RS-WC-201/301

Software Programming Reference Manual

Version 2.01

August 2012

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
1.8	Corrected the last "More Chunks" parameter value in SPI mode in Figure 17. AT_RSI_READ corrected to AT+RSI_READ
1.9	Added commands for DNS, HTTP GET and POST, soft reset, Query Group Owner Parameter, additional Modes 2 and 3 in Power save, 802.11 h and 802.11d. Added option for bypassing TCP/IP stack in SPI mode
2.00	Response ID for "Set Region" SPI command added
2.01	Corrected document formatting issues leading to error in allowed values of parameters in the following commands UART mode: at+rsi_dnsget- Primary or Secondary DNS server at+rsi_setregion- region_code SPI Mode: Set operating mode- operMode Antenna Selection- Antennaval Configure WiFi Direct P2P mode- devState Scan- securityMode, uNetworkType Join- powerLevel Set certificate- cert_type Set IP parameters- dhcpMode Open a socket- socketType Query WLAN Connection Status- state Query Network parameters- wlan_state, Sec_type, dhcp_mode, connection_type, socket_type HTTP Get- more

	HTTP Post- more rsi_spi_opermode.c- mode rsi_spi_dns_server.c- dnsmode rsi_spi_set_region.c- WiFi_region rsi_spi_socket.c- socketType rsi_spi_dns_server.c- DNSmode

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. It also describes the flow of commands to be used to configure the modules into specific functionality. 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 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.

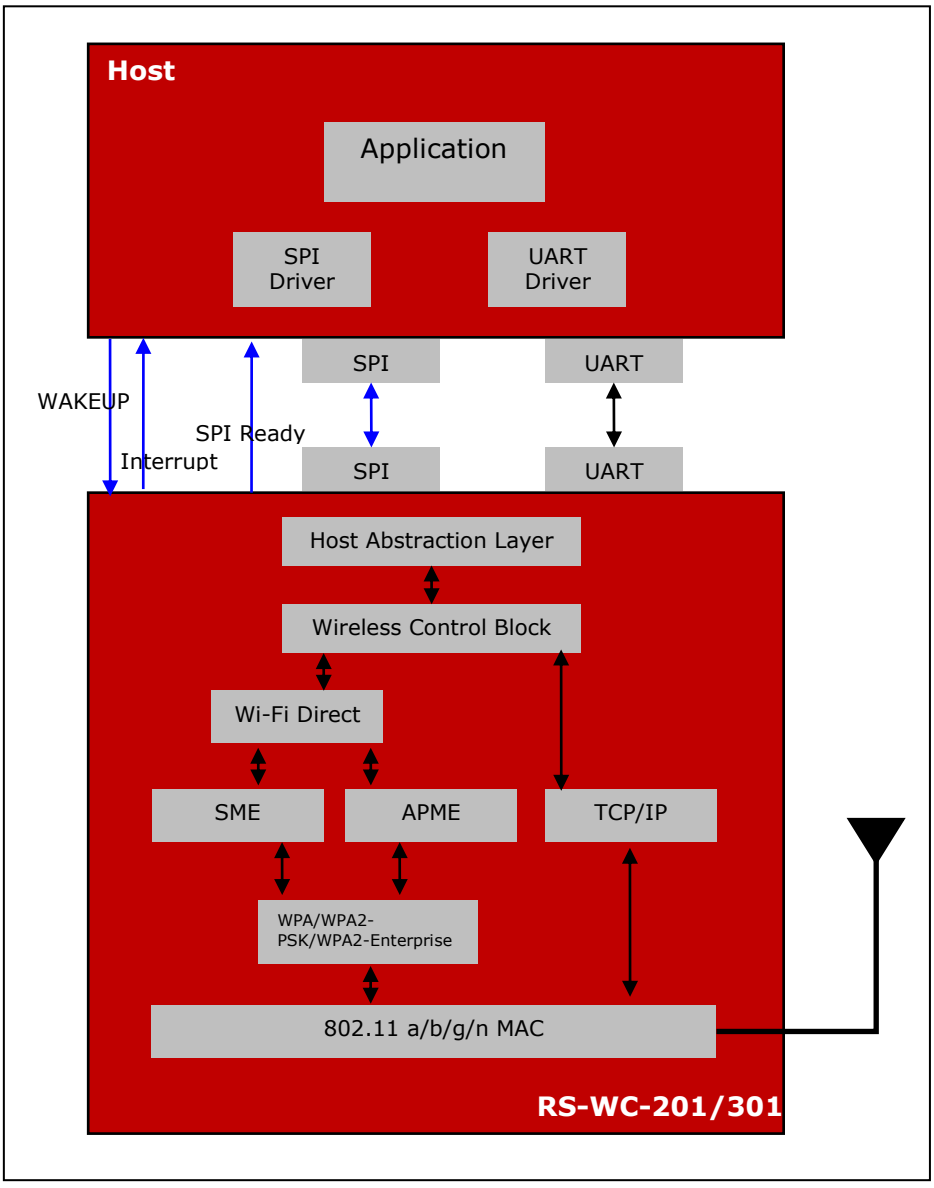


Figure 1:RS-WC-201/301 Software Architecture

The RS-WC-201/301 module is integrated with the Host using either UART or SPI interfaces. The transmission and reception of the data to/from the Host depends on the interface used.

UART mode:

The Host transmits/receives data using UART interface and AT command set when the RS-WC-201/301 module is configured for UART mode.

SPI mode:

Host transmits/receives data using SPI interface when the RS-WC-201/301 module is configured for SPI mode. A driver on the Host takes care of interacting with the Wi-Fi module through the SPI interface.

2.1 Host

The Host is any system that has applications being executed and either a UART or SPI interface to connect to the RS-WC-201/301 module.

2.1.1 UART

The UART on the Host side provides an interface for the Host to access the Wi-Fi module. UART is used to configure various parameters of the RS-WC-201/301 module and also to send and receive data.

2.1.2 SPI

The SPI on the Host side provides an interface for the Host to access the Wi-Fi module. SPI on the Host acts as the master.

The SPI driver on the Host is a driver through which the applications interact with the Wi-Fi module to send/receive the data to/from the RS-WC-201/301 module and also to configure the module over the SPI interface.

2.2 RS-WC-201/301

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

2.2.1 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.

2.2.2 UART

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

2.2.3 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.

2.2.4 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.

2.2.5 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.

2.2.6 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.

2.2.7 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.

2.2.8 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.

2.2.9 WPA 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.

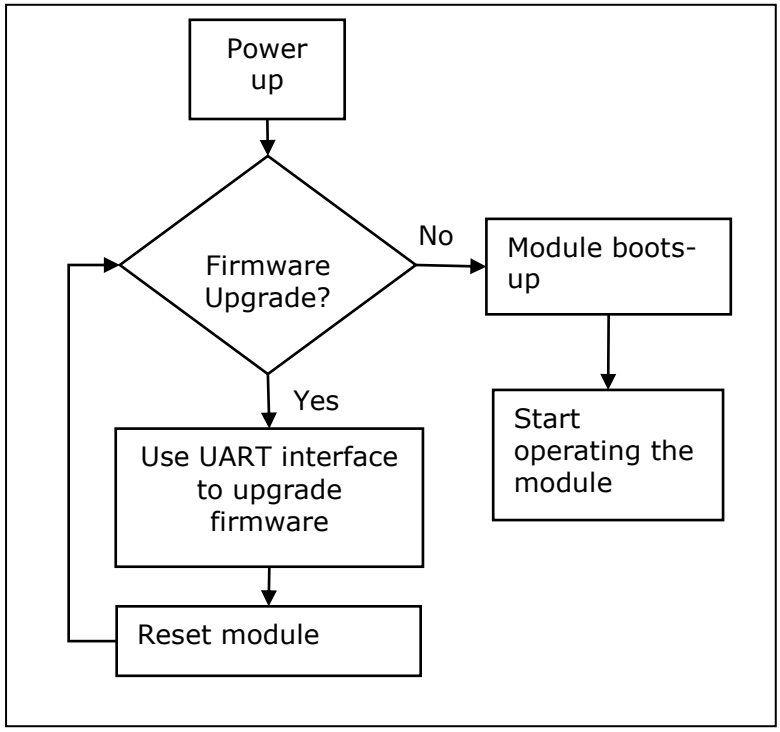


Figure 2: Firmware Upgrade and General Operation in UART modules

RS-WC-201 and RS-WC-301 modules use the following UART interface configuration for communication:

Baud Rate: 115200

Data bits: 8

Parity: None

Stop bits: 2

Flow control: None

3.1 Messages on Power-up

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
2. The module sends the message "Welcome to WiSeConnect" to the Host and then starts boot-up:

[0xFC 0xFC 0xFC 0xFC 0xFF Welcome to WiSeConnect\r\n]

……..

0xFC 0xFC 0xFC 0xFC 0xFF 0x57 0x65 0x6C 0x63 0x6F 0x6D 0x65 0x20 0x74 0x6F 0x20 0x57 0x69 0x53 0x65 0x43 0x6F 0x6E 0x6E 0x65 0x63 0x74 0x0D 0x0A

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.

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: 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>, returned in two's complement, hex format.

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: 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™** 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 *at+rsi_wfd*. 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.

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.

Note: This format of command and the corresponding byte stream under the dotted line is followed in all examples.

Response

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

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

Relevance

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

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

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 and 2.

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

Relevance

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

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

Description

This command is used to set the configuration information for Operating mode 1. There are two sub-modes inside Operating Mode 1: Wi-Fi Direct Mode and AP mode. In 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**

Command

From Host to Module

at+rsi_wfd

Asynchronous Message from Module to Host

AT+RSI_WFDDEV (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

Asynchronous Message from Module to Host

AT+RSI_WFDDEV=<device_state><device_name><device_mac><device_type
>

Parameters

From Host to Module

Group_Owner_intent (maximum of 2 bytes, ASCII): The interpretations of this parameter for the WiFi Direct and AP modes are described below.

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 15. Higher the number, higher is the willingness of the module to become a GO.

Access Point Mode: If the module needs to be configured as an Access Point with WPA2-PSK security, then a value of 16 should be used.

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 or Access Point. 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 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].

This SSID (along with the DIRECT-xy prefix) also appears if the module is configured in Access Point mode.

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 (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 or an Access Point. WPA2-PSK security mode is used in the module in GO or AP mode.

Response

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

Asynchronous Message AT+RSI_WFDDEV from Module to Host

device_state (1-byte, hex).

0x00– The remote Wi-Fi Direct node was found in a previous scan iteration

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.

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

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

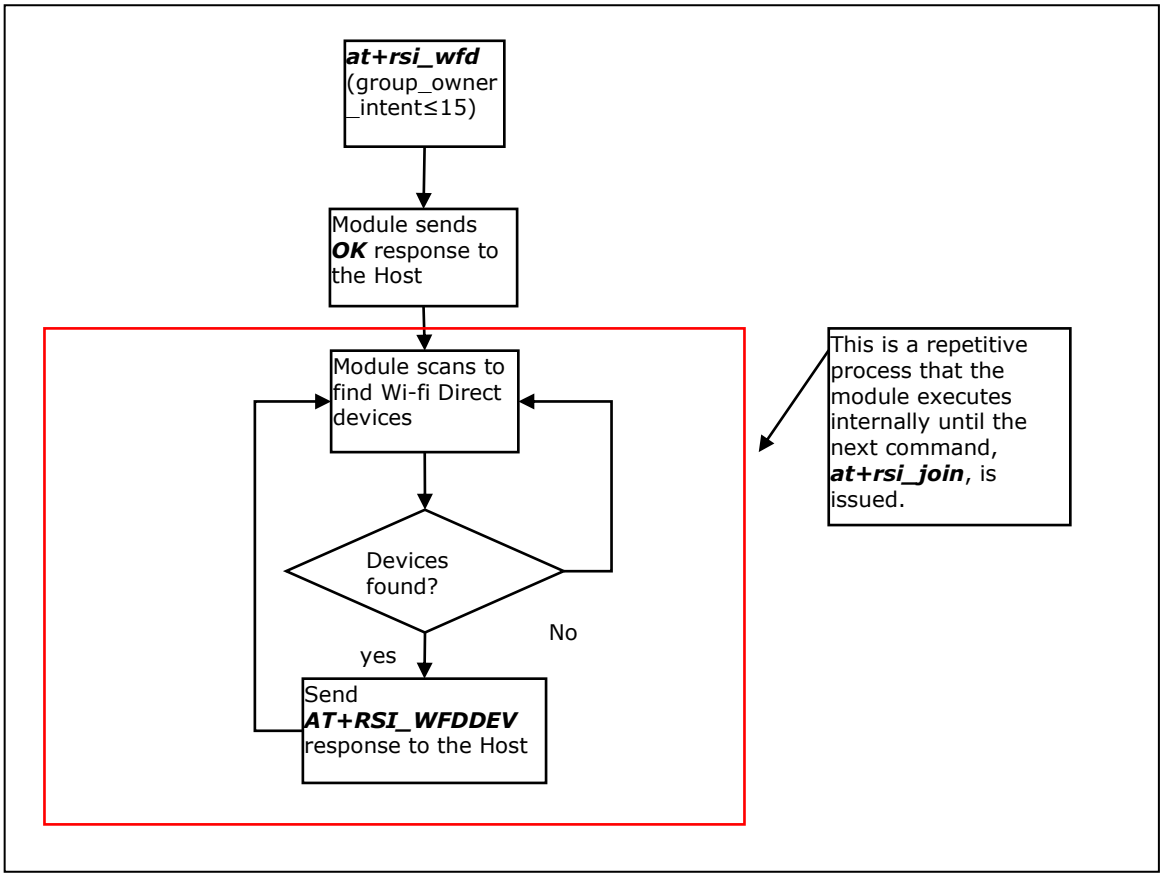


Figure 3: Operation after issuing at+rsi_wfd command

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,

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

AT+RSI_WFDDEV=1 wi_fi_phone 0x00 0x23 0x12 0x13 0x14 0x16 0x0A 0x04 0x0D 0x0A\r\n

……..

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 0x01 0x02 0x03 0x04 0x05 0x06 0x0A 0x04 0x0D 0x0A

Relevance

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

3.2.6 Scan

Description

This command makes the module scan for Access Points and gives the scan results to the host.

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

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

44	3
48	4
52	5
56	6
60	7
64	8
100	9
104	10
108	11
112	12
116	13
120	14
124	15
128	16
132	17
136	18
140	19
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.

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><Channel2><SecurityMode2><RSSIVal2><Network type2><Device Name2><BSSID2><Reserved2>……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):

Result Code	Description
	Channel number of the Access Point. *SecurityMode* (1 byte, hex): 0x00– open 0x01- WPA 0x02- WPA2 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 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. 32 byte stream, filler bytes (0x00) are put to complete 32 bytes, if actual length is not 32 bytes. *BSSID* (6 bytes, hex): BSSID of the scanned access point *Reserved* (2 bytes, hex): All '0'
ERROR<Error code>	Failure

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 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 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 and 2.

3.2.7 Join

Description

This command is used for following:

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

Associate to a remote device in WiFi Direct mode (operating mode 1)

Create an Access Point (operating mode 1)

Allow a third party to associate to a WiFi 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 *GO_Intent* parameter in *at+rsi_wfd* command is set to 16 to make the module an Access Point, this parameter becomes a part of the AP's SSID (Refer "Create an Access Point" in APPENDIX A: Sample Flow of Commands in UART).

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
MCS0	13

Data Rate (Mbps)	Value of uTxDataRate
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:

0– Low power (7+/-1 dBm)

1– Medium power (10 +/-1 dBm)

2– High power (15 +/- 2 dBm)

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

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.

Result Code	Description
	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 (ASCII "G") – if the module becomes a Group Owner (GO) after the GO negotiation stage. 0x43 (ASCII "C") – 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.
ERROR<Error code>	Failure

Relevance

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

When the module is in WiFi Direct Mode within 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.

3.2.8 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

Relevance

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

3.2.9 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

3–Power Save Mode 3

Response

Result Code	Description

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 0 or 2.

3.2.9.1 Power Save Operation

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

3.2.9.1.1 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 SLP\r\n (0x53 0x4C 0x50 0x0D 0x0A) to the Host. If the Host has any data to transmit, it should execute corresponding commands. Once that is done, the module can be put back to sleep by sending the string ACK\r\n.

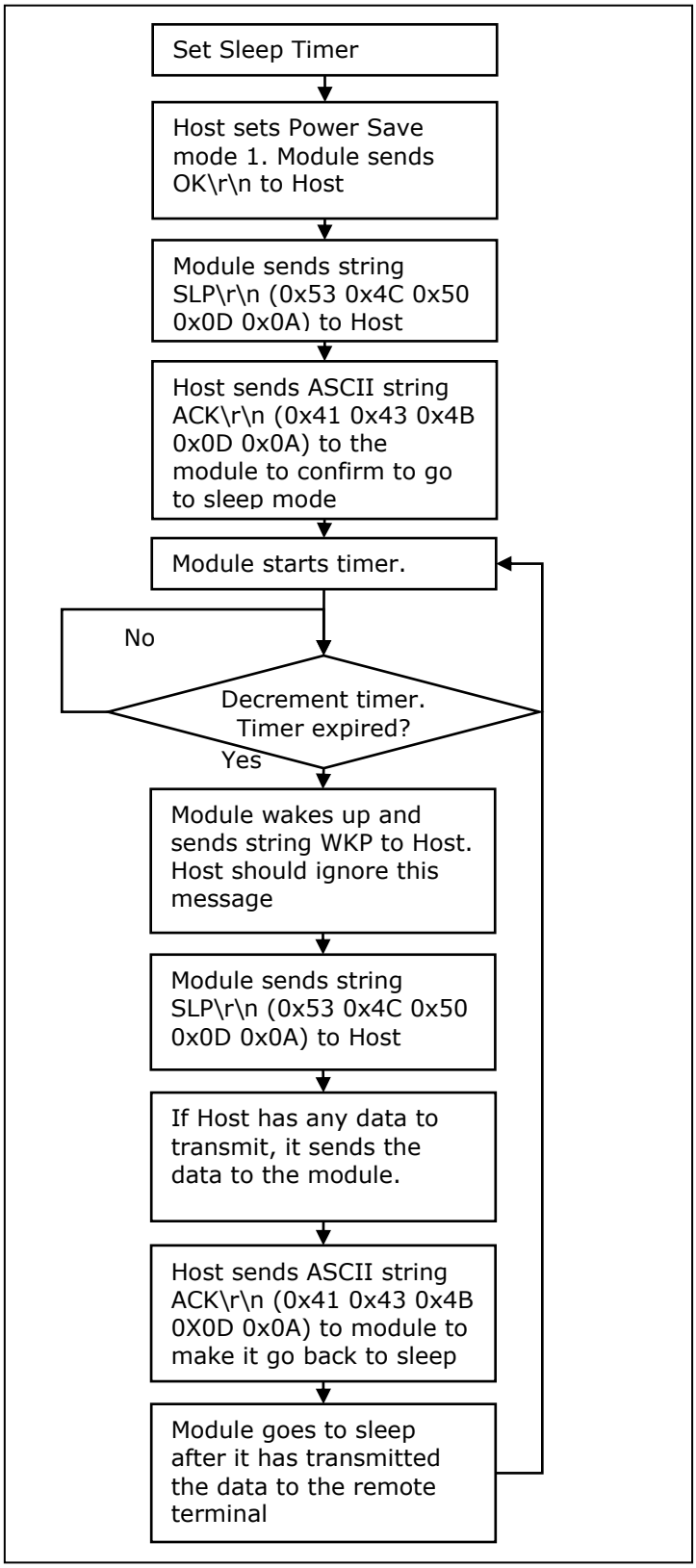


Figure 4: Setting Power Save Mode 1

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.

3.2.9.1.2 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. The Host should give commands to operate the module only when it is in awake state.

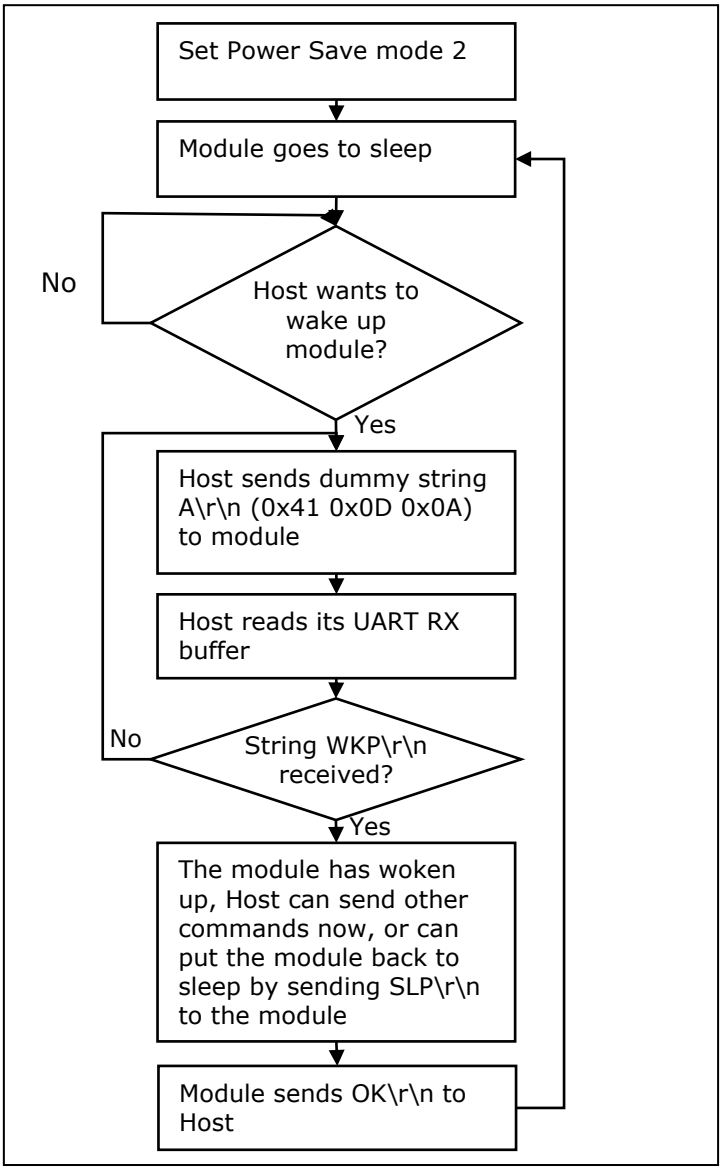


Figure 5: Power Save Mode 2

3.2.9.1.3 Power Save Mode 3

Once the module is put to power save mode 3, it can be woken up by the Host. The Host needs to send a dummy character in its UART TX line. This activity wakes up the module. For example, a string that can be sent from the Host is A\r\n. Once the module wakes up, it sends an ASCII string OK to the Host to indicate that it has woken up. The Host should give commands right from the first command "Set Operating Mode" to configure the module. All previous configuration information (such as band, Access Point it was previously connected to etc.) is lost after the module goes to sleep in this mode.

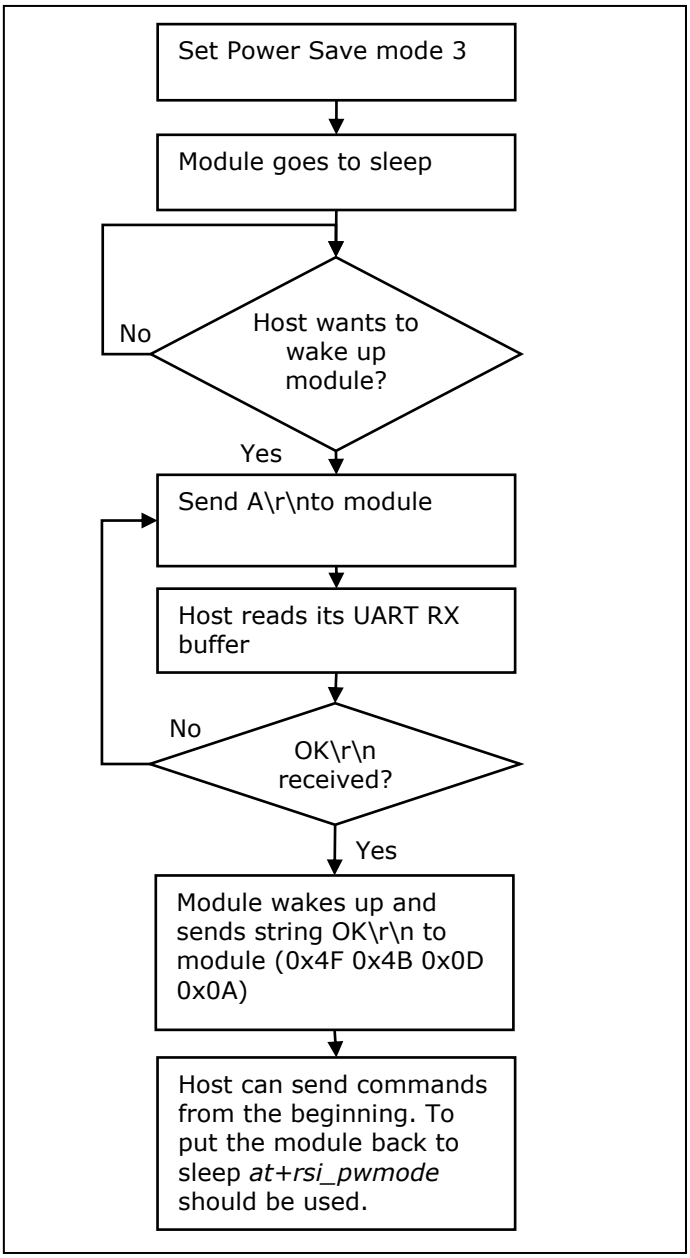


Figure 6: Power Save Mode 3

Power consumption is the lowest in Power Save Mode 3.

3.2.10 Pre Shared Key

Description

This command configures the PSK (Pre Shared Key) that is used for creating secured access. The PSK is used for WPA/WPA2-PSK. This command should be issued to the module before the *Join* command if the Access Point is in secure mode. This command can be ignored if the Access Point 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

Example 1: 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

Relevance

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

3.2.11 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

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.

3.2.12 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

key_password (): Private key password, used to generate the certificate

certificate: TLS certificate / PAC file in ASCII format. 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)
```

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.

3.2.13 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.

Command

at+rsi_disassoc

Usage

at+rsi_disassoc\r\n

Parameters

No Parameters

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 0, 2 and 1 (when module is not the GO).

3.2.14 Set IP Parameters

Description

This command configures the IP address, subnet mask and default gateway for the module in client mode. The command can also be used to configure the IP address of module in Access Point mode within Operating Mode 1. Refer "Creating an Access Point" in APPENDIX A: Sample Flow of Commands in UART.

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.

Input parameter delimiter

The input parameters are delimited by commas (0x2C in ASCII format).

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

Result Code	Description
	Subnet_Mask (4 Bytes, hex): Subnet mask *Gateway*(4 Bytes, hex): Subnet mask
ERROR<Error code>	Failure

Relevance

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

3.2.15Open 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 successfully opening and connecting the TCP socket to the Host port, a socket handle is returned. The least significant byte is returned first.

Result Code	Description
	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

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 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 or 2.

3.2.16 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).
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)
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. 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

Result Code	Description
	the socket in the module. *Module_ipaddr* (4 bytes, hex): Module's IP address.
ERROR<Error code>	Failure

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 or 2.

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.

3.2.17 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

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 or 2.

3.2.18Open 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. 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

Result Code	Description
	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

Input parameter delimiter

The input parameters are delimited by commas (0x2C in ASCII format) in this command.

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 or 2.

3.2.19 Query a Listening Socket's Active Connection Status

Description

This command retrieves handles of active socket connection established with a listening TCP socket in the module. This command can be issued when a listening/server TCP socket has been opened in the module, to get information about the remote peer's client socket.

Command

at+rsi_ctcp

Usage

at+rsi_ctcp=socket_handle\r\n

Parameters

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

Response

Result Code	Description
OK<*socket_handle*><*IP_Address*><*Port*>	*socket_handle* (1 byte, hex): Socket handle of an active socket, returned as two's complement. The least significant byte is sent first. *Socket_handle* >=0: A handle to an active connected socket. For example, if *socket_handle* = 7 (0x0007), it is returned as 0x07 0x00 *IP_Address* (4 bytes, hex): Client/remote peer's IP address. *Port* (2 bytes, hex): Port number on the client/remote peer. The least significant byte is returned first. For example, if the port number is 8001 (0x1F41), then first 0x41 is returned, then 0x1F
ERROR<Error code>	Failure

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 or 2.

3.2.20 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

Relevance

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

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 1 byte, hex. AT+RSI_CLOSE is returned in uppercase and ASCII format.

3.2.21 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,dummy,dipaddr,dport, data_stream\r\n

Parameters

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

Dummy – 1-byte dummy, set to '0'.

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
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=1,0,0,0, 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 0x30 0x2C 0x30 0x2C 0x30 0x2C 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0D 0x0A

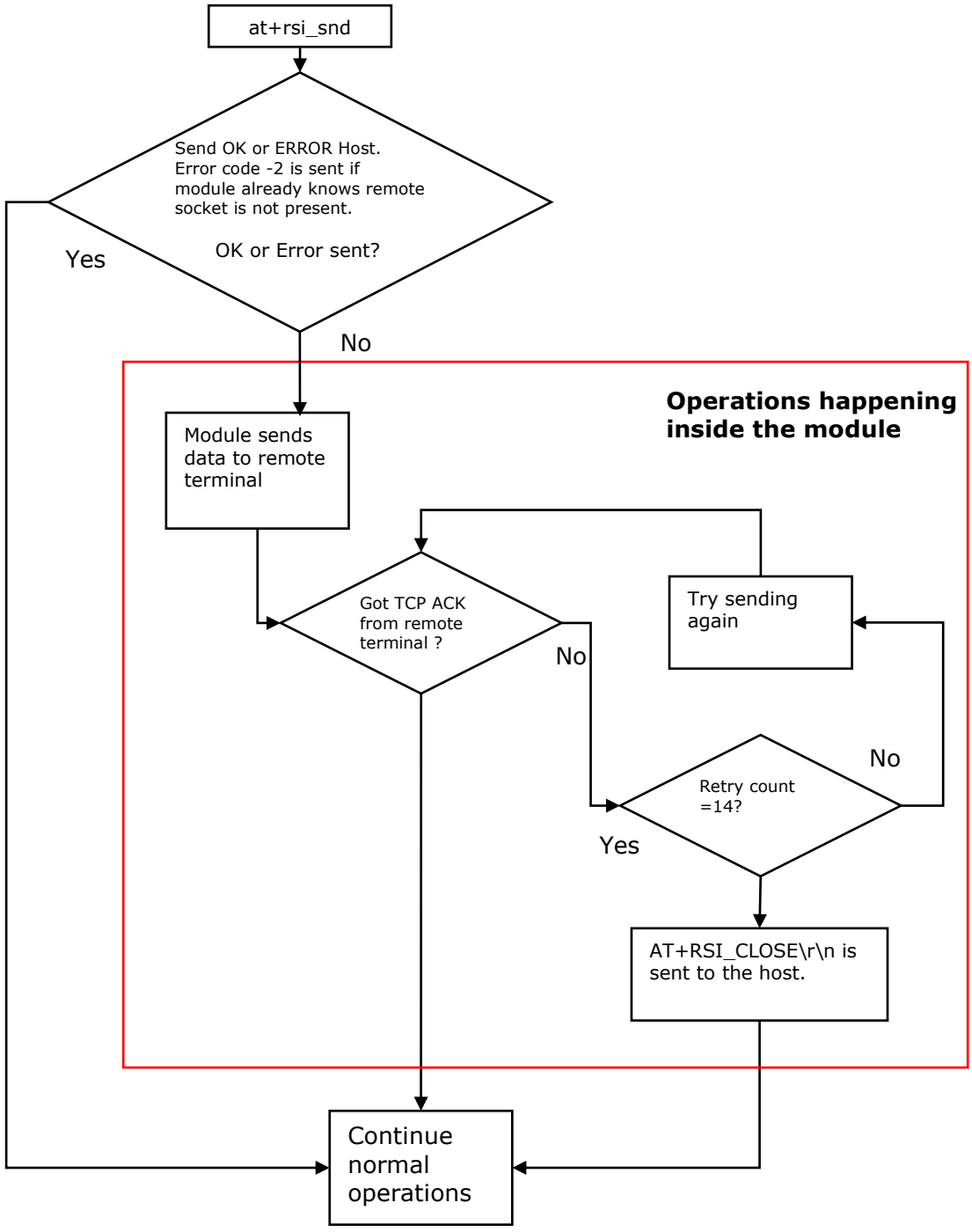
To send a data stream 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A over a UDP socket to a destination IP 192.168.1.20 and destination port 8001

at+rsi_snd=1,0,192.168.1.20,8001, 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 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

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



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 7: Send Operation

Relevance

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

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

3.2.22 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.

Message

AT+RSI_READ (returned in upper case)

Usage

AT+RSI_READ<socket_handle><size><sipaddr><Sport><stream>\r\n

Parameters

socket_handle (1 byte, hex) – socket handle of the socket over which the data is received

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 a TCP socket.

Stream – actual received data stream. A maximum of 1400 bytes can be received 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 or 2.

3.2.23 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=*webpage_len,webpage*\r\n

Parameters

webpage_len (variable length, ASCII)– The total length of the characters in the source code of the webpage. The maximum value of this parameter is 1000, planned to be increased in future firmware releases.

Webpage (variable length, ASCII)– Actual source code of the webpage.

For example, below is given 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,<source code>

……..

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

Response

Result Code	Description
OK	Success

Result Code	Description
ERROR<Error code>	Failure

The input parameters are delimited by commas (0x2C in ASCII format).

Relevance

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

> NOTE: Procedure for accessing the web page, wirelessly from a remote terminal.
>
> If the module in a GO (Group Owner), then a remote terminal can access the webpage stored in the module by typing http://192.168.100.76 (this is the IP address of the module when it is in GO mode. Refer Join) in its browser.
>
> In any other 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.

3.2.24 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.

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

3.2.25 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*(1 byte, hex): Number of IP addresses resolved *IPAddr*(4 bytes, hex): Individual IP addresses, up to a maximum of 10
ERROR	Failure.

Relevance

This command is relevant in Operating Modes 0 and 2.

3.2.26 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: HTMLGET1.00\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><length_of_data><Data>

Response Parameters

More (2 bytes, hex): This indicates whether more HTTP data for the HTTP GET response is pending. The least significant byte is sent first.

0x0001–More data pending. Additional AT+RSI_HTTPRSP messages may be sent by the module till all the data received is given to the Host.

0x0000–End of HTTP data

length_of_data (2 bytes, hex): This indicates the length of the actual data in the HTTP response. The least significant byte is sent first.

Data: Actual data in the HTTP response. This is not byte stuffed by the module when sent to the Host.

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: HTMLGET1.00\r\n\r\n

Data = <data>, the below command is used

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

Relevance

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

3.2.27 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><Length of Data><Data>

More (2 bytes, hex): This indicates whether more HTTP data for the HTTP POST response is pending. The least significant byte is sent first.

0x0001–More data pending. Additional AT+RSI_HTTPRSP messages may be sent by the module till all the data received is given to the Host.

0x0000–End of HTTP data

length_of_data (2 bytes, hex): This indicates the length of the valid HTTP data that is present in the response. The least significant byte is sent first.

Data: Actual data in the HTTP response. This is not byte stuffed by the module when sent to the Host.

The string AT+RSI_HTTPRSP is in uppercase ASCII.

Relevance

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

3.2.28 Set Region (802.11d)

Description

This command is used to set region code (US/Europe/Japan) in the module. This region code will be overridden if module finds any 802.11d element in beacon/probe responses from scanned access points.

Command

at+rsi_setregion

Usage

at+rsi_setregion=region_code\r\n

Parameters

region_code – The WiFi Region Code

1– US

2– Europe and France

3– Japan

Response

Result Code	Description
OK	Success
ERROR<Error code>	Failure

Relevance

This command is relevant in Operating Modes 0 and 2.

3.2.28.1 Module Operation in Different Geographical Domains

The module checks Access Point beacons to get geographical information. If the access point does not support 802.11d protocol, it may not embed geographical information in its beacons. In such a case, the module uses the information that is supplied by the user in the command *at+rsi_setregion*.

Below tables explain the channels supported in each regulatory domain.

2.4 GHz Band Channels:

Channel	Frequency (MHz)	US	EU	Japan
1	2412	Allowed	Allowed	Allowed
2	2417	Allowed	Allowed	Allowed
3	2422	Allowed	Allowed	Allowed
4	2427	Allowed	Allowed	Allowed
5	2432	Allowed	Allowed	Allowed
6	2437	Allowed	Allowed	Allowed
7	2442	Allowed	Allowed	Allowed
8	2447	Allowed	Allowed	Allowed
9	2452	Allowed	Allowed	Allowed
10	2457	Allowed	Allowed	Allowed
11	2462	Allowed	Allowed	Allowed
12	2467	Not allowed	Allowed	Allowed
13	2472	Not allowed	Allowed	Allowed
14	2484	Not allowed	Not allowed	Allowed

Table 6: Geographical Domains – 2.4GHz

Channel	Frequency (MHz)	US	Europe	Japan
36	5180	Allowed	Allowed	Allowed
40	5200	Allowed	Allowed	Allowed
44	5220	Allowed	Allowed	Allowed
48	5240	Allowed	Allowed	Allowed
52	5260	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Allowed – (DFS/TPC channel)
56	5280	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Allowed – (DFS/TPC channel)
60	5300	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Allowed – (DFS/TPC channel)
64	5320	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Allowed – (DFS/TPC channel)
100	5500	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
104	5520	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
108	5540	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
112	5560	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
116	5580	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
120	5600	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
124	5620	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
128	5640	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
132	5660	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
136	5680	Allowed (DFS	Allowed – (DFS/TPC	Not allowed

		channel)	channel)	
140	5700	Allowed (DFS channel)	Allowed – (DFS/TPC channel)	Not allowed
149	5745	Allowed	Not allowed	Not allowed
153	5765	Allowed	Not allowed	Not allowed
157	5785	Allowed	Not allowed	Not allowed
161	5805	Allowed	Not allowed	Not allowed
165	5825	Allowed	Not allowed	Not allowed

Table 7: Geographical Domains- 5 GHz

The *at+rsi_setregion* command should be issued before the scan command (refer "Associate to an Access Point with WPA2-PSK security as a client" in section APPENDIX A: Sample Flow of Commands in UART). If this command is not issued, the module takes US as the default region. For any geographical region, the module "active" scans channels that are allowed and "passive" scans channels that are not allowed or are designated as DFS channels in the domain. In passive scanning, the module does not send out probe requests to the Access Points.

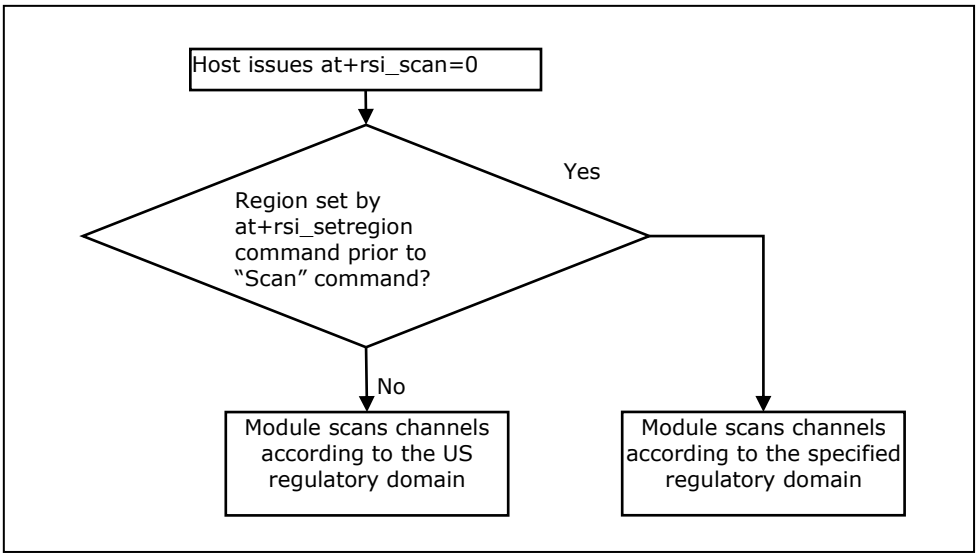


Figure 8: Configuring Regulatory Domains

3.2.29 DFS Client (802.11h)

Description

DFS client implementation is internal to the module. In Operating modes 0, 2, 3 and 5, in 5 GHz band, the module does only passive scan in DFS channels. If the Access Point detects radar signals, it indicates to the module (client) to switch to a different channel by using the "channel switch frame". The module performs

channel switch as per the AP's channel switch parameters. There is no command required to enable this feature, it is enabled by default.

3.2.30 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,Major 21.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

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 or 2.

3.2.31 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

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.

3.2.32Query 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
OK<*MAC_Address*>	*MAC_Address* (6 bytes, hex): MAC address of the module
ERROR<Error code>	Failure

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 or 2.

3.2.33 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*>[<*socket_id1*><*socket_type1*><*source_port1*><*destination_port1*><*destination_ip1*>][<*socket_id2*><*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. *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* (32 bytes, ASCII)*:* This value is the SSID of the Access Point to which the module is connected. If the actual length is less than 32, filler bytes 0x00 are used to make it 32 bytes *Network_type* (2 bytes, hex)*:* 0x0001 – Infrastructure 0x0000 – Ad-hoc Currently only Infrastructure mode is 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 0x01 – DHCP *Ipaddr* (4 bytes, hex): This is the IP

Result Code	Description
	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 below parameters are for each open socket. *Socket_id* (1 byte, hex): This indicates the socket handle. *Socket_type*(1 byte hex): 0x00 – TCP client 0x01 – UDP 0x02 – TCP server (Listening TCP) 0x04 – Listening UDP *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. *Destination_ip* (4 bytes, hex): IP of the remote terminal.
ERROR<Error Code>	Failure

Relevance

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

3.2.34 Query Group Owner Parameters

Description

This command is used to retrieve Group Owner related parameters. This command is issued to the module only if the module has become a Group Owner in Wi-Fi Direct mode.

Command

at+rsi_goparams

Usage

at+rsi_goparams?\r\n

Parameters

N/A

Response

Result Code	Description
OK<*SSID*><*reserved*><*BSSID*><*Channel_num*><*PSK*><*reserved*><*IPAddr*><*Station_count*><*MAC1*><*IP1*><*MAC2*><*IP2*>….<*MAC4*><*IP4*>	*SSID* (32 bytes, ASCII)*:* SSID of the Group Owner. If the SSID is less than 32 characters, then filler bytes 0x00 are added to make the length 32 bytes *reserved*(2 bytes): reserved, host should ignore *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 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

Result Code	Description
	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

Relevance

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

3.2.35Soft 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.

3.3 Error Codes

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

Error Code	Description
3 (0x0003)	No AP found
5 (0x0005)	Invalid band
10 (0x000A)	Invalid channel
20 (0x0014)	Wi-Fi Direct or EAP configuration is not done
21 (0x0015)	Memory allocation failed
22 (0x0016)	Information is wrong or insufficient in Join command
24 (0x0018)	Push button command given before the expiry of previous push button command.
25 (0x0019)	Access Point not found
28 (0x001C)	EAP configuration failed
29 (0x001D)	P2P configuration failed
30 (0x001E)	Unable to start Group Owner negotiation
32 (0x0020)	Unable to join
33 (0x0021)	Command given in incorrect state
34 (0x0022)	Query GO parameters issued in incorrect operating mode
35 (0x0023)	Unable to form Access Point
36 (0x0024)	Wrong Scan input parameters supplied to "Scan" command
-2 (0xFFFE)	Sockets not available. The error comes if the Host tries to open more than 8 sockets
-4 (0xFFFC)	IP configuration failed
-8 (0xF8)	Invalid command (e.g. parameters insufficient or invalid in the command). Please note that this is returned as a 1 byte value.
-8[1] (0xFFF8)	Invalid operation (e.g. power save command with the same mode given twice, accessing wrong socket, creating more than allowed sockets)
-9 (0xFFF7)	Byte stuffing error
-69 (0xBB)	Invalid content in the DNS response to the DNS

[1] Error code -8 is returned as 1 byte or a 2 byte value depending on whether it corresponds to an invalid command or an invalid operation. This will be rationalized in future releases.

Error Code	Description
	Resolution query
-70 (0xBA)	DNS Class error in the response to the DNS Resolution query
-72 (0xB8)	DNS count error in the response to the DNS Resolution query
-73 (0xB7)	DNS Return Code error in the response to the DNS Resolution query
-74 (0xB6)	DNS Opcode error in the response to the DNS Resolution query
-75 (0xB5)	DNS ID mismatch between DNS Resolution request and response
-85 (0xAB)	Invalid input to the DNS Resolution query
-92 (0xA0)	DNS response was timed out
-95 (0xFFA1)	ARP request failure
-99 (0xFF9D)	DHCP lease time expired
-100 (0xFF9C)	DHCP handshake failure
-121 (0xFF87)	This error is issued when module tried to connect to a non-existent TCP server socket on the remote side
-123 (0xFF85)	Invalid socket parameters
-127 (0xFF81)	Socket already open
-192 (0xFF40)	TCP socket close command is issued before getting the response of the previous close command

Table 8: 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

……..

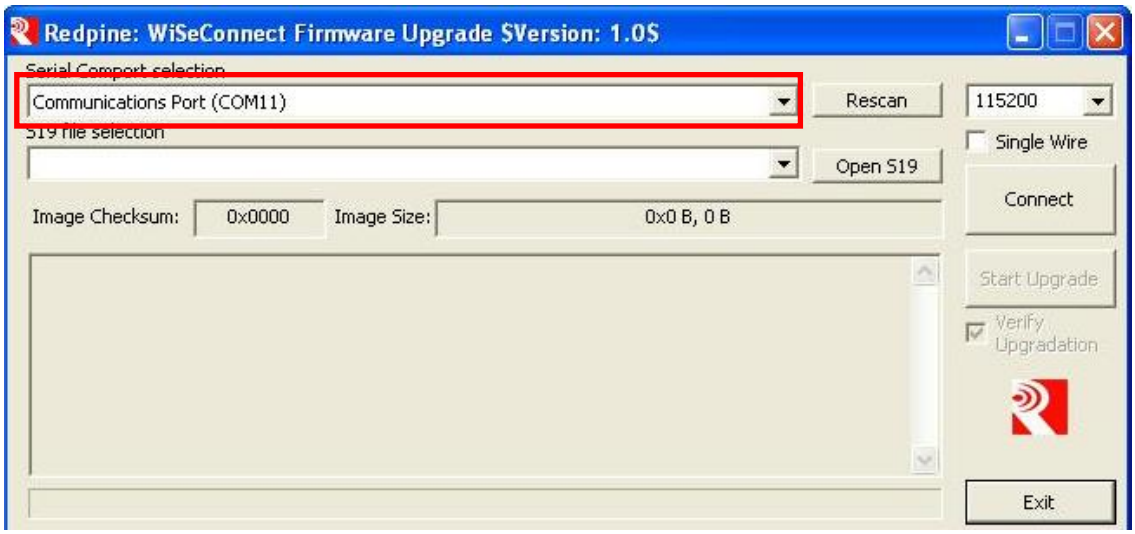
0x45 0x52 0x52 0x4F 0x52 0xFC 0xFF 0x0D 0x0A

4 Upgrading Firmware Through the UART Interface

The firmware of the module can be upgraded using the UART interface. Upgrading of firmware using the SPI interface is not supported currently. However, upgrading firmware using the wireless interface will interface will be supported in future firmware releases. If the user wants to the upgrade the firmware of the module with a newer version, the following flow should be used.

Connect a PC to the Module through the UART interface, using a UART cable.

Open the application RS.WSC.201.GENR.x.x.x\Software\Applications\tools\Firmware_upgrade\WSC_FW_Upgrade_Util.exe in 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.

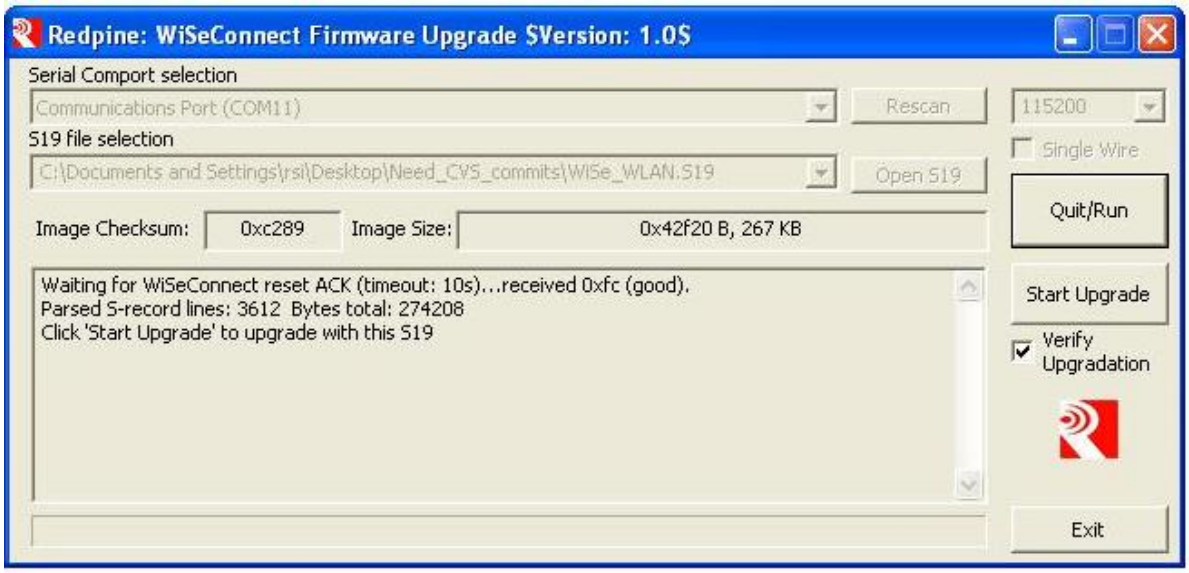


From the drop-down box, select the COM port that is connected to the module's UART interface.

Click "Open S19" button. Select the file WiSe_WLAN.S19. Now press "Connect" button.

Within 10 secs of pressing the "Connect" button, give a hard-reset to the module.

The message window of the GUI will prompt to start the upgrade. Click on "Start Upgrade" button.



The Message window prompts "Upgradation Completed". Card_Ready (Pin Number 49 for RS-WC-201 and Pin 54 for RS-WC-301) goes "Low". It may take up to 1 min for the pin to go low.

Click "Open S19" button. Select the file WiSe_Control.S19. Now press "Connect" button.

Within 10 secs of pressing the "Connect" button, give a hard-reset to the module or power cycle the module.

The message window of the GUI will prompt to start the upgrade. Click on "Start Upgrade" button.

The Message window prompts "Upgradation Completed". This completed the Firmware Upgrade Process. The module should now be reset or power cycled to operate in normal mode.

5 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.

5.1 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.

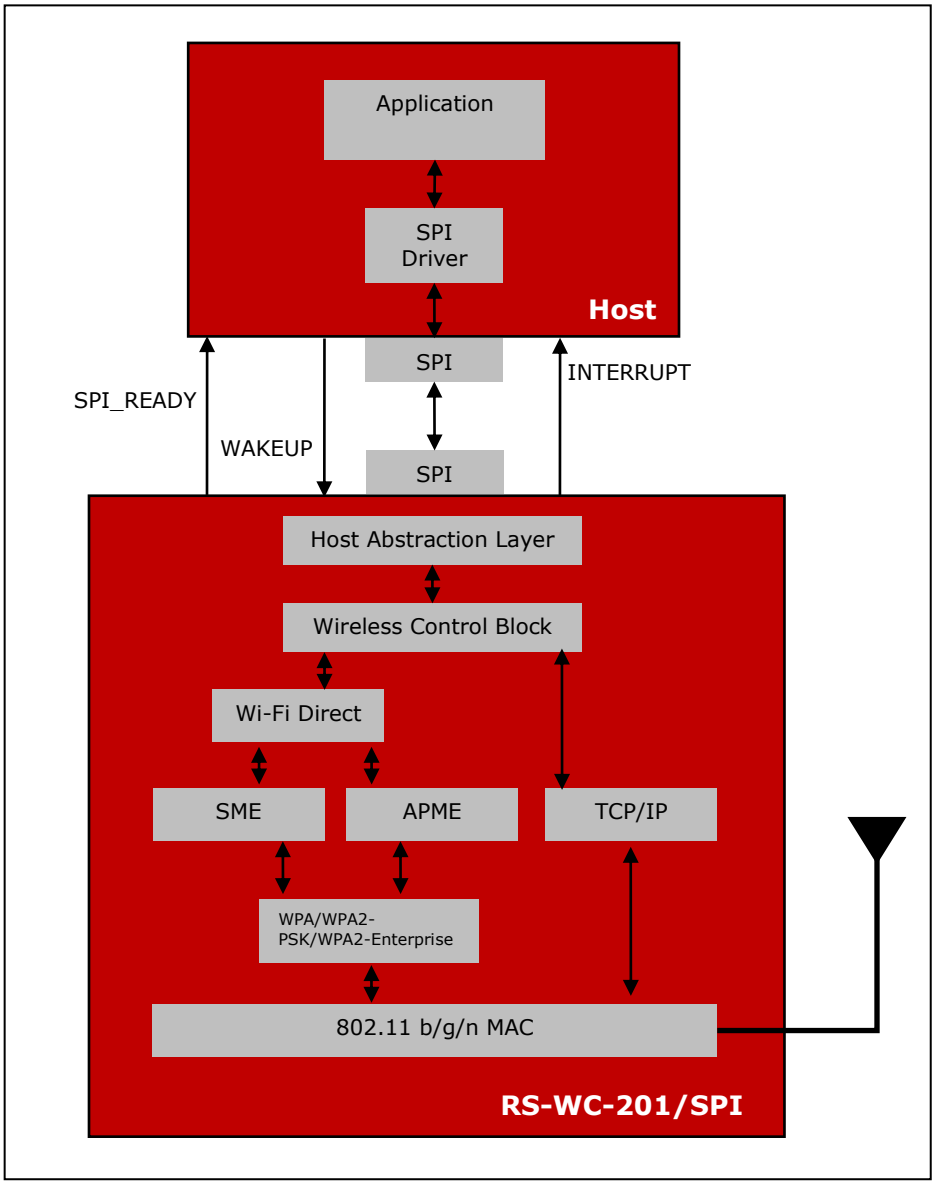


Figure 9: System Architecture with SPI Interface

5.1.1 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.

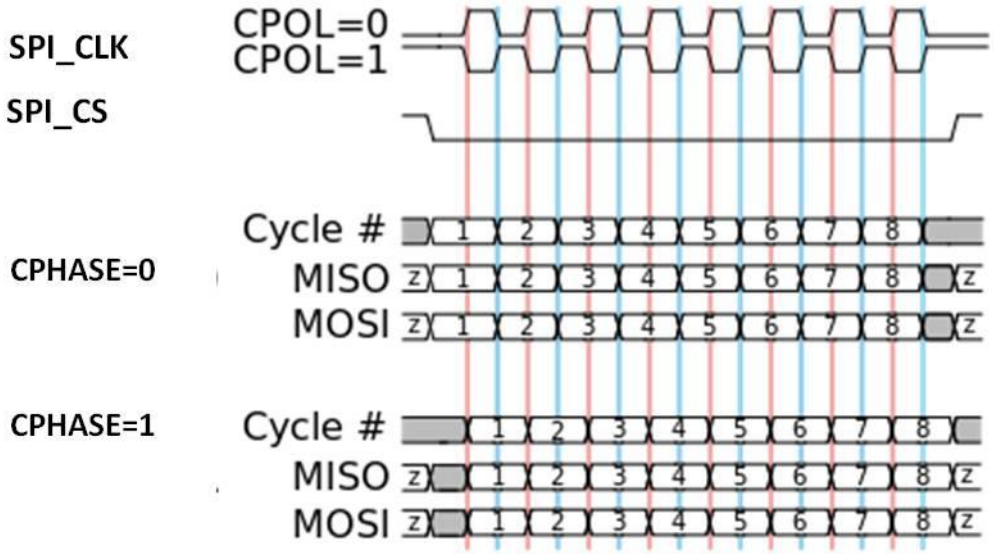


Figure 10: Clock Polarity and Clock Phase

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.

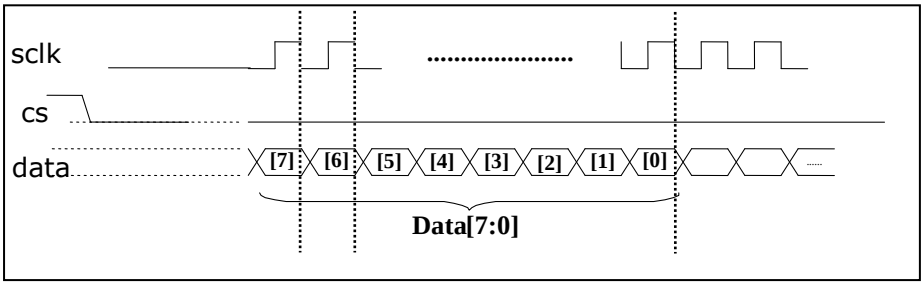


Figure 11: Endianess in Data Transfer

5.2 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.

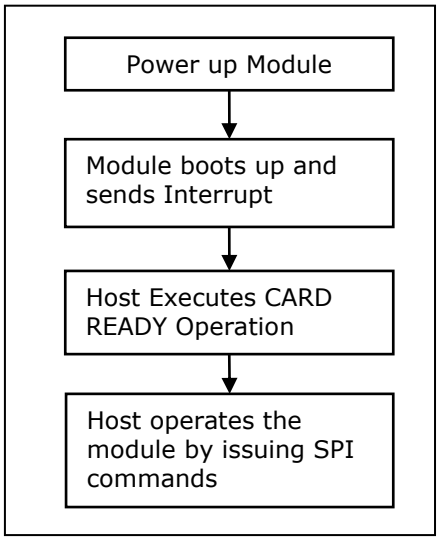


Figure 12: Module Operation

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.

5.2.1 Tx Operation

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

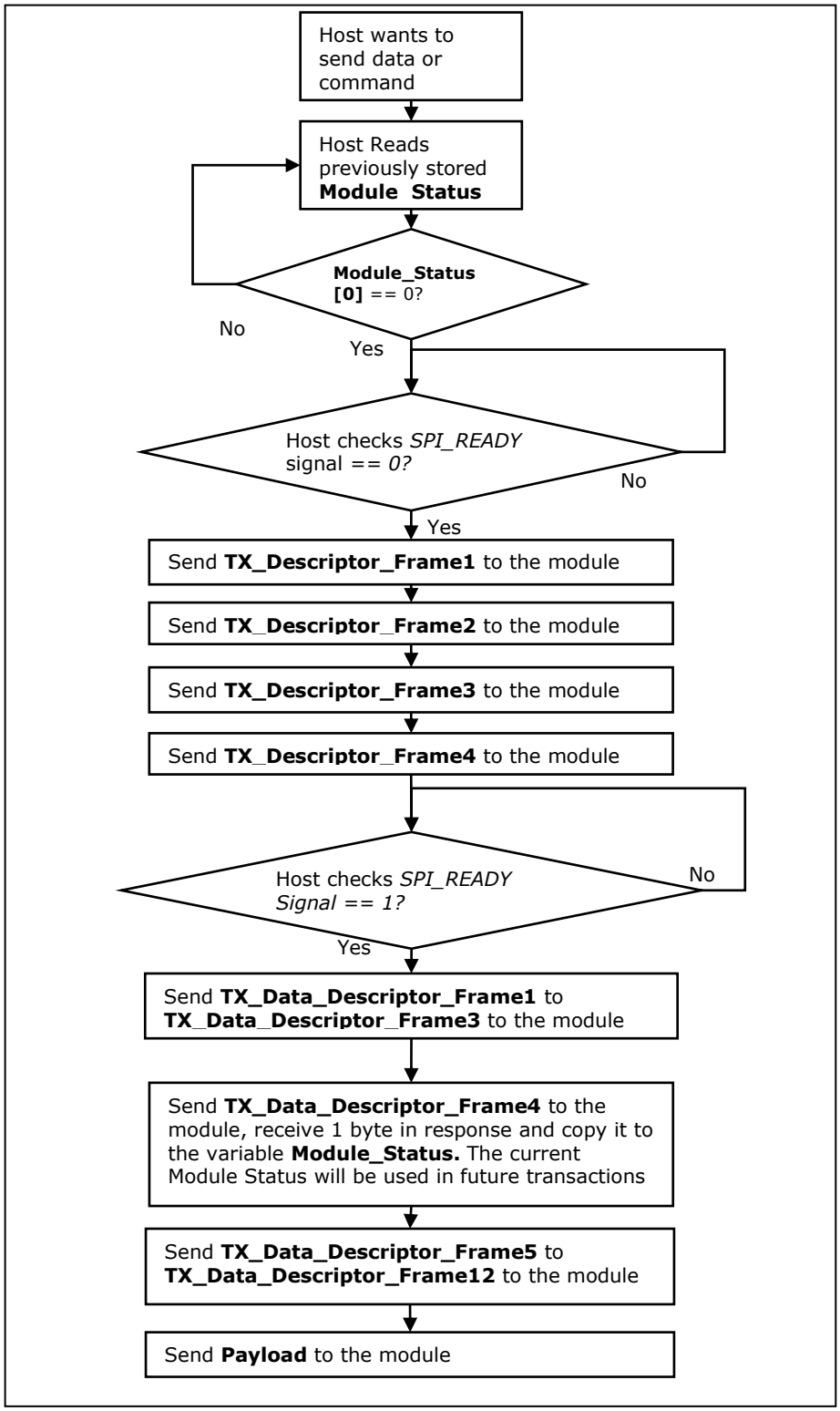


Figure 13: Tx Operation[1]

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

5.2.1.1 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 9: 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.

5.2.1.2 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 10: 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
Set Region	0x1D
Set IP Parameters	0x41
Socket Create	0x42
Socket Close	0x43
DNS Resolution	0x44
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 Webpage	0x50
HTTP GET	0x51
HTTP POST	0x52
DNS Server	0x55

Table 11: Command IDs for Tx Data Operation

5.2.1.3 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 12: Module Status

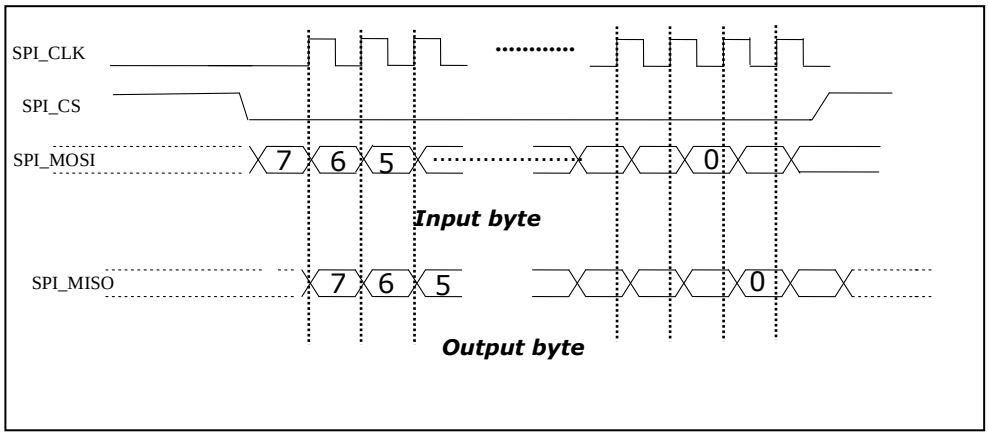


Figure 14: SPI Transactions

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.

5.2.1.4 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 .

5.2.2 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.

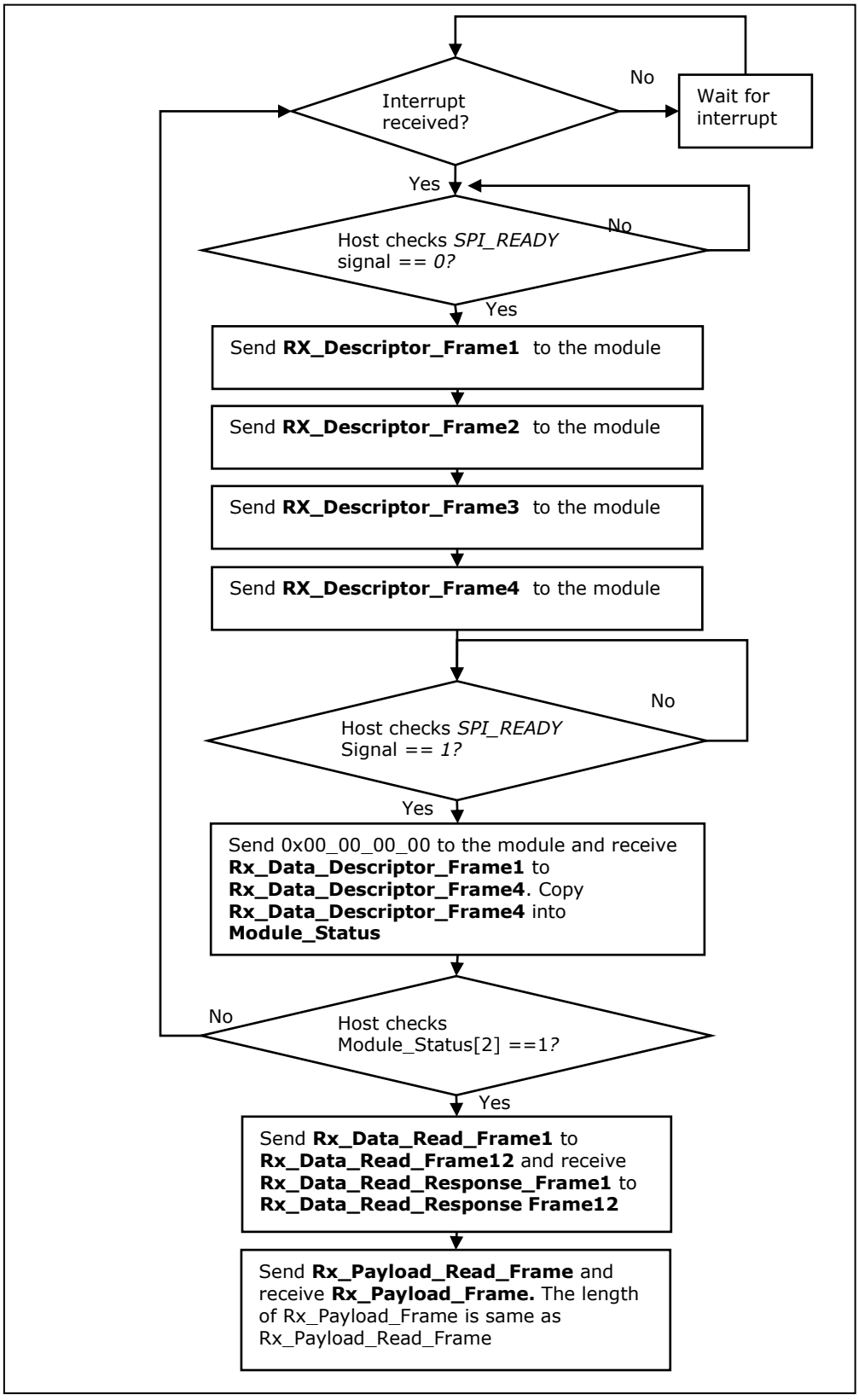

Interrupt received?
No
Wait for interrupt
Yes
Host checks SPI_READY signal == 0?
No
Yes
Send RX_Descriptor_Frame1 to the module
Send RX_Descriptor_Frame2 to the module
Send RX_Descriptor_Frame3 to the module
Send RX_Descriptor_Frame4 to the module
Host checks SPI_READY Signal == 1?
No
Yes
Send 0x00_00_00_00 to the module and receive Rx_Data_Descriptor_Frame1 to Rx_Data_Descriptor_Frame4. Copy Rx_Data_Descriptor_Frame4 into Module_Status
Host checks Module_Status[2] ==1?
No
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

5.2.2.1 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 13: 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
Rx_Data_Descriptor_Frame4	Copied to **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 14: 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.

5.2.2.2 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_Frame10	Reserved, should be ignored
Rx_Data_Read_Response_Frame11	Response ID. Refer table Response IDs for Rx OperationCommand IDs for Tx Data Operation
Rx_Data_Read_Response_Frame12	Reserved, should be ignored

Table 15: 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
Set Region	0x1D
IP Parameters Configure	0x41
Socket Create	0x42
Socket Close	0x43

DNS Resolution	0x44
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 Webpage	0x50
HTTP GET	0x51
HTTP POST	0x52
Async WFD	0x54
DNS Server	0x55
Async TCP Socket Connection Established	0x61
Async Socket Remote Terminate	0x62
Receive data	Ignore the response ID field while receiving data from a remote terminal

Table 16: 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.

5.2.2.3 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.

5.3 Card Ready Operation

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

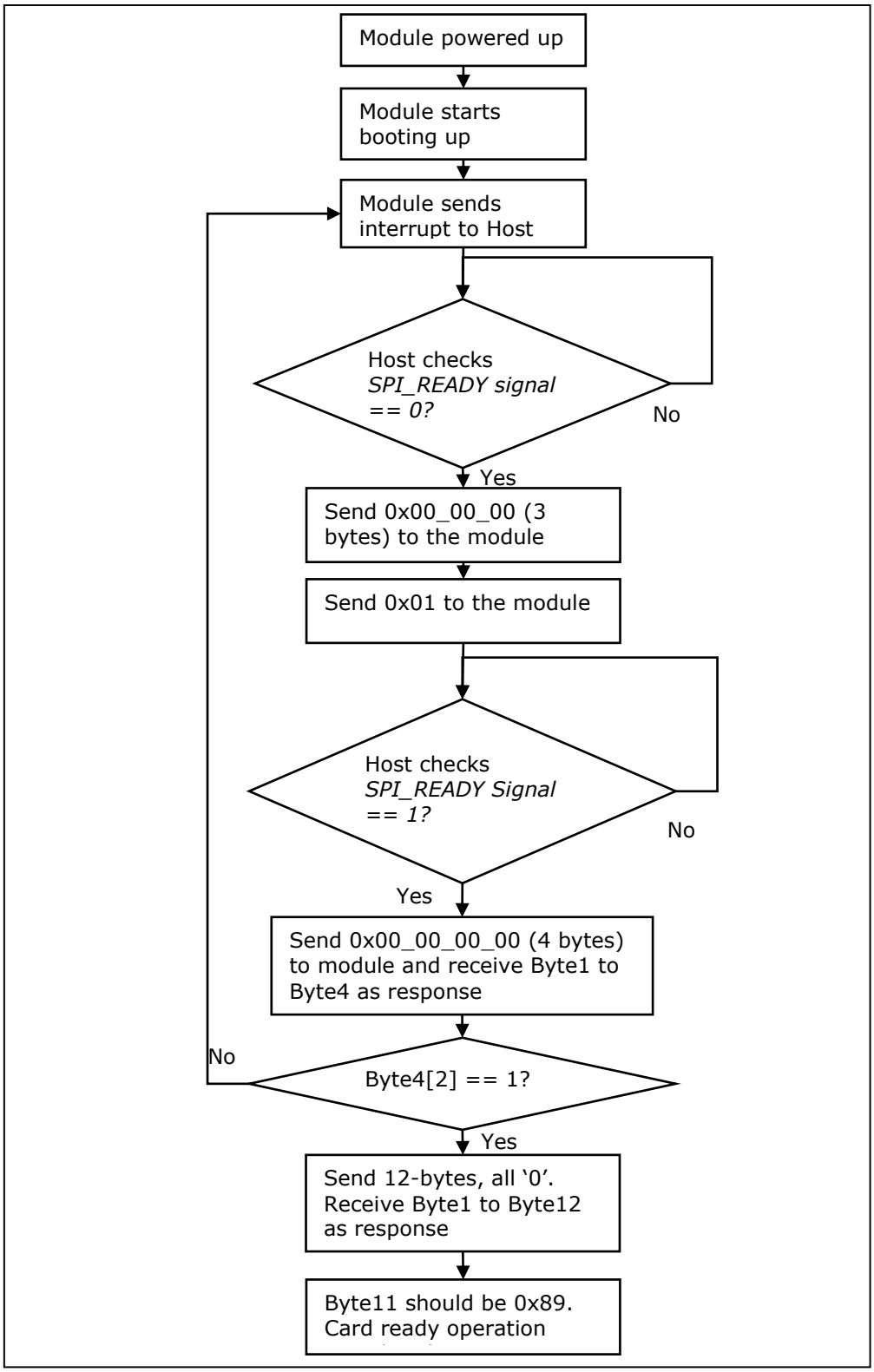
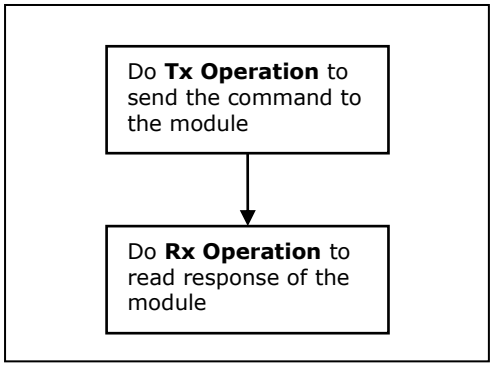


Figure 16: Card Ready Operation

5.4 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.



5.4.1 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;
```

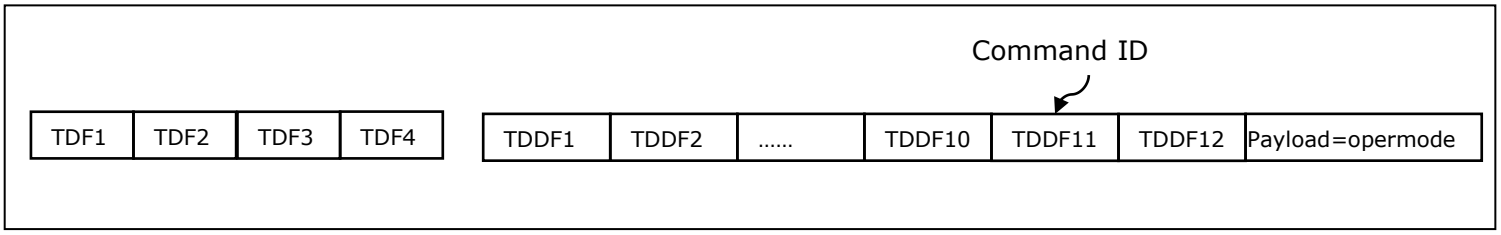


Figure 17: Sending a Command to the Module

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 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.

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

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 WiFi 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.

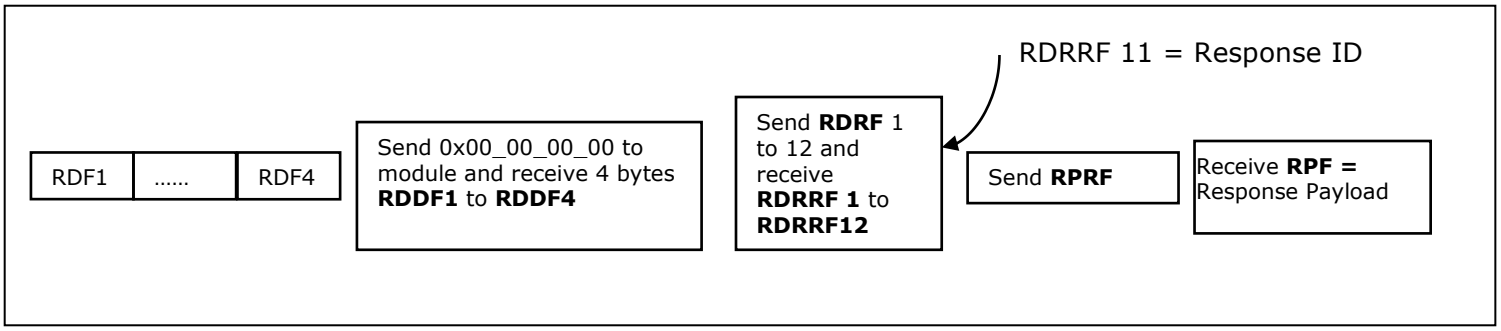


Figure 18: Receiving Response from the Module

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

5.4.2 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 within Operating Mode 1

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.

Wi-Fi Direct Mode or Access Point Mode within Operating Mode 1 is set in the command "Configure Wi-Fi Direct Peer-to-Peer Mode".

Response Payload

There is no response payload for this command.

Relevance

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

5.4.3 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.

Relevance

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

5.4.4 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.

Relevance

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

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

Description

This command is used to set the configuration information for Operating mode 1. There are two sub-modes inside Operating Mode 1: Wi-Fi Direct Mode and AP mode. In 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 15. Higher the number, higher is the willingness of the module to become a GO.

Access Point Mode: If the device wants to become an Access Point with WPA2-PSK security, then a value of 16 should be used.

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].

This SSID (along with the DIRECT-xy prefix) also appears if the module is configured in Access Point mode.

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, tt 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;
```

Parameters

devState: State of the remote Wi-Fi Direct node.

0– The remote Wi-Fi Direct node was found in a 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.

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

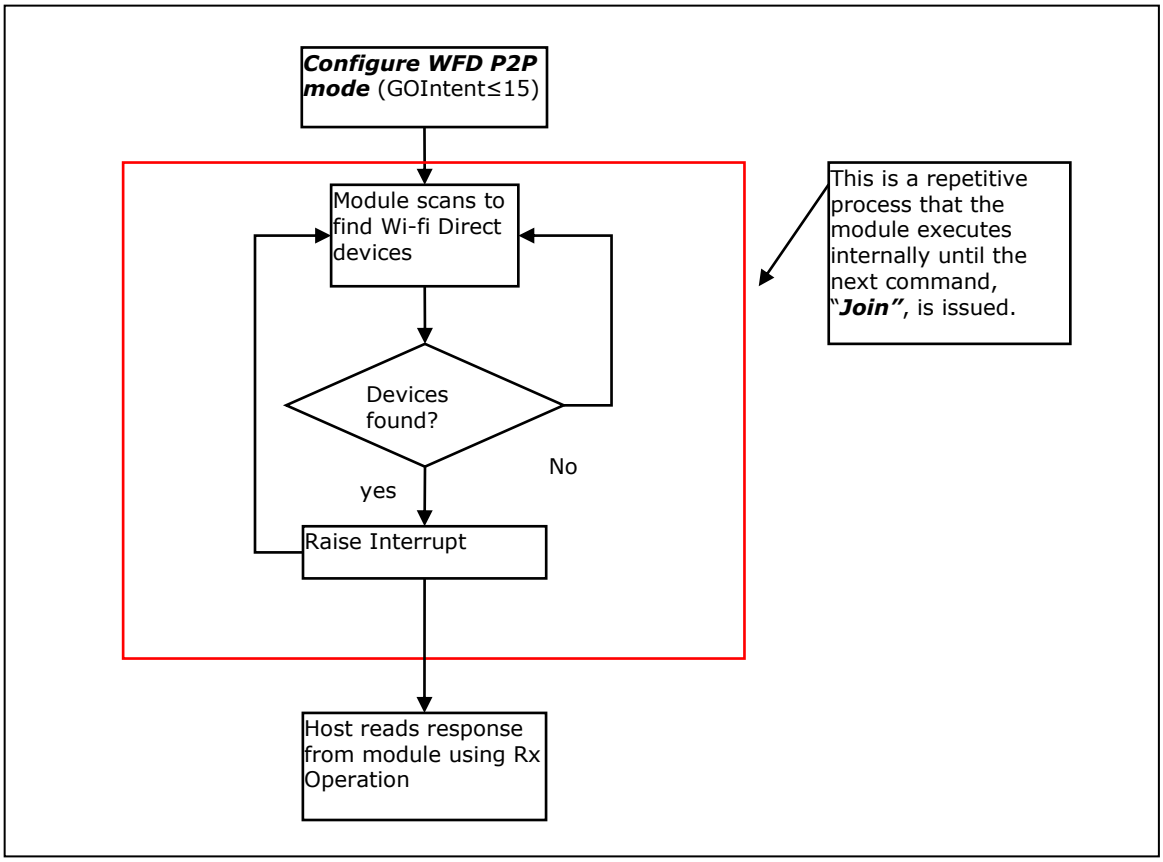


Figure 19: Operation after issuing "Configure WFD P2P Mode" command

Relevance

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

5.4.6 Scan

Description

This command scans for Access Points and gives the scan results to the host.

Payload Structure

```
struct {
        uint8       chan_num[4];
        uint8       ssid[32];     /* 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[32];
        uint8                   bssid[6];
}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
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

scancount: Number of Access Points scanned

Relevance

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

5.4.7 Join

Description

This command is used for following:

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

Associate to a remote device in WiFi Direct mode (operating mode 1)

Create an Access Point (operating mode 1)

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

Payload Structure

```
struct {
        uint8       reserved1;
        uint8       reserved2;
        uint8       dataRate;
        uint8       powerLevel;
        uint8       psk[64];
        uint8       ssid[32];
        uint8       reserved3;
        uint8       reserved4;
        uint8       reserved5;
        uint8       ssid_len ;
} joinFrameSnd;
```

Parameters

reserved1: Reserved. Set to '0'

reserved2: Reserved. Set to '0'

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

Data Rate (Mbps)	Value of dataRate
24	9
36	10
48	11
54	12
MCS0	13
MCS1	14
MCS2	15
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:

0– Low power (7dBm)

1– Medium power (10dBm)

2– High power (16 to 17dBm)

psk: Passphrase used in WPA/WPA2-PSK security mode. In open 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 *GOIntent* parameter in command "Configure Wi-Fi Peer-to-Peer" is set to 16 to make the module an Access Point, this parameter becomes a part of the AP's SSID

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.

Relevance

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

5.4.8 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.

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.

Relevance

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

5.4.9 Set 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

cert_type: Type of certificate.

1– TLS client certificate

2– FAST PAC file

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]

cert_length: Length of the current segment

keyPwd: Private key password, used to generate the certificate

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

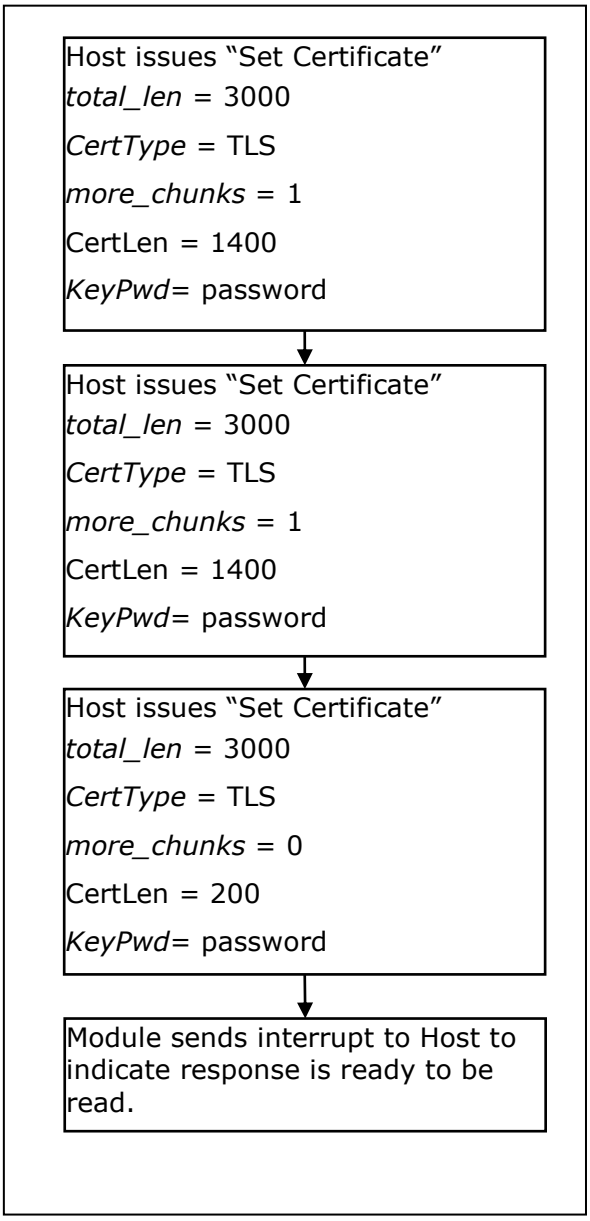


Figure 20: Loading Certificate in SPI mode

Response Payload

There is no response payload for this command.

Relevance

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

5.4.10 Set IP Parameters

Description

This command configures the IP address, subnet mask and default gateway of the TCP/IP stack in the RS-WC-201/301 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

Relevance

This command is relevant when the module is configured in Operating Mode 0, 2 and 1 (when it is not the GO or AP).

5.4.11 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)

4– Listening UDP

moduleSocket: Port number of the socket in the module. Value ranges from 1024 to 49151

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

destIpaddr: 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.

Relevance

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

5.4.12 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

Relevance

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

5.4.13 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

Relevance

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

5.4.14 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  webPageLen[2];
        uint8   webServData[1024];
}webServFrameSnd;
```

Parameters

webPageLen: Length of the webpage (in number of characters) to be loaded

weServData – Actual source code of the webpage, in ASCII characters. The maximum size of this parameter is 1000, designed to be increased in future firmware releases.

For example, below is given the source code of a reference page that can be sent to load the corresponding web page(99 characters in all including newline character.

```
<html>
<head>
```

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

Response Payload

There is no response payload for this command.

Relevance

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

> 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.

5.4.15 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 and 5.

5.4.16 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 or 2.

5.4.17 Send data

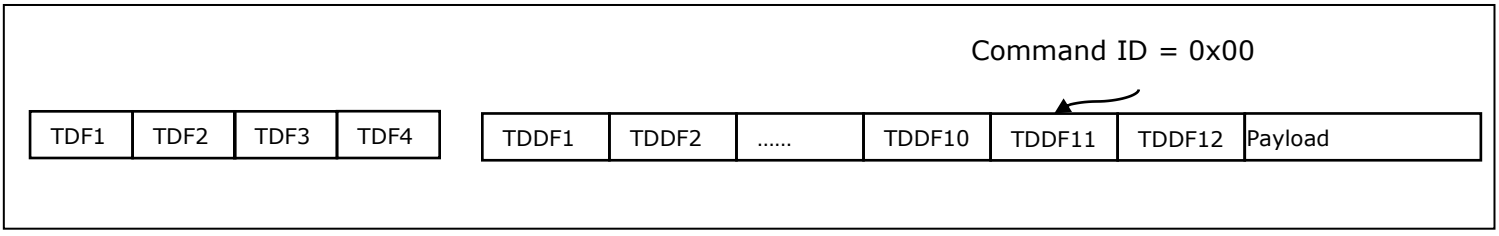
Operating Mode 0, 1 and 2

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                    *Payload;
uint32                  payloadLen;
uint8                   protocol;
} sendFrameSnd;
```



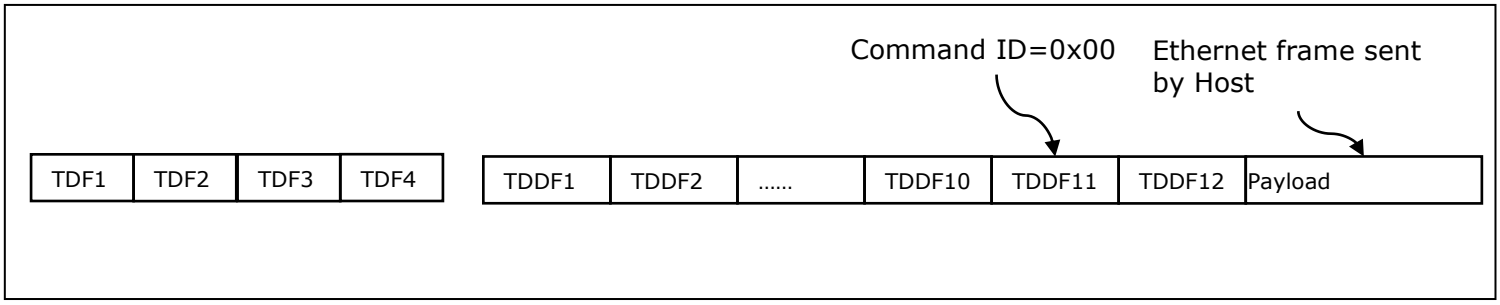
Parameters

socketDescriptor: Descriptor of the socket over which data is to be sent

Payload: Actual data to be sent. A maximum of 1400 bytes can be sent in a packet.
payloadLen: Length of the payload, in number of bytes
protocol: 0– UDP data
1– TCP data

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.



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.

5.4.18Receive data

Operating Mode 0, 1 and 2

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.

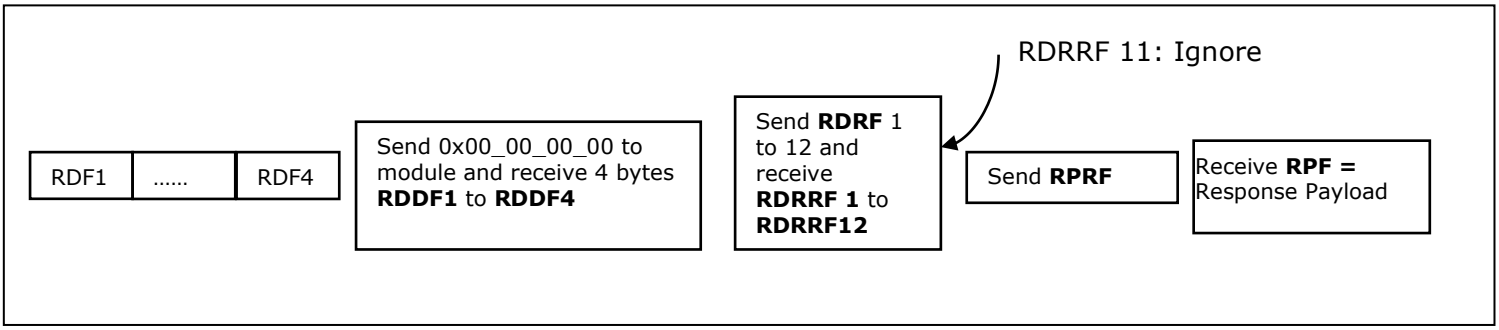
Response Payload

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;
```



Response Parameters

recvSocket: Port number 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.

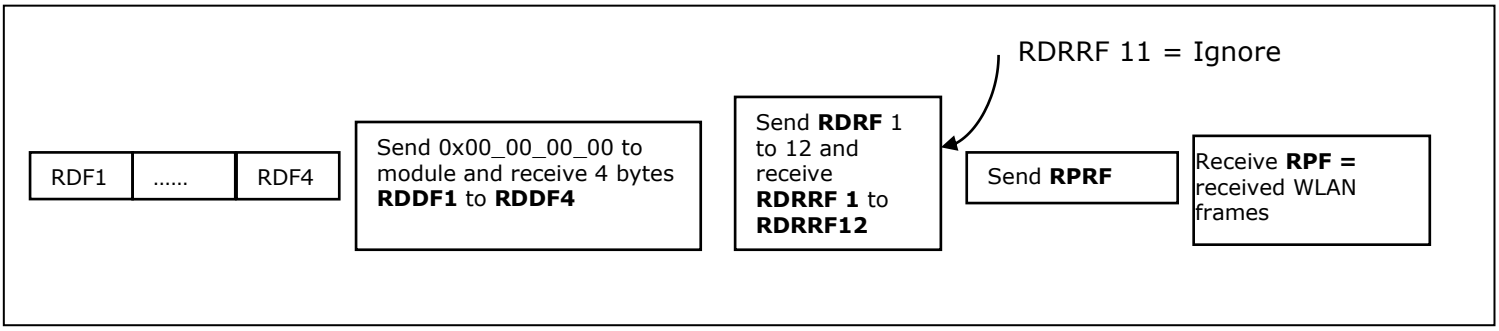
fromPortNum: The socket number of the source port

fromIpaddr: The IP address of the source terminal

recvDataBuf: Actual data sent from remote terminal

Operating Mode 3, 4 and 5

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.



5.4.19 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 or 2.

5.4.20 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

Relevance

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

5.4.21 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.

Relevance

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

5.4.22 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

3–Power Save Mode 3

If power mode 1 is set, then the below steps should be executed to confirm to the module to go to power save mode 1.

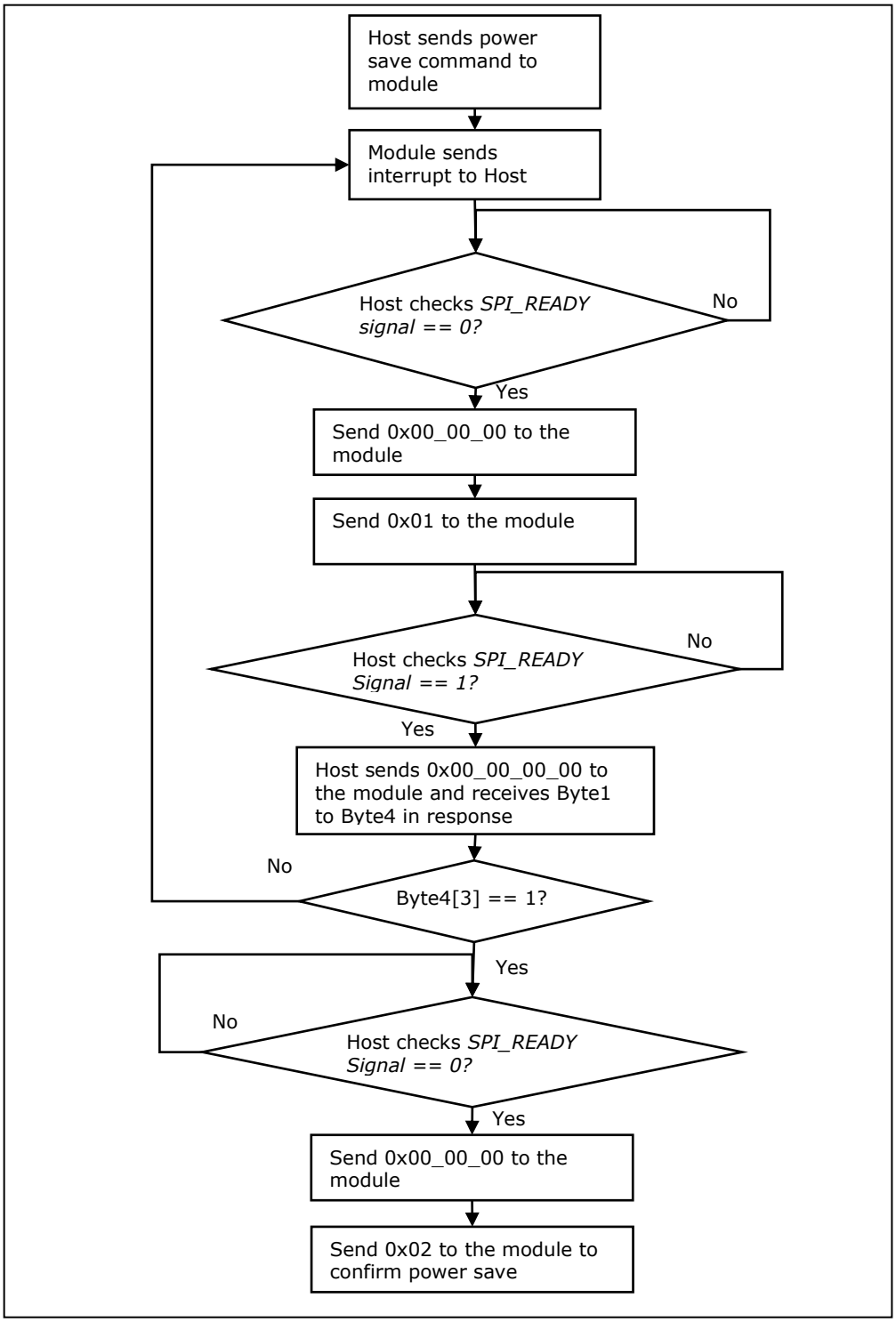


Figure 21: Setting Power Save Mode 1

If power mode 2 or 3 is set, then the below steps should be executed to confirm to the module to go to power save mode 2 or 3.

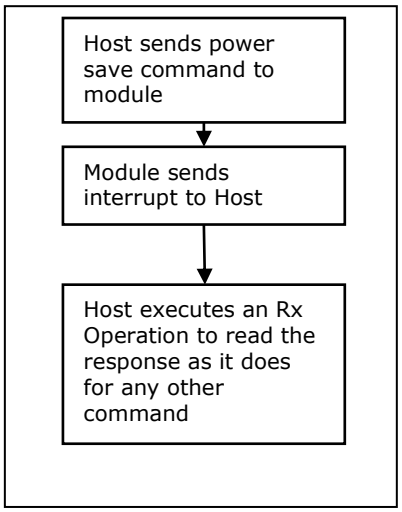


Figure 22: Setting Power Save Mode 2 and 3

Response Payload

There is no response payload for this command.

Relevance

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

5.4.22.1 Power Save Operation

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

5.4.22.1.1 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.

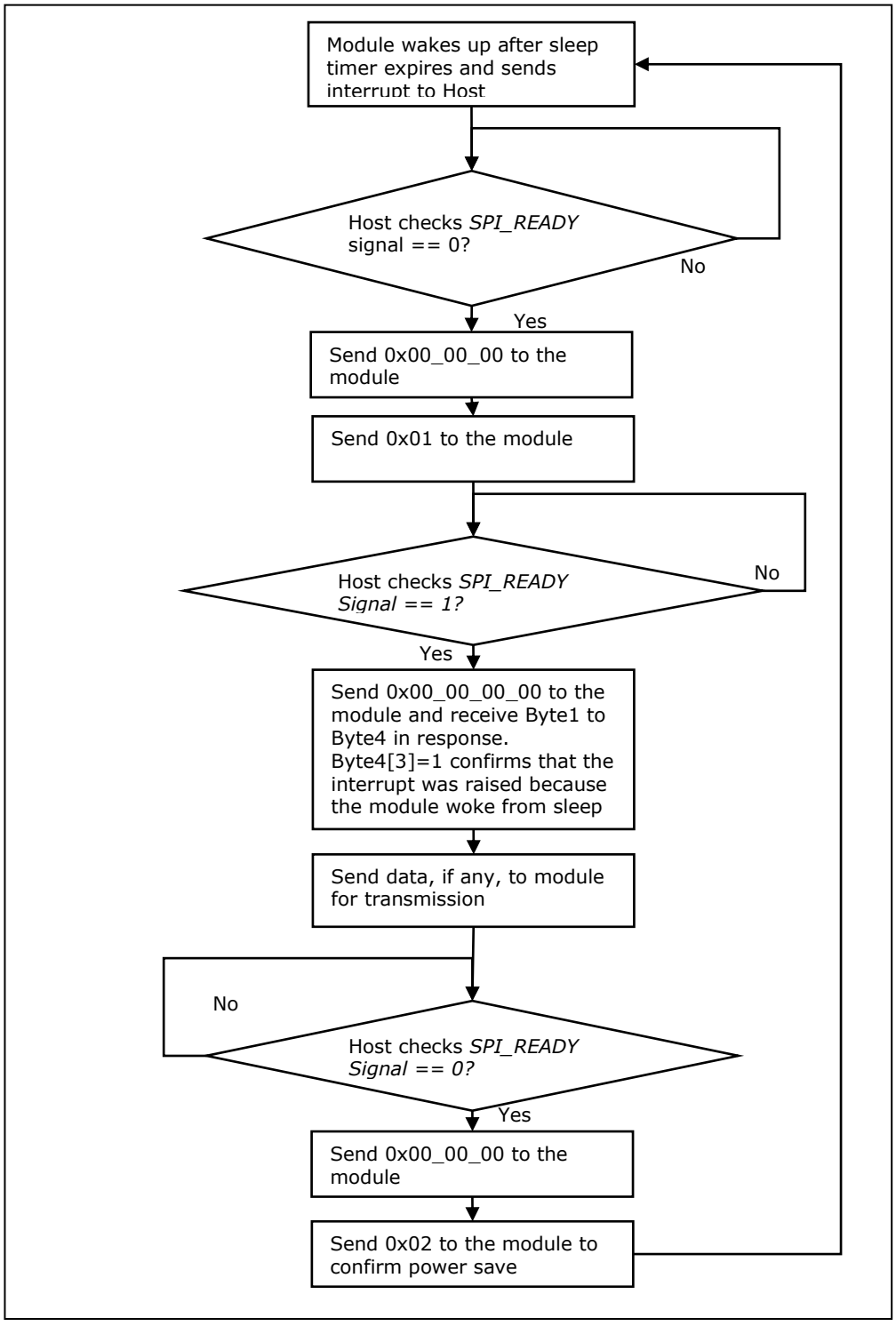


Figure 23: Power Save Operation

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.

5.4.22.1.2 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.

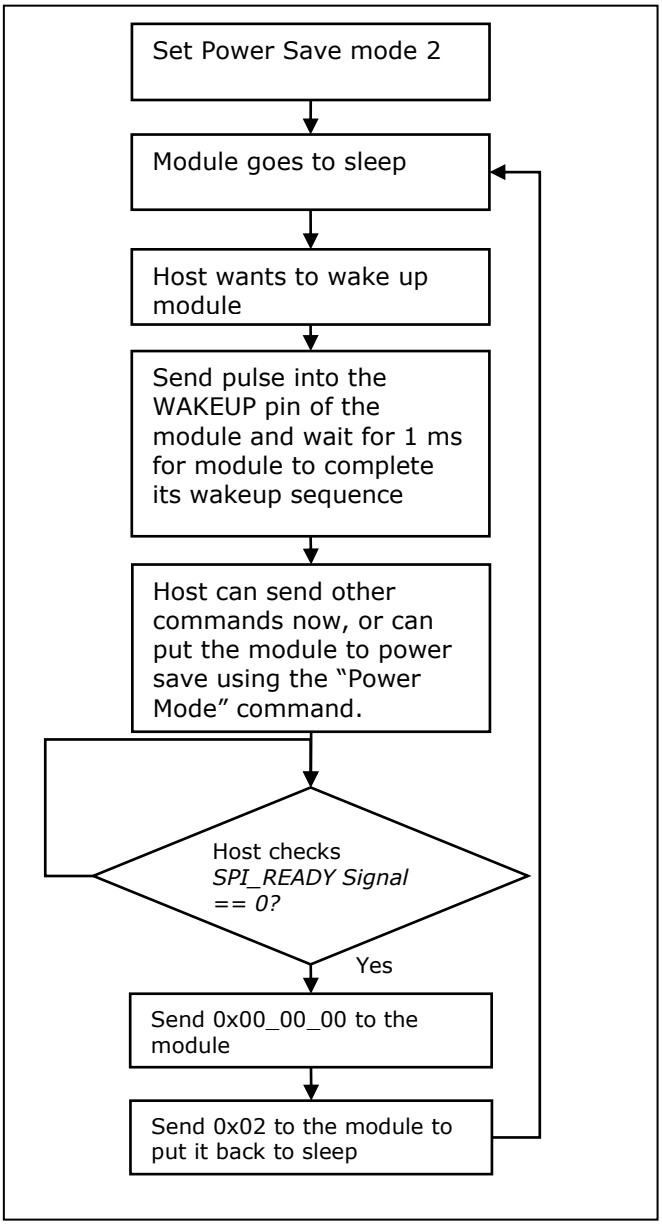


Figure 24: Power Save Mode 2

5.4.22.1.3 Power Save Mode 3

Once the module is put to power save mode 3, 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. This mode resembles a fresh power of the module, and the steps shown in figure Module Operation should be executed. The Host should give commands right from the first command "Set Operating Mode" to configure the module. All previous

configuration information (such as band, Access Point it was previously connected to etc.) is lost after the module goes to sleep in this mode.

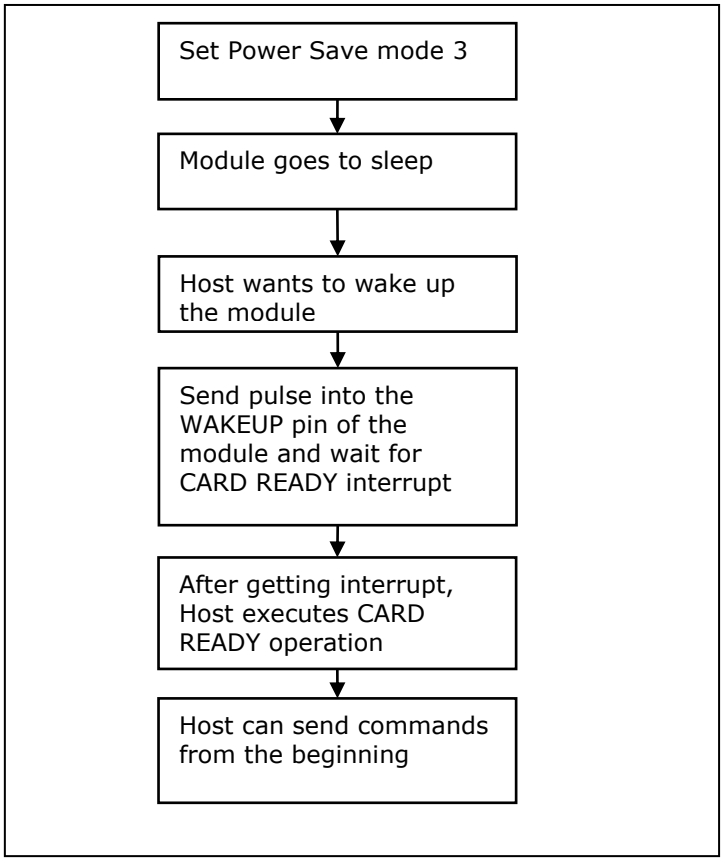


Figure 25: Power Save Mode 3

Power consumption is the lowest in Power Save Mode 3, followed by Mode 2 and Mode 1.

5.4.23 Disassociate

Description

This command is issued to request the module to disassociate (disconnect) from an Access Point. The Host can then issue a fresh set of 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.

Payload Structure

No payload required for this command.

Response Payload

There is no response payload for this command.

Relevance

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

5.4.24 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 value of the RSSI. For example, if the RSSI is -20dBm, then 20 is returned.

Relevance

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

5.4.25 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
{
```

```
    uint8   sock_id;
    uint8   socket_type;
    uint16 sPort;
    uint16  dPort;
    uint8   dIp[4];
}sock_info_query_t;
struct  EVT_NET_PARAMS
{
    uint  wlan_state;
    uint  Chn_no;
    uint  psk[64];
    uint  mac_addr[6];
    uint  ssid[32];
    uint connection_type ;
    uint  sec_type ;
    uint  dhcp_mode ;
    uint  ipaddr[4];
    uint  subnet_mask[4];
    uint  gateway[4];
    uint  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

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

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

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

Relevance

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

5.4.26 Query Group Owner Parameters

Description

This command is used to retrieve Group Owner related parameters. This command is issued to the module only if the module has become a Group Owner in Wi-Fi Direct mode.

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

Relevance

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

5.4.27DNS 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

Relevance

This command is relevant in Operating Modes 0 and 2.

5.4.28 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

Relevance

This command is relevant in Operating Modes 0 and 2.

5.4.29 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.

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

For example,

IP = 192.168.40.50

URL=/index.html

Header=HTMLGET1.00\r\n\r\n

Data=<data>

can be the valid contents of the buffer

Response

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
{
UINT16 more;
UINT16 length;
UINT8 *data;
}__attribute__((packed));
```

more: This indicates whether more HTTP data for the HTTP GET request is pending.

1–More data pending. Further interrupts may be raised by the module till all the data is transferred to the Host.
0– End of HTTP data

length: This indicates the length of the valid HTTP data that is present in the response

data: Actual data in the HTTP GET response

Relevance

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

5.4.30 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=HTMLPOST1.00\r\n

Data=<data>

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. 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
{
UINT16 more;
UINT16 length;
UINT8 *data;
}__attribute__((packed));
```

more: This indicates whether more HTTP data for the HTTP POST request is pending.

1– More data pending
0 – End of HTTP data

length: This indicates the length of the valid HTTP data that is present in the response

data: Actual data in the HTTP response

Relevance

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

5.4.31 Set Region (802.11d)

Description

This command is used to set region code (US/Europe/Japan) to the module. This region code will be overridden if module finds any 802.11d element in beacon/probe responses from scanned access points.

Payload Structure

```
struct Set_Region_Code
    {
    UINT16      WiFi_region;
    }__attribute__((packed));
```

Parameters

WiFi_region:

1-US

2-Europe and France

3-Japan

Response Payload

There is no response payload for this command.

Relevance

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

5.4.31.1 Module Operation in Different Geographical Domains

The module checks Access Point beacons to get geographical information. If the access point does not support 802.11d protocol, it may not embed geographical information in its beacons. In such a case, the module uses the information that is supplied by the user in the command "Set Region". In passive scanning, the module does not send out probe requests to the Access Points.

Tables Geographical Domains – 2.4GHz and Geographical Domains- 5 GHz explain the channels supported in each regulatory domain.

The "Set Region" command should be issued before the scan command (refer "Associate to an Access Point with WPA2-PSK security as a client" in section

APPENDIX B: Sample Flow of Commands in SPI and figure Configuring Regulatory Domains)). If this command is not issued, the module takes US as the default region. For any geographical region, the module "active" scans channels that are allowed and "passive" scans channels that are not allowed or are designated as DFS channels in the domain. In passive scanning, the module does not send out probe requests to the Access Points.

5.4.32 DFS Client (802.11h)

Description

DFS client implementation is internal to the module. In Operating modes 0, 2, 3 and 5, in 5 GHz band, the module does only passive scan in DFS channels. If the Access Point detects radar signals, it indicates to the module (client) to switch to a different channel by using the "channel switch frame". The module performs channel switch as per the AP's channel switch parameters. There is no command required to enable this feature, it is enabled by default.

5.4.33 Soft Reset

Description

This command acts gives 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 from the first command "Set Operating Mode" after issuing this command.

Payload Structure

There is no payload for this command.

Response

There is no response payload for this command.

Relevance

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

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.

5.5 Error Codes

Error Code(in decimal)	Description
3	No AP found

5	Invalid band
10	Invalid channel
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	Access Point not found
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
-2	Sockets not available. The error comes if the Host tries to open more than 8 sockets
-4	IP configuration failed
-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

-92	DNS response was timed out
-95	ARP request failure
-99	DHCP lease time expired
-100	DHCP handshake 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
-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

Table 17: Error Codes for SPI

6 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.

6.1 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.

6.2 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.

6.3 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.

6.3.1 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

6.3.2 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

6.3.3 rsi_spi_init.c

This file contains the API for the "Init" command.

API Prototype :

int16 rsi_init(void)

Parameters:

None

6.3.4 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

6.3.5 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
deviceName[64]	uint8	Device name

Structure Member Name	Structure Member type	Description
operChannel[2]	uint8	Channel in which the Device operates
ssidPostFix[64]	uint8	Post Fix for SSID
psk[64]	uint8	Pre shared Key

6.3.6 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

6.3.7 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                     reserved2;
        uint8                   dataRate;
        uint8                   powerLevel;
        uint8                   psk[RSI_PSK_LEN];
        uint8                   ssid[RSI_SSID_LEN];
        uint8                   reserved3;
        uint8                   reserved4;
        uint8                   reserved5;
        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, reserved5	uint8	Reserved, set all to '0'
Reserved2	uint8	Reserved
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 an unused input in open mode. Last character is NULL.
Ssid[32]	uint8	SSID of the access

Structure Member Name	Structure Member Type	Description
		point

6.3.8 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

6.3.9 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

6.3.10 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)
netmask[4]	uint8	Subnet mask of the TCP/IP stack (valid only in Manual mode)
gateway[4]	uint8	Default gateway of the

Structure Member Name	Structure Member type	Description
		TCP/IP stack (valid only in Manual mode)

6.3.11 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 socket has to be bound.
destSocket[2]	uint8	The destination's port. This port number is not valid for a listening socket.

Structure Member Name	Structure Member Type	Description
destIpaddr[4]	uint8	The destination's IP address. This IP address is not valid for a listening socket.

6.3.12 rsi_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.

6.3.13 rsi_spi_webserver.c

This file contains the API for the "Load webpage on Module" command.

API Prototype:

```
int16 rsi_webserver(rsi_uWebServer *uWebServer)
```

Parameters:

rsi_uWebServer *uWebServer

```
typedef union
 {
   struct {
        uint8                   webPageLen[2];
                uint8                   webServData[1024];
        }webServFrameSnd;
        uint8                   uWebServBuf[1026];
  }rsi_uWebServer;
```

Structure Member Name	Structure Member Type	Description
webPageLen[2]	uint8	Length of the page to be loaded
webServData[1024]	Uint8	web data to be loaded

6.3.14 rsi_spi_query_fwversion.c

This file contains the API for "Query Firmware Version" command.

API Prototype :

int16 rsi_query_fwversion(void)

Parameters:

None

6.3.15 rsi_spi_query_macaddress.c

This file contains the API for "Query MAC Address" command.

API Prototype:

int16 rsi_query_macaddress()

Parameters:

None

6.3.16rsi_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

Structure Member Name	Structure Member Type	Description
		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)

6.3.17 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)

6.3.18rsi_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

6.3.19 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

6.3.20 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

6.3.21 rsi_spi_disconnect.c

This file contains the API for the "Disassociate" command.

API Prototype:

int16 rsi_disconnect()

Parameters:

None

6.3.22 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

6.3.23 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

6.3.24 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

6.3.25 rsi_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

6.3.26 rsi_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 Data.

6.3.27 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 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.

6.3.28 rsi_spi_dns_query.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

6.3.29 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

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)

6.3.30 rsi_spi_set_region.c

This file contains API for the "Set Region" command.

API Prototype

int16 rsi_spi_set_region(rsi_uSetRegionReq *uSetRegionReqFrame)

Parameters

rsi_uSetRegionReq *uSetRegionReqFrame

```
typedef union {
   struct {
            uint8    uWiFi_region[2];
            uint8    uMax_power[2];
         } SetRegReqFrameSnd;
   uint8                uSetRegionBuf[4];
} rsi_uSetRegionReq ;
```

Structure Member Name	Description
WiFi_region	1-US 2-Europe and France 3–Japan

6.3.31 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

NOTE:

All the above APIs returns '-1', If buffer full is indicated from the module, then the user has to recall the API that is called earlier.

In case of a buffer full, the status will be updated by a buffer empty interrupt from module whenever the buffer is emptied.

The API Library contains other files like rsi_spi_framerdwr.c, rsi_lib_util.c which are for the library's internal usage. The user may ignore these files and their functionality.

6.4 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.

6.4.1 rsi_hal.h

This is the header file for the HAL layer.

6.4.2 rsi_hal_mcu_timers.c

This file implements MCU related delay functions.

a.Millisecond timer

API Prototype:

void rsi_delayMs (float delay)

{

}

Description:

This HAL API should contain the code to introduce a delay in milliseconds.

b.Microsecond timer

API Prototype:

```
void rsi_delayUs (float delay)
{
        volatile uint16 i;
        for(i=0;i< (delay *1000);i++);

}
```

This HAL API should contain the code to introduce a delay in micro seconds.

6.4.3 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

}

6.4.4 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)

{

a. Configure SPI MISO pin, SPI MOSI pin, SPI CLK pin, SPI CS pin
b. 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**.**
c. Configure clock polarity CPOL as '0' and clock phase CPHA as '0'
d. Configure transfer length to 8 bits.
e. Configure data mode as 'MSB first'
f. Configure SPI clock upto 12MHz (buad rate)
g. Configure external interrupt pin (Input) to receive interrupts from wifi module and register to ISR
h. Configure Reset pin (Output)
i. 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.

6.4.5 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.

6.5 Response Data Structures

This section describes important data structures used to read responses from the module.

6.5.1 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;                 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	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_MAX]	rsi_scanInfo	Scanned Access point information in following section
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_socketCloseFram			Response structure for "Close a Socket"

eRcv			
	socketDsc[2]	uint8	Descriptor of the socket closed
rsi_ippara mFrameRc v			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_qryNet ParmsFra meRcv			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[32]	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_DNSQryResponse			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 18: Read Response Data Structure in Driver

6.5.2 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];
} 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[32]	uint8	SSID of Access Point
bssid[6]	uint8	BSSID of scanned AP

6.6 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 32 – SSID length module supports for more information refer "Join" command section.

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_2P5GHZ 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 AUTO GO,17 for OPEN GO)

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

6.6.1 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.

6.6.1.1 Client mode with Personal Security

```
#define RSI_OPERMODE            0
#define CLIENT_MODE             ENABLE
#define P2P_MODE                DISABLE
#define ENTERPRISE_MODE         DISABLE
#define RSI_BAND                RSI_BAND_2P5GHZ
#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
```

6.6.1.2 WiFi Direct Mode

```
#define RSI_OPERMODE            1
#define CLIENT_MODE             DISABLE
#define P2P_MODE                ENABLE
#define ENTERPRISE_MODE         DISABLE
#define RSI_BAND                RSI_BAND_2P5GHZ
#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
```

6.6.1.3 Client Mode with Enterprise Security

```
#define RSI_OPERMODE            2
#define CLIENT_MODE             DISABLE
```

```
#define P2P_MODE                DISABLE
#define ENTERPRISE_MODE           ENABLE
#define EAP_METHOD              "TTLS"
#define INNER_METHOD             "\"auth=MSCHAPV2\""
#define USER_IDENTITY           "\"user1\""
#define PASSWORD               "\"test123\""
#define KEY_PASSWORD             "wifi"
```

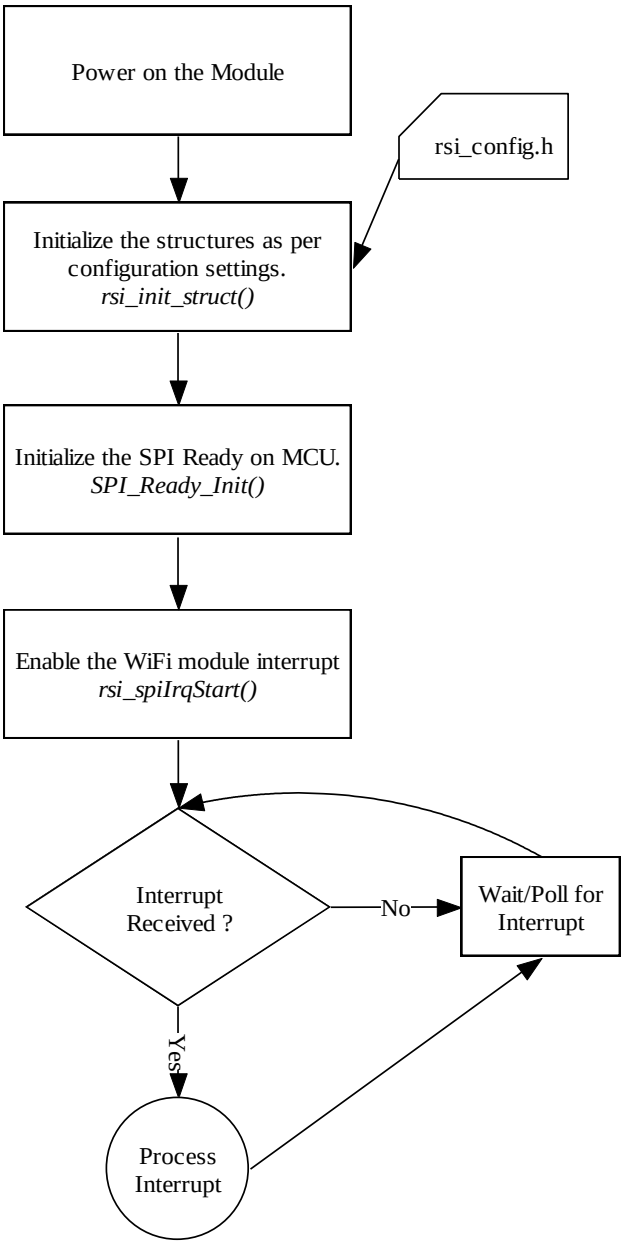
6.6.1.4 TCP/IP

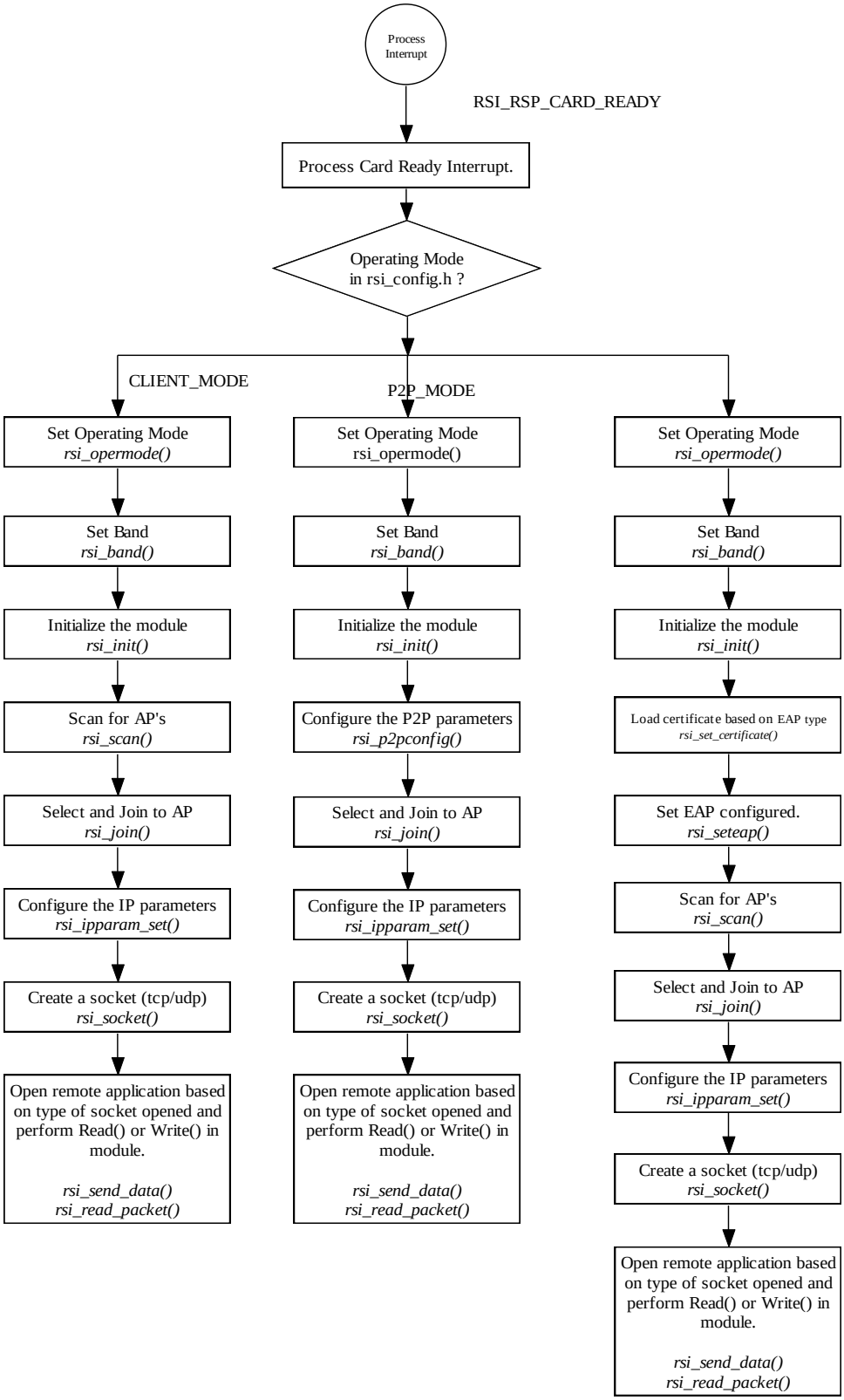
```
#define RSI_IP_CFG_MODE              RSI_DHCP_MODE_ENABLE
#define RSI_MODULE_IP_ADDRESS              "192.168.100.67"
#define RSI_NETMASK                     "255.255.255.0"
#define RSI_GATEWAY                     "192.168.100.1"
#define RSI_TARGET_IP_ADDRESS         "192.168.100.198"
#define RSI_SOCKET_TCP_CLIENT_TYPE    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
```

6.6.2 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
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()

6.6.3 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.

6.6.4 Power mode API usage

int16 rsi_power_mode(uint8 mode) :

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 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, sser 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 */
```

```
}
```

6.7 HTML Documentation

The index.html file inside RS.WSC.2.0.GENR.A.B.C.X.Y.Z\Resources\SPI\Driver\ \Documentation\html (Ref [3]) is the starting point for browsing the HTML based documentation.

7 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

7.1 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.

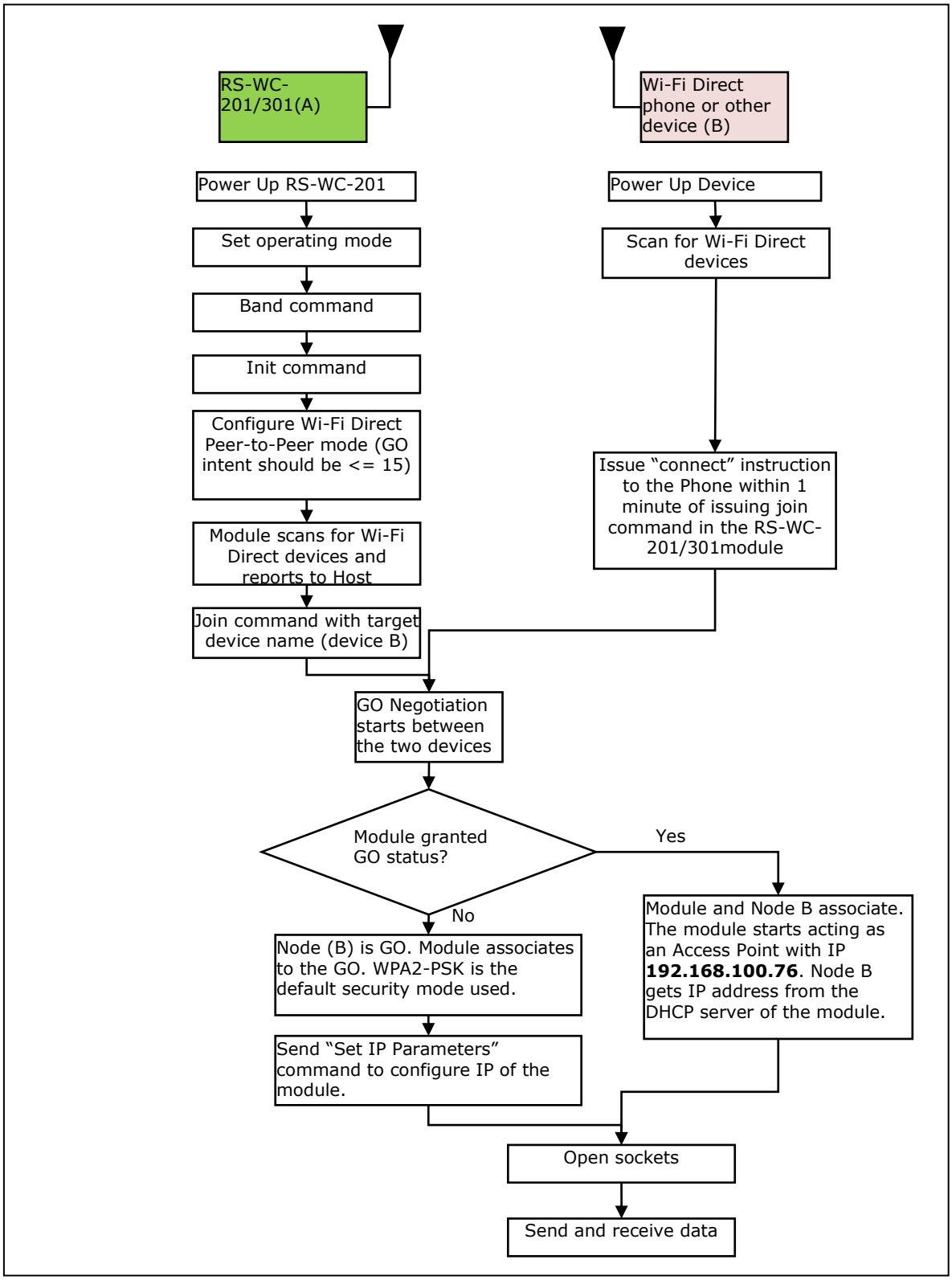


Figure 26: Wi-Fi Direct Peer-to-Peer Mode

7.2 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.

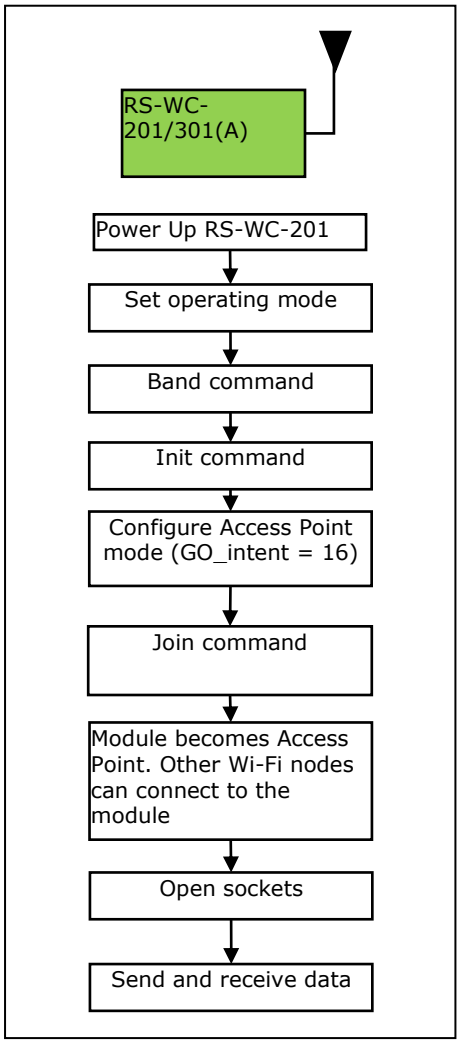


Figure 27: Access Point Mode

7.3 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.

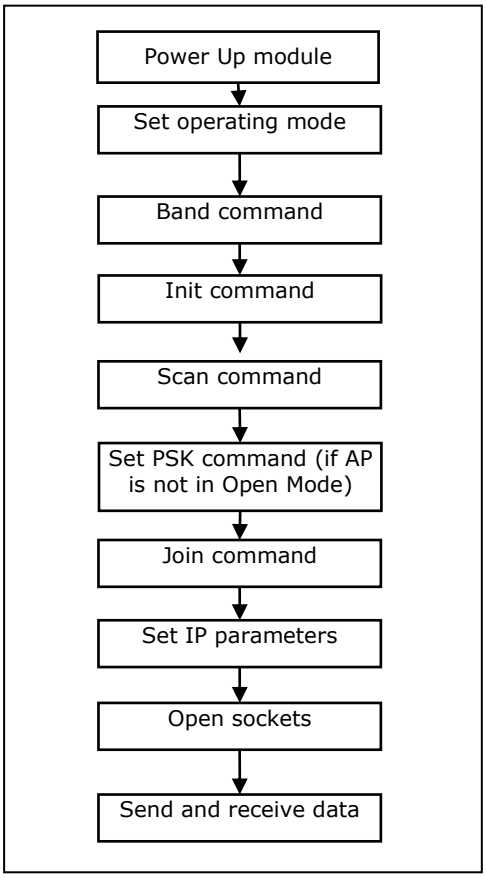


Figure 28: Client Mode with Personal Security

7.4 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.

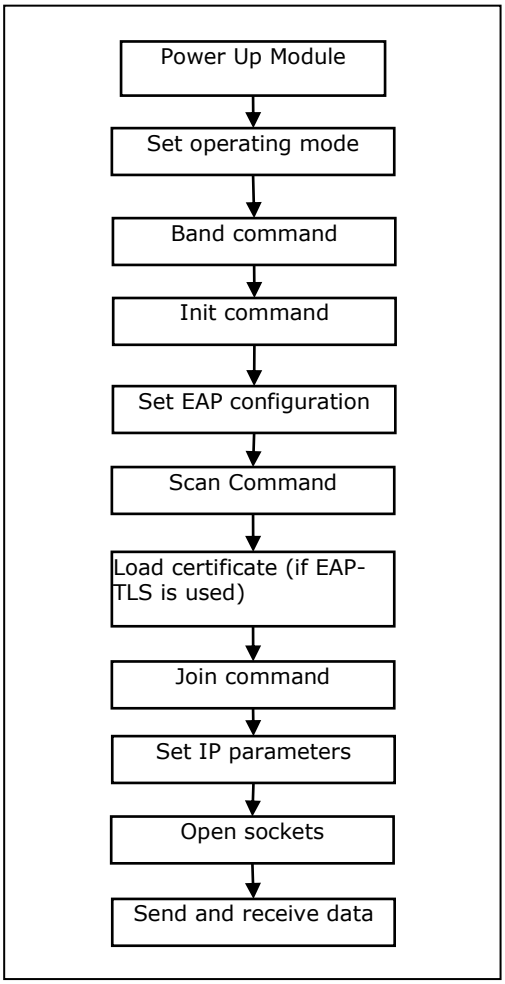


Figure 29: Client Mode with Enterprise Security

8 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

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=2,0,0,0,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=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,test,6,redpine,12345678\r\n

This command configures the module to act as an Access Point in channel 6, with SSID "DIRECT-xyredpine" and WPA2-PSK key of 12345678.

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_join=redpine,0,2\r\n

This command will create the Access Point with SSID DIRECT-xyredpine 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=1,0,0,0,This is a test\r\n

If the remote node (Device A) sends data, the module receives the data and transfers to the Host with a **AT+RSI_READ** message

Another client (Named "Device B" in this example) can also connect to the Access Point in the module and data transfer can be executed between Device A and Device B through the AP. A maximum of 4 clients are supported.

Associate to an Access Point (with WPA2-PSK security) as a client

at+rsi_opermode=0\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_setregion=1\r\n

Optional command to set the geographical domain.

at+rsi_scan=0\r\n

This command scans for Aps and reports the Aps found.

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=1,0,0,0,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=1,0,0,0,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_setregion=1\r\n

Optional command to set the geographical domain.

at+rsi_scan=0\r\n

This command scans for Aps and reports the Aps found

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=1,0,0,0,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.

9 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 WiFi 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-Di Direct Peer-to-Peer

This command starts the Wi-Fi Direct mode of the module. GOIntent should be less than or equal to 15)

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

Create an Access Point

Set Operating Mode

This command sets the operating mode of the module.

Band

This command sets the operating band of the module.

Init

This command initializes the module.

Configure Wi-Di Direct Peer-to-Peer

This command configures the module to act as an Access Point (GOIntent should be 16).

Set IP Parameters

This command can be used optionally in this flow to configure the IP 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.

Set Region

This is an optional command to set geographical domains.

Scan

This command scans for Aps and reports the Aps found.

Set PSK

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.

Set Region

This is an optional command to set geographical domains.

Scan

This command scans for Aps and reports the Aps found

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