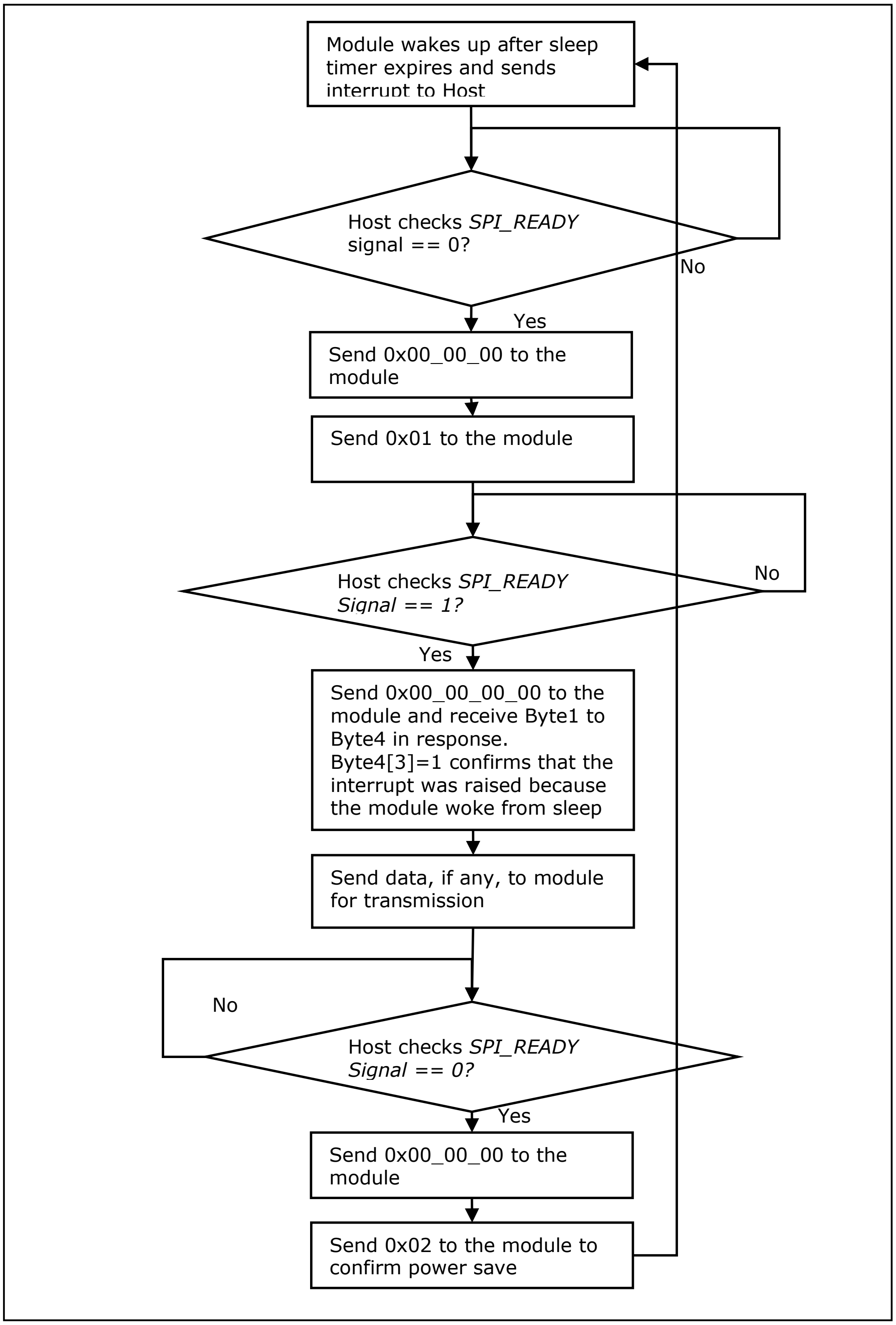

Module wakes up after sleep timer expires and sends interrupt to Host
Host checks *SPI_READY* signal == 0?
No
Yes
Send 0x00_00_00 to the module
Send 0x01 to the module
Host checks *SPI_READY* *Signal == 1?*
No
Yes
Send 0x00_00_00_00 to the module and receive Byte1 to Byte4 in response. Byte4[3]=1 confirms that the interrupt was raised because the module woke from sleep
Send data, if any, to module for transmission
No
Host checks *SPI_READY* *Signal == 0?*
Yes
Send 0x00_00_00 to the module
Send 0x02 to the module to confirm power save

Figure 25: Power Save Operation

The module can be put to Power Save mode any time after the *Init* command. After having put the module to power save mode, the Host can issue subsequent commands only after the module has indicated to the Host that it has woken up. The module can however always receive data from the remote terminal at any point of time and can send an interrupt to the Host indicating the same.

7.4.28.1.2 Power save Mode 2

Once the module is put to power save mode 2, it can be woken up by the Host. The Host should send a pulse into the WAKEUP Pin of the module (Pin # 58 in RS-WC-201 and Pin # 64 in RS-WC-301). The default value in this pin is 0, while a high pulse of minimum width 100 us should be sent to wake up the module. The Host should give commands to operate the module only when it is in awake state. The module can however always receive data from the remote terminal at any point of time and can send an interrupt to the Host indicating the same.

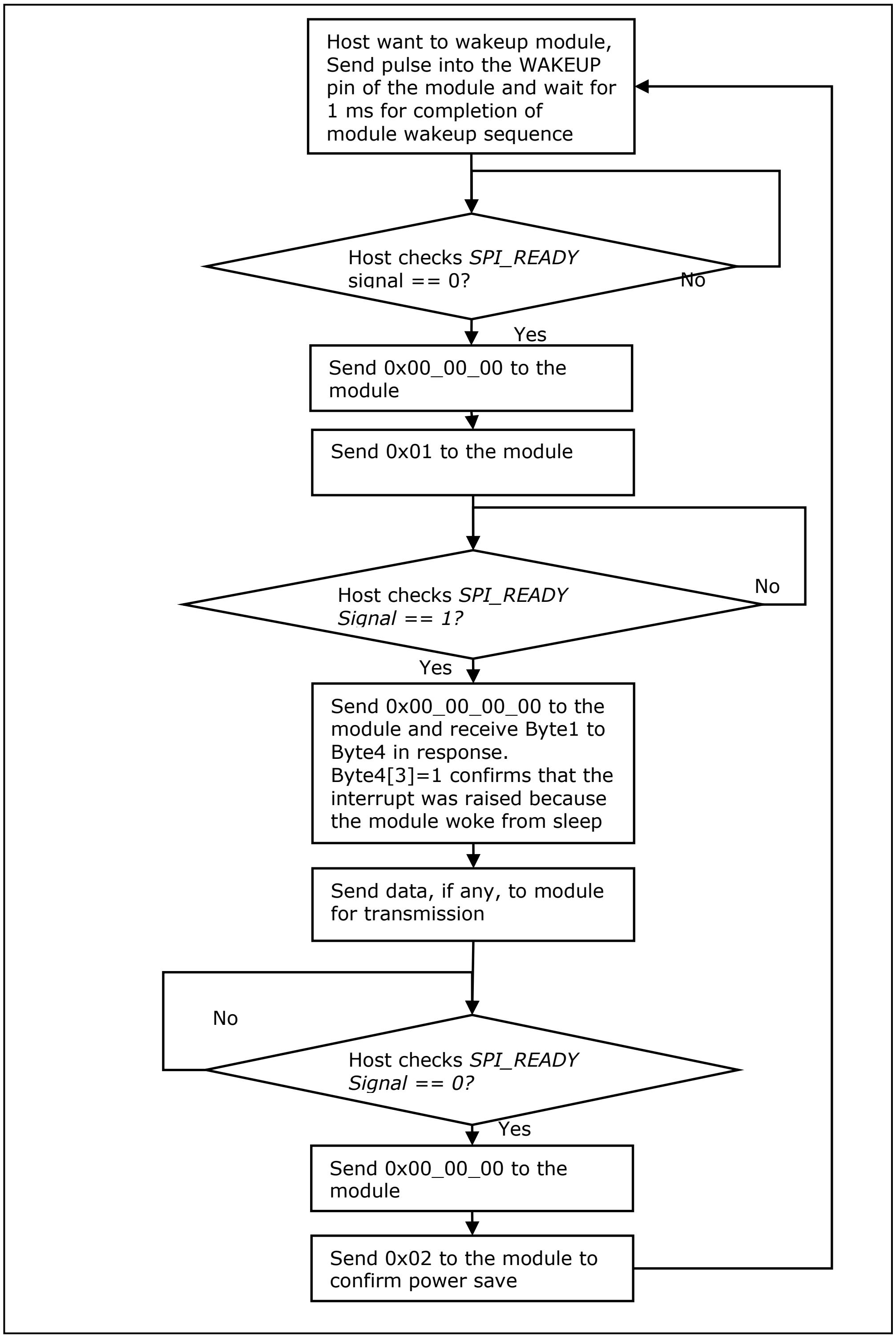

Host want to wakeup module, Send pulse into the WAKEUP pin of the module and wait for 1 ms for completion of module wakeup sequence
Host checks *SPI_READY* signal == 0?
No
Yes
Send 0x00_00_00 to the module
Send 0x01 to the module
Host checks *SPI_READY* *Signal == 1?*
No
Yes
Send 0x00_00_00_00 to the module and receive Byte1 to Byte4 in response. Byte4[3]=1 confirms that the interrupt was raised because the module woke from sleep
Send data, if any, to module for transmission
No
Host checks *SPI_READY* *Signal == 0?*
Yes
Send 0x00_00_00 to the module
Send 0x02 to the module to confirm power save

Figure 26: Power Save Mode 2

Note:

WiSeConnect doesn’t support power save modes while operating in AP or group owner mode.

After issuing PS Continue, if an interrupt comes from module to host, that interrupt should be served (read packet/ status). After reading the packet, host should issue a PS continue again to put back module into sleep.

7.4.29 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.

Payload Structure

```
struct disassoc_s
{
   uint8  mode_flag[2];
   uint8  client_mac_addr;
};
```

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 Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 19, 33, 37 and 44

Relevance

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 3, 4, 5, 6 and 7.

Note:

During rejoin, if user wants to connect to another AP, diassoc has to be issued although an error '6' will be given and after handling this error user has to start from init command.

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.

When the module is operating in AP mode, station may get disconnected from AP because of some other reasons other than disconnect (Station is idle and so). No asynchronous response will be given to host upon a station removal. Host has to issue "GO params" command to know the connected stations information.

7.4.30 Query RSSI Value

Description

This command is used to retrieve the RSSI value for Access Point to which the module is connected.

Payload Structure

No Payload required.

Response Payload

```
struct{
        uint8 rssiVal[2];
}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 are 33, 37 and 44

Relevance

This command is relevant when the module is configured in Operating Mode 0, 2, 3 and 5.

Note: The RSSI values ranges from -100 dB to -15 dB. Closer the RSSI value to '0', stronger the signal strength.

7.4.31Query SNR value

Description

This command is used to get the signal to noise ratio of the signal received from the Access Point that the module is connected to. Closer the AP, higher is the SNR

Payload Structure

No payload required

Response Payload

```
struct{
        uint8 snrVal;
}rsi_snrFrameRcv;
```

Parameters

snrVal: SNR value (in dBm)

Possible error codes

Possible error codes are 33, 37 and 44

Relevance

This command is relevant when the module is configured in Operating Mode 0,2, 3 and 5.

> Note: The SNR values ranges from 5 dB to 80 dB. Greater the SNR value, stronger the signal strength.

7.4.32 Query Network Parameters

Description

This command is used to retrieve the WLAN connection and IP parameters. This command should be sent only after the connection to an Access Point is successful.

Payload Structure

No payload required for this command.

Response Payload

The structure for the Network Parameters query frame's response is as follows:

```
struct sock_info_query_t
{
 uint16  sock_id;
 uint16  socket_type;
 uint16   sPort;
 uint16  dPort;
 uint8   dIp[4];
}sock_info_query_t;
struct  EVT_NET_PARAMS
{
        uint8  wlan_state;
        uint8  Chn_no;
        uint8  psk[64];
        uint8  mac_addr[6];
        uint8  ssid[34];
        uint16 connection_type ;
        uint8  sec_type ;
        uint8  dhcp_mode ;
        uint8  ipaddr[4];
        uint8  subnet_mask[4];
        uint8  gateway[4];
        uint16  num_open_socks;
        sock_info_query_t socket_info[10];
} EVT_NET_PARAMS;
```

Response Parameters

wlan_state: This indicates whether the module is connected to an Access Point or not.

0–Not Connected

1–Connected

Chn_no: Channel number of the AP to which the module joined or the channel number in which the AP is created when the module is operating in AP mode.

Psk: Pre-shared key used

Mac_Addr: MAC address of the module

Sec_type: Security mode of the AP.

0– Open mode

1– WPA security

2– WPA2 security

3- WEP

4– WPA-Enterprise

5– WPA2-Enterprise

SSID: This value is the SSID of the Access Point to which the module is connected

Ipaddr: This is the IP Address of the module. If the Set IP Params command was not sent to the module before Query Network Parameters command, the module returns a default value which should be ignored

Subnet_mask: This is the Subnet Mask of the module. If the Set IP Params command was not sent to the module before Query Network Parameters command, the module returns a default value which should be ignored

Gateway: This is the Gateway Address of the module. If the Set IP Params command was not sent to the module before Query Network Parameters command, the module returns a default value which should be ignored

DHCP_mode: This value indicates whether the module is configured for DHCP or Manual IP configuration.

0– Manual IP configuration
1– DHCP

Connection_type: This value indicates whether the module is operational in Infrastructure mode.

1– Infrastructure
3–AP mode

Num_open_socket: This value indicates the number of sockets currently open

Sock_id: Socket handle of an existing socket

Socket_type: Type of socket

0–TCP Client

1-UDP Client

2– TCP Server (Listening TCP)

4– Listening UDP

Sport: Port number of the socket in the module.

Dport: Destination port number, of the socket in the remote terminal. The least significant byte is returned first

Dip: IP of the remote terminal

Possible error codes

Possible error codes are 33, 37 and 44

Relevance

This command is relevant when the module is configured in Operating Mode 0, 2, 3 and 5.

7.4.33Query 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 is issued 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.

Payload Structure

There is no payload for this command.

Response Payload

```
#define MAX_STA_SUPPORT 4

struct  go_sta_info_s
{
   uint8 mac[6];
   uint8 ip_addr[4];
};

typedef struct {
   uint8   ssid[34];
   uint8   bssid[6];
   uint16  channel_number;
   uint8   psk[64];
   uint8   ip[4];
   uint16  sta_count;
   struct  go_sta_info_s sta_info[MAX_STA_SUPPORT];
}rsi_qryGOParamsFrameRcv;
```

Response Parameters

SSID: SSID of the Group Owner. If the SSID is less than 34 characters, then filler bytes 0x00 are added to make the length 34 bytes. The last two bytes of this parameter should be ignored.

BSSID: MAC address of the module

Channel_number: Channel number of the group owner

PSK: PSK that was supplied in the command "Configure Wi-Fi Direct Peer-to-Peer mode"*.* If the PSK is less than 64 bytes, filler bytes of 0x00 are added to make the parameter 64 bytes. The last byte of this parameter should be ignored. Third party clients should use this PSK while associating to the Group Owner (the Group Owner appears as an Access Point to third party clients).

IP: IP Address of the module.

Sta_count: Number of clients associated to the Group Owner. The least significant byte is returned first. A maximum of 4 clients is supported.

MAC: MAC address of the connected client

IP_addr: IP address of the connected client

Possible error codes

Possible error codes are 33, 34, 37 and 44

Relevance

This command is relevant when the module is configured in Operating Mode 1, 4, 6 and 7.

7.4.34DNS Server

Description

This command is used to provide to the module the DNS server's IP address. This command should be issued before the "DNS Resolution" command and after the "Set IP Parameter" command.

Payload Structure

```
struct TCP_CNFG_Configure

{

     UINT8 mode;

     UINT8 primary_dns_server[4];

     UINT8 secondary_dns_server[4];

};
```

Parameters

mode:

1-The module can obtain a 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_server: This is the IP address of the Primary DNS server to which the DNS Resolution query is sent. Should be set to '0' if *mode* =1.

Secondary_DNS_server: This is the IP address of the Secondary DNS server to which the DNS Resolution query is sent. If *mode* =1 or if the user does not want to specify a secondary DNS Server IP address, this parameter should be set to '0'

Response Payload

```
typedef struct{
```

```
            uint8     primary_dns_ip[4];
            uint8      secondary_dns_ip[4];
    }rsi_dnsserverResponse ;
```

Response Parameters

primary_dns_ip: IP address of the primary DNS server
secondary_dns_ip: IP address of the secondary DNS server

Possible error codes

Possible error codes are 33, 37 and 44

Relevance

This command is relevant in Operating Modes 0 and 2.

7.4.35 DNS Resolution

Description

This command is issued by the Host to obtain the IP address of the specified domain name.

Payload Structure

```
#define MAX_NAME_LEN 90

typedef struct TCP_CNFG_DNS_Req
{
     uint8    aDomainName [MAX_NAME_LEN];
     uint16   uDNSServerNumber;
}__attribute__ ((packed)) TCP_CNFG_DNS_Req;
```

Parameters

aDomainName: This is the domain name of the target website. A maximum of 90 characters is allowed.
uDNSServerNumber: Used to indicate the DNS server to resolve the Query.

1-Primary DNS server

2-Secondary DNS server

Response Payload

```
#define MAX_DNS_REPLY 10

typedef struct TCP_EVT_DNS_Resp
{
    uint16  uIPCount;
    uint8   aIPaddr[MAX_DNS_REPLY][4];
```

}__attribute__((packed))TCP_EVT_DNS_Resp;

Response Parameters

uIPCount: Number of IP addresses resolved
aIPaddr: Individual IP addresses, up to a maximum of 10

Possible error codes

Possible error codes are 33, 37, 44 and -190

Relevance

This command is relevant in Operating Modes 0 and 2.

7.4.36 HTTP GET

Description

This command is used to transmit an 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. The Host connected to the module acts as a HTTP client when this command is used

Payload Structure

```
struct TCP_HTTP_Req
{
UINT16 ipaddr_len;
UINT16 url_len;
UINT16 header_len;
UINT16 reserved;
UINT8  buffer[1200];
}__attribute__((packed));
```

Parameters

ipaddr_len – The length of the IP Address (including the digits and dots). For example, if the IP address of www.website.com is 192.168.40.86, *ipaddr_len* = 13

url_len – The length of the URL. For example, if www.website.com/index.html is the webpage, then *url_len* = 11 for "/index.html", www.website.com is not included as it is specified in the IP address

header_len – The length of the header of the HTTP GET request

reserved – Set this value to 0

Buffer – Buffer contains actual values in the order of <IP Address>,<URL>, <Header> and <Data>. *Data* is the actual data involved in the HTTP GET request. The parameter *Buffer* is a character buffer of size 1200 bytes.

For example,

IP = 192.168.40.50

URL=/index.html

Header= User-Agent: HTMLGET 1.00\r\n\r\n

Data=<data>

Can be the valid contents of the buffer

Response Payload

After the module sends out the HTTP GET request to the remote server, it may take some time for the server response to come back. An interrupt is raised by the module to indicate to the Host that the response from the remote server has been received. The Host should perform an Rx Operation. On performing the Rx operation, it receives the code for **HTTP GET** (table Response IDs for Rx Operation) and the below structure as the *Payload*

```
struct TCP_EVT_HTTP_Data_t
{
UINT32 more;
UINT32 offset;
UINT32 data_len;
UINT8 *data;
}__attribute__((packed));
```

more: This indicates whether more HTTP data for the HTTP GET request is pending.

0– More data pending. Further interrupts may be raised by the module till all the data is transferred to the Host.
1– End of HTTP data

offset: This indicates the offset of the valid HTTP data that is present in the response

data_len: This indicates the length of the data in HTTP response

data: Actual data in the HTTP GET response

Possible error codes

Possible error codes are 33, 37 and 44.

Relevance

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

7.4.37 HTTP POST

Description

This command is used to transmit an 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. The Host connected to the module acts as a HTTP client when this command is used.

Payload Structure

```
struct TCP_HTTP_Req
{
UINT16 ipaddr_len;
UINT16 url_len;
UINT16 header_len;
UINT16 data_len;
UINT8  buffer[1200];
}__attribute__((packed));
```

Parameters

ipaddr_len – The length of the IP Address (including the digits and dots). For example, if the IP address of www.website.com is 192.168.40.86, *ipaddr_len* = 13

url_len – The length of the URL. For example, if www.website.com/index.html is the webpage, then *url_len* = 11 for "/index.html", www.website.com is not included as it is specified in the IP address

header_len – The length of the header of the HTTP POST request

Data_len – This is the length of the data field in the *Buffer* parameter

Buffer – Buffer contains actual values in the order of <IP Address>,<URL>, <Header><data>. *Data* is the actual data to be posted to the server. The parameter *Buffer* is a character buffer.

For example,

IP = 192.168.40.50

URL=/index.html

Header= User-Agent: HTMLGET 1.00\r\n\r\n

Data=<data>

Response Payload

After the module sends out the HTTP POST request to the remote server, it may take some time for the response to come back. An interrupt is raised by the module to indicate to the Host that the response from the remote server has been received. The Host should perform an Rx Operation. On performing the Rx operation, it receives the code for **HTTP POST** (table Response IDs for Rx Operation) for the Response ID and the below structure as the *Payload*

```
struct TCP_EVT_HTTP_Data_t
{
UINT32 more;
UINT32 offset;
UINT32 data_len;
UINT8 *data;
}__attribute__((packed));
```

more: This indicates whether more HTTP data for the HTTP GET request is pending.

0– More data pending. Further interrupts may be raised by the module till all the data is transferred to the Host.
1– End of HTTP data

offset: This indicates the offset of the valid HTTP data that is present in the response

data_len: This indicates the length of the data in HTTP response

data: Actual data in the HTTP GET response

Possible error codes

Possible error codes are 33, 37 and 44.

Relevance

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

Note: There is no timeout programmed for HTTP Get/HTTP Post requests. The module will wait for infinite time for the response to come.

7.4.38 Soft Reset

Description

This command gives 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 from the first command "Set Operating Mode" after issuing this command.

Payload Structure

There is no payload for this command.

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 33, 37 and 44

Relevance

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 3, 4, 5, 6 and 7.

After the module is reset, it sends an interrupt to the Host. The Host should execute the Card Ready Operation and start giving commands from the beginning as if a fresh power-up has happened.

7.4.39 Configure GPIOs

Description

This command configures the GPIO pins that are coming out of WiSeConnect™ module as output pins.

Payload Structure

```
struct rsi_gpio_conf_s
{
     uint8 pin_no;
     uint8 pin_direction;
     uint8 pin_val;
     uint8 reserved;
}rsi_gpio_conf_t;
```

Parameters

pin_no – pin number[1] to be configured, either '1' or '2'

pin_direction – it should be set to '1'.

pin_val – value on the pin, '0' for making it low and '1' for making it high.

Response Payload

There is no response payload for this command.

Possible error codes

Possible error code is 38.

Relevance

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 3, 4, 5, 6 and 7.

[1] In WiSeConnect™ 201/301 module, user can configure two pins with this command. For 201, these 1 and 2 are pin-25 and pin-26 respectively. For 301, these 1 and 2 are pin-30 and pin-31 respectively.

7.4.40 Ping from module

Description

This command sends the ping request from WiSeConnect module to target IP address.

Payload Structure

```
typedef struct rsi_ping_request_s{
     uint8 ping_IP[4];
     uint8 data_size[2];
}rsi_ping_request_t;
```

Parameters

`ping_IP` – Target IP address to ping

`data_size` – Ping size[1]

Response Payload

```
typedef struct {
     uint8 rsp_len[2];
     uint8 ping_ip[4];
}rsi_pingRspRcv;
```

Possible error codes

Possible error codes are 37, 46, 47 and 49.

Relevance

This command is relevant to operating modes 0, 1, 2 and 6.

> Note: Module will give the error for the ping requests until the ARP entry for the destination IP is successful.

7.4.41 Get socket information

Description

This command is used to query the information of the TCP socket with the socket handle requested.

Payload Structure

[1] WiSeConnect supports up to ping of data size from 1 to 54 bytes.

```
typedef union{
     struct {
           uint8 sock_handle;
     } SockinfoFrameSnd;
}rsi_socket_info_t;
```

Parameters

`sock_handle` – Socket descriptor

Response Payload

```
typedef struct {
     uint16 uSockDesc;
     uint16 mss;
     uint8 reserved[8];
}rsi_socketinfo_response;
```

Possible error codes

Possible error codes are 37 and 38.

Relevance

This command is relevant to the operating modes 0, 1, 2 and 6.

7.4.42 Get statistics

Description

This command is used to query the statistics of transmitted and received packets from/to WiSeConnect module.

Payload Structure

None

Parameters

None

Response Payload

```
typedef    struct {
     uint16 tx_mgmt_rate;
     uint16 tx_retries;
     uint16 rx_retries;
     uint16 signal_rssi;
     uint16 snr_value;
```

```
        uint16 reserved[5];
    }rsi_total_stats_t;
```

Possible error codes

Possible error codes are 37 and 38.

Relevance

This command is relevant to the operating modes 0, 2, 3 or 5.

> Note: The stats values (tx_retries, rx_retries) wrap around to zero after reaching the maximum value (0xffff) or for every time “get stats” command is issued from host.

7.5 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, 1 (Wi-Fi Direct mode), 2, 3, 4 (Wi-Fi Direct mode) ,5 ,6 (AP mode) and 7 (AP mode).

7.5.1 Storing Configuration Parameters in Client mode

7.5.1.1 Store Configuration in Flash Memory

Description

This command is used to save in internal memory the parameters of an access point to connect to (in auto-join mode) or that of the Access Point to create when the module is powered up (in auto-create mode).

Payload Structure

There is no payload for this command

Response Payload

There is no response payload for this command

Possible error codes

Possible error codes are 33, 37 and 44

7.5.1.2 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.

Payload Structure

```
struct cfgenable
{
UINT8 cfg_enable_val;
};
```

Parameters

`cfg_enable_val:`

0-Disables auto-join or auto-create

1-Enables auto-join or auto-create

Response Payload

There is no response payload for this command.

Possible error codes

Possible error codes are 33, 37 and 44

7.5.1.3 Get Information about Stored Configuration

Description

This command is used to get the configuration values that have been stored in the module's memory and that are used in auto-join or auto-create modes.

Payload Structure

There is no response for this command

Response Payload

```
{
 UINT8 cfg_enable;
 UINT8 opermode;
 uint8 band;
 uint8 reserved;
 uint8 ssid[34];
 UINT8 uRate;
```

```
  UINT8 uTxPower;
  UINT8 psk[64];
  UINT8 cnum;
  UINT8 dhcp_enable;
  UINT8 ip_addr[4];
  UINT8 snmask[4];
  UINT8 dgw[4];
  uint8 eap_method[32];
  uint8 reserved;
  uint8 user_identity[64];
  uint8 passwd[128];
   uint8 sec_type;
   uint8 encryption_type;
   uint8 beacon interval[2];
   uint8 dtim_period[2];
};
```

Response Parameters

cfg_enable (1 byte, hex):

0x00- auto-join or auto-create modes are disabled

0x01- auto-join or auto-create modes are enabled

Opermode (1 byte, hex):

0x00- Auto-join mode enabled Client mode with personal security (WPA/WPA2-PSK)

0x01- Auto-create mode enabled

0x02- Auto-join mode enabled with Enterprise Security

Band (1 byte, hex):

0x00- Module configured to operate in 2.4 GHz

0x01- Module configured to operate in 5 GHz

reserved (1 byte): Host should ignore this value

ssid (34 bytes, ASCII): 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, hex): Data rate configured in the module. Refer table Data Rate Parameter

uTXPower (1 byte, hex): Tx power configured in the module.

psk (64 bytes, ASCII): PSK configured in the module 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.

Cnum (1byte, hex): Channel number of the module in auto-join or auto-create mode

Dhcp_enable (1 byte, hex):

0x00- DHCP client is disabled in module

0x01- DHCP client is enabled in module

IP_addr (4 bytes, hex): 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, hex): Subnet mask

dgw(4 bytes, hex): Default gateway

*Eap_method (*1 byte, hex*):*

0x01- TLS,

0x02- TTLS,

0x03- PEAP,

0x04- FAST

*Reserved (*1 byte, hex*):*

user_identity (64 bytes, hex): User ID in enterprise security. Refer to the parameter *user_identity* in the command "Set EAP Configuration".

Passwd(128 bytes, ASCII): Password configured for enterprise security. Refer to the parameter *Password* in the command "Set EAP Configuration"*.* Filler bytes of 0x00 are used to make the length 128 bytes, of the original length is less than 128 bytes.

sec_type(1 byte, Hex): Security type of the AP when the module is in AP mode.

0x00 – OPEN,

0x01 – WPA,

0x02 – WPA2

encryption_type(1 byte, Hex): Security type of the AP when the module is in AP mode.

0x00 – OPEN,

0x01 – TKIP,

0x02 – AES

beacon_interval(2 bytes, Hex): Beacon interval in AP mode.

dtim_period(2 bytes, Hex): DTIM period in AP mode.

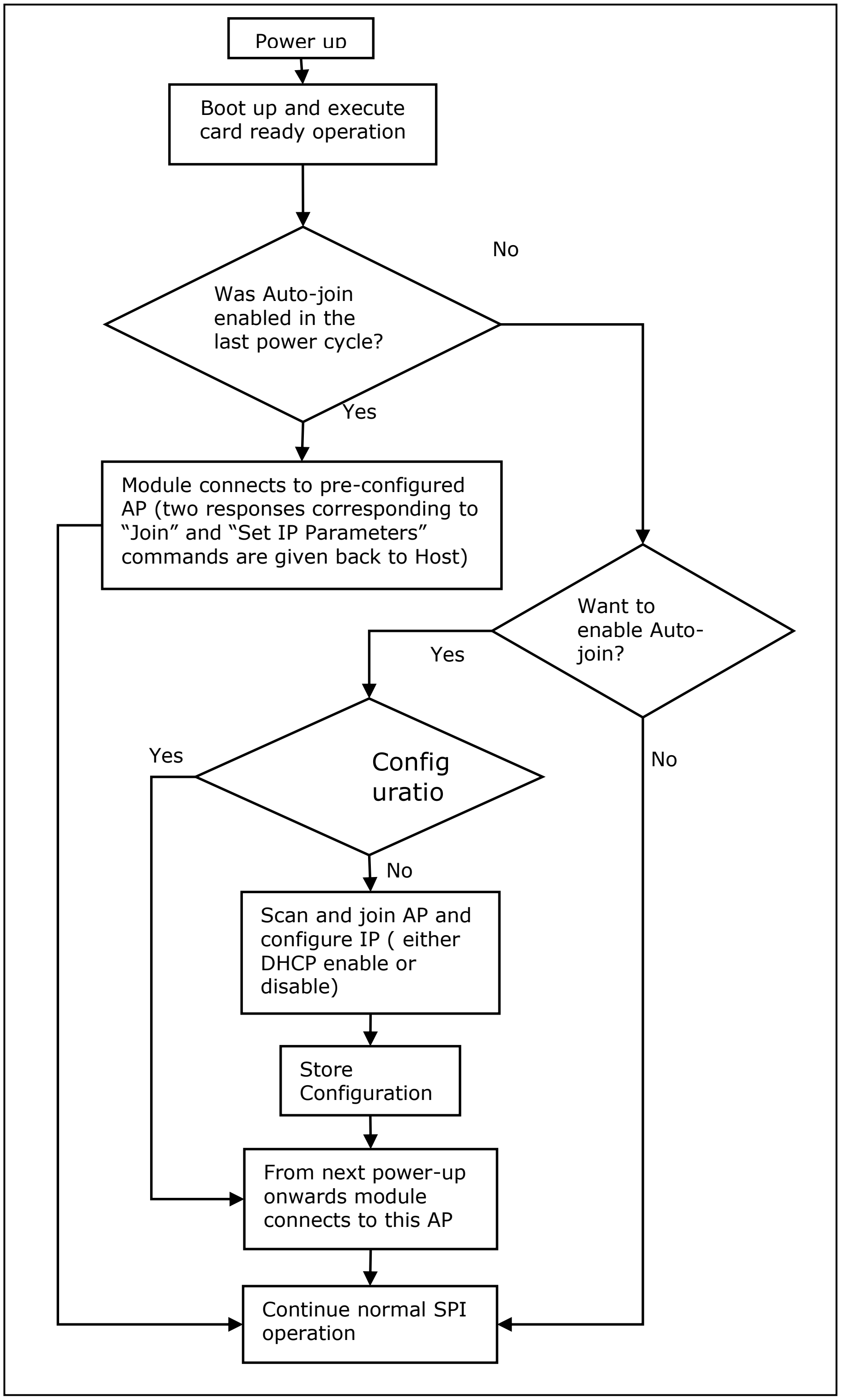

Power up
Boot up and execute card ready operation
Was Auto-join enabled in the last power cycle?
No
Yes
Module connects to pre-configured AP (two responses corresponding to “Join” and “Set IP Parameters” commands are given back to Host)
Want to enable Auto-join?
Yes
No
Config uratio
Yes
No
Scan and join AP and configure IP (either DHCP enable or disable)
Store Configuration
From next power-up onwards module connects to this AP
Continue normal SPI operation

Figure 27: Connecting to pre-configured AP

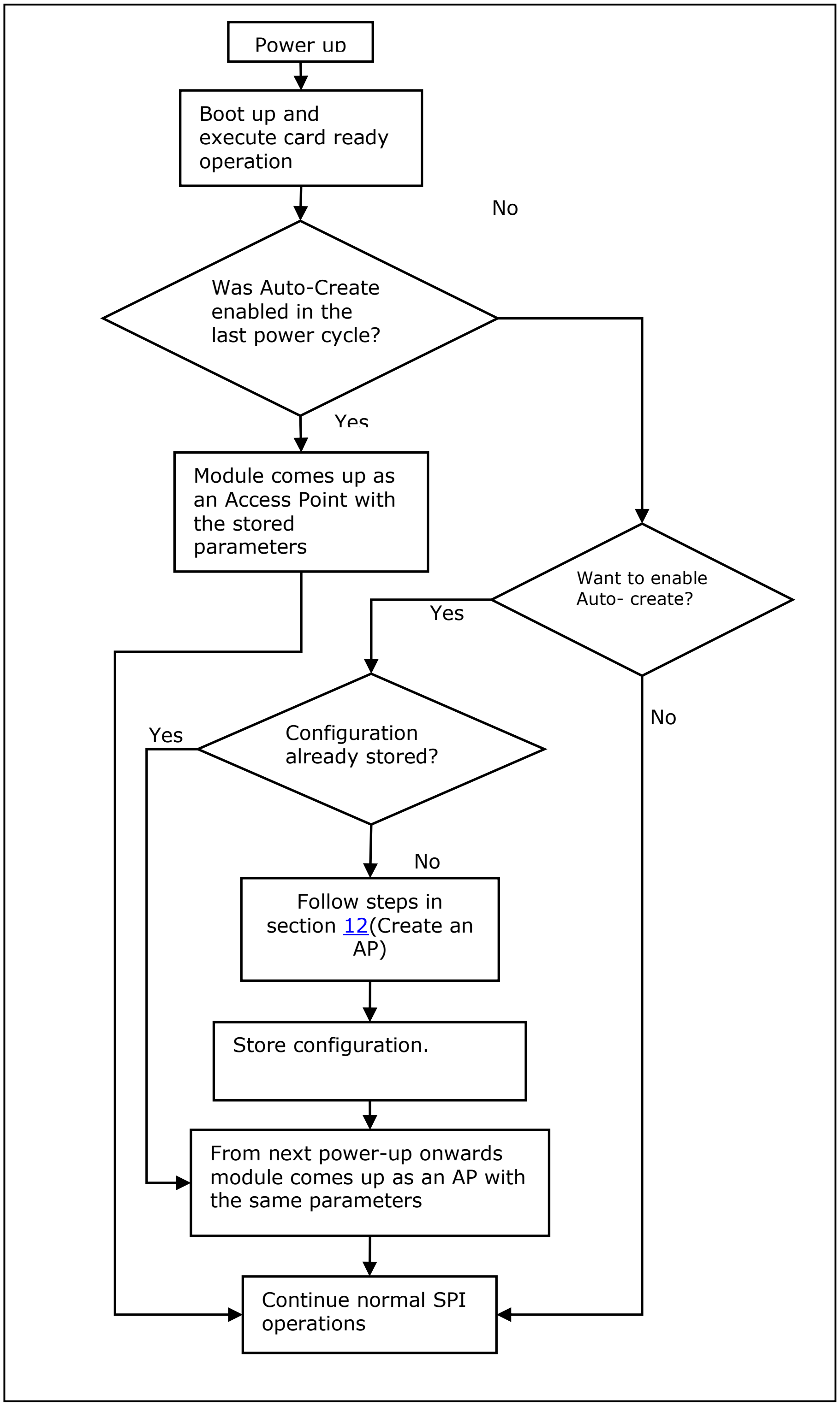

Power up
Boot up and execute card ready operation
No
Was Auto-Create enabled in the last power cycle?
Yes
Module comes up as an Access Point with the stored parameters
Want to enable Auto- create?
Yes
No
Yes
Configuration already stored?
No
Follow steps in section 12(Create an AP)
Store configuration.
From next power-up onwards module comes up as an AP with the same parameters
Continue normal SPI operations

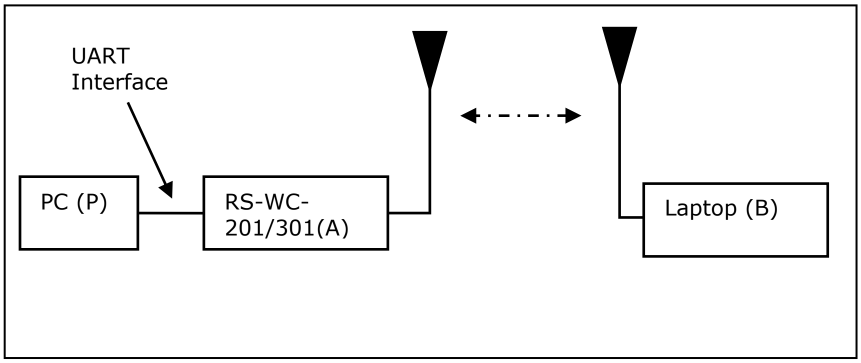
Figure 28: Creating a Pre-configured AP

7.6 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").

7.6.1 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.



11. Connect a PC or Host to the module through the UART interface and power up the module.
12. Configure the module to become an AP by issuing commands from PC (P) (refer APPENDIX B: Sample Flow of Commands in SPI).
13. Connect a Laptop (B) to the created AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.1, then the URL is http://192.168.100.1/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
14. 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.

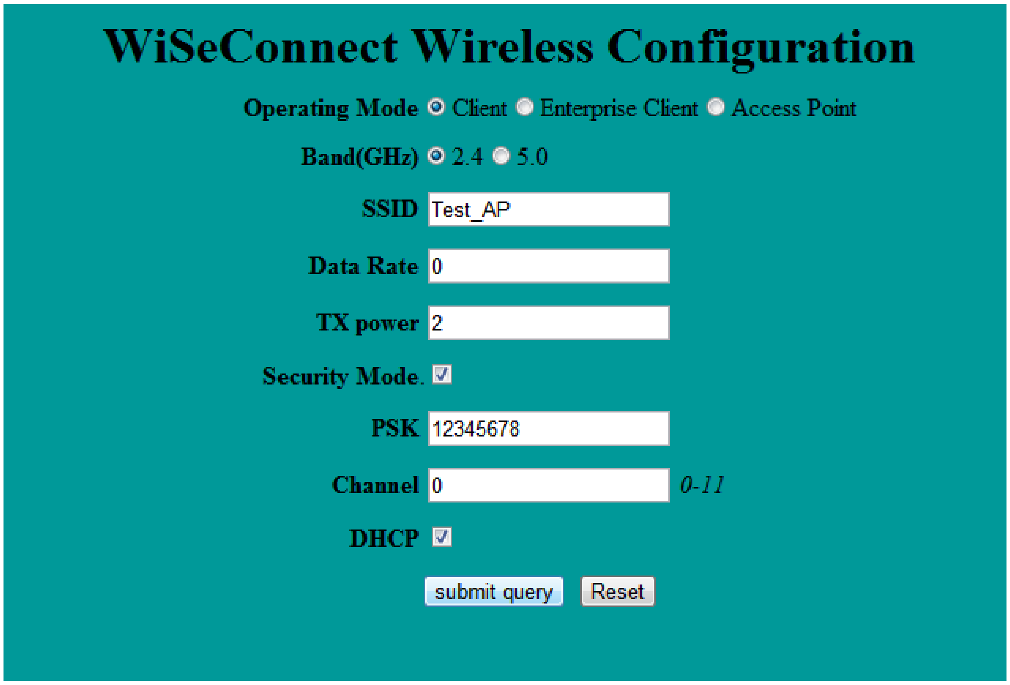
    Data rate: Physical data rate (refer Data Rate Parameter).

    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 Channels in 2.4 GHz and Channels in 5 GHz.



WiSeConnect Wireless Configuration

Operating Mode ● Client ○ Enterprise Client ○ Access Point

Band(GHz) ● 2.4 ○ 5.0

SSID Test_AP

Data Rate 0

TX power 2

Security Mode. ☑

PSK 12345678

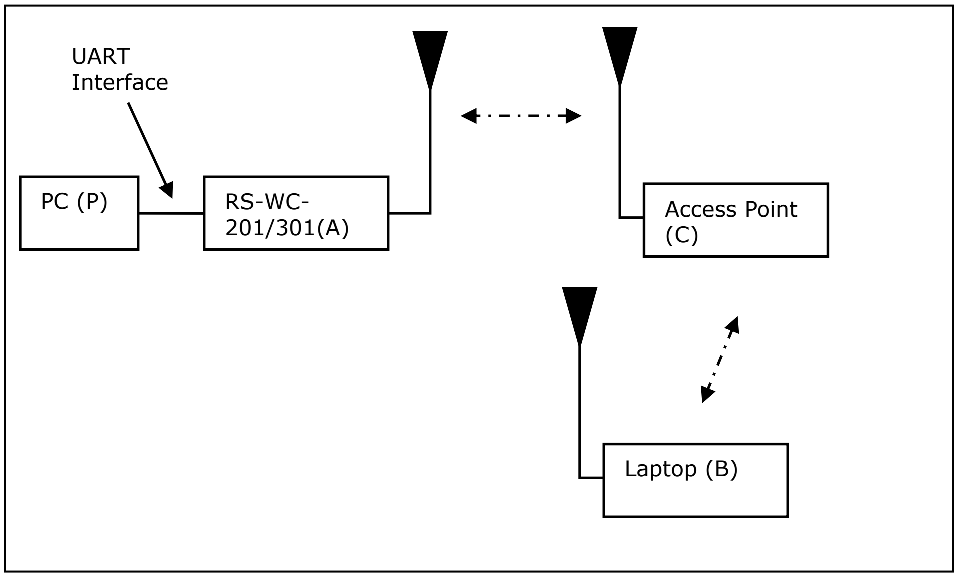
Channel 0 *0-11*

DHCP ☑

Click on "Submit Query" button. The information is sent to the module and stored in its internal flash.

15. 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 internally given "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.



10. Connect a PC or Host to the module through the UART interface and power up the module.
11. Configure the module to become a client and connect to an AP, by issuing commands from PC (P) (refer APPENDIX B: Sample Flow of Commands in SPI).
12. Connect a Laptop (B) to the same AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.20, then the URL is http://192.168.100.20/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
13. 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.

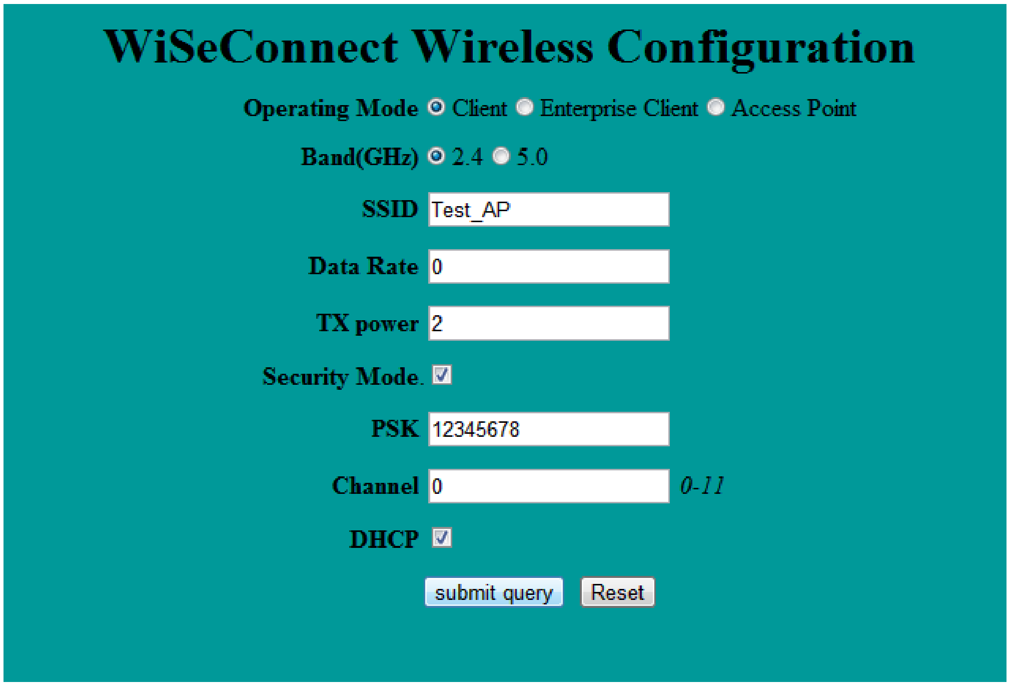
    Data rate: Physical data rate (refer Data Rate Parameter).

    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 Channels in 2.4 GHz and Channels in 5 GHz.



WiSeConnect Wireless Configuration

Operating Mode ● Client ○ Enterprise Client ○ Access Point

Band(GHz) ● 2.4 ○ 5.0

SSID Test_AP

Data Rate 0

TX power 2

Security Mode. ☑

PSK 12345678

Channel 0 *0-11*

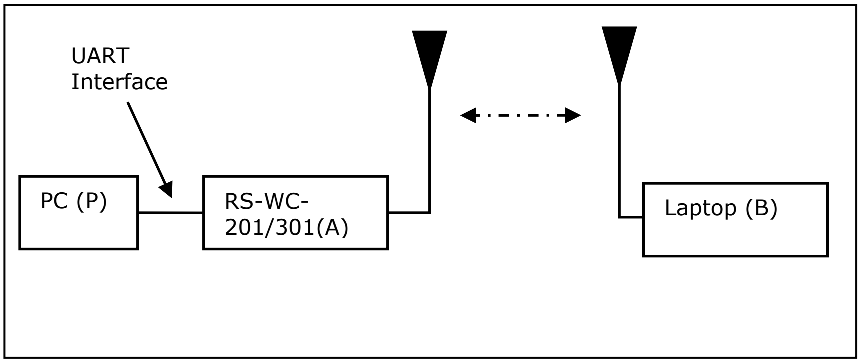
DHCP ☑

Click on "Submit Query" button. The information is sent to the module and stored in its internal flash.

14. 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 internally given "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.

7.6.2 Configuration 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.



6. Connect a PC or Host to the module through the UART interface and power up the module.
7. Configure the module to become an AP by issuing commands through PC (P). (refer APPENDIX B: Sample Flow of Commands in SPI).
8. Connect a Laptop (B) to the created AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.1, then the URL is http://192.168.100.1/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
9. 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.

   Data rate: Set the data rate to '0'.

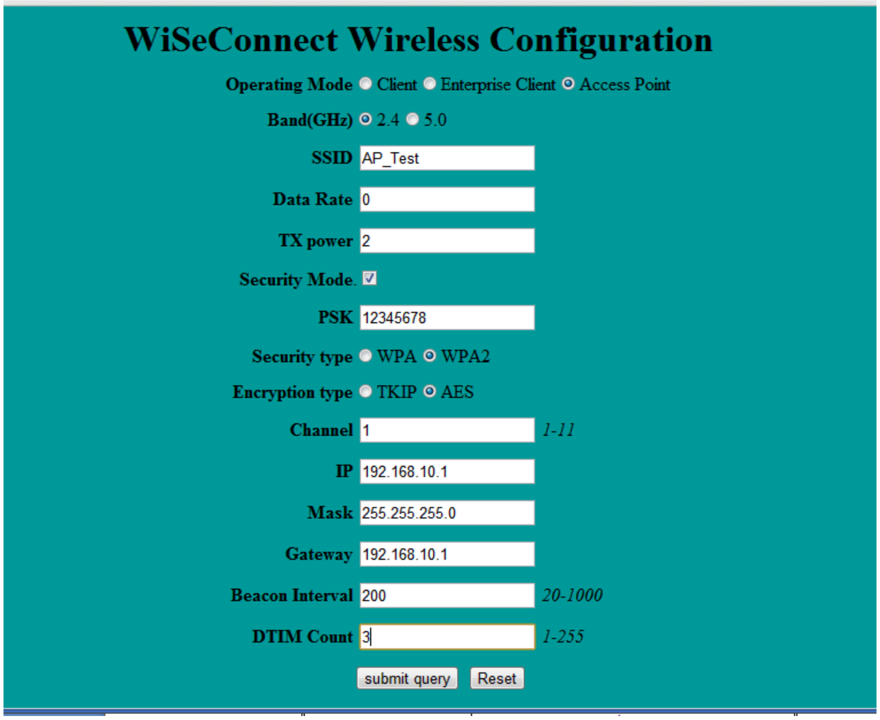
   Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.

   Security mode , 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 Channels in 2.4 GHz and Channels in 5 GHz. Value of '0' is not allowed.

   IP, Mask, 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.



WiSeConnect Wireless Configuration

Operating Mode ○ Client ○ Enterprise Client ● Access Point

Band(GHz) ● 2.4 ○ 5.0

SSID AP_Test

Data Rate 0

TX power 2

Security Mode ☑

PSK 12345678

Security type ○ WPA ● WPA2

Encryption type ○ TKIP ● AES

Channel 1 *1-11*

IP 192.168.10.1

Mask 255.255.255.0

Gateway 192.168.10.1

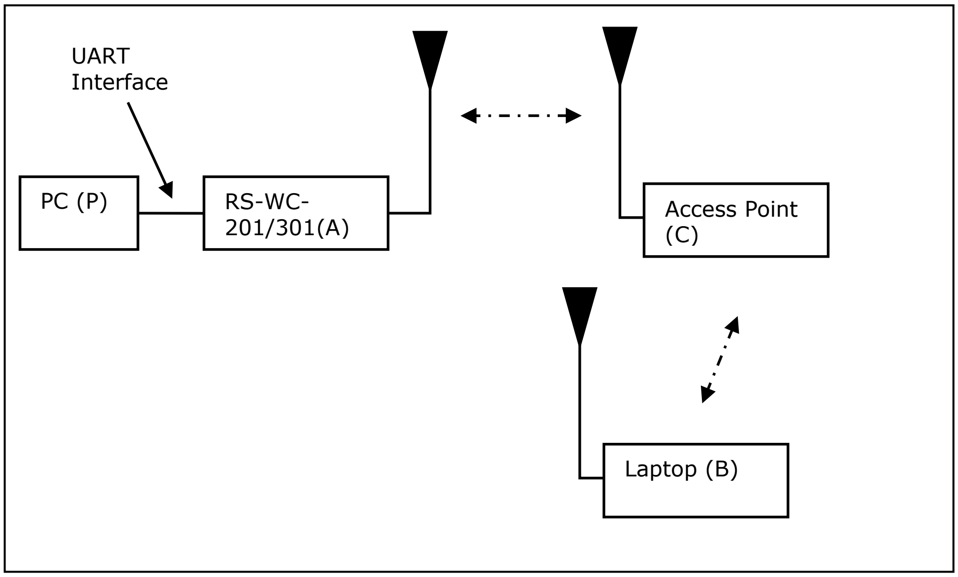
Beacon Interval 200 *20-1000*

DTIM Count 3 *1-255*

Click on "Submit Query" button. The information is sent to the module and stored in its internal flash.

10. The module should now be power cycled or hard reset. It boots up and then automatically creates and AP with the configured parameters. The module will send out two responses to the Host, the first corresponds to the internally given "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.



6. Connect a PC or Host to the module through the UART interface and power up the module.
7. Configure the module to become a client and connect to an AP by issuing commands from the PC (P) (refer APPENDIX B: Sample Flow of Commands in SPI).
8. Connect a Laptop (B) to the created AP. Open the URL **http://<Module's IP address>/config.htm** in the Laptop. For example, if the module was configured to have an IP of 192.168.100.20, then the URL is http://192.168.100.20/config.htm. Make sure the browser in the laptop does not have any proxies enabled.
9. 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.

   Data rate: Set the data rate to '0'.

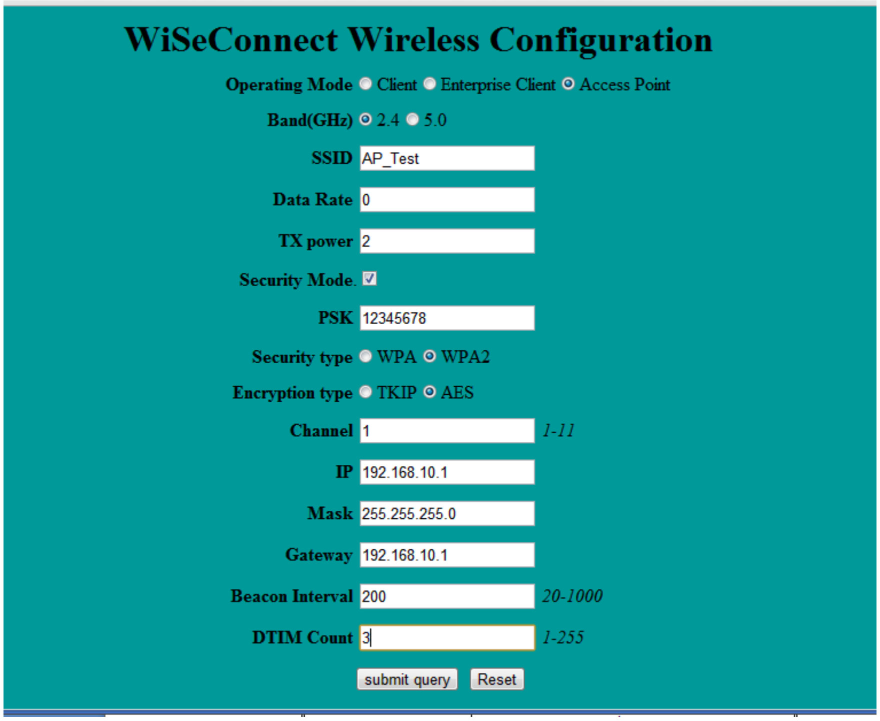
   Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.

   Security mode , 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 Channels in 2.4 GHz and Channels in 5 GHz. Value of '0' is not allowed.

   IP, Mask, 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.



WiSeConnect Wireless Configuration

Operating Mode	○ Client ○ Enterprise Client ● Access Point	
Band(GHz)	● 2.4 ○ 5.0	
SSID	AP_Test	
Data Rate	0	
TX power	2	
Security Mode	☑	
PSK	12345678	
Security type	○ WPA ● WPA2	
Encryption type	○ TKIP ● AES	
Channel	1	1-11
IP	192.168.10.1	
Mask	255.255.255.0	
Gateway	192.168.10.1	
Beacon Interval	200	20-1000
DTIM Count	3	1-255

Click on "Submit Query" button. The information is sent to the module and stored in its internal flash.

10. The module should now be power cycled or hard reset. It boots up and then automatically creates and AP with the configured parameters. The module will send out two responses to the Host, the first corresponds to the internally given "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.

7.7 Error Codes

Error Code(in decimal)	Description

2	Scan command issued while module is already associated with an Access Point
3	No AP found
4	Wrong PSK is issued while the module client tries to join an Access Point with WEP security enabled
5	Invalid band
6	Association not done or in unassociated state
8	De-authentication received from AP
9	Failed to associate to Access Point during “Join”
10	Invalid channel
14	1. Authentication failure during “Join” 2. Unable to find AP during join which was found during scan.
15	Missed beacon from AP during join
19	Non-existent MAC address supplied in “Disassociate” command
20	Wi-Fi Direct (or) EAP configuration is not done
21	Memory allocation failed
22	Information is wrong or insufficient in Join command
24	Push button command given before the expiry of previous push button command.
25	1.Access Point not found 2.Rejoin failure
26	Frequency not supported
28	EAP configuration failed
29	P2P configuration failed
30	Unable to start Group Owner negotiation
32	Unable to join
33	Command given in incorrect state
34	Query GO parameters issued in incorrect operating mode
35	Unable to form Access Point

36	Wrong Scan input parameters supplied to “Scan” command
37	Command issued during re-join in progress
38	Wrong parameters passed in command (e.g. SSID given is greater than 32 bytes, webpage length is given wrong in the command, more web fields are given, wrong values passed for GPIO configuration command)
40	PSK of wrong length is configured
41	Failed to clear or to set the Enterprise Certificate (Set Certificate)
42	Group Owner negotiation failed in Wi-Fi Direct mode
43	Association between nodes failed in Wi-Fi Direct mode
44	If a command is issued by the Host when the module is internally executing auto-join or auto-create
45	WEP key is of wrong length
46	ICMP request timed out
47	Ping size given is beyond the maximum ping size supported
48	Send data packet exceeded the limit or length that is mentioned
49	ARP Cache entry not found
-1	Listening TCP socket in module is not connected to the remote peer, or the LTCP socket is not yet opened in the module
-2	Sockets not available. The error comes if the Host tries to open more than 8 sockets or If the host tries to send data over socket which is already closed
-4	IP configuration failed
-8	Invalid operation (e.g. power save command with the same mode given twice, accessing wrong socket, creating more than allowed sockets)
-69	Invalid content in the DNS response to the DNS Resolution query
-70	DNS Class error in the response to the DNS Resolution query

-72	DNS count error in the response to the DNS Resolution query
-73	DNS Return Code error in the response to the DNS Resolution query
-74	DNS Opcode error in the response to the DNS Resolution query
-75	DNS ID mismatch between DNS Resolution request and response
-85	Invalid input to the DNS Resolution query
-91	IGMP error
-92	DNS response was timed out
-95	ARP request failure
-99	DHCP lease time expired
-100	DHCP handshake failure/ DHCP renewal failure
-121	This error is issued when module tried to connect to a non-existent TCP server socket on the remote side
-123	Invalid socket parameters
-127	Socket already open
-128	Attempt to open more than the maximum allowed number of sockets
-130	Data length is beyond maximum segment size (mss)
-151	Invalid command in sequence
-191	HTTP socket creation failed
-192	TCP socket close command is issued before getting the response of the previous close command
-190	DNS response timed out
-211	TCP ACK failed for TCP SYN-ACK
-205	TCP keep alive timed out

Table 15: Error Codes for SPI

8 Driver Porting Guide for SPI

The source code of a sample driver, application and API Library for the SPI interface is provided in the software package. The developer can port this reference driver to the target platform by incorporating appropriate HAL changes. It is assumed for the purposes of usage of this driver that the TCP/IP stack in the module is NOT bypassed. This section describes the driver source code and the APIs. It also discusses the requirements of the MCU's HAL APIs. This driver must be used in conjunction with the SPI section of this document (RS-WC-201/301 in SPI Mode) for successful operation of the module because all the parameters used in driver's functions are described in detail in the SPI section. The API library is platform independent and is written in C language.

8.1 Porting Steps

The following are the general steps required to port the driver into the target platform.

1. Modify the Hardware abstraction layer (HAL) based on hardware MCU platform. This involves configuring the SPI pins (SPI_MISO /SPI_MOSI/SPI_CLK/SPI_CS), Interrupt and the SPI_READY signal of module, Clock polarity (CPOL), Clock phase (CPHASE) and Data Endianess (figure Endianess in Data Transfer).
2. Modify the application to fit the actual evaluation set-up. The settings of files rsi_global.h and rsi_config.h can be modified to reflect intended mode of operation. The files are present in RS.WSC.x.x.GENR.A.B.C.X.Y.Z\Resources\SPI\Driver\Applications\MCU
3. Build the APIs along with the application using tool chain provided with the Host MCU.

8.2 File Structure

Path References:

References	Name
1	SPI API Library: RS.WSC.x.x.GENR.A.B.C.X.Y.Z\Resources\SPI\Driver\API_Lib
2	SPI MCU Applications: RS.WSC.x.x.GENR.A.B.C.X.Y.Z\Resources\SPI\Driver\Applications\MCU
3	SPI API Library Doxygen documentation (HTML based) : RS.WSC.x.x.GENR.A.B.C.X.Y.Z\Resources\SPI\Driver\\Documentation\html

The file/folder structure and contents of this library are as follows:

1. API_Lib (present in RS.WSC.x.x.GENR.A.B.C.X.Y.Z\Resources\SPI\Driver\API_Lib): The source code of the API Library to interact with the module over SPI interface.
2. Applications (present in RS.WSC.x.x.GENR.A.B.C.X.Y.Z\Resources\SPI\Driver\Applications\MCU): Contains sample applications to be run on the Host MCU connected to the module.

   Main.c – Top level application that calls the APIs

   rsi_config_init.c, rsi_config.h and rsi_global.h – These files contain the initial configuration and global macros for the API Library.

   Rsi_app_util – These files contain utility functions needed for configuration initialization and debug prints.

   Wifiuser.pem – Sample certificate file to be used in EAP-TLS mode to show the usage of rsi_set_certificate() API.

3. Documentation\HTML (Present in RS.WSC.x.x.GENR.A.B.C.X.Y.Z\Resources\SPI\Driver\ \Documentation\html): Contains the documentation for the API Library's source code in HTML form.

8.3 API Library

The API Library provides APIs which are called by the Application of the MCU in order to configure the Wi-Fi module and also exchange data over the network. In addition to this, a Makefile is provided for the user to compile the library with a standard compiler such as gcc. For detailed descriptions of the parameters please refer to RS-WC-201/301 in SPI Mode. The parameters are only briefly described here.

8.3.1 rsi_spi_opermode.c

This file contains the API for the "Set Operating Mode" command. This API is used to select the Legacy client mode or p2p mode or Enterprise Security. This API is the first to be called after CARD READY operation.

API Prototype :

int16 rsi_opermode(uint8 mode)

Parameters:

uint8 mode

Structure Member Name	Structure Member Type	Description

Structure Member Name	Structure Member Type	Description
mode	uint8	0-Client mode 1-WiFi Direct or AP mode 2-Enterprise security mode 6-AP mode

8.3.2 rsi_spi_band.c

This file contains the API for the "Band" command.

API Prototype:

int16 rsi_band(uint8 band)

Parameters:

uint8 band

Structure Member Name	Structure Member Type	Description
band	uint8	0– 2.4GHz 1– 5GHz

8.3.3 rsi_spi_init.c

This file contains the API for the "Init" command.

API Prototype :

int16 rsi_init(void)

Parameters:

None

8.3.4 rsi_spi_antenna_selection.c

This file contains API for the "Antenna Selection" command.

API Prototype:

int16 rsi_select_antenna(uint8 antenna_val)

Parameters:

Uint8 antenna_val – Parameter to select between internal and external antenna.

Structure Member Name	Structure Member Type	Description
Antenna_Val	uint8	1– Internal antenna selected 2– uFL selected for external antenna

8.3.5 rsi_spi_p2pcmd.c

This file contains the API for the “Configure Wi-Fi Direct Peer-to-Peer Mode” command. This API is used to set the WiFi-Direct arameters to the Wi-Fi module. This API should be used only in p2p mode. This API should be called only after rsi_init API.

API Prototype:

int16 rsi_p2pconfig(rsi_uConfigP2p *uConfigP2p)

Parameters:

rsi_uConfigP2p *uConfigP2p

```
typedef union
{
  struct {
        uint8           GOIntent[2];
        uint8           deviceName[64];
        uint8                   operChannel[2];
        uint8           ssidPostFix[64];
        uint8           psk[64];
  }configP2pFrameSnd;
 uint8                  uConfigP2pBuf[196];
}rsi_uConfigP2p;
```

Structure Member Name	Structure Member type	Description
GOIntent[2]	uint8	Group Owner Intent

Structure Member Name	Structure Member type	Description
deviceName[64]	uint8	Device name
operChannel[2]	uint8	Channel in which the Device operates
ssidPostFix[64]	uint8	Post Fix for SSID
psk[64]	uint8	Pre shared Key

8.3.6 rsi_spi_scan.c

This file contains the API for the “Scan” command.

API Prototype:

int16 rsi_scan(rsi_uScan *uScanFrame)

Parameters:

rsi_uScan *uScanFrame – Pointer scan parameter structure.

```
Typedef union
{
     struct{
           uint8                    channel[4] ;
           uint8                    ssid[RSI_SSID_LEN] ;
           }scanFrameSnd ;
           uint8                    uScanBuf[RSI_SSID_LEN + 4] ;
} rsi_uScan;
```

Structure Member Name	Member Type	Description
channel[4]	uint8	Channel Number of the Access Point.
Ssid[RSI_SSID_LEN]	uint8	SSID of the Access Point

8.3.7 rsi_spi_join.c

This file contains the API for the “Join” command.

API Prototype :

int16 rsi_join(rsi_uJoin *uJoinFrame)

Parameters:

rsi_uJoin *uJoinFrame – Pointer to join parameter structure.

```
Typedef union
{
struct {
        uint8                           reserved1;
     uint8                              wep_shared;
        uint8                           dataRate;
        uint8                           powerLevel;
        uint8                           psk[RSI_PSK_LEN];
        uint8                           ssid[RSI_SSID_LEN];
        uint8                           reserved2;
        uint8                           reserved3;
        uint8                           reserved4;
        uint8                           ssid_len;
    } joinFrameSnd;
     uint8              uJoinBuf[RSI_SSID_LEN + RSI_PSK_LEN + 8];

} rsi_uJoin ;
```

Structure Member Name	Structure Member Type	Description
reserved1, reserved2, reserved3, reserved4,	uint8	Reserved, set all to '0'
wep_shared	uint8	1 – To connect an AP in WEP shared mode 0 – For remaining all modes
dataRate	uint8	Rate at which the data has to be transmitted.
Powerlevel	uint8	Transmit Power level of the module.
Psk[64]	uint8	Pre-shared key (Only in Security mode). It is

Structure Member Name	Structure Member Type	Description
		an unused input in open mode. Last character is NULL.
Ssid[34]	uint8	SSID of the access point

8.3.8 rsi_spi_seteap.c

This file contains the API for the "Set EAP Configuration" command.

API Prototype

int16 rsi_seteap(rsi_uSetEap *uSetEap)

Parameters:

rsi_uSetEap *uSetEap

```
typedef union
{
        Struct
        {
        uint8           eapMethod[32];
        uint8           innerMethod[32];
        uint8                   userIdentity[64];
        uint8           password[128];
        }setEapFrameSnd;
        uint8                   uSetEapBuf[260];
}rsi_uSetEap;
```

Structure Member Name	Structure Member Type	Description
eapMethod[32]	uint8	Enterprise mode- TTLS FAST PEAP TLS
innerMethod[32]	uint8	MSCHAPV2
userIdentity[64]	uint8	username
Password[128]	uint8	password

8.3.9 rsi_set_certificate.c

This file contains the API for “Set certificate” command.

API Prototype:

int16 rsi_set_certificate(uint8 *buffer)

Parameters:

uint8 *buffer, pointer to the certificate buffer

8.3.10 rsi_spi_ipparam.c

This file contains the API for “Set IP Parameters” command.

API Prototype:

int16 rsi_ipparam_set(rsi_uIpparam *uIpparamFrame)

Parameters:

rsi_uIpparam *uIpparamFrame – Pointer to the ip configuration parameter structure.

```
Typedef union
{
    struct {
       uint8                          dhcpMode;
       uint8                          ipaddr[4];
       uint8                          netmask[4];
       uint8                          gateway[4];
       } ipparamFrameSnd;
       uint8                                 uIpparamBuf[13];
} rsi_uIpparam;
```

Structure Member Name	Structure Member type	Description
dhcpMode	uint8	The mode with which the TCP/IP stack has to be configured. 0– Manual 1– DHCP
ipaddr[4]	uint8	IP address of the TCP/IP stack (valid only in Manual mode)

Structure Member Name	Structure Member type	Description
netmask[4]	uint8	Subnet mask of the TCP/IP stack (valid only in Manual mode)
gateway[4]	uint8	Default gateway of the TCP/IP stack (valid only in Manual mode)

8.3.11 rsi_spi_socket.c

This file contains the API to open a socket inside the Wi-Fi module using "Open a Socket" command.

API Prototype:

int16 rsi_socket(rsi_uSocket*uSocketFrame)

Parameters:

rsi_uSocket *uSocketFrame – Pointer to socket create parameter structure.

```
Typedef union
{
        struct {
                uint8                           socketType[2];
                uint8                           moduleSocket[2];
            uint8                               destSocket[2];
                uint8                           destIpaddr[4];
                uint8               padding[2];
            } socketFrameSnd;
            uint8                       uSocketBuf[12];                      }
rsi_uSocket;
```

Structure Member Name	Structure Member Type	Description
socketType[2]	uint8	Type of the socket 0– TCP Client 1– UDP Client 2– TCP Server 4– Listening UDP
moduleSocket[2]	uint8	Local port on which the

Structure Member Name	Structure Member Type	Description
		socket has to be bound.
destSocket[2]	uint8	The destination's port. This port number is not valid for a listening socket.
destIpaddr[4]	uint8	The destination's IP address. This IP address is not valid for a listening socket.

8.3.12rsi_spi_socket_close.c

This file contains the API to "Close a socket" command.

API Prototype:

```
int16 rsi_socket_close(
        uint16 socketDescriptor
        )
```

Parameters:

uint16 socketDescriptor – Socket number to close. The socket descriptor is returned by the module at the time of socket creation.

8.3.13 rsi_spi_webserver.c

This file contains two APIs. One for loading the webpage into module's flash. Second is to give the content of the webpage with the URL requested by the module.

8.3.13.1 Load Webpage

This API used for the "Load webpage on Module" command.

API Prototype:

```
int16 rsi_load_webpage(rsi_uWebServer *uWebServer)
```

Parameters:

rsi_uWebServer *uWebServer

```
#define MAX_WEBPAGE_SEND_SIZE 1024
typedef struct
{
        uint8 total_len[2];
        uint8 current_len[2];
```

```
            uint8 more_chunks;

            uint8 webpage[MAX_WEBPAGE_SEND_SIZE];

    }WebpageSnd_t;

typedef union
 {
   struct {
         WebpageSnd_t   Webpage_info;
         }webServFrameSnd;
         uint8                      uWebServBuf[1029];
  }rsi_uWebServer;
```

Structure Member Name	Structure Member Type	Description
total_len[2]	uint8	Total Length of the page to be loaded
current_len[2]	uint8	Length of the current chunk
more_chunks	uint8	More chunks 0 – last chunk 1- More chunks
webpage[MAX_WEBPAGE_SEND_SIZE]	Uint8	web data to be loaded

8.3.13.2 URL response to the module

This APi is used to give URL response to the module.

API Prototype:

int16 rsi_send_url_rsp(WebpageSnd_t *uUrlRsp)

Parameters:

WebpageSnd_t *uUrlRsp

```
#define MAX_WEBPAGE_SEND_SIZE 1024

typedef struct

{

        uint8 total_len[2];

        uint8 current_len[2];

        uint8 more_chunks;

        uint8 webpage[MAX_WEBPAGE_SEND_SIZE];
}WebpageSnd_t;
```

Structure Member Name	Structure Member Type	Description
total_len[2]	uint8	Total Length of the page to be loaded
current_len[2]	uint8	Length of the current chunk
more_chunks	uint8	More chunks 0 – last chunk 1- More chunks
webpage[MAX_WEBPAGE_SEND_SIZE]	Uint8	web data to be loaded

8.3.14rsi_spi_webfields.c

This file contains the API for the “Load web fields on Module” command.

API Prototype:

int16 rsi_webFields (rsi_uWebFields *uWebFields)

Parameters:

rsi_uWebFields *uWebFields

```
#define MAX_NO_OF_FIELDS  10
 typedef struct field_st{
      uint8 field_index;
   uint8 field_val[64];
}field_st;
typedef union {
 struct {
        uint8 field_cnt;
        struct field_st field_st[MAX_NO_OF_FIELDS];
        }webFieldsFrameSnd;
        uint8   uWebFieldBuf[680];
}rsi_uWebFields;
```

Structure Member Name	Structure Member Type	Description
field_index	uint8	Index of the individual field
field_val	Uint8	Actual value in the field
field_cnt	Uint8	Number of fields the user wants to update

8.3.15 rsi_spi_query_fwversion.c

This file contains the API for "Query Firmware Version" command.

API Prototype :

int16 rsi_query_fwversion(void)

Parameters:

None

8.3.16 rsi_spi_query_macaddress.c

This file contains the API for "Query MAC Address" command.

API Prototype:

int16 rsi_query_macaddress()

Parameters:

None

8.3.17rsi_spi_send_data.c

This file contains the API to send application data payloads to the Wi-Fi module, which then transmits them over the Wi-Fi network using "Send data" command.

API Prototype:

```
int16 rsi_send_data(
        uint16 socketDescriptor,
        uint8 *payload,
        uint32 payloadLen,
        uint8 protocol
        )
```

Parameters:

Structure Member Name	Structure Member Type	Description
socketDescriptor	Uint16	Socket descriptor, used to identify the socket on which data has to be transmitted
payload		Pointer to data payload buffer which has to be transmitted
payloadLen		Length of the data payload
protocol		Type of the protocol (TCP/UDP)

8.3.18 rsi_spi_read_packet.c

This file contains the API to receive responses from the Wi-Fi module for ALL the commands (described above and in the following sections) that are sent to the module.

API Prototype:

```
int16 rsi_read_packet(
        rsi_uCmdRsp *uCmdRspFrame
        )
```

Parameters:

rsi_uCmdRsp *uCmdRspFrame – This is an output parameter to hold the response frame from the module. It is described further in Read Response Data Structure (From module)

8.3.19rsi_spi_send_raw_data.c

This file contains the API to send raw data to the module in TCP/IP bypass mode.

API Prototype:

int16 rsi_send_raw_data(uint8 *payload, uint32 payloadLen)

Parameters:

uint8 *payload – Pointer to the payload buffer

unt32 payloadLen – Length of the payload

8.3.20 rsi_spi_sleeptimer.c

This file contains the “Set Sleep Timer” command to set SPI Sleep timer.

API Prototype:

int16 rsi_sleeptimer(uint8 time)

Parameters:

uint8 time

8.3.21 rsi_spi_power_mode.c

This file contains the API for putting the Wi-Fi module to power save mode using “Power Mode” command.

API Prototype :

int16 rsi_power_mode(uint8 powerMode)

Parameters:

uint8 powerMode

API Prototype :

int16 rsi_pwrsave_continue(void)

Parameters:

None

8.3.22 rsi_spi_disconnect.c

This file contains the API for the “Disassociate” command.

API Prototype:

int16 rsi_disconnect(rsi_disassoc_t *disassoc_frame)

Parameters:

rsi_disassoc_t *disassoc_frame

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

8.3.23 rsi_spi_query_rssi.c

This file contains the API for "Query RSSI Value" command.

The Module should get associated to an AP.

API Prototype:

int16 rsi_query_rssi(void)

Parameters:

None

8.3.24 rsi_spi_query_net_parms.c

This file contains the API for querying the network parameters of the Wi-Fi module using "Query Network Parameters"

API Prototype:

int16 rsi_query_net_parms(void)

Parameters:

None

8.3.25 rsi_spi_query_conn_status.c

This file contains the API for the command "Query WLAN Connection Status".

API Prototype:

int16 rsi_query_conn_status(void)

Parameters:

None

8.3.26rsi_spi_query_go_parms.c

This file contains API for guery of GO parameters using "Query GO Parameters" command.

API Prototype
int16 rsi_query_go_parms(void)

Parameters

None

8.3.27rsi_spi_http_get.c

This file contains the API for the "HTTP GET" command.

API Prototype

int16 rsi_spi_http_get(rsi_uHttpReq * uHttpGetReqFrame)

Parameters

```
 rsi_uHttpReq * uHttpGetReqFrame
```

```
typedef union {
   struct {
      uint8             ipaddr_len[2];
      uint8             url_len[2];
      uint8             header_len[2];
      uint8             data_len[2];
      uint8             buffer[1200];
   } HttpReqFrameSnd;
   uint8                uHttpReqBuf[1208];
} rsi_uHttpReq;
```

Structure Member Name	Description
ipaddr_len	The length of the IP Address. This is required since the IP address is sent as a string in this frame. For example, if the IP address of www.website.com is 192.168.30.6, the ipaddr_len = 12
url_len	The length of the URL. For example, if www.website.com/welcome.php is the webpage, then url_len = 12 (for "/welcome.php", www.website.com is not included in the url).
Header_len	The length of the header of the HTTP GET request.
Data_len	This is the length of the data string.
Buffer	Buffer contains actual values in the order of IP Address, URL, Header and

Structure Member Name	Description
	Data.

8.3.28 rsi_spi_http_post.c

This file contains API for the “HTTP POST” command.

API Prototype

int16 rsi_spi_http_post(rsi_uHttpReq *uHttpReqFrame)

Parameters

```
rsi_uHttpReq *uHttpReqFrame
```

```
typedef union {
    struct {
        uint8           ipaddr_len[2];
        uint8           url_len[2];
        uint8           header_len[2];
        uint8           data_len[2];
        uint8           buffer[1200];
    } HttpReqFrameSnd;
    uint8               uHttpReqBuf[1208];
} rsi_uHttpReq;
```

Structure Member Name	Description
ipaddr_len	The length of the IP Address. This is required since the IP address is sent as a string in this frame. For example, if the IP address of www.website.com is 192.168.30.6, the ipaddr_len = 12
url_len	The length of the URL. For example, if www.website.com/welcome.php is the webpage, then url_len = 12 (for “/welcome.php”, www.website.com is

Structure Member Name	Description
	not included in the url).
Header_len	The length of the header of the HTTP GET request.
Data_len	This is the length of the data string.
Buffer	Buffer contains actual values in the order of IP Address, URL, Header and Data.

8.3.29 rsi_spi_dns_get.c

This file contains API for the “DNS Resolution” command.

API Prototype

```
int16 rsi_dns_query(rsi_uDnsQry  *uDnsQryFrame)
```

Parameters

rsi_uDnsQry *uDnsQryFrame – Pointer to DNS query frame.

```
#define RSI_MAX_DOMAIN_NAME_LEN 90
typedef union{
        struct {
                uint8   DomainName[RSI_MAX_DOMAIN_NAME_LEN];
                uint8   DnsNumber[2];
        } dnsQryFrameSnd;
        uint8   uDnsBuf[92];
 } rsi_uDnsQry;
```

Structure Member Name	Description
DomainName	Domain name, example: www.website.com . A maximum of 90 characters are allowed.
DnsNumber	To select DNS server to resolve the Query. 1-Primary DNS server 2-Secondary DNS server

8.3.30 rsi_spi_dns_server.c

This file contains API for the "DNS Server" command.

API Prototype:
int16 rsi_dns_server(rsi_uDns *uDnsFrame)

Parameters:

```
 typedef union {
      struct {

            uint8               DNSMode;
            uint8                         primary_dns_ip[4];
            uint8               secondary_dns_ip[4];
      }dnsServerFrameSnd;
      uint8                         uDnsBuf[9];
 }rsi_uDnsServer;
```

Structure Member Name	Description
DNSMode	1-Dynamic 0– Manual mode of entry for the DNS servers IP addresses
primary_dns_server	Primary DNS server IP address (valid only in Manual mode)
secondary_dns_server	Secondary DNS server IP address(valid only in Manual mode)

8.3.31 rsi_spi_module_reset.c

This file contains the API for the command "Soft Reset".

API Prototype

int16 rsi_spi_module_reset(void)

Parameters

None

8.3.32 rsi_spi_query_snr.c

This file contains the API for “Query SNR Value” command.

The Module should get associated to an AP.

API Prototype:

int16 rsi_query_rssi(void)

Parameters:

None

8.3.33 rsi_spi_featsel.c

This file contains the API for “Feature Select” command.

This needs to be given before the rsi_opermode API.

API Prototype:

int16 rsi_featsel(uint32 featsel_bitmap)

Parameters:

uint8 featsel_bitmap[4]

8.3.34 rsi_spi_cfgsave.c

This file contains the API for “Configuration Save” command.

This needs to be given after successful join command,

API Protoype:

int16 rsi_cfg_save(void)

Parameters:

None.

8.3.35 rsi_spi_cfgget.c

This file contains the API fpr “Get Configuration” command.

This is used to get the configuration saved through save configuration command.

API Prototype:

int16 rsi_cfg_get(void)

Parameters:

None.

8.3.36 rsi_spi_cfgenable.c

This file contains the API for “configuration Enable” command.

API Prototype:

int16 rsi_cfg_enable(uint8 cfg_enable_val)

Parameters:

uint8 cfg_enable_val

8.3.37 rsi_spi_wireless_fwupgrade.c

This file contains the API to control the behavior of the pins WF_HNDSHKE1 and WF_HNDSHKE2 as described in section Upgrading Firmware Wirelessly.

API Prototype:

int16 rsi_spi_wireless_fwupgrade(void)

Parameters:

None.

8.3.38 rsi_spiapconfCmd.c

This file contains API for apconf command. This needs to be called after init command. If the user wants to configure the IP address of the AP manually, IP Config should be called after init and then AP Config command should be called.

API Prototype

int16 rsi_apconfiguration(rsi_apconfig *apconf)

Parameters

rsi_apconfig *apconf

```
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 reserved1[2];
        uint8 max_sta_support[2];
}rsi_apconfig;
```

Structure Member Name	Description

Structure Member Name	Description
max_sta_support	Max number of clients that module can support in AP mode
dtim_period	This is the dtim period
beacon_interval	This takes value for beacon interval of AP
Channel_no	This is the channel number in which AP exist.
Ssid	This is ssid of AP
Security type	This is security type 0- No security 1-WPA 2-WPA2
encryp_mode	0 -No encryption 1-TKIP 2-CCMP
Psk	This is the pre shared key
Rserved1	Reserved for future use

8.3.39 rsi_spi_wepkeyCmd.c

This file contains API for wepkey command.

API Prototype

```
int16 rsi_set_wepkey(rsi_wepkey *wepkey)
```

Parameters

```
rsi_wepkey *wepkey

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

Structure Member Name	Description

Structure Member Name	Description
Index	This is the index of the key
Key	This two dimentional array takes four keys

8.3.40 rsi_spi_send_ludp_data.c

This file contains API for sending data over LUDP socket

API Prototype

```
int16 rsi_send_ludp_data
        (uint16 socketDescriptor,
        uint8 *payload,
         uint32 payloadLen,
        uint8 protocol,
        uint8 *destIp,
        uint16 destPort)
```

Parameters

uint16 socketDescriptor – socket descriptor for LUDP socket

uint8 *payload – Pointer to the payload data buffer

uint32 payloadLen – Length of the payload o be transferred

uint8 protocol – Protocol of the data transfer

uint8 *destIP – Destination IP address

uint16 destPort – Destination Port

8.3.41rsi_spi_gpio_config.c

This file contains the API to configure GPIOs.

API Prototype

int16 rsi_gpio_config(rsi_gpio_conf_t *gpio)

Parameters

rsi_gpio_conf_t *gpio

```
typedef struct rsi_gpio_conf_s
{
    uint8   pin;
    uint8   direction;
    uint8   value;
    uint8 reserved;
}rsi_gpio_conf_t;
```

uint8 pin – pin number to be configured

uint8 direction – Direction to be configured for the pin. For output it is '1'.

uint8 value – Value to be configured on the pin

uint8 reserved

8.3.42 rsi_spi_ping.c

This file contains the API to ping from WiSeConnect module.

API Prototype

```
int16  rsi_ping(rsi_ping_request_t *pingReq)
```

Parameters

```
rsi_ping_request_t  *pingReq
typedef struct rsi_ping_request_s{
        uint8 ping_IP[4];
        uint8 data_size[2];
}rsi_ping_request_t;
```

uint8 ping_IP[4] – Target IP to ping

uint8 data_size[2] – Size of the ping

8.3.43 rsi_spi_socketinfo.c

This file contains the API to query the TCP socket information.

API Prototype

```
int16 rsi_socketinfo(rsi_socket_info_t *uSocketinfo)
```

Parameters

```
rsi_socket_info_t *uSocketinfo
typedef union{
        struct {
                        uint8 sock_handle;
                } SockinfoFrameSnd;
        uint8 sock_handle;
}rsi_socket_info_t;
```

uint8 sock_handle – socket handle for the TCP socket

8.3.44rsi_spi_query_stats.c

This file contains the API to query the stats from WiSeConnect module.

API Prototype

int16 rsi_query_stats(void)

Parameters

None

8.4 Hardware Abstraction Layer (HAL) Files

The HAL files included in the API Library have placeholders for HAL APIs which need to be provided by the MCU's BSP. These can be filled with the MCU's HAL APIs directly or some more code might be needed to be written as wrappers if the MCU's HAL APIs are not directly compatible with them.

The HAL files are listed below.

8.4.1 rsi_hal.h

This is the header file for the HAL layer.

8.4.2 rsi_hal_mcu_timers.c

This file implements MCU related delay functions.

a.Millisecond timer

API Prototype:

```
void rsi_delayMs (uint16 delay)
{
}
```

Description:

This HAL API contains the code to introduce a delay in milliseconds.

b.Microsecond timer

API Prototype:

```
void rsi_delayUs (uint16 delay)
{


}
```

This HAL API contains the code to introduce a delay in micro seconds.

8.4.3 rsi_hal_mcu_spi.c

This API is used to transact the data to the Wi-Fi module through the SPI interface.

a.Sending data through SPI interface

API Prototype:

int16 rsi_spiSend(uint8 *ptrBuf, uint16 bufLen,uint8 *valBuf)

Parameters:

uint8 *ptrBuf – Pointer to the buffer containing the data to be sent through SPI interface.

Uint16 bufLen – Length of the data to be sent through SPI interface.

Uint8 *valBuf – Pointer to a four byte buffer to hold first two bytes of data received from the module while sending data through SPI interface.

```
{
   1. Control SPI Chip select pin to low if it is being controlled by user
   2. Write the data to be sent in the TX register.
   3. Wait for SPI TX to be completed.
   4. Ready dummy bytes from SPI RX register
   5. Repeat steps ‘b’ to ‘d’ to complete transfer of all bytes
   6. Control SPI chip select pin to high if it is being controlled by user
}
```

b.Receive data through SPI interface

API Prototype:

int16 rsi_spiRecv(uint8 *ptrBuf, uint16 bufLen)

Parameters:

uint8 *ptrBuf – Pointer the buffer to hold the received data from module through SPI interface.

Uint16 bufLen – Number of bytes to read from the module.

```
{

   1. Control SPI Chip select pin to low if it is being controlled by user
   2. Write dummy data to be sent in the TX register.
   3. Wait for SPI TX to be completed.
   4. Ready actual valid data from SPI RX register
```

5. Repeat steps ‘b’ to ‘d’ to complete transfer of all bytes

6. Control SPI chip select pin to high if it is being controlled by user

}

API Prototype:

int16 rsi_hal_mcu_spi_init(void)

Parameters:

None.

{

To configure the SPI settings for Host MCU.

}

8.4.4 rsi_hal_mcu_ioports.c

This file contains API to control different pins of the microcontroller which interface with the module and other components related to the module.

a.Reset the Module

API Prototype:

void rsi_moduleReset(uint8 state)

Parameters:

uint8 state

```
        void rsi_moduleReset(uint8 state)
{
  if (state == RSI_TRUE) {
      /*  Set Reset */
  }
  else {
      /*  Clear Reset */
  }
}
```

This HAL API is used to set or clear the active- low reset pin of the Wi-Fi module.

b. Configure SPI READY pin as input

API Prototype:

void SPI_Ready_Init(void)

Parameters:

No Parameters

```
void SPI_Ready_Init(void)
{
        1. Configure SPI MISO pin, SPI MOSI pin, SPI CLK pin, SPI CS pin
        2. Configure SPI in master mode. Configure SPI MISO pin, SPI
        MOSI pin, SPI CLK pin, SPI CS pin. Configure SPI peripheral in full
        duplex mode.
        3. Configure clock polarity CPOL as '0' and clock phase CPHA as '0'
        4. Configure transfer length to 8 bits.
        5. Configure data mode as 'MSB first'
        6. Configure SPI clock upto 12MHz (buad rate)
        7. Configure external interrupt pin (Input) to receive interrupts
        from wifi module and register to ISR
        8. Configure Reset pin (Output)
        9. Configure WLAN module "SPI ready" pin as input.
}
```

This HAL API is used to configure Host GPIO as input to receive SPI_Ready signal from module.

c. Configure WF_HNDSHKE2 as output

This API is used to configure Host GPIO as output to give wakeup interrupt to the module.

API Prototype:

void rsi_wsc_wakeup_pin_init(void)

Parameters:

None.

d. Configure WF_HNDSHKE1 as input

API Prototype:

void WF_ HNDSHKE1_Init(void)

Parameters:

None.

e. Configure WF_HNDSHKE2 as output

API Prototype:

void WF_HNDSHKE2_Init(void)

Parameters:

None.

8.4.5 rsi_hal_mcu_interrupt.c

This file contains the list of functions for configuring the microcontroller interrupts.

API Prototype:

```
void rsi_spiIrqStart(void)
{
}
```

This HAL API should contain the code to initialize the register related to interrupts.

API Prototype :

```
void rsi_spiIrqDisable(void)
{
}
```

This HAL API should contain the code to disable interrupts.

API Prototype :

```
void rsi_spiIrqEnable(void)
{
}
```

This HAL API should contain the code to enable interrupts.

API Prototype:

```
void rsi_spiIrqClearPending(void)
{
}
```

This HAL API should contain the code to clear the handled interrupts.

API Prototype:

```
void rsi_spi_interrupt_handler(void)
{
        pkt_pending++;
        interrupt_rcvd++;
}
```

This HAL API contains the code to update variables for SPI interrupt.

8.5 Response Data Structures

This section describes important data structures used to read responses from the module.

8.5.1 Read Response Data Structure (From module)

This important data structure is used by the library to pass the values received from the Wi-Fi module to the application. This structure is updated for each call of the rsi_spi_read_packet API with the appropriate information. The rsi_uCmdRsp structure is a union of multiple structures and is explained below.

```
typedef struct
{
        uint8                   rspCode[2];
        uint8                   status[2];
union {
    rsi_initResponse            initResponse;
    rsi_scanResponse         scanResponse;
    rsi_joinResponse         joinResponse;
    rsi_wfdDevRsp           wfdDevRespone;
    rsi_rssiFrameRcv         rssiFrameRcv;
    rsi_socketFrameRcv        socketFrameRcv;
    rsi_socketCloseFrameRcv   socketCloseFrameRcv;
    rsi_ipparamFrameRcv       ipparamFrameRcv;
    rsi_conStatusFrameRcv     conStatusFrameRcv;
    rsi_qryNetParmsFrameRcv   qryNetParmsFrameRcv;
    rsi_qryFwversionFrameRcv  qryFwversionFrameRcv;
    rsi_recvFrameUdp          recvFrameUdp;
    rsi_recvFrameTcp          recvFrameTcp;
    rsi_recvRemTerm           recvRemTerm;
    rsi_recvLtcpEst          recvLtcpEst;
```

```
            rsi_qryMacFrameRcv recvMacFrame;
             uint8 uCmdRspPayloadBuf[56+1400+100];
           }uCmdRspPayload;
    } rsi_uCmdRsp;
```

Structure/ Union Name	Structure Member name	Structure Member Type	Description
rsi_uCmdRsp			Common structure for reading the responses
	rspCode[2]	uint8	response code of the command executed (refer response code table)
	status[2]	uint8	returns 0 for success of the command executed returns Error code for Error in command. (Refer to PRM for error codes)
rsi_initResponse			Response structure for **Init** command
	macAddress[6]	uint8	returns mac address after initialization
rsi_wfdDevRsp			Response structure corresponding to the asynchronous interrupt received after issuing the command Configure Wi-Fi Direct Peer-to-Peer Mode
	devstate	uint8	1 for New Device 0 for left device
	devName[4]	uint8	Name of the scanned WFD device
	macAddress[6]	Uint8	Mac Address of the WFD Device scanned
	devType[2]	Uint8	1st byte indicates Primary Device type 2nd byte indicates sub category
rsi_scanResponse			Response structure for "Scan" command
	scanCount[4]	uint8	Number of access points found.
	strScanInfo[RSI_AP_SCANNED_M	rsi_scanInfo	Scanned Access point information in following section

	AX]		
rsi_joinResponse			Response structure for "Join" command
	operState	Uint8	State of the device in 'G'-GO 'C'-Client
rsi_rssiFrameRcv			Response structure for "Query RSSI" command
	rssiVal[2]	uint8	RSSI Value
rsi_socketFrameRcv			Response structure for "Open a Socket" command
	socketType[2]	uint8	Type of the socket created
	socketDescriptor [2]	uint8	Created socket descriptor (or handle).Need to use this number while sending data through this socket using rsi_send_data and close this socket using rsi_socket_close API's.
	moduleSocket[2]	uint8	Local port number
	moduleIpaddr[4]	uint8	Local ipaddress
rsi_socketCloseFrameRcv			Response structure for "Close a Socket"
	socketDsc[2]	uint8	Descriptor of the socket closed
rsi_ipparamFrameRcv			Response structure for "Set IP Parameters" command
	macAddr[6]	uint8	Mac address of WiFi module
	ipaddr[4]	uint8	IP address of Wi-Fi module
	netmask[4]	uint8	Network mask configured
	gateway[4]	uint8	Gateway configured
rsi_qryNetParmsFrameRcv			Response structure for "Query Network Parameters"
	wlanState[2]	uint8	This indicates whether the module is connected to an Access Point or not. 0 – Not connected 1 – Connected

	Chn_No	uint8	Channel number
	Psk[64]	uint8	PreShared Key
	mac_addr[6]	uint8	Mac address of AP
	sec_type	uint8	Security type (open or enterprise mode)
	ssid[34]	uint8	This value is the SSID of the Access Point to which the module is connected.
	Ipaddr[4]	uint8	This is the IP Address configured to Wi-Fi module.
	subnetMask[4]	uint8	This is the Subnet Mask configured to Wi-Fi module.
	Gateway[4]	uint8	This is the gateway configured to WiFi module
	dhcpMode[2]	uint8	This value indicates whether the module is configured for DHCP or Manual IP configuration. 0 – Manual IP configuration 1 – DHCP
	connType[2]	uint8	This value indicates whether the module is operational in Infrastructure mode 1 – Infrastructure mode
	num_open_socks [2]	Unit8	Number of sockets open
rsi_qryFwversionFrameRcv			Response structure for "Query Firmware Version"
	Fwversion[20]	uint8	Version of the firmware loaded in the module. This is given in string format. The firmware version format is x.y.z,a.b.c (e.g., 1.0.0,0.0.6) 1.0.0 WiSe Control version 0.0.6 WiSe WLAN version
rsi_recvFrameUdp			Structure for receiving UDP data
	recvSocket[2]	uint8	Socket descriptor on which data received
	recvBufLen[4]	uint8	Receive packet length

	recvDataOffsetSize[2]	uint8	Offset where the actual payload data start in the buffer.
	fromPortNum[2]	uint8	Port number of remote machine (from where this packet received)
	fromIpaddr[4]	uint8	IP address of remote machine (from where this packet received)
	recvDataOffsetBuf[26]	uint8	Dummy data before actual payload start, need to ignore this content.
	recvDataBuf[1400]	uint8	Actual payload data.
	Padding[2]	uint8	padding
rsi_recvFrameTcp			Structure for receiving TCP data
	recvSocket[2]	uint8	Socket descriptor on which data received
	recvBufLen[4]	uint8	Receive packet length
	recvDataOffsetSize[2]	uint8	Offset where the actual payload data start in the buffer.
	fromPortNum[2]	uint8	Port number of remote machine (from where this packet received)
	fromIpaddr[4]	uint8	IP address of remote machine (from where this packet received)
	recvDataOffsetBuf[38]	uint8	Dummy data before actual payload start, need to ignore this content.
	recvDataBuf[1400]	uint8	Actual payload data.
	Padding[2]	uint8	Padding
rsi_recvRemTerm			Response structure for Remote Socket Closure.
	Socket[2]	uint8	Socket descriptor for which the Remote termination has happened.
Rsi_qryMacFrameRcv			Response structure for "Query Mac Address"
	macAdress[6]	uint8	MAC Address of the module
rsi query go parms			Query GO params response structure.
	Ssid[34]	uint8	SSID of the P2P GO

	bssid[6]	uint8	BSSID of the P2P GO
	Channel number	uinit8	operating channel of GO
	psk[64]	uint8	PSK of the GO
	ip	uint8	IP address of GO
	sta_count	uint8	Number clients associated to clients
	sta_info	go_sta_info_s	associated clients information. Refer the next section for details
rsi_HttpGetFrameRcv			Response to HTTP GET request
	Ipaddr_len	Uint16	The length of the IP Address. This is required since the IP address is sent as a string in this frame. For example, if the IP address of www.website.com is 192.168.30.6, the ipaddr_len = 12
	url_len	Uint16	The length of the URL. For example, if www.website.com/index.html is the webpage, then url_len = 11 (for "/index.html", www.website.com is not included in the url).
	Header_len	Uint16	The length of the header of the HTTP GET request.
	Data_len	Uint16	This value has to be assigned 0.
	Buffer[1200]	Uint8	Buffer contains actual values in the order of IP Address, URL, Header and Data.
Rsi_HttpGetFrameRcv			Response to HTTP POST request
	Ipaddr_len	Uint16	The length of the IP Address. This is required since the IP address is sent as a string in this frame. For example, if the IP address of

			www.website.com is 192.168.30.6, the ipaddr_len = 12
	url_len	Uint16	The length of the URL. For example, if www.website.com/index.html is the webpage, then url_len = 11 (for "/index.html", www.website.com is not included in the url).
	Header_len	Uint16	The length of the header of the HTTP GET request.
	Data_len	Uint16	This value has to be assigned 0.
	Buffer[1200]	Uint8	Buffer contains actual values in the order of IP Address, URL, Header and Data.
Rsi_DNS QryRespo nse			Response to DNS query.
	uIPCount	uint16	This indicates number of Ips resolved for the given domain name
	aIPaddr[10][4]	uint8	This returns the resolved IP addresses. A maximum of 10 IP addresses can be returned. User should read the number of IP addresses indicated by uIPCount.

Table 16: Read Response Data Structure in Driver

8.5.2 Scan information data structure

The below structure is part of "rsi_uCmdRsp"

```
typedef struct {
        uint8 rfChannel;
        uint8 securityMode;
        uint8 rssiVal;
        uint8   uNetworkType;
        uint8 ssid[RSI_SSID_LEN];
        uint8 BSSID[6];
#ifndef RSI_FEATSEL_ENABLE[1]
        uint8    reserved[2];
#else
        uint8    snr;
        uint8    reserved;
        uint8    ap_name[16];
#endif

} rsi_scanInfo;
```

Structure Member Name	Structure Member Type	Description
rfchannel	uint8	Channel of the scanned AP
securityMode	uint8	Security Mode 0-Open mode 1-WPA 2-WPA2 4- WPA Enterprise 5-WPA2 Enterprise
rssiVal	uint8	RSSI Value of scanned AP
uNetworkType	uint8	1-Infrastructure Mode
ssid[34]	uint8	SSID of Access Point
bssid[6]	uint8	BSSID of scanned AP
reserved[2]	uint8	Reserved, two bytes if there is no Feature select, other wise only one byte of reserved field.
snr	uint8	Signal to Noise Ratio

[1] This Macro need to be defined in rsi_global.h to use feature select.

Structure Member Name	Structure Member Type	Description
ap_name[16]	uint8	Cisco AP name

8.6 Applications

The files in the Applications folders (Ref[2]) contains files for the application layer of the Host MCU. These have to be modified to setup the application for the system which the user wants to realize. The user has to call the APIs provided in the API library to setup the wireless connection and exchange data over the network.

1. main.c – This file contains the entry point for the application. It also has the initialization of parameters of the global structure and the operations to control & configure the module, like scanning networks, joining to an Access Point etc. Here we just provided sample code for the user to understand the flow of commands. This is not must to use the same. User can write his own application code instead of that.
2. rsi_app_util.h and rsi_app_util.c – These files contain list of utility functions which are used by rsi_config_init API and debug prints.
3. rsi_config.h and rsi_config_init.c – These files contain all the parameters to configure the module. Some example parameters are SSID of the Access Point to which the module should connect, IP address to be configured in the module, etc.

   To facilitate Application development we have defined a data structure named rsi_api as described below.This structure is initialized by the application using rsi_init_struct API of the rsi_config_init.c file (application layer file) and then uses to pass the parameter to the library API calls. The user may change the values assigned to the macros without worrying about understanding the contents of the structure.

   The contents of this structure are explained in brief below, using the declaration of the structure in rsi_global.h file (which is also an application layer file).

```
Typedef struct {
     uint8                           band;
     uint8               opermode;
   uint8                             powerMode;
   uint8             antSel;
   uint16                socketDescriptor;
     uint8                         macAddress[6];  rsi_uScan
          uScanFrame;                    rsi_uJoin
```

```
        uJoinFrame;                   rsi_uIpparam
        uIpparamFrame;                rsi_uSocket
uSocketFrame;

        rsi_uWebServer         uWebData;

        rsi_uWebFields         uWebField;

        rsi_uSetEap            usetEap;

        rsi_uConfigP2p         uconfigP2p;

} rsi_api;
```

Following are list of macro's need to define in rsi_global.h based application requirement.

a. #define RSI_MAX_PAYLOAD_SIZE 1400 – To configure maximum packet size.

b. #define RSI_AP_SCANNED_MAX   11 – Maximum number of access points can scan, please refer "Scan" command section for maximum number access point module can return.

c. #define RSI_MAXSOCKETS 8 –Maximum number of sockets module support, refer "Open a Socket" command section for more information.

d. #define RSI_PSK_LEN 64 – PSK length for more information refer "Join" command section.

e. #define RSI_SSID_LEN 34 – SSID length module supports for more information refer "Join" command section.

f. #define RSI_LITTLE_ENDIAN 0 – Comment this incase the host processor is big endian

g. #define RSI_FEATSEL_ENABLE   0 – Comment this if feature select command is not used at all

h. #define RSI_HOST_CPU_CLK_IN_MHZ   48 - To use delay functions and timeouts, user need to configure this MACRO with the host CPU clock value in MHz.

Following are list of macro's need to define in rsi_config.h based application requirement. This Macro's used by rsi_init_struct function to initialize rsi_api data structure.

a. #define RSI_OPERMODE To configure the operating mode

b. #define RSI_MODULE_IP_ADDRESS To configure IP address to module.

c. #define RSI_GATEWAY To configure gateway IP address to module.

d. #define RSI_TARGET_IP_ADDRESS To configure target IP address.

e. #define RSI_NETMASK To configure network mask to the module.

f. #define RSI_SECURITY_TYPE To configure RSI_SECURITY_OPEN or RSI_SECURITY_WPA1 or RSI_SECURITY_WPA2

g. #define RSI_PSK To configure PSK, if security mode is enabled(\0 for open mode).

h. #define RSI_SCAN_SSID To scan only particular access point configure this macro.

i. #define RSI_SCAN_CHANNEL To scan only particular channel configure this macro, if 0 module will scan all the channels.

j. #define RSI_JOIN_SSID To configure SSID to join.

k. #define RSI_IP_CFG_MODE To enable or disable DHCP while IP configuration (RSI_DHCP_MODE_DISABLE or RSI_DHCP_MODE_ENABLE)

l. #define RSI_NETWORK_TYPE To select network type (RSI_INFRASTRUCTURE_MODE)

m. #define RSI_BAND To select BAND (RSI_BAND_2P4GHZ or RSI_BAND_5GHZ).

n. #define RSI_DATA_RATE To select data rate auto or fixed data rate (RSI_DATA_RATE_AUTO or RSI_DATA_RATE_(1, 2, 5P5, 11, 6, 9, 12)).

o. #define RSI_POWER_LEVEL To select power level (RSI_POWER_LEVEL_LOW or RSI_POWER_LEVEL_MEDIUM or RSI_POWER_LEVEL_HIGH)

p. #define P2P_DEVICE_NAME To give P2P device name of the Wi-Fi Module

q. #define POST_FIX_SSID To give the Post_Fix_SSID of the P2P device

r. #define GO_INTENT_VALUE To set the GO_INTENT of the WiSeconnect(0-15 for P2P ,16 for Access Point)

s. #define OPER_CHANNEL To set the Operating channel of the device

t. #define EAP_METHOD To set the EAP security method(TLS,TTLS.PEAP,FAST)

u. #define INNER_METHOD To set the inner method of the EAP Method(MSCHAPV2)

v. #define USER_IDENTITY To set the user name (user1)

w. #define PASSWORD To set the password of authentication server ex:RADIUS (test123)

x. #define KEY PASSWORD To set the key password which is used to generate the certificate.

y. #define WEB_PAGE_LENGTH To set the length of the web page to be loaded

Note: The above way of configuring global structure is optional. If the user wants to use global rsi_api and initialize using rsi_init_struct, then the above MACROs should be defined in rsi_config.h

8.6.1 Using rsi_config.h for various modes

This function will help the user to define the MACROs in rsi_config.h file to configure the module to different operating modes.

8.6.1.1 Client mode with Personal Security

```
#define RSI_OPERMODE               0
#define CLIENT_MODE                ENABLE
#define P2P_MODE                   DISABLE
#define ENTERPRISE_MODE            DISABLE
#define AP_MODE                    DISABLE
#define RSI_BAND                   RSI_BAND_2P4GHZ
#define RSI_SECURITY_TYPE          RSI_SECURITY_WPA2
#define RSI_NETWORK_TYPE           RSI_INFRASTRUCTURE_MODE
#define RSI_PSK                    “12345678”
#define RSI_SCAN_SSID              “”
#define RSI_SCAN_CHANNEL           0
#define WEB_PAGE_LENGTH            26
#define RSI_JOIN_SSID              “DLINK_DMA”
#define SSID_LEN                   9
#define RSI_DATA_RATE              RSI_DATA_RATE_AUTO
#define RSI_POWER_LEVEL            RSI_POWER_LEVEL_HIGH
```

8.6.1.2 WiFi Direct Mode

```
#define RSI_OPERMODE               1
#define CLIENT_MODE                DISABLE
#define P2P_MODE                   ENABLE
#define ENTERPRISE_MODE            DISABLE
```

```
#define AP_MODE                  DISABLE
#define RSI_BAND                         RSI_BAND_2P4GHZ
#define WEB_PAGE_LENGTH             26
#define RSI_PSK                            “12345678”
#define P2P_DEVICE_NAME            “WSC1.0”
#define POST_FIX_SSID              “WSC_1_0_0”
#define GO_INTENT_VALUE            16
#define OPER_CHANNEL              11
```

8.6.1.3 Client Mode with Enterprise Security

```
#define RSI_OPERMODE                 2
#define CLIENT_MODE                 DISABLE
#define P2P_MODE                   DISABLE
#define ENTERPRISE_MODE               ENABLE
#define AP_MODE                  DISABLE
#define EAP_METHOD                  “TTLS”
#define INNER_METHOD                 “\”auth=MSCHAPV2\””
#define USER_IDENTITY               “\”user1\””
#define PASSWORD                    “\”test123\””
#define KEY_PASSWORD                  “\”wifi\””
```

8.6.1.4 TCP/IP

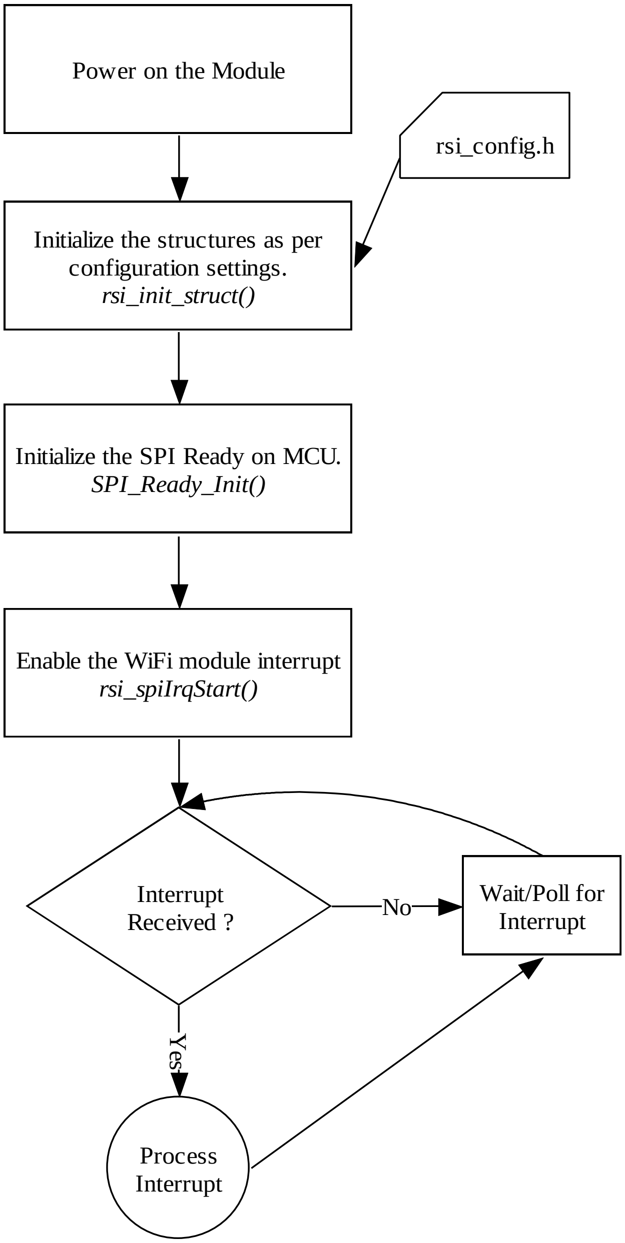
```
 #define RSI_IP_CFG_MODE              RSI_DHCP_MODE_ENABLE
#define RSI_MODULE_IP_ADDRESS                 “192.168.100.67”
#define RSI_NETMASK                          “255.255.255.0”
#define RSI_GATEWAY                          “192.168.100.1”
#define RSI_TARGET_IP_ADDRESS          “192.168.100.198”
#define RSI_SOCKET_TCP_CLIENT_TYPE     RSI_SOCKET_TCP_CLIENT
#define RSI_SOCKET_TCP_SERVER_TYPE     RSI_SOCKET_TCP_SERVER
#define RSI_SOCKET_UDP_CLIENT_TYPE     RSI_SOCKET_UDP_CLIENT
#define RSI_SOCKET_LUDP_TYPE           RSI_SOCKET_LUDP
#define RSI_MODULE_SOCKET_ONE                  25000
#define RSI_TARGET_SOCKET_ONE                  25000
```

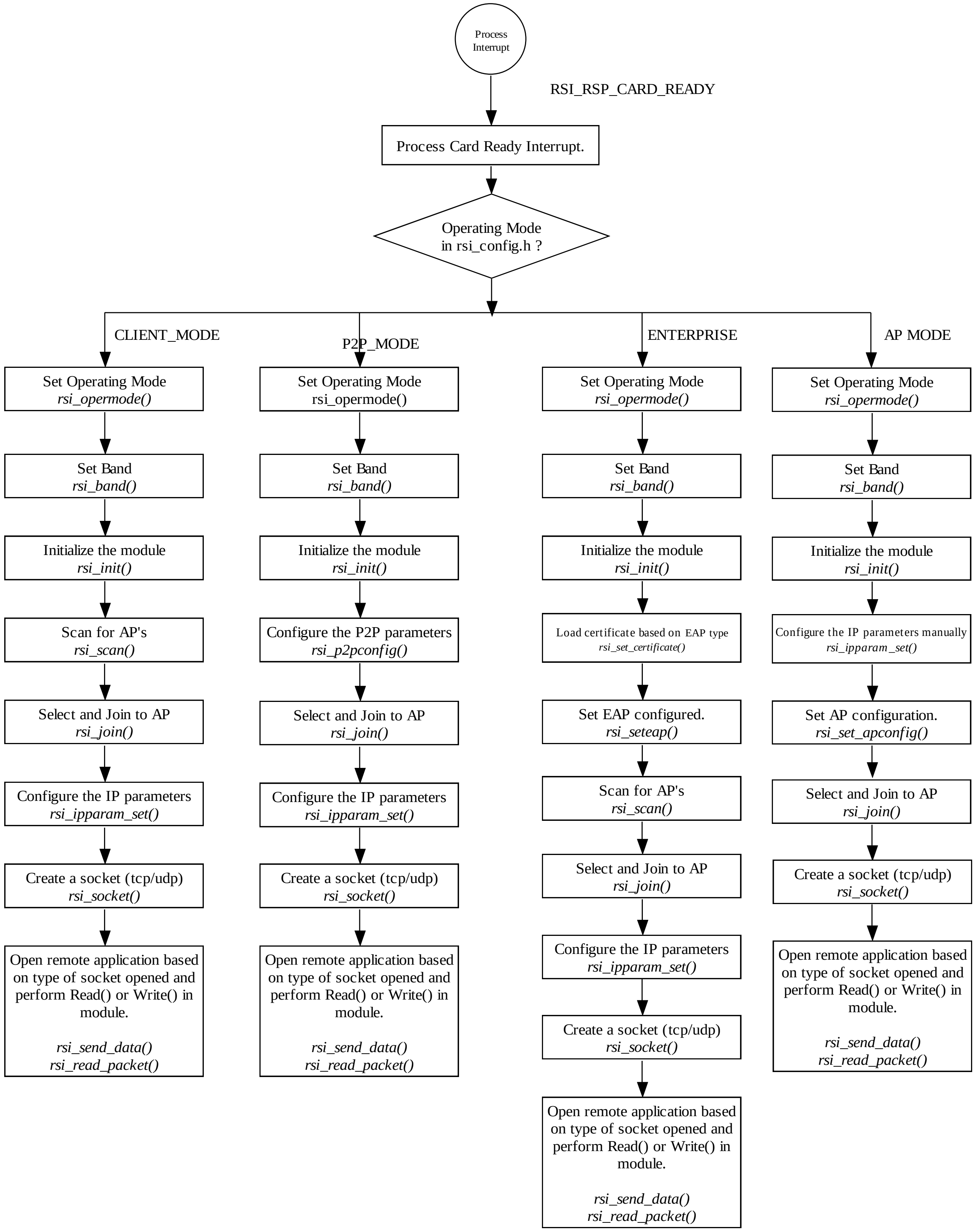
8.6.1.5 Configuring in AP mode

```
#define RSI_OPERMODE          6
#define CLIENT_MODE           DISABLE
#define P2P_MODE            DISABLE
#define ENTERPRISE_MODE         DISABLE
#define AP_MODE           ENABLE
#define RSI_BAND               RSI_BAND_2P4GHZ
#define RSI_PSK                   "12345678"
#define RSI_DTIM_PERIOD           4
#define RSI_BEACON_INTERVAL       100
#define RSI_AP_CHANNEL_NUM             1
#define RSI_SECURITY_TYPE       RSI_SECURITY_NONE
#define RSI_ENCRYPTION_TYPE     RSI_ENCRYPTION_NONE
#define MODE_FLAG               1
#define MAX_NO_OF_CLIENTS       4
```

8.6.2 Command Sequence

The figure below shows an example sequence of commands that need to be sent to the Wi-Fi module.


Power on the Module
rsi_config.h
Initialize the structures as per configuration settings.
rsi_init_struct()
Initialize the SPI Ready on MCU.
SPI_Ready_Init()
Enable the WiFi module interrupt
rsi_spiIrqStart()
Interrupt Received ?
No
Wait/Poll for Interrupt
Yes
Process Interrupt


Process Interrupt
RSI_RSP_CARD_READY
Process Card Ready Interrupt.
Operating Mode in rsi_config.h ?
CLIENT_MODE
P2P_MODE
ENTERPRISE
AP MODE
Set Operating Mode rsi_opermode()
Set Band rsi_band()
Initialize the module rsi_init()
Scan for AP's rsi_scan()
Select and Join to AP rsi_join()
Configure the IP parameters rsi_ipparam_set()
Create a socket (tcp/udp) rsi_socket()
Open remote application based on type of socket opened and perform Read() or Write() in module.
rsi_send_data()
rsi_read_packet()
Set Operating Mode rsi_opermode()
Set Band rsi_band()
Initialize the module rsi_init()
Configure the P2P parameters rsi_p2pconfig()
Select and Join to AP rsi_join()
Configure the IP parameters rsi_ipparam_set()
Create a socket (tcp/udp) rsi_socket()
Open remote application based on type of socket opened and perform Read() or Write() in module.
rsi_send_data()
rsi_read_packet()
Set Operating Mode rsi_opermode()
Set Band rsi_band()
Initialize the module rsi_init()
Load certificate based on EAP type rsi_set_certificate()
Set EAP configured. rsi_seteap()
Scan for AP's rsi_scan()
Select and Join to AP rsi_join()
Configure the IP parameters rsi_ipparam_set()
Create a socket (tcp/udp) rsi_socket()
Open remote application based on type of socket opened and perform Read() or Write() in module.
rsi_send_data()
rsi_read_packet()
Set Operating Mode rsi_opermode()
Set Band rsi_band()
Initialize the module rsi_init()
Configure the IP parameters manually rsi_ipparam_set()
Set AP configuration. rsi_set_apconfig()
Select and Join to AP rsi_join()
Create a socket (tcp/udp) rsi_socket()
Open remote application based on type of socket opened and perform Read() or Write() in module.
rsi_send_data()
rsi_read_packet()

8.6.3 Typical Usage of APIs

This section describes a typical sequence to call the APIs of the library in the application. The application has to first call the API for a command, then wait for a interrupt to occur and then service it. The application can perform other tasks during this wait period. Once the data pending interrupt event is received, the application has to call the rsi_spi_read_packet API to read the response from the module and parse the response and handle it appropriately.

8.6.4 Power mode API usage

int16 rsi_power_mode(uint8 mode) :

a. When called with mode value ‘1’ enables the power save. Here some part of the module goes to sleep.

   The upper layer will send a sleep message to host, by raising an interrupt. Once the host MCU receives the POWER_SAVE bit set in the status, it needs to send an ack message for the sleep request from the module. This can be done by calling rsi_pwrsave_continue() API from the host. The upper layer will then go to sleep. It will send a sleep request again after waking up through a timer timeout which is configured by the rsi_sleeptimer() API from the user.

   Once the sleep request came from the module, user can send an ack to make the upper layer to go to power save or can send the data/cmd to the module. After the user application completes sending of data, the user has to give the ack for the sleep message.

   Example Usage:

```
if(rsi_status & POWER_SAVE)
{
   sleep_received = 1;
}
 if(sleep_received == 1)
{
   rsi_pwrsave_continue();
   //giving ack to send the upper layer to sleep
        (OR)
   /* User can send the pending commands here,
   After giving all the commands, to send the module
   to complete sleep call rsi_pwrsave_continue() API */
}
```

b. When the power mode API is called with value ‘2’, then host can wakeup the module by toggling the wake up pin for the module. Once the module wakes up, then host can send the commands.

When host is trying to send the command or reading the response, first it waits for the ready signal to go low. If it is not low, then that means the module missed the wake up interrupt or toggle given from host. In this case host needs to do the wakeup again.

```
/* wait for ready signal to go low */
while(SPI_READY_VAL != RSI_FALSE)
{
     if(powermode == 2)  // only if power mode is ‘2’
     {
         rsi_delayMs(3); // have some delay of 3ms
         rsi_wakeup_from_host();     // wakeup pin toggle
     }
}
```

Once the module is wake up and it is ready to go to sleep again, it gives a sleep request to host. Now, host can give the power save confirm to put the module back to sleep.

```
if(rsi_status & POWER_SAVE)
{
   sleep_received = 1;
}
 if(sleep_received == 1)
{
   rsi_pwrsave_continue();
   //giving ack to send the upper layer to sleep
        (OR)
   /* User can send the pending commands here,
   After giving all the commands, to send the module
   to complete sleep call rsi_pwrsave_continue() API */
}
```

8.7 HTML Documentation

The index.html file inside RS.WSC.x.x.GENR.x.x.x.x.x.x\Resources\SPI\Driver\ \Documentation\html is the starting point for browsing the HTML based documentation.

9 Using the module in Different Operational Modes

The module can be configured in the following modes (

Set Operating Mode):

1. Wi-Fi Direct™ mode
2. Access Point Mode
3. Client Mode to connect to an AP in open mode or with Personal Security
4. Client mode to connect to an AP with Enterprise Security

9.1 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 RS-WC-201/301 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 RS-WC-201/301 Wi-Fi Direct network.In this mode, the module connects to a Wi-Fi direct node 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* (Configure Wi-Fi Direct Peer-to-P). The node with a higher value of *Group_Owner_intent* would get preference over a lower value in becoming a GO. If the values advertized 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.

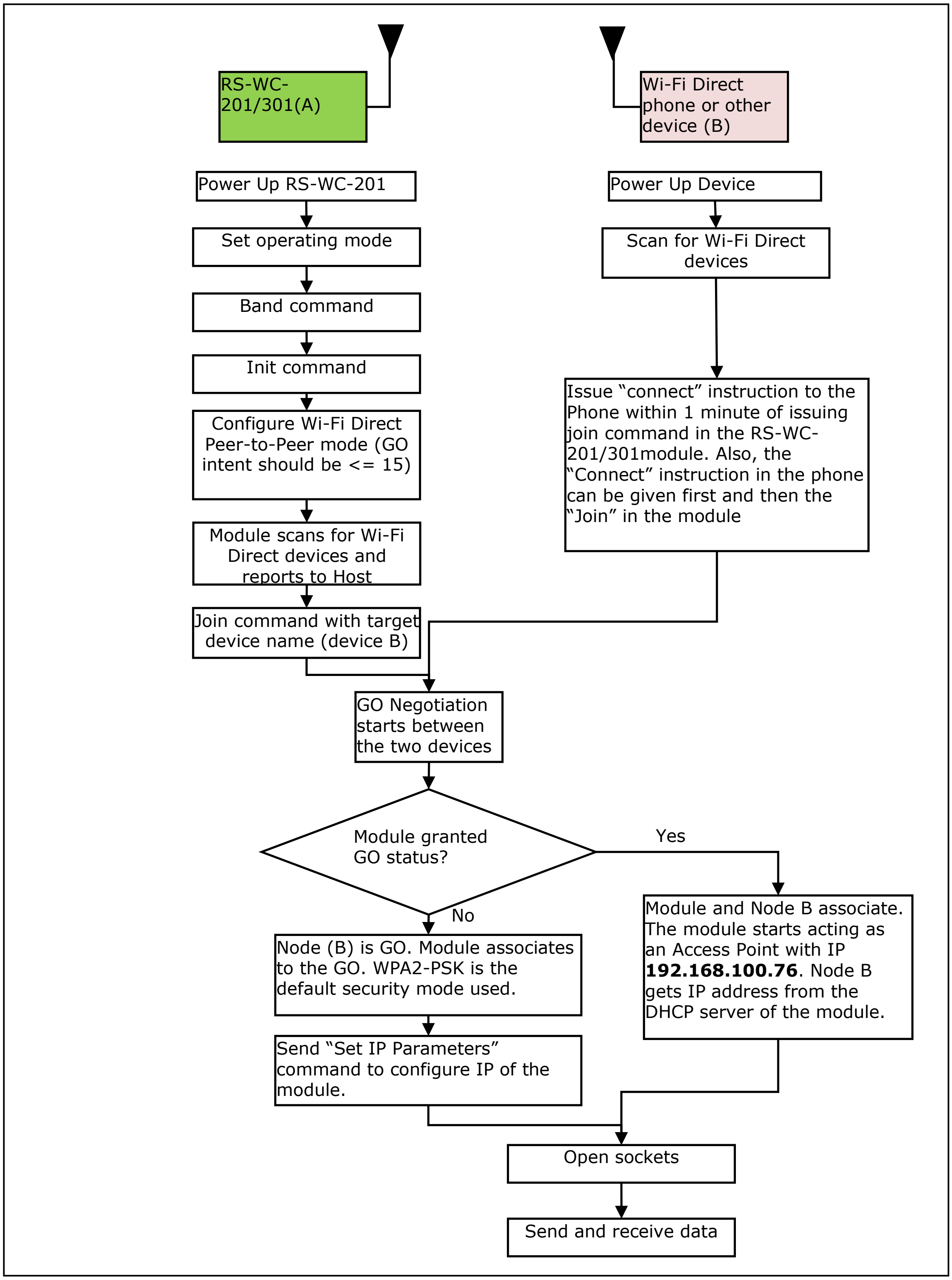

RS-WC-201/301(A)
Wi-Fi Direct phone or other device (B)
Power Up RS-WC-201
Set operating mode
Band command
Init command
Configure Wi-Fi Direct Peer-to-Peer mode (GO intent should be <= 15)
Module scans for Wi-Fi Direct devices and reports to Host
Join command with target device name (device B)
Power Up Device
Scan for Wi-Fi Direct devices
Issue "connect" instruction to the Phone within 1 minute of issuing join command in the RS-WC-201/301module. Also, the "Connect" instruction in the phone can be given first and then the "Join" in the module
GO Negotiation starts between the two devices
Module granted GO status?
Yes
No
Node (B) is GO. Module associates to the GO. WPA2-PSK is the default security mode used.
Send "Set IP Parameters" command to configure IP of the module.
Module and Node B associate. The module starts acting as an Access Point with IP 192.168.100.76. Node B gets IP address from the DHCP server of the module.
Open sockets
Send and receive data

Figure 29: Wi-Fi Direct Peer-to-Peer Mode

9.2 Access Point Mode

The following sequence of commands should be used to create an Access Point in the module. The module can support four external clients when it is configured in Access Point mode. The module acts as a DHCP server. The module cannot act simultaneously as an Access Point and a client.

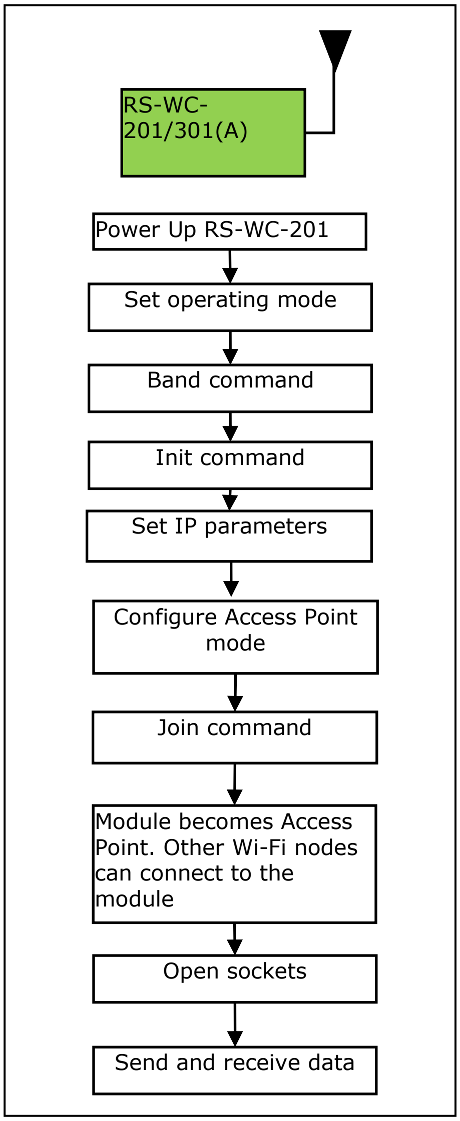


Figure 30: Access Point Mode

9.3 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.

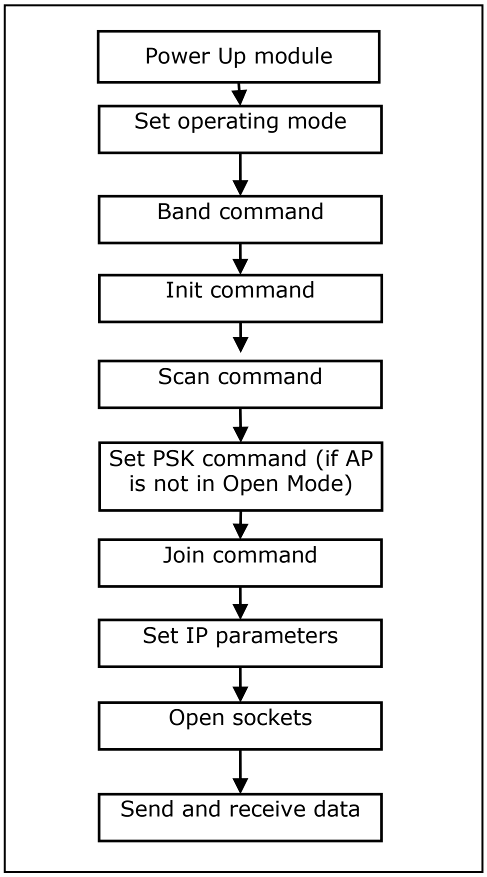


Figure 31: Client Mode with Personal Security

9.4 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.

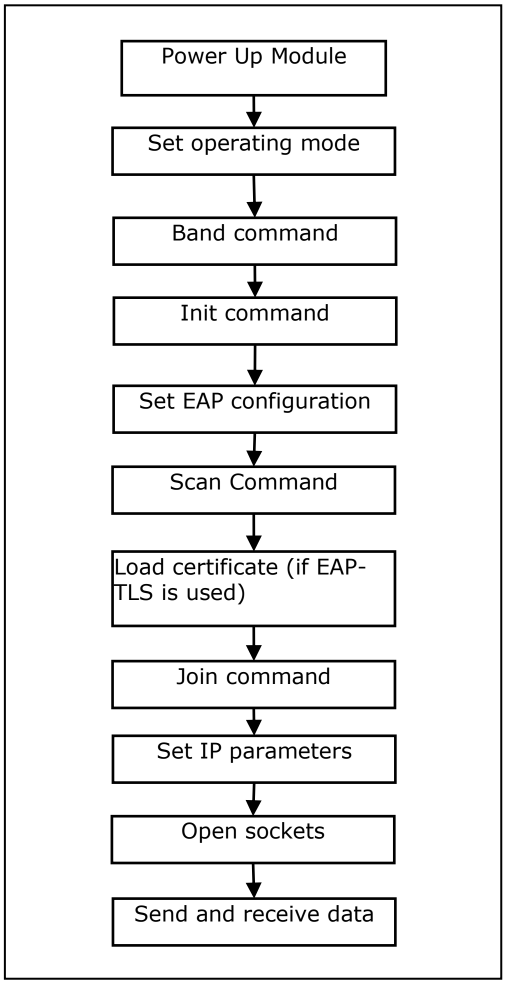


Figure 32: Client Mode with Enterprise Security

10 Wireless Firmware Upgrade

The firmware of the module can be upgraded wirelessly. This feature is available from firmware version 2.1.0.1.2.5 onwards. The following sections describe the process.

10.1 Users of Older Firmware

The user should first upgrade to version 2.1.0.1.2.5 to use the feature of wireless firmware upgrade. Refer to the section Upgrading Firmware Through the UART Interface.

10.2 Upgrading Firmware Wirelessly

If the module already has firmware version 2.1.0.1.2.5 or above in the module, this section should be followed. To upgrade the firmware of the module, pins WF_HNDSHKE1 (pin #13 in RS-WC-201 and #92 in RX-WC-301) and WF_HNDSHKE2 (pin #14 in RS-WC-201 and pin #90 in RS-WC-301) should be connected to corresponding GPIO pins of the Host. WF_HNDSHKE1 is an input to the module; WF_HNDSHKE2 is an output from the module.

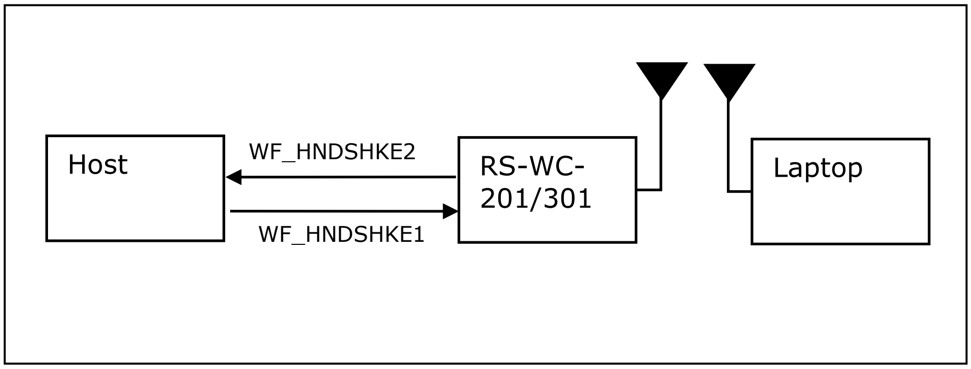
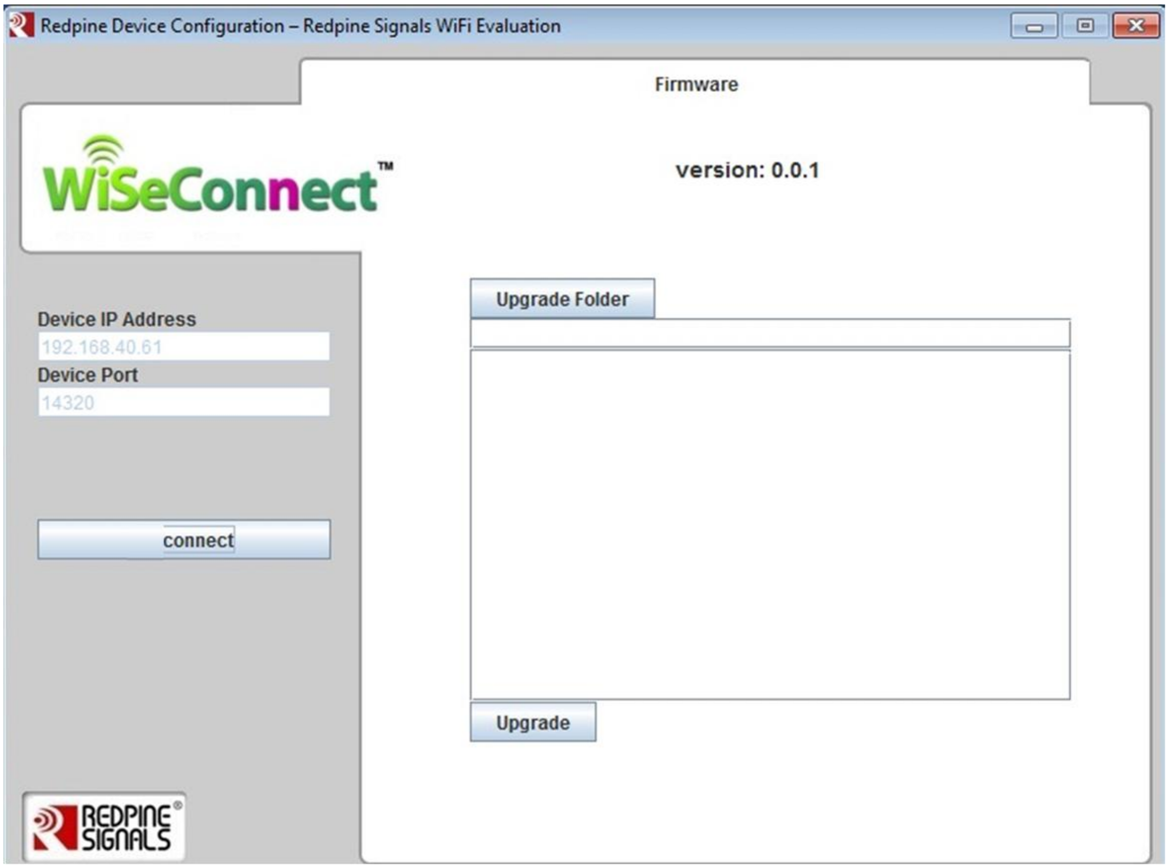


Figure 33: Set-up for Wireless Firmware Upgrade

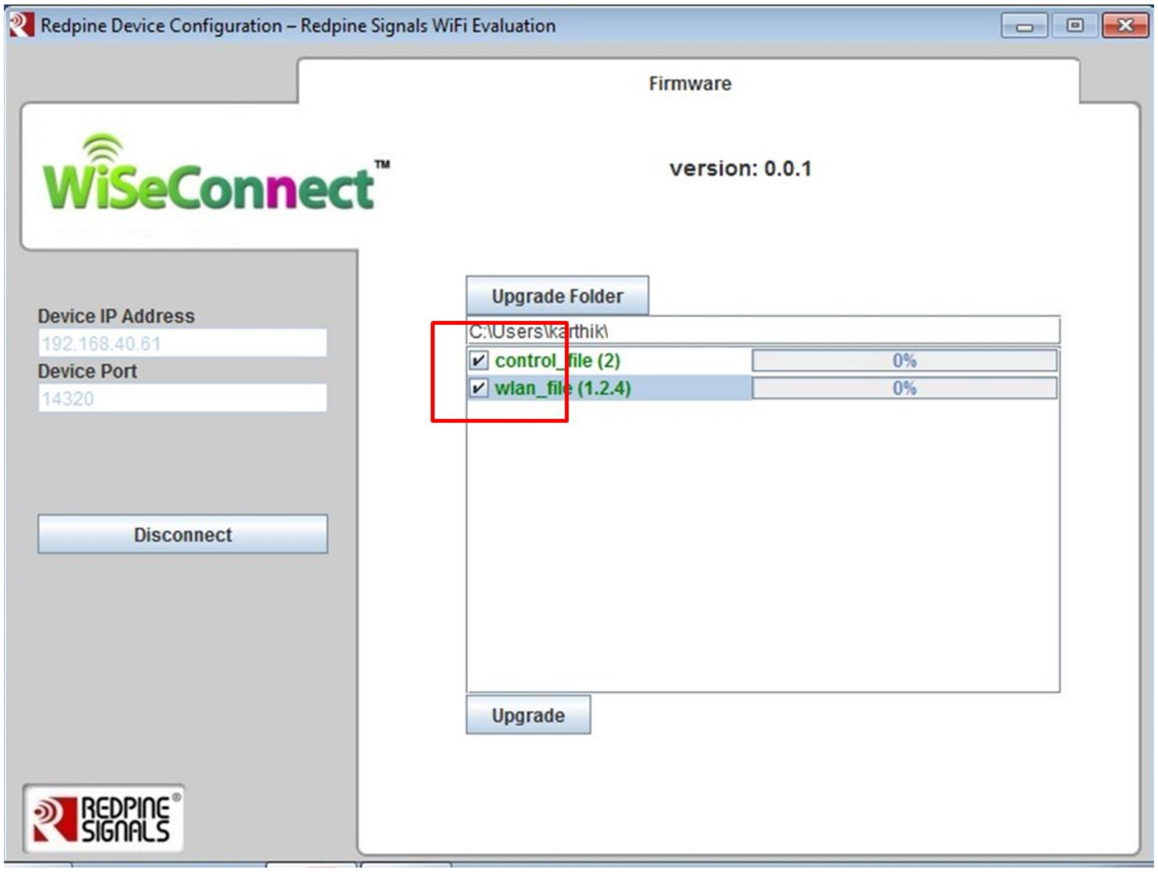
The steps are mentioned below:

1. Set WF_HNDSHKE1 to logic '1' from the Host
2. Power up the module
3. Module boots up and comes up as an Access Point in open mode with with SSID REDPINE_<MAC> where <MAC> are the last 3 bytes of the MAC address of the module. The default IP address of the module is 192.168.40.61.
4. Connect a Laptop to the Access Point
5. Open the application RS.WSC.x.x.GENR.x.x.x.x.x.x\Firmware\WFU WiSeConfigGUI.exe in the Laptop



Device IP Address
192.168.40.61
Device Port
14320

6. Click on the “Connect” button. Then click on the button “Upgrade Folder”. Select the files RS.WSC.x.x.GENR.x.x.x.x.x.x\Firmware\WFU\control_file.rps and RS.WSC.x.x.GENR.x.x.x.x.x.x\Firmware\WFU\wlan_file.rps in the window that comes up.


Redpine Device Configuration – Redpine Signals WiFi Evaluation
Firmware
WiSeConnect™
Upgrade Folder
Disconnect
Upgrade

version: 0.0.1

Device IP Address: 192.168.40.61

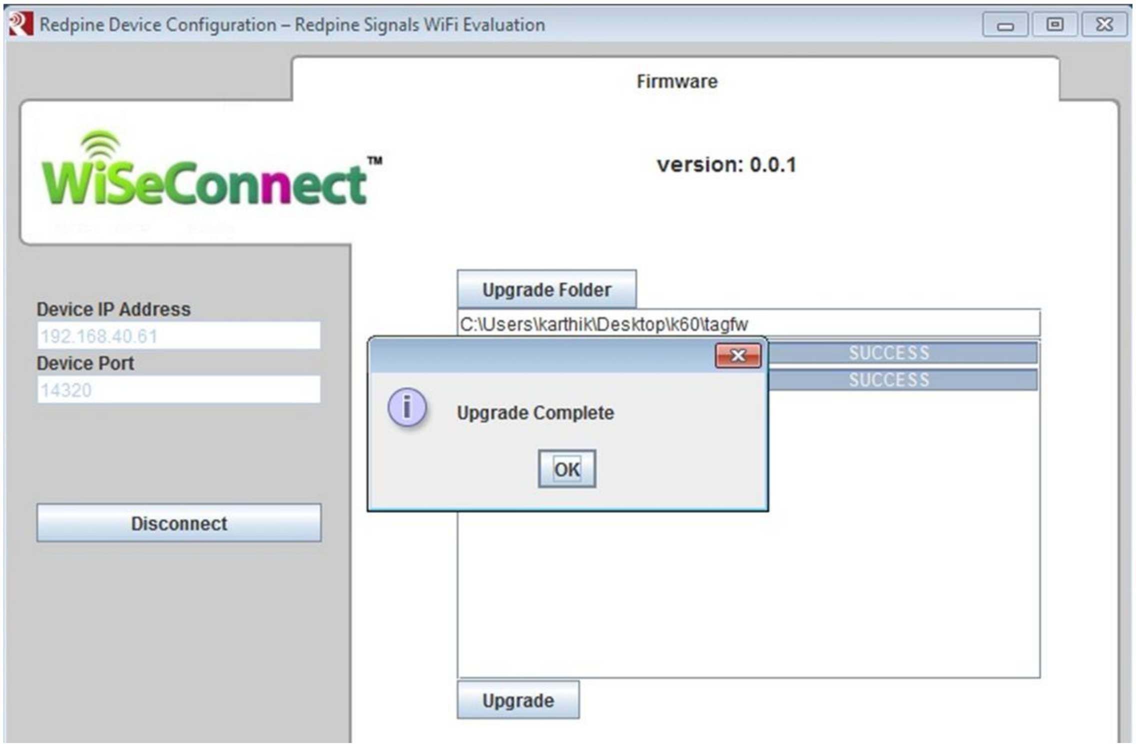
Device Port: 14320

C:\Users\karthik\

☑ control_file (2) 0%

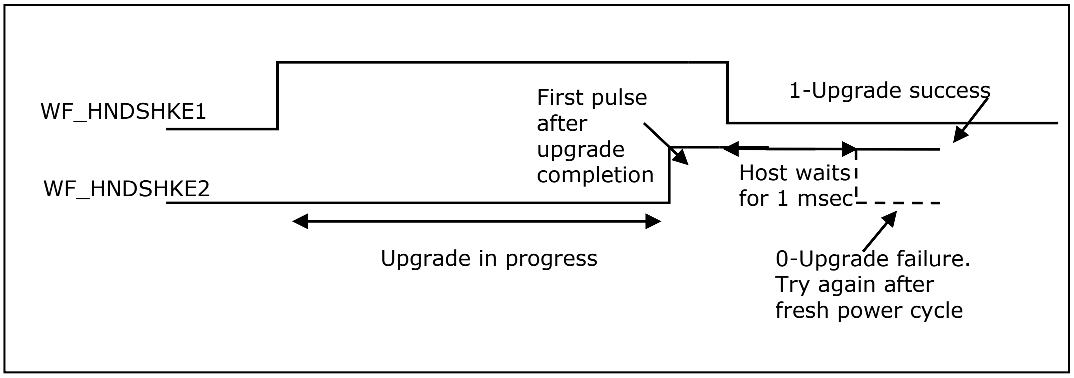
☑ wlan_file (1.2.4) 0%

7. Tick the check boxes and click on the “Upgrade” button. The progress of the upgrade is shown in the progress bars. After the files are transferred, it may take up to 1 min for the final upgrade confirmation to come in as shown below.



Redpine Device Configuration – Redpine Signals WiFi Evaluation

Meanwhile, the Host can keep polling the signal WF_HNDSHKE2, and the moment it goes high, WF_HNDSHKE1 should be pulled low to complete the upgrade process. After the upgrade process is over, the module will set the signal to high and will retain high or transition it to low depending on whether the upgrade was successful, after a delay of 1 msec after WF_HNDSHKE1 had been pulled low. This process would confirm the final upgrade status to the Host. The module can now be power cycled for normal operation (WF_HNDSHKE1 should be kept '0' in normal operation)



WF_HNDSHKE1
WF_HNDSHKE2
First pulse after upgrade completion
1-Upgrade success
Host waits for 1 msec
Upgrade in progress
0-Upgrade failure. Try again after fresh power cycle

Figure 34: Signal Status During Firmware Upgrade

After the confirmation, the module should be power cycled and operated normally thereafter.

NOTE:

1. If a user is using firmware version 2.1.0.1.2.5 or above in the module, during normal operation of the module the pin WF_HNDSHKE1 should be set to '0'. The ONLY scenario where this signal should be set to '1' is when the user wants to upgrade the module's firmware wirelessly.

The feature of Wireless Firmware Upgrade is not dependent on the interface (UART/SPI/USB).

The file RS.WSC.x.x.GENR.x.x.x.x.x.x\Firmware\WFU\control_file.rps is exactly same as RS.WSC.x.x.GENR.x.x.x.x.x.x\Firmware\WiSe_Control.S19. The former is in a different format and is used to wirelessly upgrade the firmware, while the latter is used to upgrade the firmware using the UART interface. Same is the case with RS.WSC.x.x.GENR.x.x.x.x.x.x\Firmware\WFU\wlan_file.rps and RS.WSC.x.x.GENR.x.x.x.x.x.x\Firmware\Wise_WLAN.S19

2. It is recommended to upgrade all the files in a release package through UART firmware upgrade utility before upgrading the module with any other firmware through wireless.

11 APPENDIX A: Sample Flow of Commands in UART

Sample command sequences are shown below for operating the module in different modes.

Operate in Wi-Fi Direct Mode to associate to a WiFi Direct Phone

at+rsi_opermode=1\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=15,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 highest value of 15 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

In case, any P2P device initiates the connection to the module, host will receive asynchronous AT+RSI_CONNECT message. This happens usually when the “redpine” P2P device is selected from the available scanned P2P devices list and clicks on “Connect” in WiFi Direct phone.

at+rsi_join=AndroidP2P,0,2\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 module has become a Group Owner. The IP address of the module would be 192.168.100.76. The phone will acquire an IP address from the module. 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\r\n

opens a server TCP socket inside the module with port number 5001.

A client socket at the remote node (phone) 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=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\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\r\n

opens a server TCP socket inside the module with port number 5001.

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.

<u>Associate to an Access Point (with WPA2-PSK security) as a client</u>

at+rsi_opermode=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=12345678\r\n

This command configures the PSK to be used to associate to the Access Point.

at+rsi_join=Test_AP,0,2\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\r\n

This command opens a server TCP socket inside the module with port number 5001.

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\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\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\r\n

This command opens a server TCP socket inside the module with port number 5001.

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\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=WPS_SSID,0,2\r\n

This command associates the module to the AP using WPS push button method. Note that WPS_SSID is a constant string and not the SSID of 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\r\n

This command opens a server TCP socket inside the module with port number 5001.

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\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=TLS,cert_len,key_password,<path of certificate file>\r\n

This command provides the TLS certificate to the module.

at+rsi_join=Test_AP,0,2\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\r\n

This command opens a server TCP socket inside the module with port number 5001.

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.

<u>Operate in Wi-Fi Direct Mode as an autonomous GO</u>

at+rsi_opermode=1\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=16,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 value of 16 in this case. The module responds with "OK".

at+rsi_join=directrp,0,2\r\n

This command makes the module to become an autonomous group owner. The IP address of the module would be 192.168.100.76. After this any once can connect to module in legacy client or P2P mode. The phone/device which is connected to module will acquire an IP address from the module. A ping can be issued from the phone/device 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\r\n

opens a server TCP socket inside the module with port number 5001.

A client socket at the remote node (phone) 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=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.

12 APPENDIX B: Sample Flow of Commands in 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. 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 highest value of 15 in this case.The module responds with sucess.
After issuing this command, the module starts scanning for Wi-Fi Direct devices, and reports any that are found through the asynchronous message

In case, any P2P device initiates the connection to the module, host will receive asynchronous AT+RSI_CONNECT message. This happens usually when the "redpine" P2P device is selected from the available scanned P2P devices list and clicks on "Connect" in WiFi Direct phone.

Join

This command initiates the association operation between the module and the Wi-Fi Direct phone.

Assuming that the module has become a Group owner and the phone has acquired an IP,

Open Socket and transfer data

Operate in Wi-Fi Direct Mode to become autonomous GO

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. GO Intent is 16.

Join

This command initiates the autonomous GO. SSID parameter in join command is same as P2P device name given in configure Wi-Fi Direct command.

Assuming that the module has become a Group owner and the phone has acquired an IP,

Open Socket and transfer data

<u>Create an Access Point</u>

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 WPA2-PSK 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.

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”

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 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 WPS_SSID is a constant string to be used for the SSID parameter.

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.

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"

Set Certificate

This command provides the TLS certificate to the module.

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

Set Operating Mode=3

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

Create an Access Point, with TCP/IP stack bypassed in the module

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 AP Mode

This command will configure the parameters of the AP.

Join

This command will create the Access Point.

Configure Network Interface at the Host

Refer to the note in section Set Operating Mode.

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.