Embedded BLE Software Programming Reference Manual (PRM)

Version 1.5

October 2018

Redpine Signals, Inc.

2107 North First Street, Suite #540, San Jose, California 95131,

United States of America.

Phone: +1-408-748-3385, Fax: +1-408-705-2019

Email: sales@redpinesignals.com

Website: www.redpinesignals.com

Disclaimer

The information in this document pertains to information related to Redpine Signals, Inc. products. This information is provided as a service to our customers, and may be used for information purposes only.

Redpine assumes no liabilities or responsibilities for errors or omissions in this document. This document may be changed at any time at Redpine's sole discretion without any prior notice to anyone. Redpine is not committed to updating this document in the future.

Table of Contents

About the Document

This document describes the commands to operate RS9116-WiSeConnect. The parameters in the commands and their valid values with the expected responses from the modules are also described. The document should be used by the developer to write software on the Host MCU to control and operate the module. Section AT Command Mode describes commands to operate the module using the UART / USB-CDC interfaces. Section Binary Command Mode describes Binary commands to operate the module using the SPI / USB / UART / USB-CDC / CORTEX-M4 interfaces.

1 Embedded BLE Overview

This document describes the commands to operate RS9116-WiSeConnect™. The parameters in the commands and their valid values with the expected responses from the modules are also described. The document should be used by the developer to write software on the Host MCU to control and operate the module. Section AT Command Mode describes commands to operate the module using the UART / USB-CDC interfaces. Section Binary Command Mode describes Binary commands to operate the module using the SPI / USB / UART / USB-CDC / CORTEX-M4 interfaces.

1.1 Architecture

RS9116-WiSeConnect APIs are designed in layers, where each layer is independent and uses the service of underlying layers.

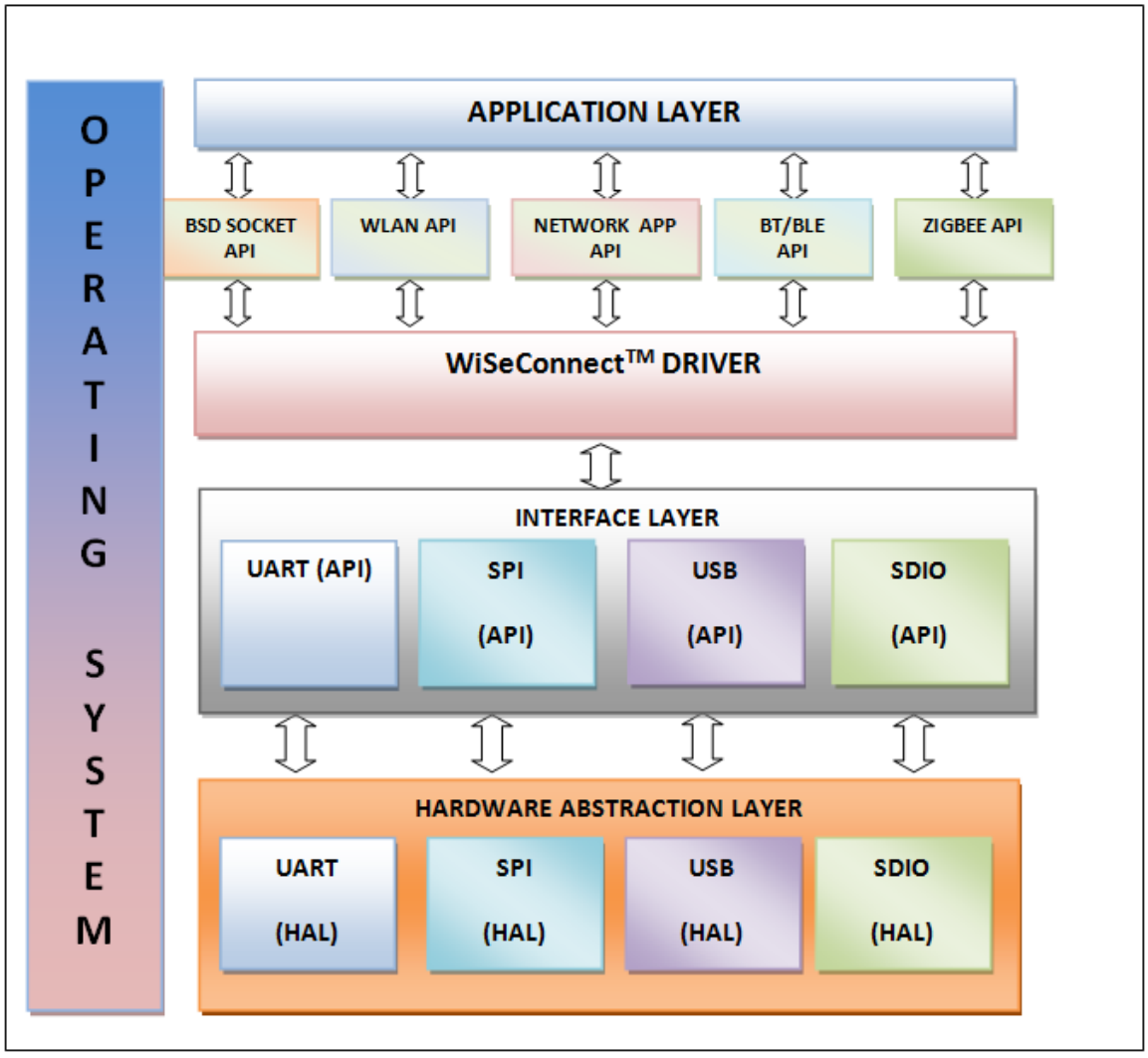


Figure 1: Architecture Overview

1.1.1 Bluetooth Software Architecture

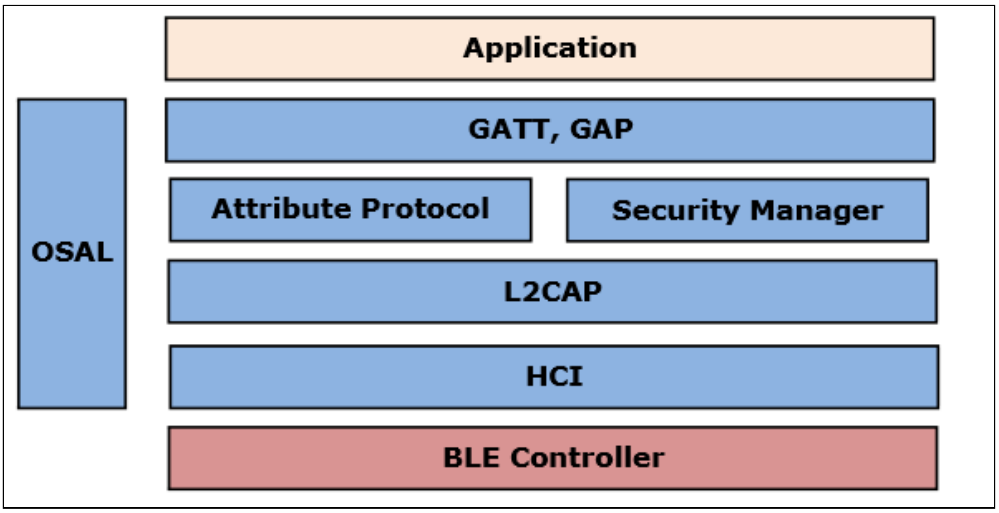


Figure 2: Bluetooth Software Architecture

1.1.2 Application

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

1.1.3 Profiles

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

1.1.4 Bluetooth Core

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

- SMP
- ATT
- L2CAP
- BLE Controller
- SMP is a security management protocol which provides services like pairing and key distribution.
- ATT (Attribute Protocol) provides a server to expose attribute values.
- L2CAP (Logical Link Control and Adaption Protocol) provides connection-oriented and connection less data services to upper layer protocols with data packet size upto 64KB in length.
- L2CAP performs the segmentation and reassemble of I/O packets from the baseband Controller.
- BLE Controller which includes link controller layers.

1.1.5 OS Abstraction Layer

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

1.1.6 Host-Module Interface

This document is primarily concerned with the host-module interface. The host can interface with RS9116-WiSeConnect using following list of interfaces to configure and send/receive data.

- SPI

- UART
- USB

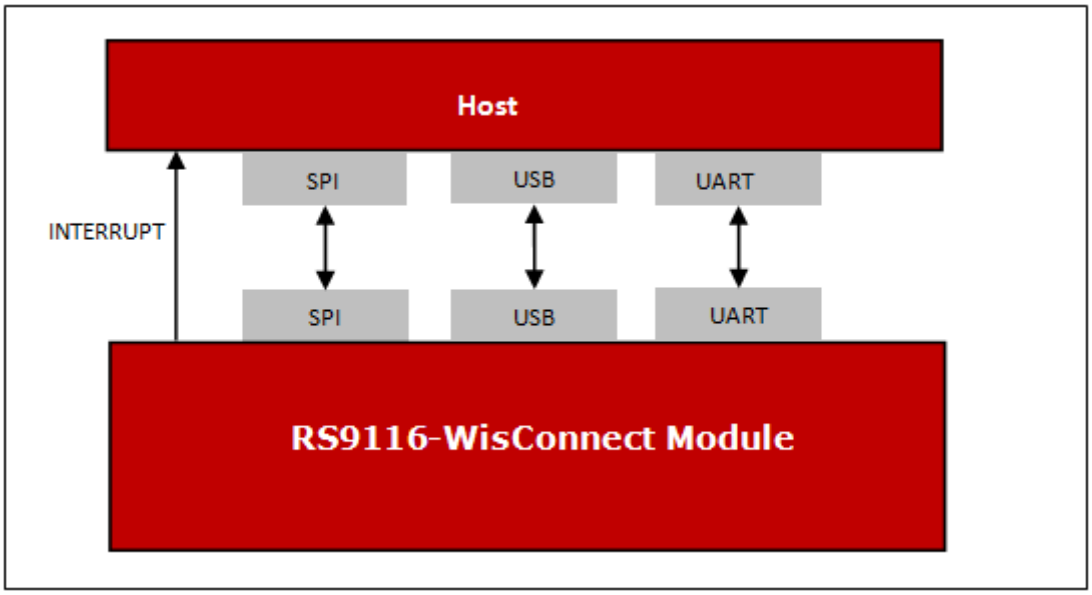


Figure 3: Host Interface Block Diagram

Note:
USB-CDC/UART/USB/SPI are the host interfaces for WiSeConnect.

2 Embedded BLE Related Resources

Name	Location on Redpine Document Portal
RS9116 Connectivity Family Common Documents	
RS9116 Connectivity Module Family Datasheet	/RS9116 Connectivity/Datasheets, Manuals, and Guides/
RS9116 EVK User Guide	/RS9116 Connectivity/EVK/
Module Integration Guide	/RS9116 Connectivity/Datasheets, Manuals, and Guides/
Regulatory and Compliance Certificates	
3D Models	/RS9116 Connectivity/CAD Files/
IBIS Models	/RS9116 Connectivity/CAD Files/
Application Notes	/Application Notes/
RS9116 WiSeConnect Documents	
WiSeConnect Getting Started Guide	TBD
WiSeConnect Software Package including examples	TBD
WiSeConnect Programmer's Reference Manual	TBD
WiSeConnect SAPI Guide	TBD
WiSeConnect SAPI Porting Guide	TBD
RS9116 n-Link Documents	
n-Link Software Package	TBD
n-Link Technical Reference Manual	TBD

3 Embedded Bootloader and Interfaces

3.1 Bootloader

3.2 Features

This document contain features that are supported by the Network and Security Processor (NWP) bootloader.

Basic Features

- Load Default Firmware
- Load Selected Firmware
- Upgrade Firmware from host
- Selecting Default images
- Enable / Disable Host interaction Bypass
- Supports for multiple host interfaces (SDIO / SPI / UART / USB / USB-CDC)
- Firmware Integrity Check
- Upgrading Keys
- JTAG Selection

The RS9116W supports two Boot loading modes:

1. **Host interaction (Non-bypass) mode:** In this mode host can interact with the bootloader and can give boot up options (commands) to configure different boot up operations. The host tells the module what operations it has to perform based on the selections made by the user.
2. **Bypass mode:** In this mode bootloader interactions are completely bypassed and use the stored bootup configurations (which are selected in host interaction mode) & load default firmware image in the module. This mode is recommended for final production software to minimize the boot up time.

3.3 Host Interaction Mode

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

3.3.1 Host Interaction Mode in UART / USB-CDC

This section explains the host interaction mode in UART / USB CDC mode.

3.3.1.1 Startup Operation

After powering up, Host is required to carry out ABRD (Auto Baud Rate Detection) operation and after successful ABRD, the module will display menu of boot up options to host. Host needs to select the appropriate option.

Note :

On powerup, bootloader checks the integrity of the bootup options. If the integrity fails, it computes the integrity from backup. If integrity passes, it copies the backup to the actual location. If integrity of the backup options also fail, the boot up options are reset/cleared. In either of the cases, bootloader bypass is disabled and corresponding error messages are given. In the case of integrity failure, **"LAST CONFIGURATION NOT SAVED**" is displayed when the backup integrity passes and **"BOOTUP OPTIONS CHECK SUM FAILED"** is displayed when the backup integrity also fails before displaying the bootup options.

Hyper Terminal Configuration

RS9116W uses the following UART interface configuration for communication:

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

Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

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

- Open HyperTerminal and enter any name in the **"Name"** field. After this, click **"OK"** button. Here, **"WiSeConnect"** is entered as shown below:

Note:

Default baud rate of the module is 115200.

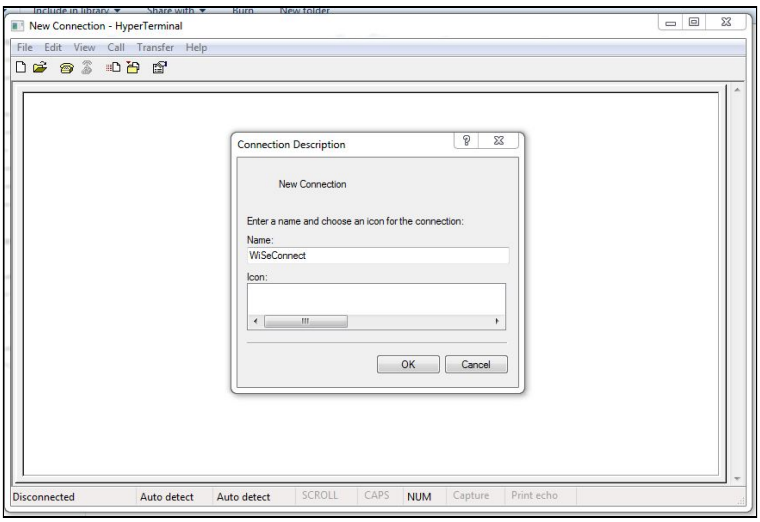


Figure 4: HyperTerminal Name field Configuration

- After clicking **"OK"**, the following dialog box is displayed as shown in the figure below.

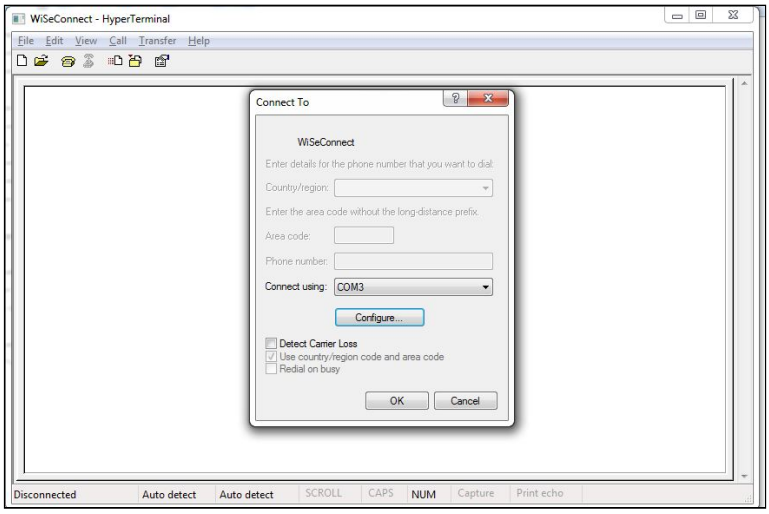


Figure 5: HyperTerminal COM port field Configuration

- In the **"Connect using"** field, select appropriate com port. In the figure above COM3 is selected. Click **"OK"** button.
- After clicking **"OK"** button the following dialog box is displayed as shown in the figure below.

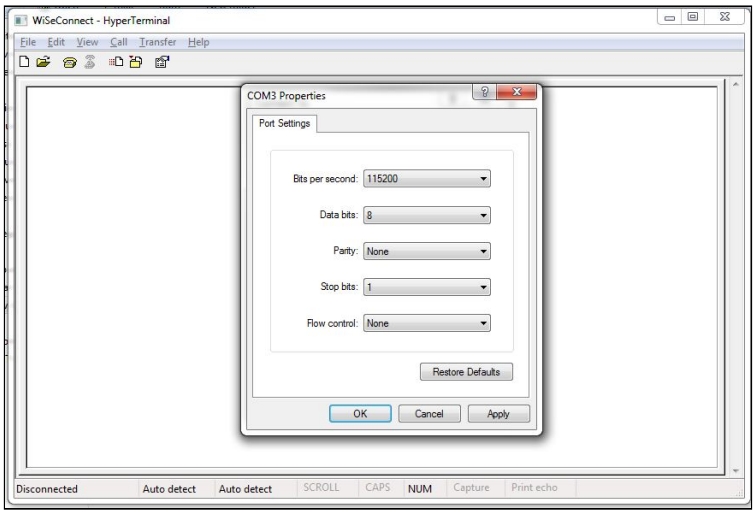


Figure 6: HyperTerminal Baud rate field Configuration

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

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

Auto Baud Rate Detection (ABRD)

The RS9116 automatically detects the baud rate of the Host's UART interface by exchanging some bytes. The Host should configure the UART interface for the following parameters for ABRD detection.
RS9116 uses the following UART interface configuration for communication:

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

Data bits: 8

Stop bits: 1

Parity: None

Flow control: None

To perform ABRD on the RS9116W, the host must follow the procedure outlined below.

1. Configure the UART interface of the Host at the desired baud rate.
2. Power on the RS9116W.
3. The Host, after releasing the module from reset, should wait for 20 ms for initial boot-up of the module to complete and then transmit 0x1C at the baud rate with which its UART interface is configured. After transmitting '0x1C' to the module, the Host should wait for the module to transmit 0x55 at the same baud rate.
4. If the '0x55' response is not received from the module, the host has to re-transmit 0x1C, after a delay of 200ms.
5. After finally receiving '0x55', the host should transmit '0x55' to the module. The module is now configured with the intended baud rate.

Note:

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

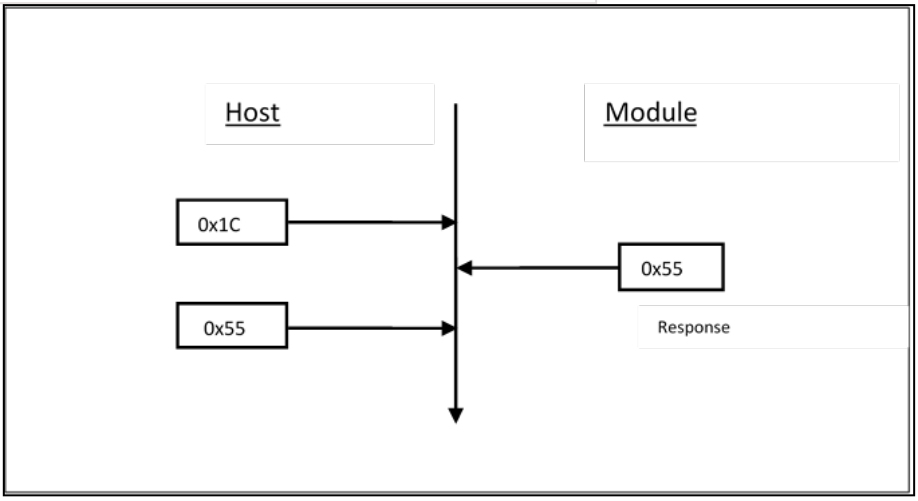


Figure 7: ABRD exchange between Host and module

Below are the bootup options, firmware upgrade and firmware loading procedure for WiSeConnect product.

3.3.1.2 Start up messages on power-up

After powering up the module and performing ABRD you will see a welcome message on host, followed by boot up options:

Note:

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

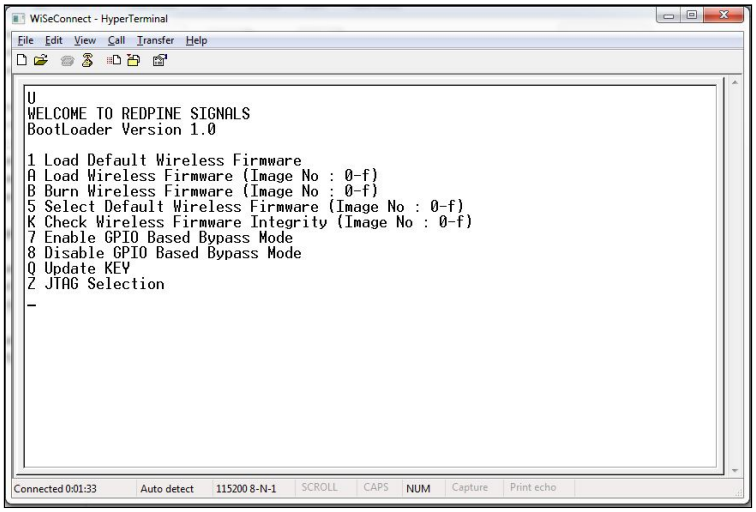


Figure 8: RS9116-WiSeConnect Module UART / USB-CDC Welcome Message

3.3.1.3 Loading the default wireless firmware in the module

To load the default firmware flashed onto the module, choose Option 1: "Load Default Wireless Firmware " .

3.3.1.4 Load Default Wireless Firmware

- After welcome message is displayed as shown in the above figure, select option 1 "Load Default Wireless Firmware " for loading image.

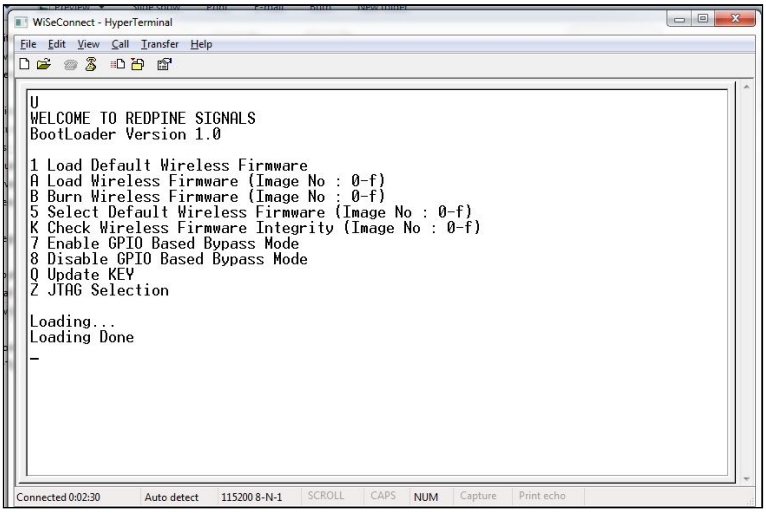


Figure 9: RS9116-WiSeConnect Module UART / USB-CDC Default Firmware Loaded

3.3.2 Loading selected Wireless Firmware in the Module

To load the selected firmware (from flash) onto the module, choose Option A: "Load Wireless Firmware (Image No : 0-f)" .

3.3.2.1 Load Wireless Firmware

- After welcome message is displayed as shown in the above figure, select option A "Load Wireless Firmware (Image No : 0-f)" for loading Image.
- In response to the option A, Module ask to enter image no.
- Select the image number to be loaded from flash.
- After successfully loading the default firmware, "Loading Done" message is displayed.
- After firmware loading is completed, module is ready to accept commands.

Note:

1. In order to use host bypass mode user has to select one of the image as default image by selecting options 5 (Select Default Wireless Firmware).
2. In Host interaction mode if there is no option selected after bootup menu for 20 seconds then bootloader will load selected Wireless default image.
3. If valid firmware is not present, then a message prompting "Valid firmware not present" will be displayed.

Firmware Upgradation
After powering up the module, a welcome message is displayed.

Upgrade NWP firmware Image

- After the welcome message is displayed, select option B "Burn Wireless Firmware (Image No : 0-f)" to upgrade Wireless Image.
- The message "Enter Wireless Image No (0-f)"
- Then select the Image no to be upgraded.
- The message "Send RS9116.NBZ.WC.GENR.x.x.x.rps" should appear as shown in the figure below.

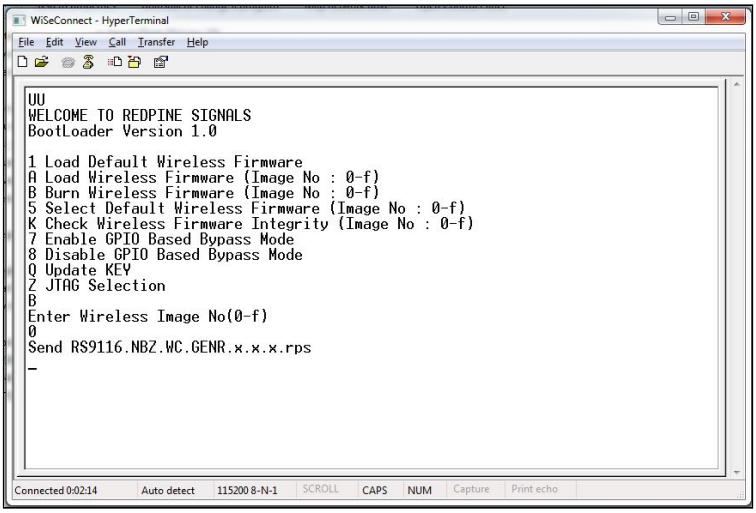


Figure 10: RS9116-WiSeConnect Module Firmware Upgrade File Prompt Message

- In the "File" menu of HyperTerminal, select the "send file" option. A dialog box will appear as shown in the figure below . Browse to the path where "RS9116.NBZ.WC.GENR.X.X.X.rps" is located and select Kermit as the protocol option. After this, click the "Send" button to transfer the file.

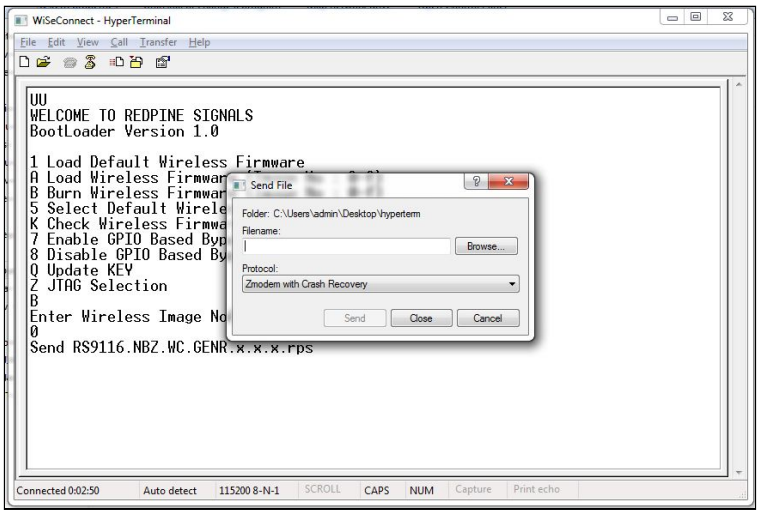


Figure 11: RS9116-WiSeConnect Module Firmware Upgrade File Selection Message

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

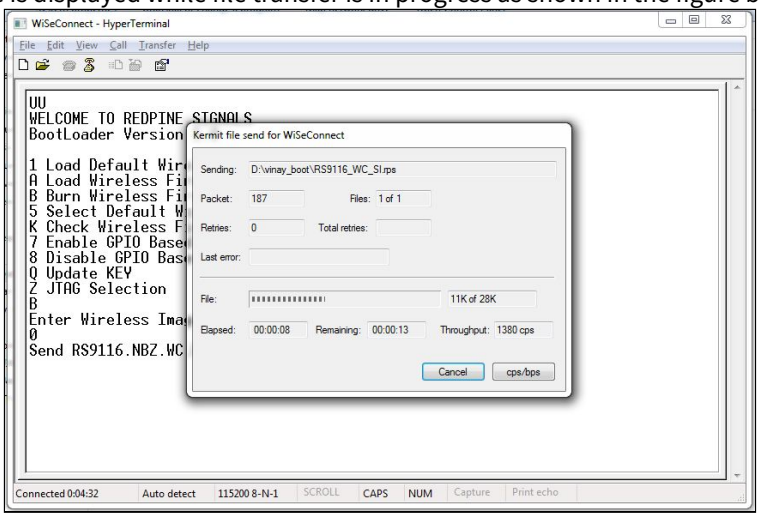


Figure 12: RS9116-WiSeConnect Module Firmware Upgrade File transfer Message

After successfully completing the file transfer, module computes the integrity of the image and displays "@@@@@@@Upgradation Failed, re-burn the image@@@@@@@ " in the case of failure and "@@Upgradation Failed and default image invalid, Bypass disabled @@" in the case of both failure and corruption of the default image.

- In the case of success, module checks if bootloader bypass is enabled and computes the integrity of the default image selected. If the integrity fails, it sends "Upgradation successful, Default image invalid, gpio bypass disabled." If integrity passes or gpio bypass not enabled, it sends "Upgradation Successful" message on terminal as shown in the figure below.

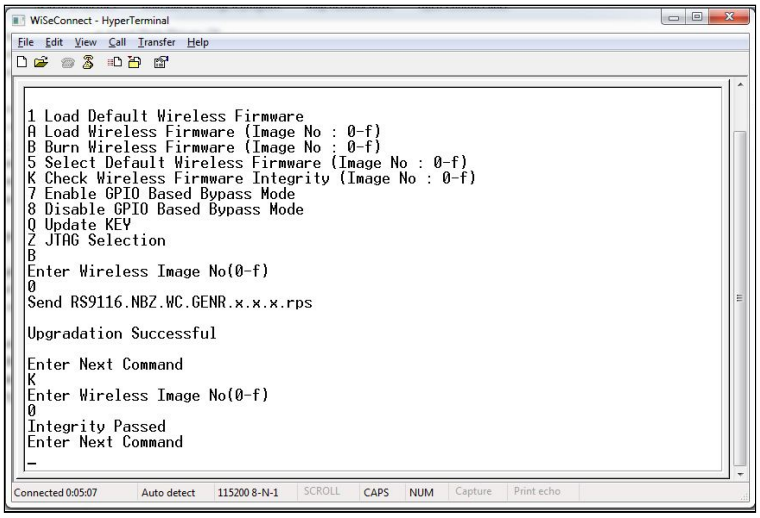


Figure 13: RS9116-WiSeConnect Module Firmware Upgrade Completion Message

- At this point, the upgraded firmware image is successfully flashed to the module.
- User can again cross check the integrity of the Image by selecting the Option K " Check Wireless Firmware Integrity (Image No : 0-f)" for Wireless Image.
- Follow the steps mentioned in the section **Loading the Default Wireless Firmware in the Module** to load the firmware from flash, select Option 1 from the above shown figure.
- The module is ready to accept commands from the Host.

Note:

This host interaction mode is **not applicable** to WiSeMCU.

3.3.3 Host Interaction Mode in SPI / USB

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

3.3.4 SPI Startup Operations

If the selected host interface is SPI or USB, the Bootloader uses two of the module hardware registers to interact with the host. In case of SPI host, these registers can be read or written using SPI memory read/write operations. In case of USB host vendor specific reads and writes on control end point are used to access these registers. Bootloader indicates its current state through HOST_INTF_REG_OUT and host can give the corresponding commands by writing onto HOST_INTF_REG_IN.
The significance of 4 bytes of the value written in to HOST_INTF_REG_IN register is as follows:

Nibble[1:0]	Message code.
Nibble[2]	Represents the Image number based on command type.
Nibble[3]	Represents the validity of the value in HOST_INTF_REG_IN register. Bootloader expects this value to be 0xA(HOST_INTERACT_REG_VALID). Booltloader validates the value written into this register if and only if this value is 0xA
Nibble[7:4]	Represents Mode for JTAG selection

Table 3- 1: HOST_INTF_REG_IN Register values

HOST_INTF_REG_OUT	0x4105003C
HOST_INTF_REG_IN	0x41050034
PING Buffer	0x18000
PONG Buffer	0x19000

Table 3 - 2: Bootloader message exchange registers

HOST_INTERACT_REG_IN_VALID		0xA000
HOST_INTERACT_REG_OUT_VALID		0xAB00
LOAD_DEFAULT_NWP_FW	'1'	0x31
LOAD_NWP_FW	'A'	0x41
BURN_NWP_FW	'B'	0x42
SELECT_DEFAULT_NWP_FW	'5'	0x35
ENABLE_GPIO_BASED_BYPASS	'7'	0x37
DISABLE_GPIO_BASED_BYPASS	'8'	0x38
CHECK_NWP_INTEGRITY	'K'	0x4B
KEY_UPDATE	'Q'	0x51
JTAG_SELECTION	'Z'	0x5A
LOAD_DEFAULT_NWP_FW_ACTIVE_LOW		0x71
LOAD_NWP_FW_ACTIVE_LOW		0x72
SELECT_DEFAULT_NWP_FW_ACTIVE_LOW		0x75
EOF_REACHED	'E'	0x45
PING_BUFFER_VALID	'I'	0x49
JUMP_TO_ZERO_PC	'J'	0x4A
INVALID_ADDRESS	'L'	0x4C
PONG_BUFFER_VALID	'O'	0x4F
POLLING_MODE	'P'	0x50
CONFIGURE_AUTO_READ_MODE	'R'	0x52
DISABLE_FWUPGRADE	'T'	0x54
ENABLE_FWUPGRADE	'N'	0x4E
DEBUG_LOG	'G'	0x47

Table 3 - 3: Bootloader Message Codes

LOADING_INITIATED	'1'	0x31

VALID_FIRMWARE_NOT_PRESENT	'#'	0x23
SEND_RPS_FILE	'2'	0x32
UPGRADATION_SUCCESFULL	'S'	0x53
PONG_AVAIL	'O'	0x4F
PING_AVAIL	'I'	0x49
CMD_PASS		0xAA
CMD_FAIL		0xCC
FLASH_MEM_CTRL_CRC_FAIL		0xF1
FLASH_MEM_CTRL_BKP_CRC_FAIL		0xF2
INVALID_CMD		0xF3
UPGRADATION_SUCCESFULL_INVALID_DEFAULT_IMAGE		0xF4
INVALID_DEFAULT_IMAGE		0xF5
UPGRADATION_FAILED_INVALID_DEFAULT_IMAGE		0xF6
IMAGE_STORED_IN_DUMP		0xF7
FLASH_NOT_PRESENT		0xFA
IMAGE_INTEGRITY_FAILED		0xFB
BOOT_FAIL		0xFC
BOARD_READY	(BL_VER_HIGH << 4 | BL_VER_LOW)	0x10
LAST_CONFIG_NOT_SAVED		0xF1
BOOTUP_OPTIONS_INTEGRITY_FAILED		0xF2
FWUP_BUFFER_VALID		0x5AA5

Table 3 - 4: Bootloader Response Codes

3.3.5 SPI Startup Messages on Powerup

Upon power-up the HOST_INTF_REG_OUT register will hold value 0xABxx. Here 0xAB (HOST_INTERACT_REG_OUT_VALID) signifies that the content of OUT register is valid. Bootloader checks for the integrity of the boot up options by computing CRC. If the integrity fails, it check the integrity from backup. If integrity passes, it copies the backup to the actual location and writes (HOST_INTERACT_REG_OUT_VALID | LAST_CONFIG_NOT_SAVED) in HOST_INTF_REG_OUT register. If integrity of backup options also fails, the boot up options are reset and (HOST_INTERACT_REG_OUT_VALID | BOOTUP_OPTIONS_INTEGRITY_FAILED) is written in HOST_INTF_REG_OUT register. In either of the cases, bootloader bypass is disabled. If the boot up options integrity passes, HOST_INTF_REG_OUT register contains 0xABxx where xx represents the two nibble bootloader version. This message is referred as BOARD_READY indication throughout the document.
For instance, for bootloader version 1.0, value of register will be 0xAB10. Host is expected to poll for one of the three values and should give any succeeding command (based on error codes if present) only after reading the correct value in HOST_INTF_REG_OUT reg.

3.4 Loading Default Wireless Firmware in the Module

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

3.4.1 Load Default Wireless Firmware

Upon receiving Boardready, if host wants to load default wireless firmware, it is expected to write value (HOST_INTERACT_REG_IN_VALID | LOAD_DEFAULT_NWP_FW) or (HOST_INTERACT_REG_IN_VALID | LOAD_DEFAULT_NWP_FW_ACTIVE_LOW) in HOST_INTF_REG_IN register. If host command is (HOST_INTERACT_REG_IN_VALID | LOAD_DEFAULT_NWP_FW) it is assumed that host platform is expecting active

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

Note:

For USB host interface mode interrupt configuration is not required, host should send (HOST_INTERACT_REG_IN_VALID | LOAD_DEFAULT_TA_FW) to load default wireless Image.

3.5 Loading Selected Wireless Firmware in the Module

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

3.5.1 Load Selected Wireless Firmware

Upon receiving Boardready, if host wants to load a selected wireless firmware, it is expected to write value (HOST_INTERACT_REG_IN_VALID | LOAD_NWP_FW | (IMAGE_NO << 8)) or (HOST_INTERACT_REG_IN_VALID | LOAD_NWP_FW_ACTIVE_LOW | (IMAGE_NO << 8)) in HOST_INTF_REG_IN register. If host command is (HOST_INTERACT_REG_IN_VALID | LOAD_NWP_FW | (IMAGE_NO << 8)) it is assumed that host platform is expecting active high interrupts. If command is (HOST_INTERACT_REG_IN_VALID | LOAD_DEFAULT_NWP_FW_ACTIVE_LOW) it is assumed that host is expecting active low interrupts and SPI hardware will be configured accordingly. After sending this command host should wait for interrupt for card ready message from loaded firmware.

Note:

For USB host interface mode interrupt configuration is not required, host should send (HOST_INTERACT_REG_IN_VALID | LOAD_NWP_FW | (IMAGE_NO << 8)) to load selected wireless Image

3.6 Upgrading Wireless Firmware in the Module

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

3.6.1 Upgrading Wireless Firmware

Steps for firmware upgradation sequence after receiving board ready are outlined below:

1. After reading the valid BOARD_READY (i.e. HOST_INTERACT_REG_OUT_VALID | BOOTLOADER VERSION) value in HOST_INTF_REG_OUT, host writes (HOST_INTERACT_REG_IN_VALID | BURN_NWP_FW | (IMAGE_NO << 8)) in HOST_INTF_REG_IN and host sTArts polling for HOST_INTF_REG_OUT.
2. Module polls for HOST_INTF_REG_IN register. When module reads a valid value (i.e. HOST_INTERACT_REG_IN_VALID | BURN_NWP_FW | (IMAGE_NO << 8)) in HOST_INTF_REG_IN, module writes (HOST_INTERACT_REG_OUT_VALID | SEND_RPS_FILE) in HOST_INTF_REG_OUT.
3. When host reads valid value (i.e. HOST_INTERACT_REG_OUT_VALID | SEND_RPS_FILE) in HOST_INTF_REG_OUT, host will write first 4Kbytes of firmware image in PING Buffer and writes (HOST_INTERACT_REG_IN_VALID | PING_VALID) in HOST_INTF_REG_IN register. Upon receiving PING_VALID command module starts burning this 4k bytes chunk onto the flash. When module is ready to receive data in PONG Buffer it sets value PONG_AVAIL (HOST_INTERACT_REG_OUT_VALID | PONG_AVAIL) in HOST_INTF_REG_OUT. Host is required to wait for this value to be set before writing next 4Kbytes chunk onto the module.
4. On reading valid value (i.e. HOST_INTERACT_REG_OUT_VALID | PONG_AVAIL) in HOST_INTF_REG_OUT, host starts memory write on PONG location and start polling for HOST_INTF_REG_OUT to read valid value (i.e. HOST_INTERACT_REG_OUT_VALID | PING_AVAIL). Module reads (HOST_INTERACT_REG_OUT_VALID | PONG_VALID) 0xAB4F value in HOST_INTF_REG_OUT and begin to write the data from PONG location into the flash.

5. This write process continues until host has written all the data into the PING-PONG buffers and there is no more data left to write.
6. Host writes a (HOST_INTERACT_REG_IN_VALID | EOF_REACHED) in to HOST_INTF_REG_IN register and start polling for HOST_INTF_REG_OUT.
7. On the other side module polls for HOST_INTF_REG_IN register. When module reads (HOST_INTERACT_REG_IN_VALID | EOF_REACHED) in HOST_INTF_REG_IN , it computes integrity for entire received firmware image. Then it checks if bypass is enabled. If enabled, it checks for the validity of the default image. If integrity is correct and default image is valid/GPIO bypass not enabled, module writes (HOST_INTERACT_REG_OUT_VALID | FWUP_SUCCESSFUL) in HOST_INTF_REG_OUT register . If integrity is correct and default image is invalid (bypass enabled), module writes (HOST_INTERACT_REG_OUT_VALID | UPGRADATION_SUCCESFULL_INVALID_DEFAULT_IMAGE) in HOST_INTF_REG_OUT register and bypass is disabled. If the integrity is failed and default image is valid / GPIO bypass is not enabled, then the module writes(HOST_INTERACT_REG_OUT_VALID | CMD_FAIL) in HOST_INTF_REG_OUT register . If the integrity is failed and default image is invalid (bypass enabled), then the module writes (HOST_INTERACT_REG_OUT_VALID | UPGRADATION_FAILED_INVALID_DEFAULT_IMAGE) in HOST_INTF_REG_OUT register and bypass is disabled.

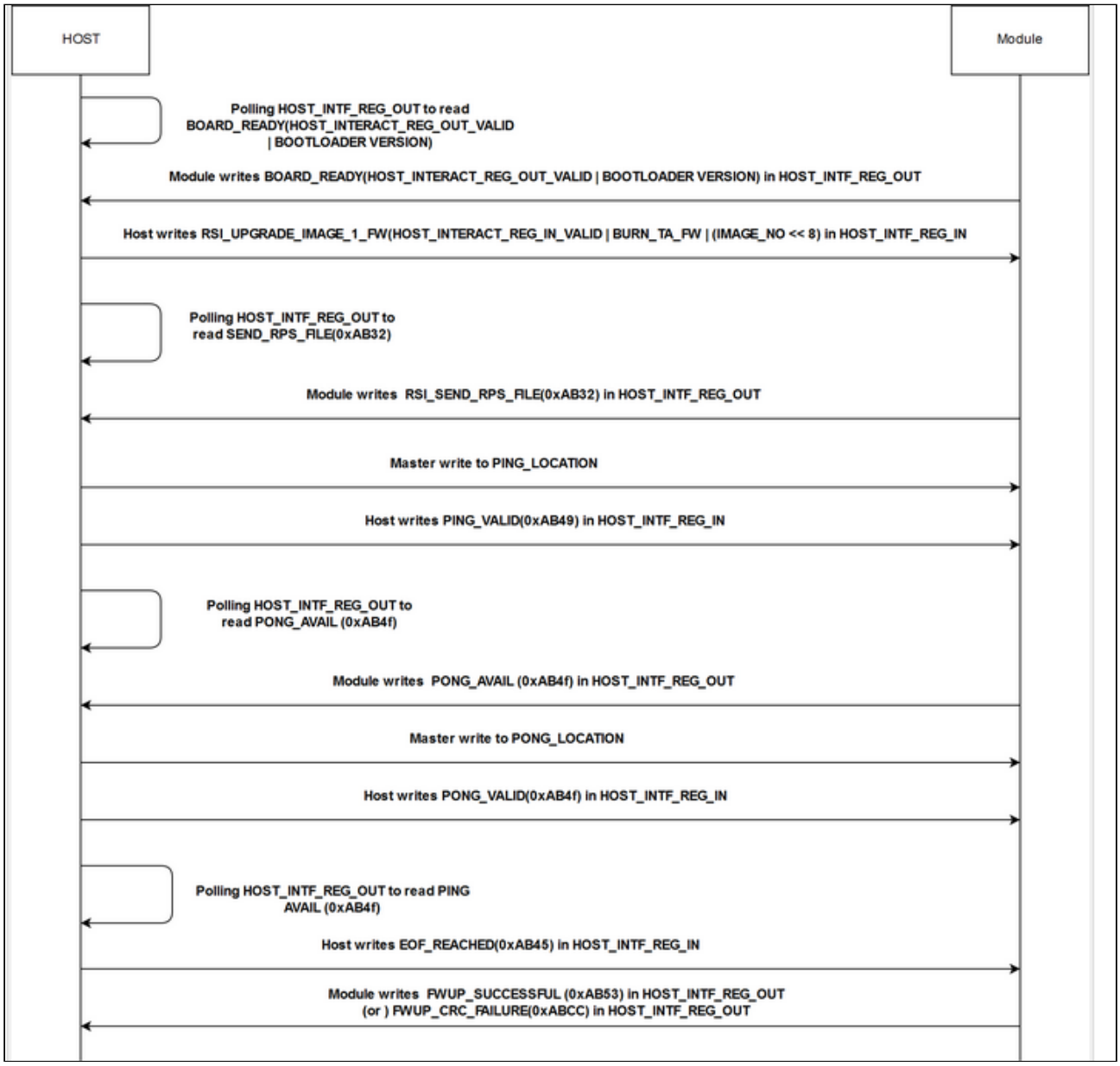


Figure 14: Wireless Firmware upgradation through SPI/USB

3.7 GPIO Based Bootloader Bypass Mode for WiSeConnect Product

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

1. Host need to select default wireless image to load in bypass mode and "Enable" Bootloader bypass mode.
2. Assert LP_WAKEUP to Bypass Bootloader on powerup.

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

Note:

This host interaction mode is **not applicable** to WiSeMCU.

3.8 Bypass Mode in UART / USB-CDC

3.8.1 Making Default Wireless Firmware Selection

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

3.8.1.1 Selecting a valid image as the default image

- After the welcome message is displayed, we can select option 5 "Select Default Wireless Firmware (Image No : 0-f)".
- The message "Enter Wireless Image No (0-f)"
- Then select the image no.
- It is better to check the integrity of image before selecting it as default image.
- When default image is selected, module checks for the validity of the image selected and displays "Configuration saved".

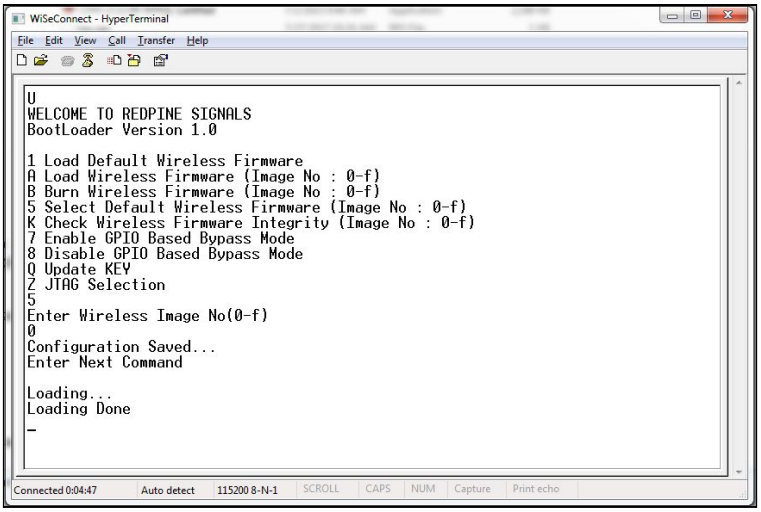


Figure 15: Making Image no - 0 as default image

3.8.2 Enable/Disable GPIO Based Bypass Option

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

3.8.2.1 Enabling the GPIO Based Bypass Mode

- If you select option 7, GPIO based Bootload bypass gets enabled.
- When this option is selected, module checks for the validity of the image selected and displays "Configuration saved" if valid.
- If valid default image is not present, then a message saying "Default image invalid" will be displayed.
- Once enabled, from next bootup, Bootloader will latch the value of LP_WAKEUP. If asserted, it will bypass the whole boot loading process and will load the default firmware image selected.
- After the welcome message is displayed, select option 5 "Select Default Wireless Firmware (Image No : 0-f)".
- The window will display "Enter Wireless Image No. (0-f)".

- Select the required image no.
- It is better to check the integrity of image before selecting it as default Image.
- When default image is selected, the module checks for the validity of the image selected and displays "Configuration saved".
- After this, select option 7 "Enable GPIO Based Bypass Mode".
- The module responds to select the host interface in Bypass mode (0 - UART , 1 - SDIO , 2 - SPI , 4 - USB , 5 - USB-CDC).
- Select the required interface.

If the default image is valid, then it enables GPIO Bypass mode , other wise it will not enable the GPIO Bypass mode.

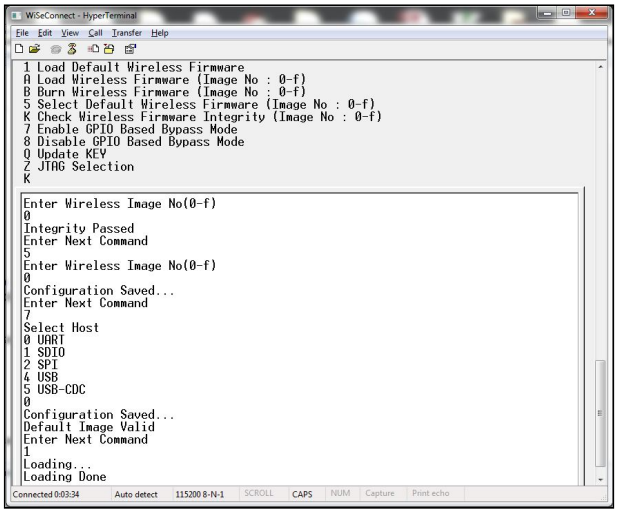


Figure 16(a):Enabling the GPIO based bypass mode; Valid Default Firmware

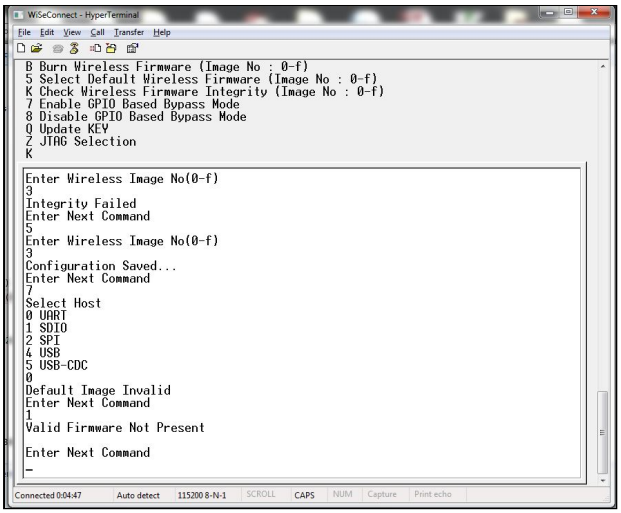


Figure 16(b):Enabling the GPIO based bypass mode; Invalid Firmware

3.8.2.2 Disabling the GPIO Based Bypass Mode

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

Note:

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

3.8.3 Check Integrity of the Selected Image

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

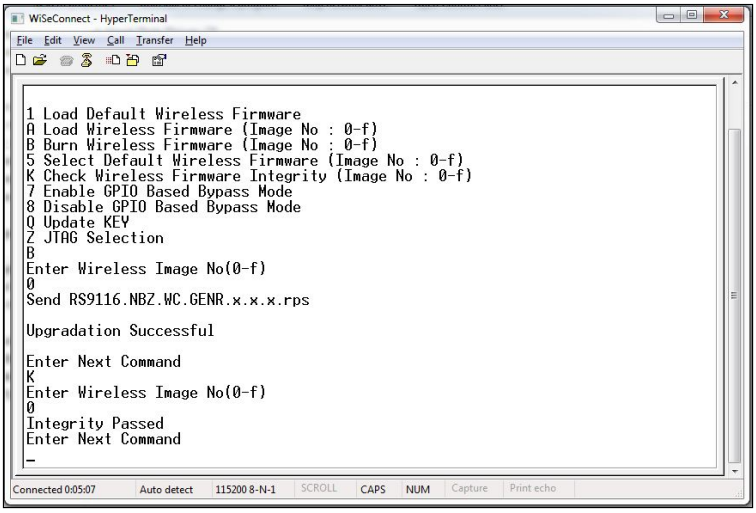


Figure 17: Integrity Check passed

Note:

This host interaction mode is **not applicable** to WiSeMCU.

3.9 Bypass Mode in SPI / USB

3.9.1 Selecting the Default Image

Upon receiving Boardready, if host wants to select default functional image, it is expected to write value (HOST_INTERACT_REG_IN_VALID | SELECT_DEFAULT_NWP_FW | (IMAGE_NO << 8)) in HOST_INTF_REG_IN register. Host is expected to receive confirmation (HOST_INTERACT_REG_OUT_VALID | CMD_PASS) in HOST_INTF_REG_OUT register if default image is valid. If default image is invalid, (HOST_INTERACT_REG_OUT_VALID | CMD_FAIL) is written in HOST_INTF_REG_OUT register.

3.9.2 Enable/Disable GPIO Based Bootloader Bypass Mode

1. Upon receiving Boardready, if host wants to enable or disable GPIO based bypass mode, it is expected to write value (HOST_INTERACT_REG_IN_VALID | ENABLE_GPIO_BASED_BYPASS) or (HOST_INTERACT_REG_IN_VALID | DISABLE_GPIO_BASED_BYPASS) in HOST_INTF_REG_IN register.
2. The Host expected to receive (HOST_INTERACT_REG_IN_VALID | ENABLE_GPIO_BASED_BYPASS) or (HOST_INTERACT_REG_IN_VALID | DISABLE_GPIO_BASED_BYPASS) in HOST_INTF_REG_OUT register if command is successful. If default image is invalid, (HOST_INTERACT_REG_OUT_VALID | INVALID_DEFAULT_IMAGE) is written in HOST_INTF_REG_OUT register. The host is expected to reboot the board after receiving confirmation (HOST_INTERACT_REG_IN_VALID | ENABLE_GPIO_BASED_BYPASS) or (HOST_INTERACT_REG_IN_VALID | DISABLE_GPIO_BASED_BYPASS) or (HOST_INTERACT_REG_OUT_VALID | INVALID_DEFAULT_IMAGE) in HOST_INTF_REG_OUT register.

3. If GPIO based bypass is enabled, from next bootup onwards, bootloader will latch the state of LP_WAKEUP. If LP_WAKEUP is asserted, bootloader will not give board ready and it will directly load the default functional image selected.

Note:

LP_WAKEUP need to be de-asserted on power up to move to host interaction mode, to select bootup options like disable Bypass mode or to change default image.

3.10 Other Operations

This section contains additional, less frequently used bootloader options.

3.11 Update KEY

NOTE: This feature is not enabled in current release.

3.12 JTAG Selection

NOTE: This feature is not enabled in current release.

Note :

This host interaction mode is **not applicable** to WiSeMCU.

3.13 SPI Interface

This section describes RS9116-WiSeConnect SPI interface, including the commands and processes to operate the module via SPI.

3.14 Features

The features are as follows:

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

3.15 Hardware Interface

The SPI interface on the RS9116-WiSeConnect works in slave mode. It is a 4-wire interface. In addition to the SPI interface, the module provides additional interrupt pin to signal events to the host.
The interrupt is raised by the module in SPI mode for the following condition.

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

The interrupt from module is active high and host has to configure the interrupt in level trigger mode.
The RS9116W also supports edge triggered interrupts, provided the host is configured to handle these appropriately (This is generic and does not entail any specific implementations). The host must check for pending interrupts before clearing / acknowledging an interrupt to avoid missing interrupts and data.

Note:

By default, the interrupt from the module is active high but, it can be configured as active low as well (this can be done from the bootloader menu).

3.15.1 SPI Signals

The SPI Interface is a full duplex serial Host interface, which supports 8-bit and 32-bit data mode. The SPI interface of the module consists of the following signals:
SPI_MOSI (Input) - Serial data input for the module.
SPI_MISO (Output) - Serial data output for the module.
SPI_CS (Input) - Active low slave select signal. This should be low when SPI transactions are to be carried out.
SPI_CLK (Input) - SPI clock. Maximum value allowed is 80 MHz
INTR (Output) - Active high (Default), Active low, level interrupt output from the module.
The module acts as a SPI slave only, while the Host is the SPI master.

Following parameters should be in the host SPI interface.
CPOL (clock polarity) = 0,
CPHA (clock phase) = 0.

3.15.2 Interrupt

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

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

3.15.3 SPI Interface Initialization

This section explains the initialization of the SPI slave interface in the module. Following is the series of steps for SPI initialization:

1. Host sends 0x00124A5C to module.
2. On successful initialization, the module sends 0x58 (SPI success) to the host or else, the module sends 0x54 (SPI busy) or 0x52 (SPI failure).

3.15.3.1 Operations through SPI

The RS9116-WiSeConnect can be configured and operated from the Host by sending commands through the SPI interface.
The SPI interface is programmed to perform a certain transfer using commands C1, C2, C3 and C4 and an optional 32-bit address. For all the Commands and Addresses, the Host is configured to transmit data with **8-bit mode**. At the end of all the Commands and Addresses, the Host is reconfigured to transmit data with 8-bit or 32-bit mode depending on the commands issued. The Module responds to all the commands with a certain response pattern. The four commands i.e. C1, C2, C3, and C4 indicate to the SPI interface for all the aspects of the transfer.

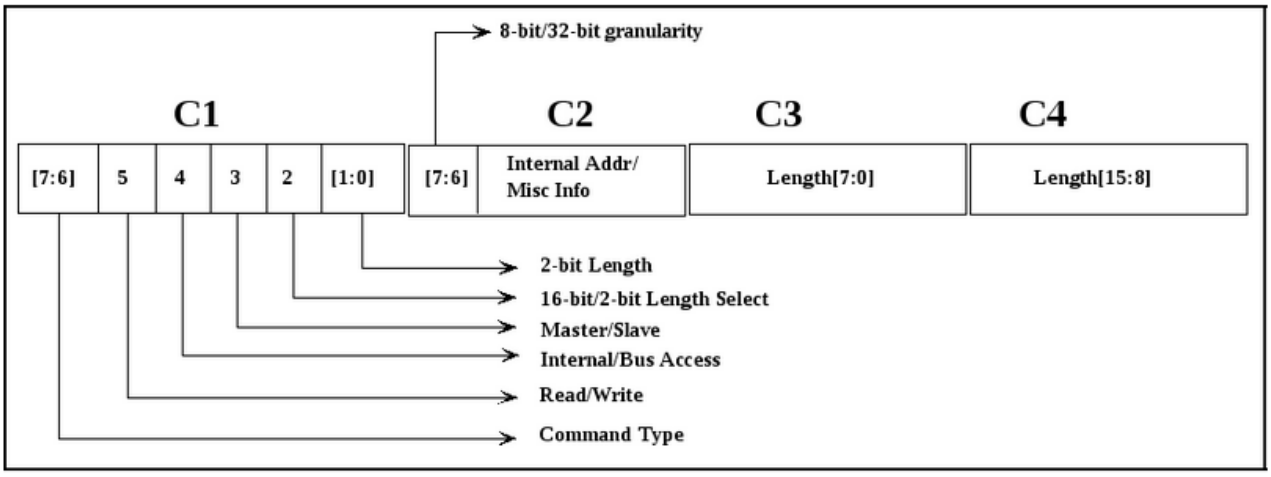


Figure 24: SPI Command Description

The command descriptions are as follows:

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

Command	Bit Number	Description
	5:0	This carries Register address if bit 4 for Command C1 is cleared (i.e. register read/write selected). Otherwise, reserved for future use.
C3	7:0	Length (7:0) LSB of the transfer's length (which is in terms of bytes) in case bit 2 of C1 is set. This command is skipped if bit 2 of C1 is cleared.
C4	7:0	MSB of the transfer Length (15:8) (Which is in terms of bytes) in case bit 2 of C1 is set. This command is skipped if bit 2 of C1 is cleared i.e. if 2-bit length is selected.

Table 3-6 : SPI Command Description

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

3.15.3.2 Module Response

The RS9116 WiSeConnect gives responses to the host SPI command requests through the SPI interface. These are as follows:

- A success / failure response at the end of receiving the command. This response is driven with 8-bit mode during the Command and Address phase and is then switched to 8-bit or 32-bit mode during the Data phase as per the command issued.
  - Success: 0x58 or 0x00000058
  - Failure: 0x52 or 0x00000052
- An 8-bit or 32-bit start token is transmitted once the four commands (C1, C2, C3, C4) indicating a read request are received and the Module is ready to transmit data. The start token is immediately followed by the read-data.
  - Start Token: 0x55 or 0x00000055
- An 8-bit or 32-bit busy response in case a new transaction is initiated while the previous transaction is still pending from the slave side.
  - Busy Response: 0x54 or 0x00000054

3.15.3.3 Module Bit Ordering of SPI Transmission / Reception

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

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

Where, B0 is sent first, then B1, then B2 and so on.
In each of the bytes, the MSB is sent first. For example, when B0 is sent, B0 [7] is sent first, then B0 [6], then B0[5] and so on. Same is the case while receiving data. In this example, B0 [7] is expected first by the receiver, then B0 [6] and so on. Also, as can be seen, the data changes value at the falling edge and is latched at the rising edge.

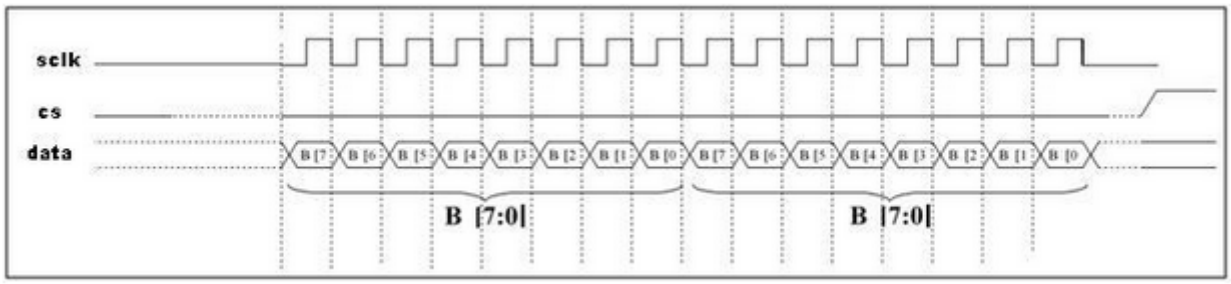


Figure 25 : 8-bit Mode

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

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

Where, W0 is sent first, then W1, then W2 and so on.
In each of the 32-bit words, the MSB is sent first. For example, when W0 is sent, W0[31] is sent first, then W0[30], then W0[29] and so on. Same is the case when receiving data. In this example, W0 [31] is expected first by the receiver, then W0[30] and so on. Also, as can be seen, the data changes value at the falling edge and is latched at the rising edge.

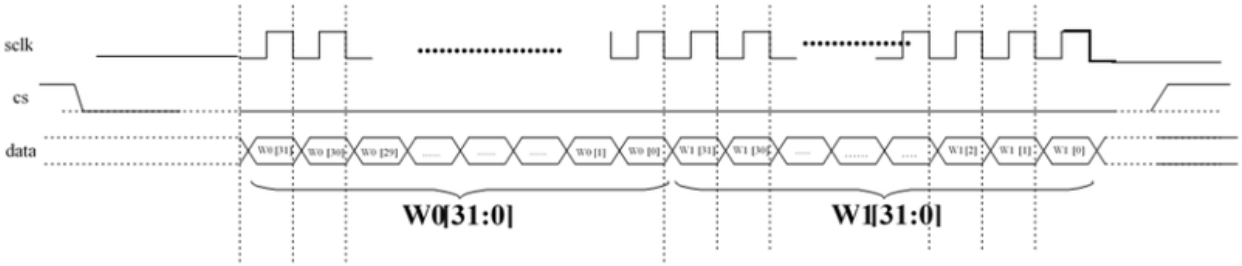


Figure 26: 32-bit Mode

Bit Ordering of Module Response
The bit ordering is same as explained in **Module bit Ordering of SPI Transmission/Reception**. For example, 0x58 response for 8-bit success is sent as
0->1->0->-1->1-> 0 -> 0 -> 0 . That is 0 is sent first, then 1, then 0, then 1, and so on.

3.15.4 Module SPI Interface Initialization

The Initialization command is given to the module to initialize the SPI interface. The SPI interface remains non-functional to all the commands before initialization and responds only after successful initialization. Initialization should be done only once after the power is on. The module treats the subsequent initialization of any command before it is reset as errors. SPI initialization is 24 bit command (0x124A5C) followed by an 8-bit dummy data. 24 bit SPI Initialization command should be send in a sequence of 0x12, 0x4A, 0x5C bytes. Status response from the SPI Interface is driven during the transmission of the dummy data i.e. after the transfer of 8-bits of command C1.

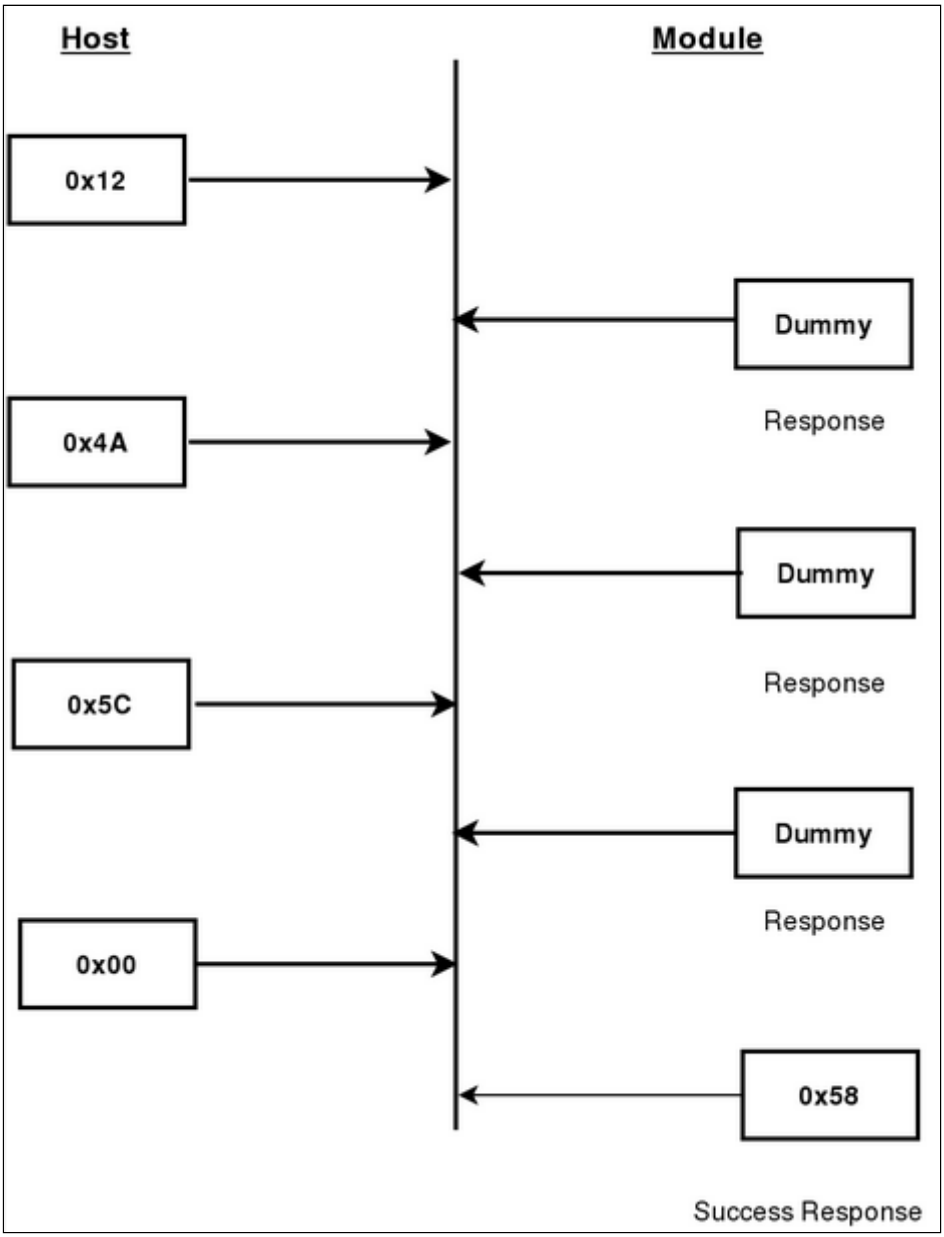


Figure 27: SPI Initialization exchanges between Host and Module

Host Interactions Using SPI Command

This section describes the procedures to be followed by the Host to interact with the RS9116-WiSeConnect using SPI commands.
The Host interactions to the module can be categorized as below.

Command

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

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

Table 3-7: Command Types

3.15.4.1 Memory Type

Host need to access the memory / registers of the RS9116-WiSeConnect for configuration and operation.
To write data into a memory / register address, the **memory write** command has to be framed as described in the figure below. If a "busy" or "failure" response is sent from the module, the host should either resend the command or reset the module.

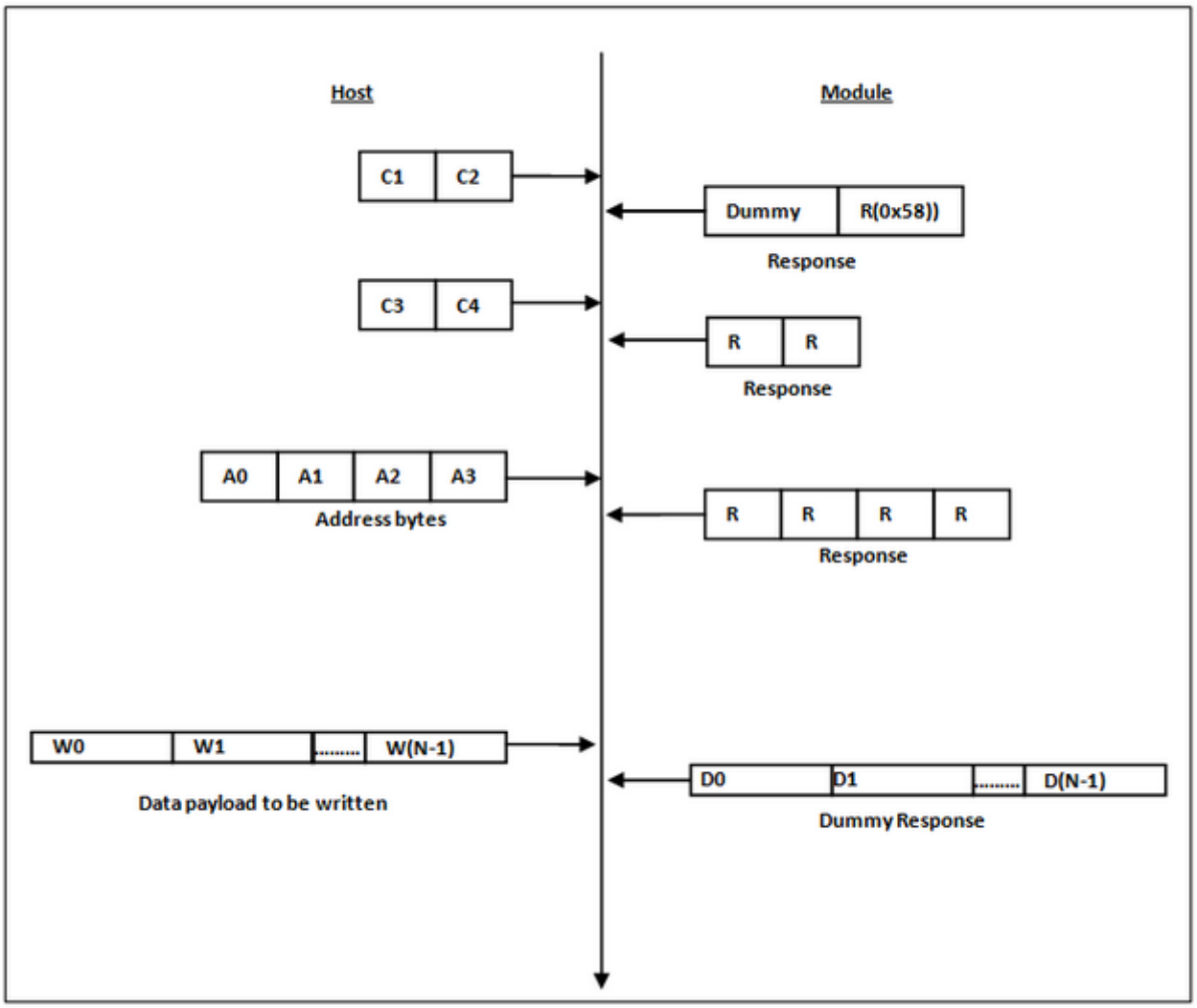


Figure 28: Memory Write

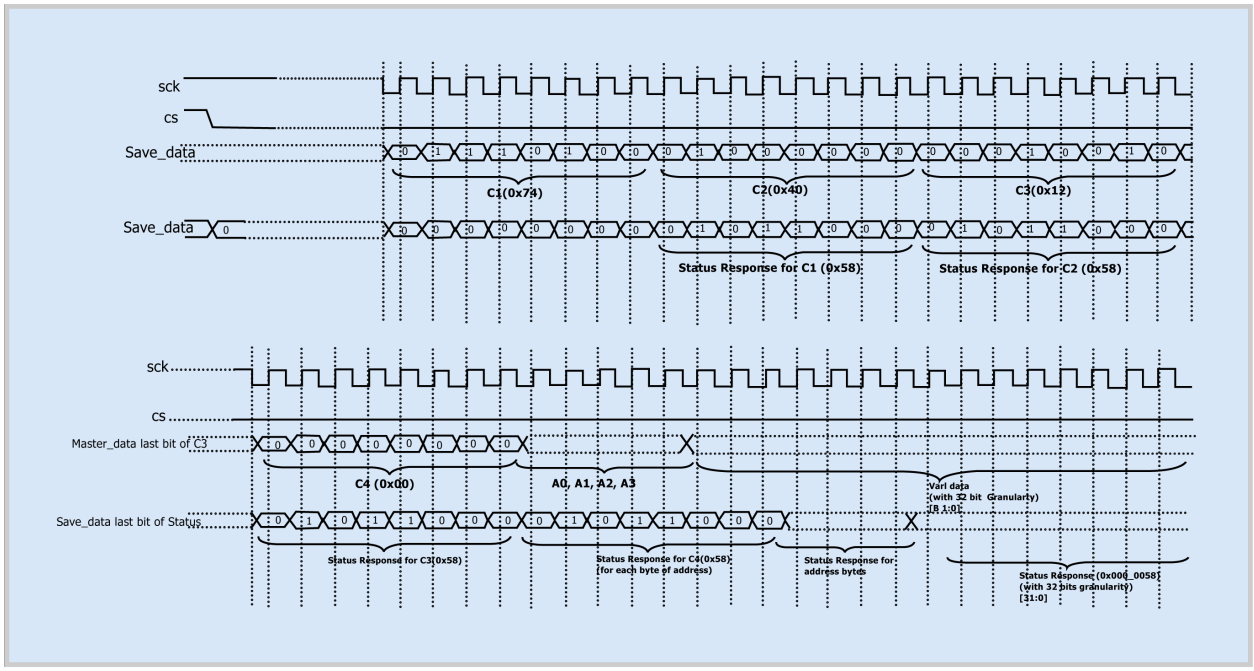


Figure 29: Interactions in the physical interface

Note:

This structure is similar for all following read/write operations.

The following procedure is to be followed by the Host.

1. Send the commands C1, C2.
2. Read the response (R) from the Module. The response will appear as described in **RS9116-WiSeConnect module Module Response**. Status 0x58 indicates that the module is ready.
3. Host should send the commands C3 and C4, followed by the 4 byte address (corresponding to the memory / register address) and data. Status 0x54 indicates that the device is busy. The host has to retry. Status 0x52 indicates a failure response.

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

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

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

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

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

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

3.15.4.2 Frame Write

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

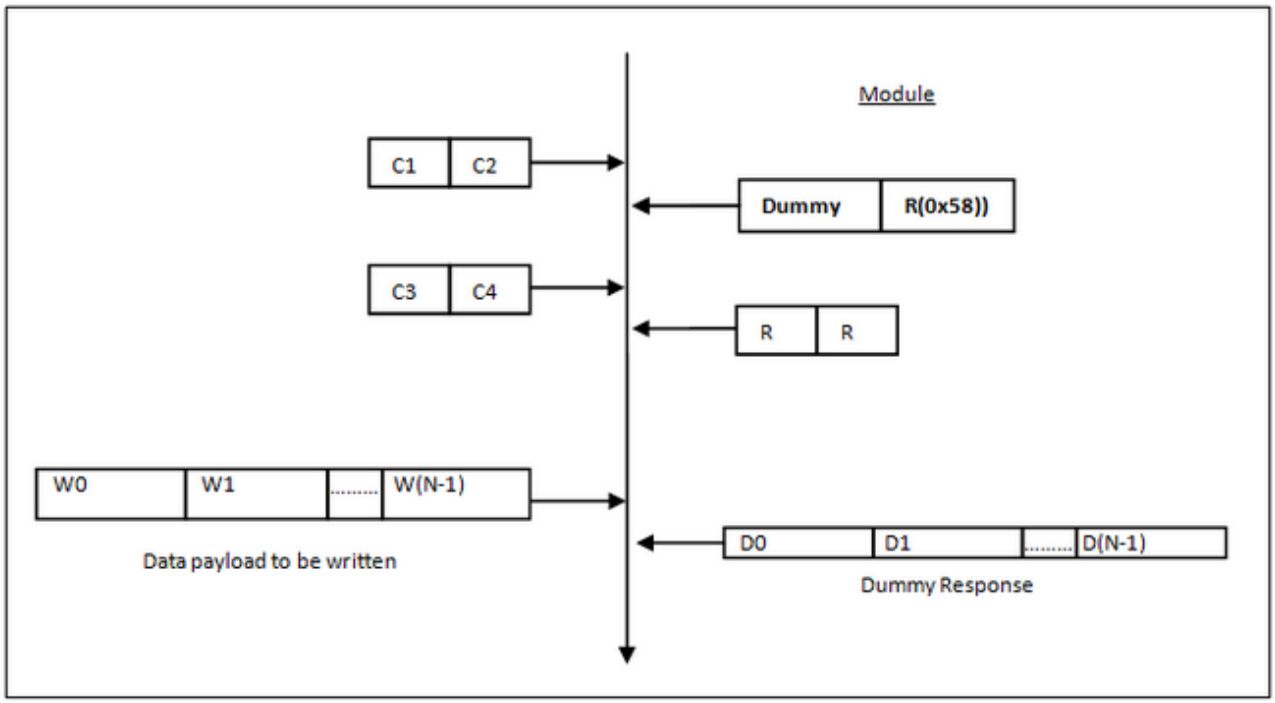


Figure 30: Frame Write

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

1. Prepare and send the commands C1, C2 together as described for Frame write.
2. Read the response (R) from the Module (RS9116-WiSeConnect).
3. Status 0x58 indicates that the Module is ready. The Host can then send the commands C3 and C4, followed by the data. Status 0x54 indicates that the device is busy. The host has to retry. Status 0x52 indicates a failure response.

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

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

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

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

3.15.4.3 Memory Read

To read data from a memory / register address, the host must form and send a Memory Read command.
The following figure gives the flow for memory reads between the Host and the Module.

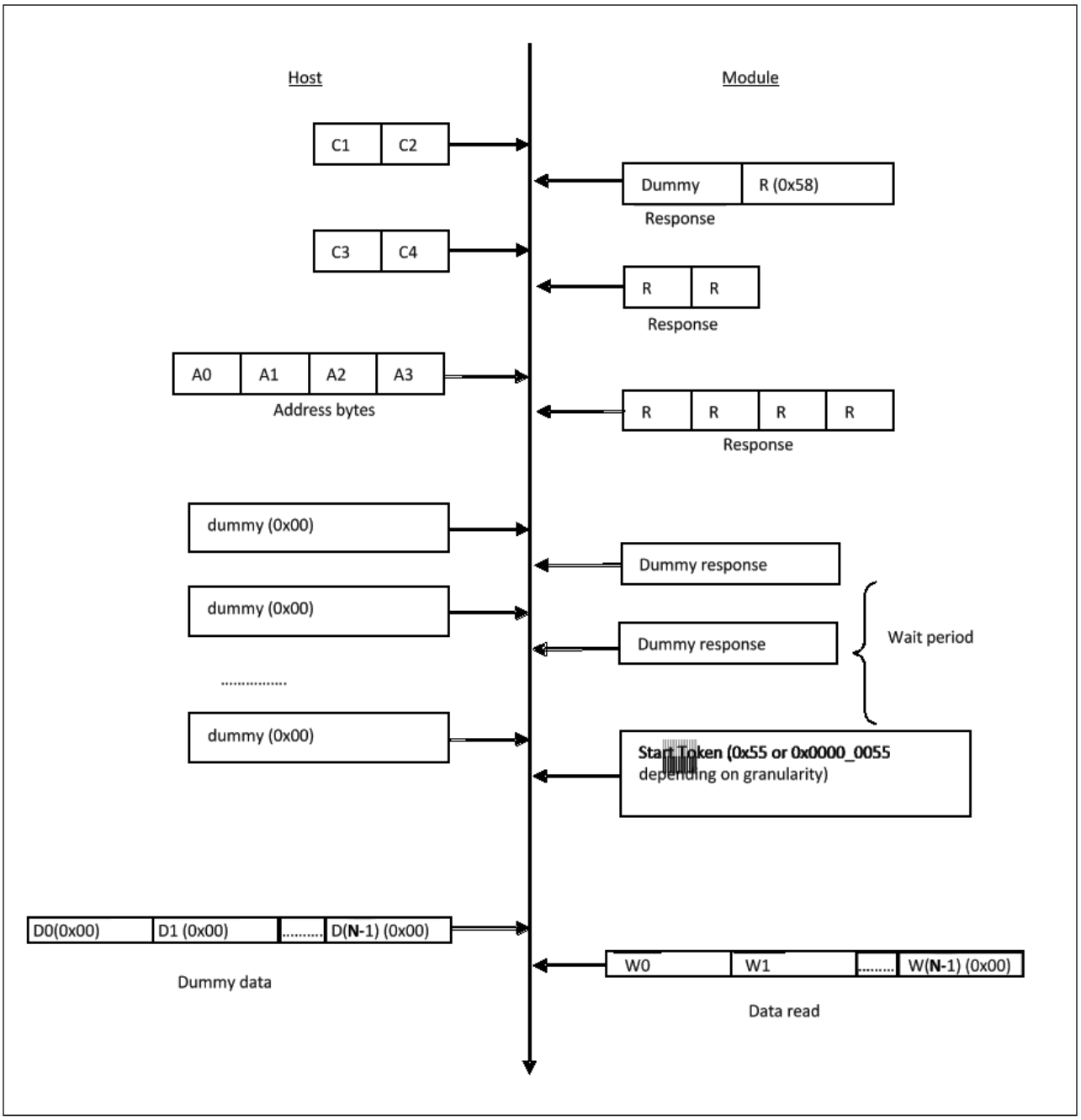


Figure 31: Memory Read

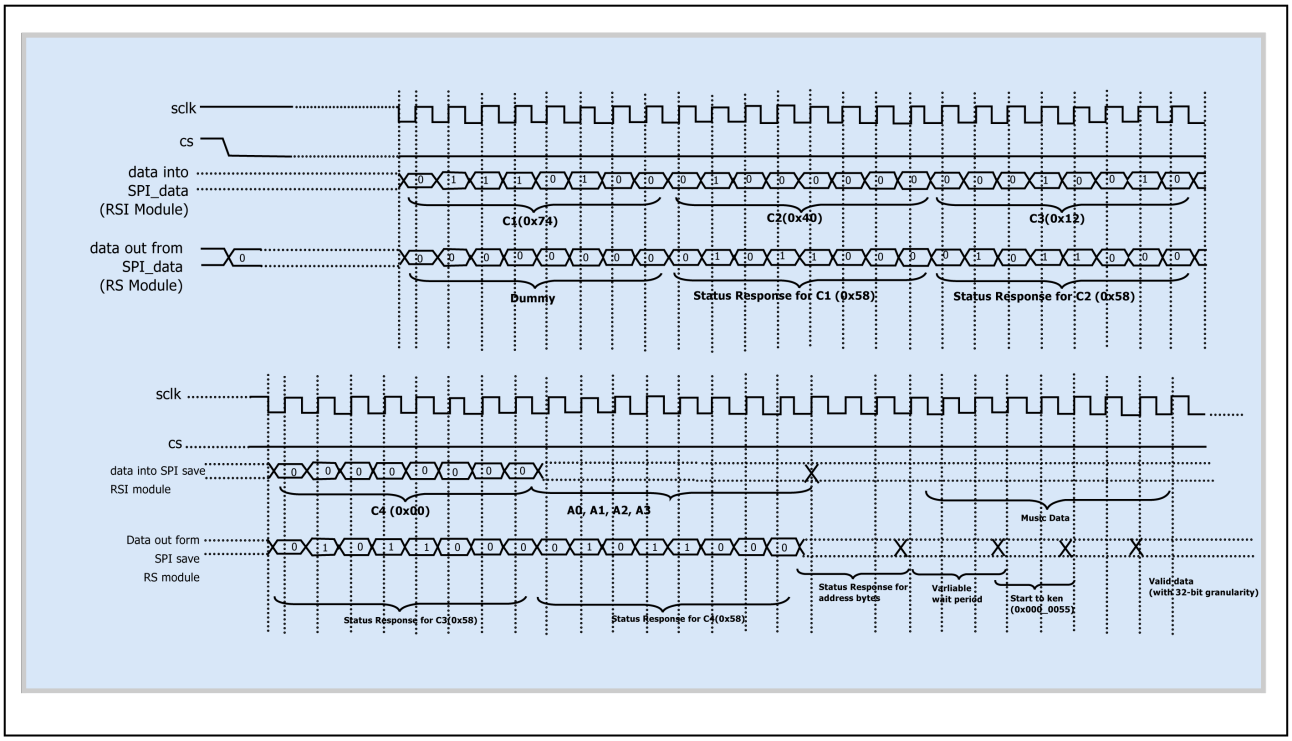


Figure 32: Memory Read at Physical Interface

To perform a memory read, the host must follow the procedure below.

1. Prepare and send the commands C1, C2 as described for Memory read.
2. Read the response from the RS9116-WiSeConnect .
3. Status 0x58 indicates that the slave is ready. Host should next send the commands C3, C4, which indicate the length of the data to be read, followed by the address of the memory location to be read.
4. After sending/receiving C3, C4 commands/response and the addresses, the host should wait for a start token (0x55). The host writes a stream of dummy bytes to read the data from the module each time. The Host should read this data until the start token is received. The data following the start token should be interpreted as the frame of specified length that is read from the Module.
5. Status 0x54, after C1 and C2 bytes, indicates that the device is busy. Host has to retry.
6. Status 0x52, after C1 and C2 bytes, indicates a failure response from the Module.

There is a variable wait period, during which dummy data is sent. After this period, a start token is transmitted from the Module to the Host. The start token is followed by valid data. The start token indicates the beginning of valid data to the Host. To read out the valid data, the host must send dummy data i.e. [D0, D1, D2, …. D (N-1)]. Where, N is the number of 8bit or 32 bit dummy words (based on mode selected) need to send in order to receive specified length of valid data from the module.
The commands C1, C2, C3 and C4 are sent in the following sequence (explained in Module bit Ordering of SPI Transmission/Reception) : C1 first, then C2, C3 and finally C4. The bit ordering is

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

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

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

3.15.4.4 Frame Read

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

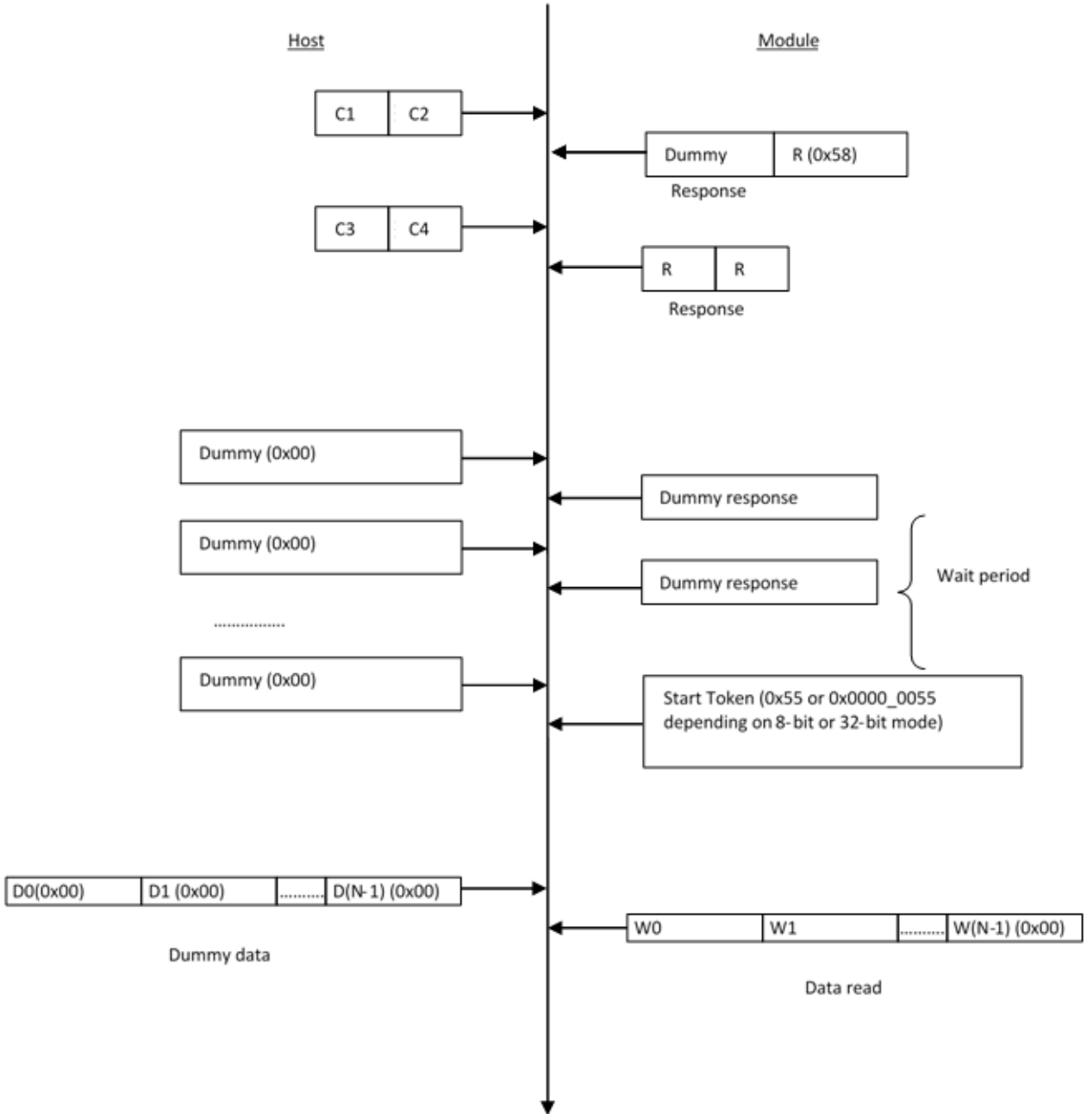


Figure 33: Frame Read

To perform a Frame Read, the host must follow the procedure below.

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

2. Read the response from the RS9116-WiSeConnect .
3. Status 0x58 indicates that the Module is ready. The host should send the commands C3, C4 which indicate the length of the data to be read.
4. After sending/receiving C3, C4 commands/response, the host should wait for a start token (0x55). The data then follows after the start token. The host writes a dummy byte to read the data from the module each time. The Host should read this data until the start token is received. The data following the start token should be interpreted as the frame of specified length that is read from the Module. Status 0x54, after C1 and C2 bytes, indicates that the device is busy. Host has to retry. Status 0x52, after C1 and C2 bytes, indicates a failure response from the Module.

The commands C1, C2, C3 and C4 are sent in the following sequence (explained in Module bit Ordering of SPI Transmission/Reception) : C1 first, then C2, C3 and finally C4. The bit ordering is
C1[7] -> C1[6] … C1[0] -> C2[7] -> C2[6] … C2[0] -> C3[7] -> C3[6] … C3[0] -> C4[7] -> C4[6] … C4[0]. That is, C1[7] bit is sent first, then C1[6] and so on.
If <D3 [7:0]> <D2[7:0]> <D1[7:0]> <D0[7:0]> is the data read, then D0 is sent from module first, then D1 and so on.
D0[7] -> D0[6] … D0[0] -> D1[7] -> D1[6] … D1[0] -> D2[7] -> D2[6] … D2[0] -> D3[7] -> D3[6] … D3[0]

3.15.4.5 Register Read

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

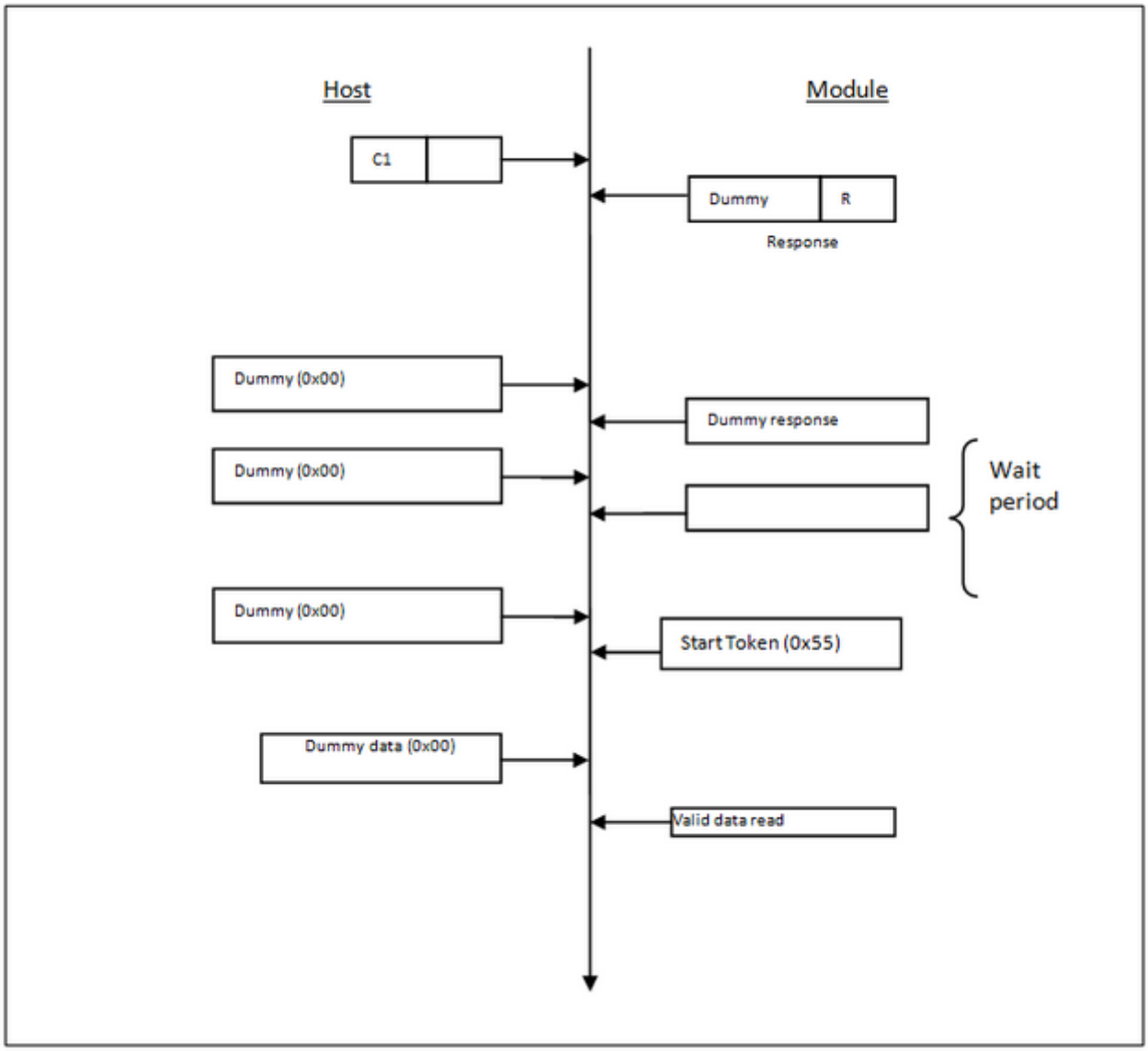


Figure 34: Register Read

3.15.4.6 Register Writes

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

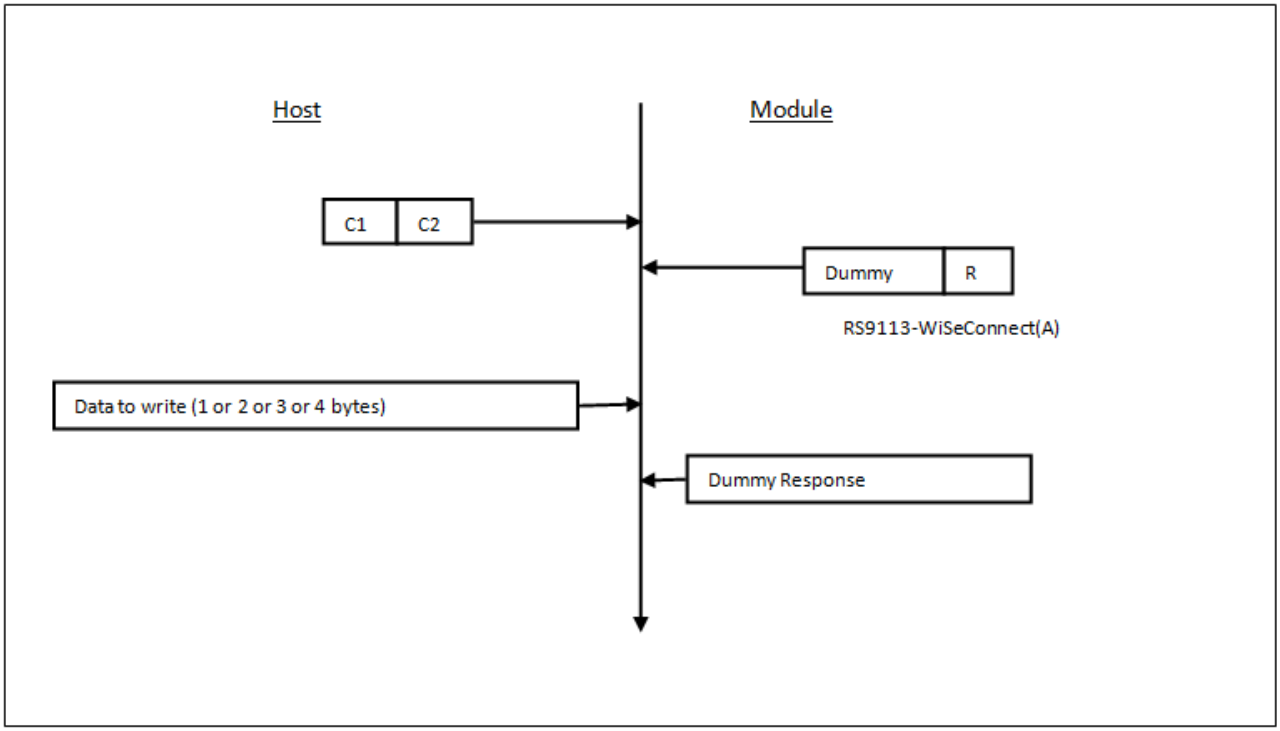


Figure 35: Register Write

3.15.5 Register Summary

Register

Register	Address
SPI_HOST_INTR	0x00

Table 3-8 : RS9116-WiSeConnect Module Register Description

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

SPI_HOST_INTR				
Register Address: 0x00				
Bit	**Access**	**Function**	**Default Value**	**Description**
[7:0]	Read only	SPI_HOST_INTR	0x00	These bits indicate the interrupt status value Bit 0: If '1', Buffer Full condition reached. When this bit is set host shouldn't send data packets. This bit has to be polled before sending each packet. Bit 1: Reserved Bit 2: Reserved Bit 3: If '1', indicates data packet or response to Management frames is pending. This is a self-clearing bit and is cleared after the packet is read by host. Bit 4: Reserved Bit 5: Reserved Bit 6: Reserved

Table 3-8 : SPI Host Interrupt Register

3.16 Software Protocol

This section explains the procedure that the host needs to follow in order to send Wi-Fi commands frames to module and to receive responses from the module.

Tx Operation

The Host uses Tx operations:

1. To send management commands to the module from the Host
2. To send actual data to the module which is to be transmitted onto the air

Host should follow the steps below to send the command frames to the Module:

1. Host should check buffer full condition by reading interrupt status register using register read.
2. If buffer full bit is not set in interrupt status register, the host needs to send Command frame in two parts:
   First it is required to send 16 byte Frame descriptor using Frame write.
   Second it is required to send optional Frame body using Frame write.

Note:

Frame write is an API provided to the host which is present in the release package.

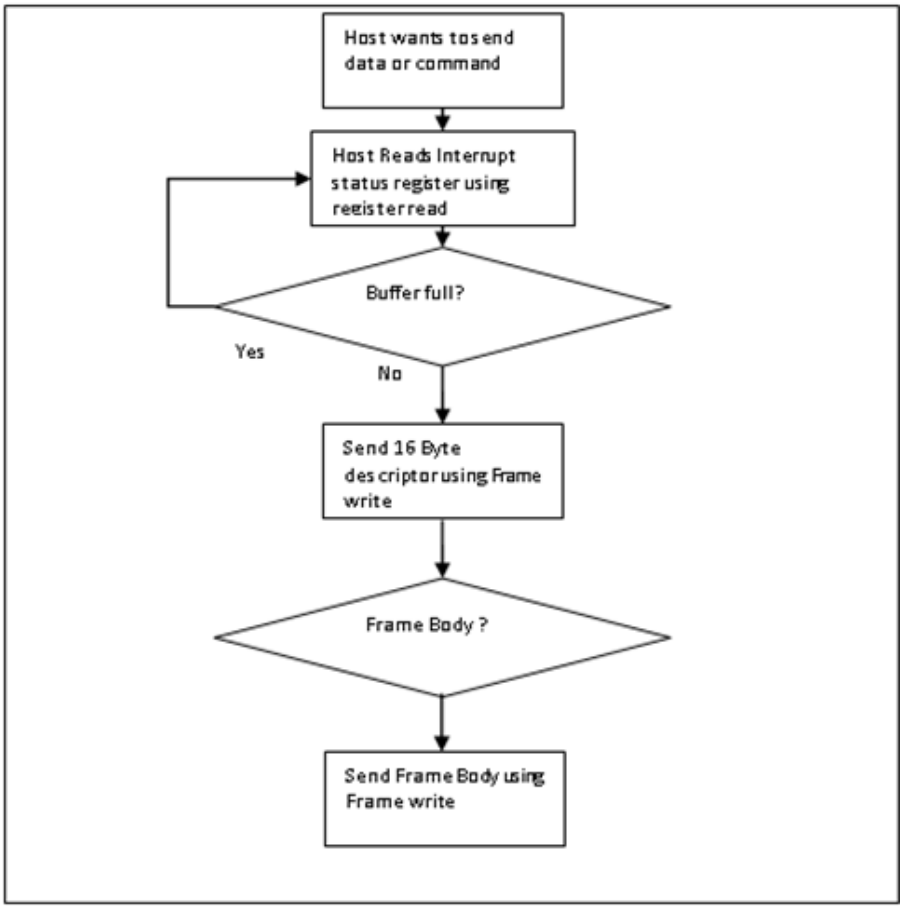


Figure 36: Tx From Host to Module

Management / Data Frame Descriptor and Frame body of command frames are sent to the module by using two separate frame writes as shown below:

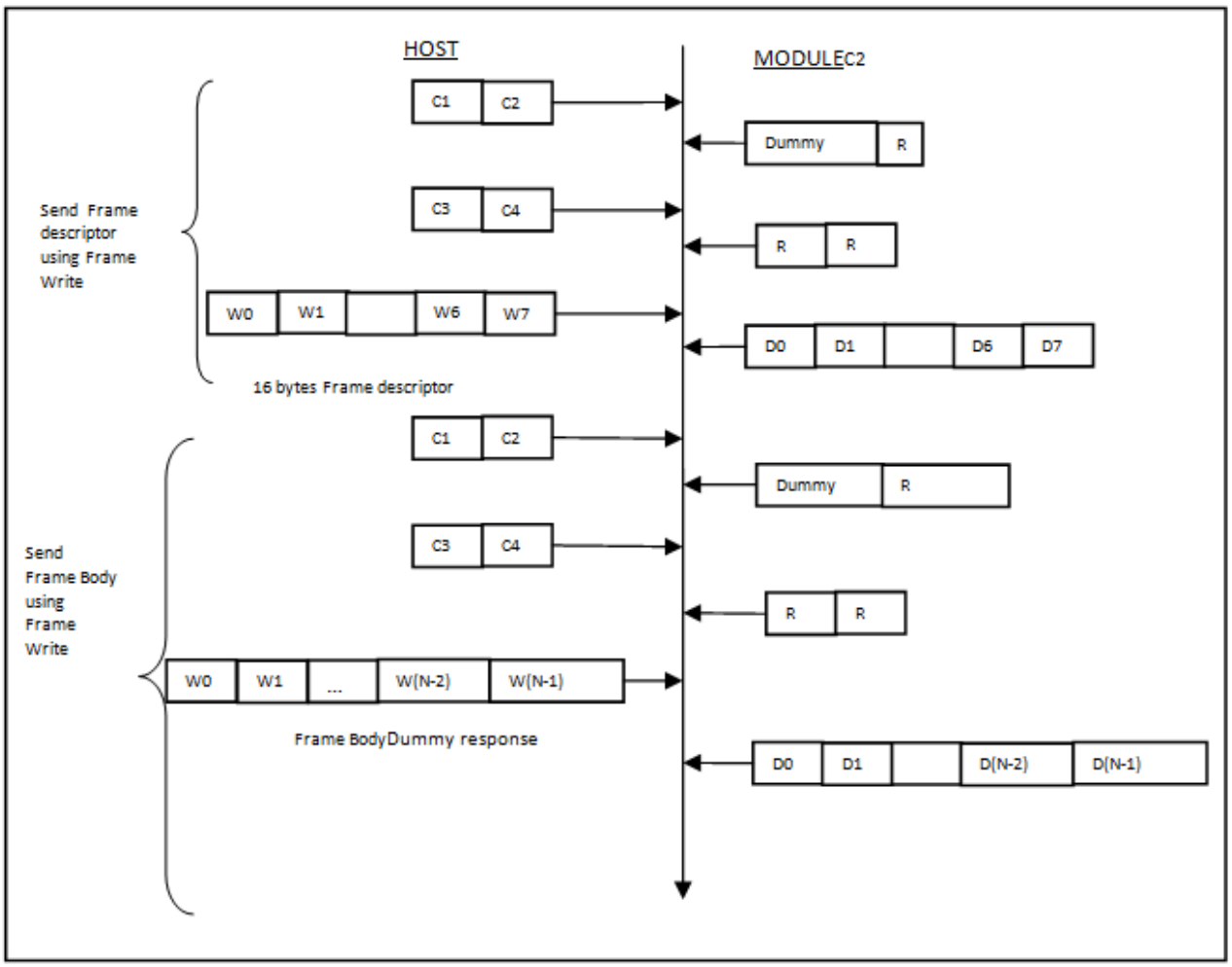


Figure 37: Exchanges between Host and Module for Tx operation

Rx Operation

The Host uses this operation:

- Module's responses, for the commands
- To read data received by the module from the remote peer.

Module sends the response/received data to Host in a format as shown below:

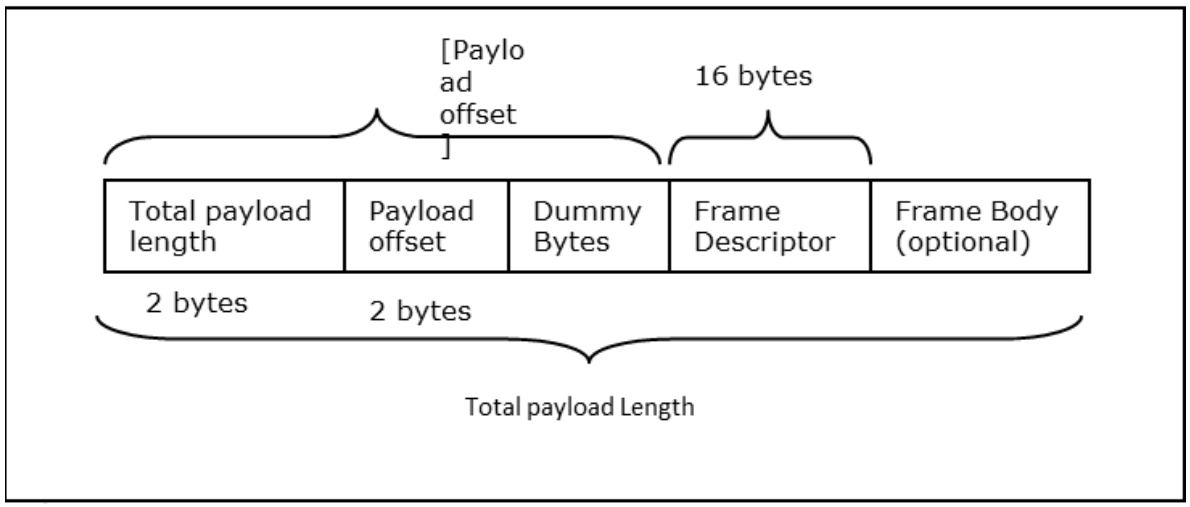


Figure 38: RX Frame Format

Note:
If Payload offset is 'x', 'x-4' dummy bytes will be added before Frame Descriptor.

Host should follow the steps below to read the frame from the Module:

1. If any Data / Management packet is pending from module, module raises an interrupt to HOST.
2. Host needs to check the reason for interrupt by reading interrupt status register using register read.
3. If data pending bit is set in interrupt status register, follow the steps below:
    a. Read 4 bytes using Frame read.
    b. Decode Total payload length and payload offset.

Read remaining payload by sending Frame read with (total payload length – 4 bytes), discard Dummy bytes and then decode Frame descriptor and Frame Body.

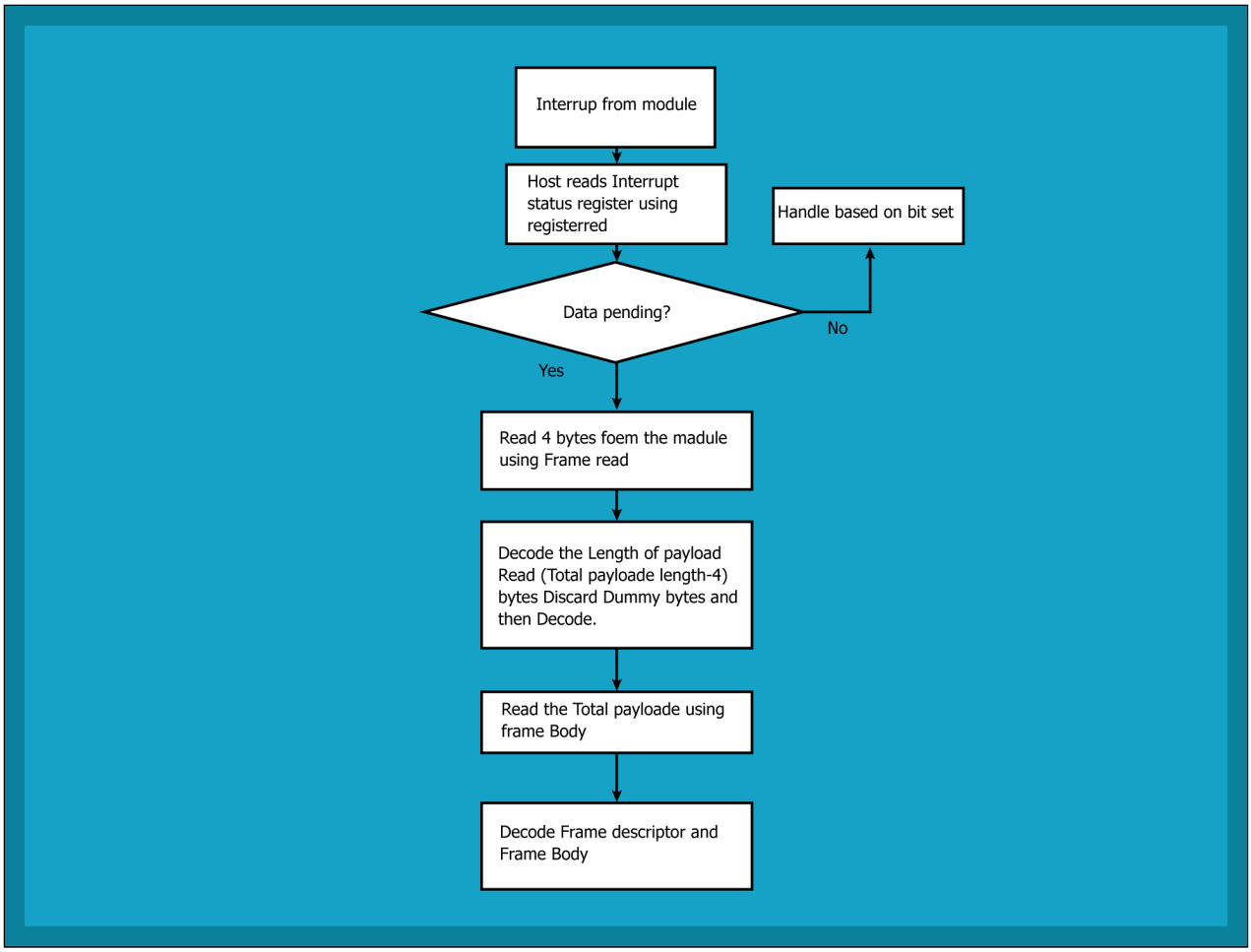


Figure 39: RX Operation from Module to Host

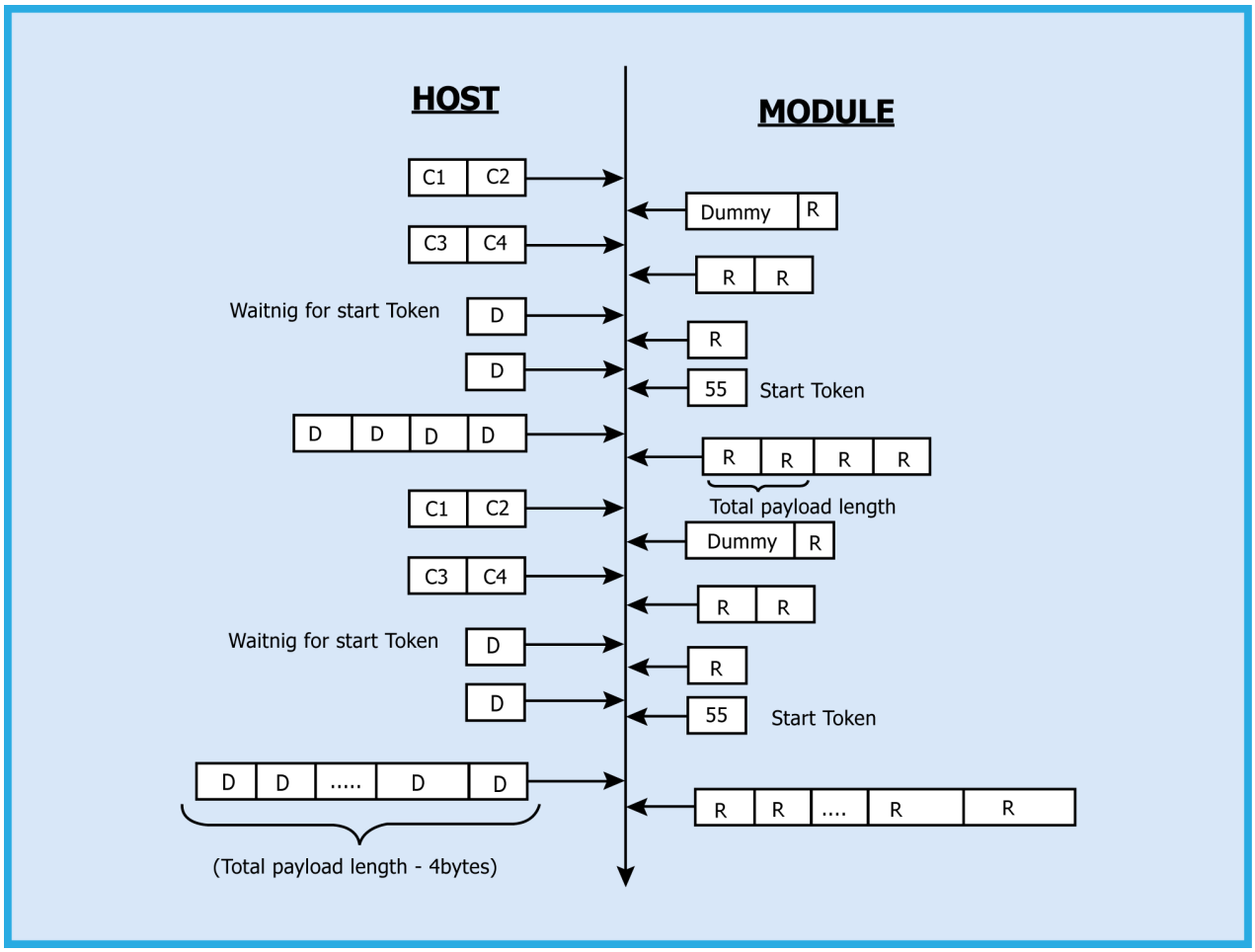


Figure 40: Message exchanges between Host and Module for Rx operation

3.17 Commands

The SPI Interface supports binary commands only. To learn more about binary commands, consult the section **Binary Command Mode**.

3.18 UART Interface

The UART on the RS9116-WiSeConnect is used as a host interface to configure the module to send data and also to receive data.

3.19 Features

- Supports hardware (RTS/CTS) flow control.
- Supports following list of baud rates
  - 9600 bps
  - 19200 bps
  - 38400 bps
  - 57600 bps
  - 115200 bps
  - 230400 bps
  - 460800 bps
  - 921600 bps

Note:

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

3.20 Hardware Interface

The UART interface on the RS9116-WiSeConnect transmits / receives data to / from the Host in UART mode. The RS9116W uses TTL serial UART at an operating voltage of 3.3V.
The Host UART device must be configured with the following settings:

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

3.21 Software Protocol

3.21.1 AT+ command mode

This section explains the procedure that the host needs to follow in order to send Wi-Fi commands frames to module and to receive responses from the module in AT+ command mode.

TX Operation

The Host uses TX operations:

1. To send management commands to the module from the Host
2. To send actual data to the module which is to be transmitted onto the air
3. If host receives error code indicating packet dropped,host has to wait for a while and send the next command /data
4. Host should send next data packet only if it receives "OK<number of bytes sent>" response for the previous one

Rx Operation

The RS9116W responds with either an 'OK' or 'ERROR' string, for Management or Data frames along with a result or error code.

3.21.2 Binary command mode

This section explains the procedure that the host needs to follow in order to send Wi-Fi commands frames to module and to receive responses from the module.

TX Operation

The Host uses TX operations:

1. To send management commands to the module from the Host
2. To send actual data to the module which is to be transmitted onto the air
3. If host receives error code indicating packet dropped,host has to wait for a while and send the next command /data
4. Host should send next data packet only if it receives ACK frame for the previous one

Host should follow the steps below to send the command frames to the Module:

1. First it is required to send 16 byte Frame descriptor using Frame write.
   Second it is required to send optional Frame body using Frame write.

RX Operation

The RS9116W responds with frame of same frame type with or with error code in the error code field.

The Host uses this operation:

- module's responses, for the commands
- To read data received by the module from the remote peer.
- Receives asynchronous information from the RS9116W

Module sends the response/received data to Host in a format as shown below:

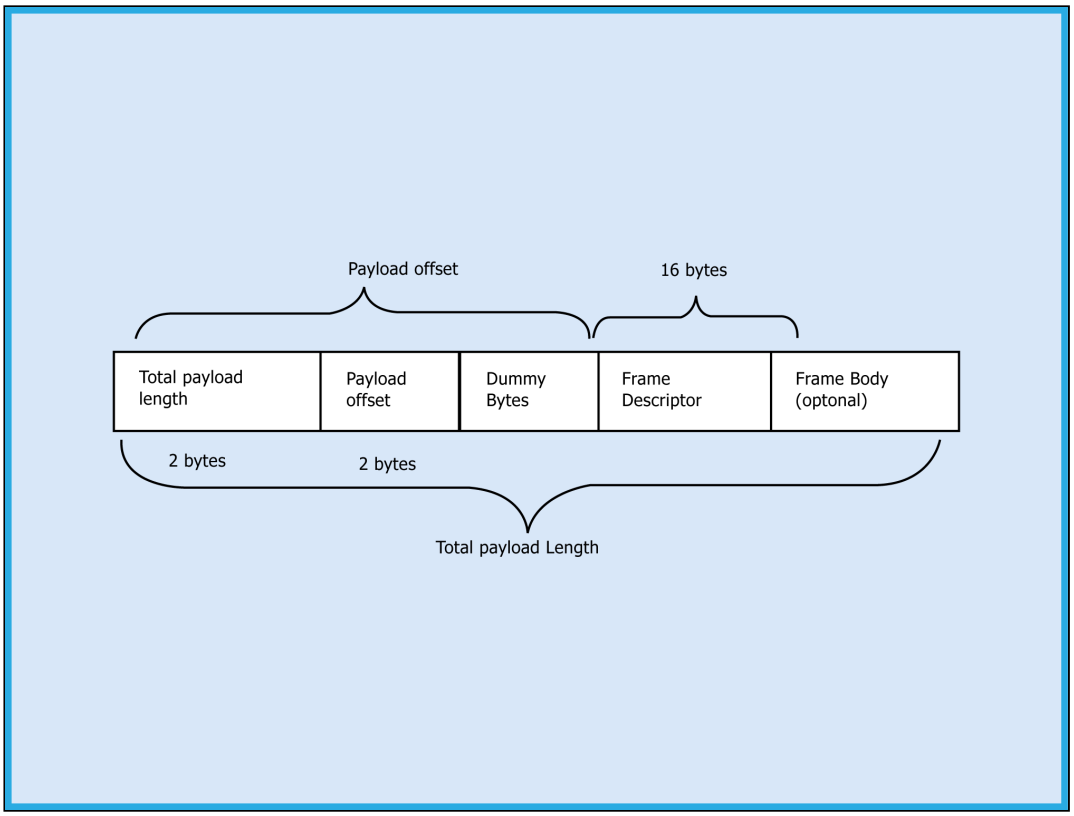


Figure 41: RX Frame format

Note:

If Payload offset is 'x', 'x-4' dummy bytes will be added before Frame Descriptor.

Host should follow the steps below to read the frame from the Module:

Read 4 bytes using Frame read.

1. Decode Total payload length and payload offset.
2. Read remaining payload by sending Frame read with (total payload length – 4 bytes), discard Dummy bytes and then decode Frame descriptor and Frame Body.

3.22 Commands

UART mode supports both AT and binary commands. To learn about AT Commands, see **AT Command Mode**. To learn about Binary Commands, see **Binary Command Mode.**
For concrete examples of AT commands over UART, consult, Appendix A.

3.23 USB Interface

RS9116-WiSeConnect supports USB interface which allows the host to configure and send / receive data through the module via USB.

3.24 Features

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

3.25 Hardware Interface

USB mode uses the standard USB 2.0 interface.

3.26 Software Protocol

This section explains the procedure of how to configure and send the Wi-Fi commands to the module and receive response from the module using USB.

RS9116 WiSeConnect supports two modes by using USB interface.

- USB mode
- USB CDC mode

3.26.1 USB Mode

In USB mode all Wi-Fi command frame formats (Frame Descriptor), Command ID's / response ID's and Error codes are exactly same as Wi-Fi SPI commands.
RS9116-WiSeConnect USB interface support 2 endpoints:

- Control endpoint: control endpoint used during enumeration process.
- Bulk endpoint: Bulk endpoint used to send / receive data between host and module through USB interface.

In USB mode the transfer of command / data packets from host to the module (Tx packet) required headroom as shown in the figure below:

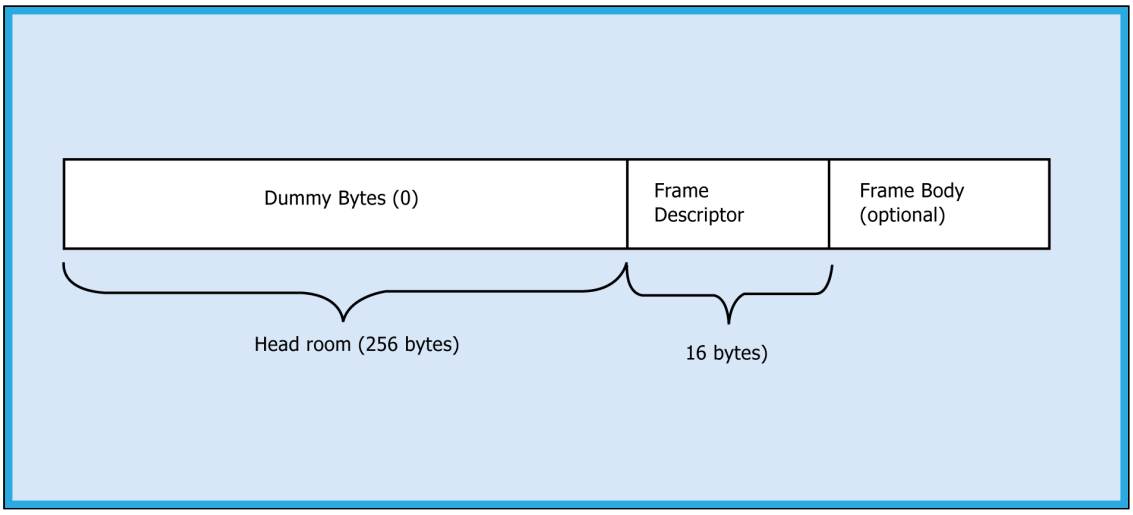


Figure 42: Command/Data Packet format from host to module in USB mode

Note:

Head room size 256 bytes.

In receive path first 4 bytes contain total packet length and descriptor offset. Frame Descriptor starts at descriptor offset location from packet start.

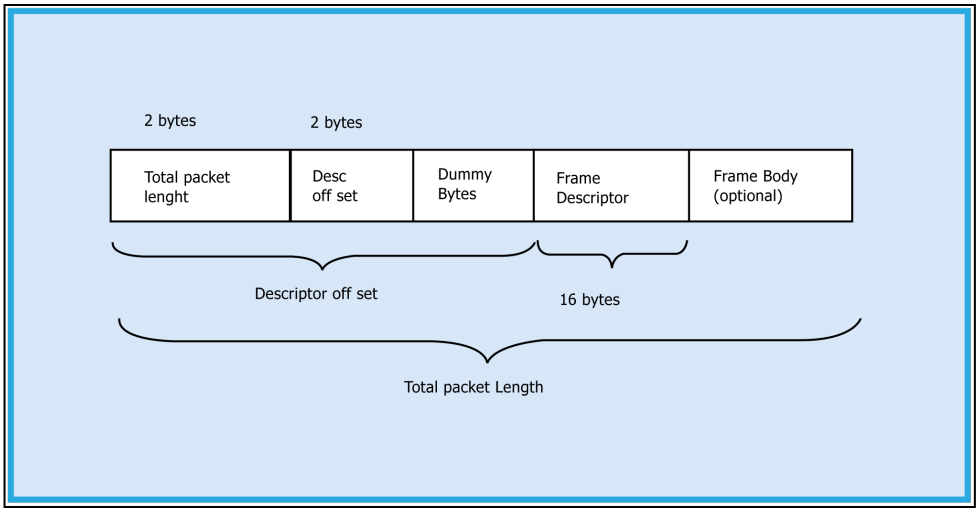


Figure 43: Command/Data Packet format from module to host in USB mode

Operations through USB interface:

This section explains the procedure to be followed by the host to send Wi-Fi command frame to the module and to receive response from the module.

TX operation :
Following are the sequence of steps to be followed to send command frame to the module through USB interface.

- Prepare command frame with headroom as shown in figure **43 : Command/Data Packet format from host to module in USB mode**.
- Forward packet to module.

RX operation:
Following are the sequence of steps to be followed in order to receive response from the module .
Send an empty buffer from host .
After receiving packet from the module, extract the frame by ignoring the process of the Frame Descriptor and Frame Body accordingly.

3.26.2 USB CDC-ACM Mode

The USB interface in the mode corresponds to the CDC-ACM class and presents itself as a USB Device to the Host USB. In order to communicate with the module the user should install a driver file (provided with the software package) in the Host .USB-CDC follows the same command structure as UART Software Protocol

USB CDC-ACM mode usage:

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

1. Connect the module's USB port to USB interface of the PC. The PC prompts for installing the USB-CDC driver. Install the driver file from RS9116.WC.GENR.x.x.x\utils\usb_cdc\rsi_usbcdc.inf. The file needs to be installed only once.
2. Power cycle the module. Check the list in **"Ports"** in the **Device Manager** Settings of the PC. It should show the device as **"RSI WSC Virtual Com Port"**.

3.27 Commands

USB mode supports binary commands only. USB-CDC mode supports both AT and binary commands. To learn about AT Commands, see **AT Command Mode**. To learn about Binary Commands, see **Binary Command Mode**.

4 Embedded BLE Command Mode Selection

This section describes the AT command mode or Binary mode selection in UART and USB-CDC.
After bootloader interaction, the module gives "Loading Done" string in ASCII format to host. After receiving "Loading Done", based on first command received from the host, the module selects command mode.
The module reads the first 4 bytes, if it matches with "AT+R" , select AT command mode, otherwise select Binary mode. Once mode is selected, it will remain in same mode until it is reset or power cycle.
There is an option in bootloader to select AT+mode or binary mode.

Note:

"AT+R" is not case sensitive.

5 Embedded BLE Command Format

This section explains the general command format. The commands should be sent to the Module in the specified format. The format is same for both Classic and LE modes.
The commands are sent to the module and the responses are read from the module using frame write/frame read (as mentioned in the preceeding sections). These commands are called as command frames.
The format of the command frames are divided into two parts:
1. Frame descriptor
2. Frame Body(Frame body is often called as Payload)

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

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

W1[15:0] Packet type Status word W6[15:0]
(for frames sent from module to host)

Frame BodyW4W7W6W5 W0 W1 W2 W3

Frame DescriptorW0[11:0] Packet Length
W0[15:12] Queue type

Payload

Figure 44: Command frame format

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

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

Table 5-1: Frame Descriptor

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

1. Request/Command frames - These are sent from Host to Module. Each Request/ Command has an associated response with it.
2. Response frames – These are sent from Module to Host. These are given in response to the previous Request/Command from the Host. Each command has a single response.
3. Event frames – These are sent from Module to Host. These are given when there are multiple responses for a particular Request/ Command frame. There is Asynchonous message to be sent to host.

The following are the types of frame requests and responses and the corresponding codes. The commands are different for both Classic and LE modes. The below Table lists the Command, Response and Event frames in LE mode.
In both the modes, the corresponding code is to be filled in W1 [15:0] mentioned in the Table above.

Command	Command ID
Set Local Name	0x0001
Get Local Name	0x0002
Get RSSI	0x0005
Get Local BD Address	0x0007
Advertise	0x0075
Scan	0x0076
Connect	0x0077
Disconnect	0x0078
Query Device State	0x0079
Connection parameter update	0x007A
Start Encryption	0x007B
SMP Pair Request	0x007C
SMP Response	0x007D
SMP Passkey	0x007E
Query Profiles list	0x007F
Query Profile	0x0080
Query Characteristic Services	0x0081
Query Include Services	0x0082
Read Characteristic Value by UUID	0x0083
Query Attribute Descriptor	0x0084
Query Attribute Value	0x0085
Query Multiple Attribute Values	0x0086
Query Long Attribute Values	0x0087
Set Attribute Value	0x0088

Command	Command ID
Set Attribute Value No Ack	0x0089
Set Long Attribute Value	0x008A
Set Prepare Long Attribute Value	0x008B
Execute Long Attribute Value Write	0x008C
Initialize BLE module	0x008D
De-initialize BLE module	0x008E
Antenna Select	0x008F
Add New Service	0x0092
Add New Attribute	0x0093
Set local attribute value	0x0094
Get local attribute value	0x0095
Notify request	0x0096
Set advertise data	0x009C
Get LE ping timeout	0x00A1
Set LE ping timeout	0x00A2
Set Random Address	0x00A3
Data Encrypt	0x00A4
GATT Read	0x00A5
Scan Response	0X00A8
White List	0X00AA
Remove Service	0X00AB
Remove Attribute	0X00AC
Resolvlist	0X00AD
Get Resolvlist Size	0X00AE
Set Resolution Enable	0X00AF
Read Phy	0X00B0
Set Phy	0X00B1
Set Data Length	0x00B2
Read Data Length	0x00B3
Set Privacy Mode	0x00B4
CBFC Connection Request	0x00B5
CBFC Connection Response	0x00B6
CBFC Tx Data	0x00B7
CBFC Disconnect	0x00B8

Command	Command ID
LE LTK Request Reply	0x00BA
Rx Test Mode	0x00BB
Tx Test Mode	0x00BC
End Test Mode	0x00BD
Vendor Specific	0x00BE
PER Tx Mode	0x00BF
PER Rx Mode	0x00C0

Table 5-2: Types of frame requests and responses and the corresponding codes

Response	Response ID
Card Ready	0x0505
Set Local Name	0x0001
Get Local Name	0x0002
Get RSSI	0x0005
Get Local BD Address	0x0007
Advertise	0x0075
Scan	0x0076
Connect	0x0077
Disconnect	0x0078
Query Device State	0x0079
Connection parameter update	0x007A
Start Encryption	0x007B
SMP Pair Request	0x007C
SMP Response	0x007D
SMP Passkey	0x007E
Query Profiles list	0x007F
Query Profile	0x0080
Query Characteristic Services	0x0081
Query Include Services	0x0082
Read Characteristic Value by UUID	0x0083
Query Attribute Descriptor	0x0084
Query Attribute Value	0x0085
Query Multiple Attribute Values	0x0086

Response	Response ID
Query Long Attribute Values	0x0087
Set Attribute Value	0x0088
Set Attribute Value No Ack	0x0089
Set Long Attribute Value	0x008A
Set Prepare Long Attribute Value	0x008B
Execute Long Attribute Value Write	0x008C
Initialize BLE module	0x008D
Deinitialize BLE module	0x008E
Antenna Select	0x008F
Add New Service	0x0092
Add New Attribute	0x0093
Set local attibute value	0x0094
Get local attribute value	0x0095
Notify Response	0x0096
Set advertise data	0x009C
Get le ping timeout	0x00A1
Set le ping timeout	0x00A2
Set Random Address	0x00A3
Data Encrypt	0x00A4
GATT Read	0x00A5
Scan Response	0X00A8
White List	0X00AA
Remove Service	0X00AB
Remove Attribute	0X00AC
Resolvlist	0X00AD
Get Resolvlist Size	0X00AE
Set Resolution Enable	0X00AF
Read Phy	0X00B0
Set Phy	0X00B1
Set Data Length	0x00B2
Read Data Length	0x00B3
Set Privacy Mode	0x00B4

Response	Response ID
CBFC Connect Request	0x00B5
CBFC Connect Response	0x00B6
CBFC Tx Data	0x00B7
CBFC Disconnect	0x00B8
LE LTK Request Reply	0x00BA
Rx Test Mode	0x00BB
Tx Test Mode	0x00BC
End Test Mode	0x00BD
PER Tx Mode	0x00BE
PER Rx Mode	0x00BF

Table 5-3 : Response IDs in BLE mode

Event	Event ID
Disconnected	0x1006
Scan Response	0x150E
Connection Status	0x150F
SMP Request	0x1510
SMP Response	0x1511
SMP Passkey	0x1512
SMP Failed	0x1513
GATT Notification	0x1514
GATT Indication	0x1515
Encrypt Status	0x1516
GATT Write	0x1517
LE ping timeout expired	0x1518
GATT Read	0x151B
MTU size	0x151C
SMP passkey display	0x151D
Phy Update Complete	0x151E
Data length change event	0x151F
SMP SC Passkey	0x1520
Enhanced Connection Event	0x1521
Directed Advertising Report	0x1522
Security Keys	0x1523
PSM Connection Request	0x1524

Event	Event ID
PSM Connection Complete	0x1525
PSM Rx Data	0x1526
PSM Disconnect	0x1527
LE LTK Request	0x152A
Connection Update Complete	0x152B

Table 5-4: Event IDs in BLE mode

6 Embedded BLE Commands

The following sections will explain various RS9116-WiSeConnect Bluetooth LE commands, their structures, the parameters they take and their responses. For API prototypes of these commands, please refer to the API Library Section

Note:
1. All BT/BLE AT commands are case sensitive and lower case.
2. A command **should not** be issued by the Host before receiving the response of a previously issued command from the module.

6.1 Generic commands

6.1.1 Set Operating Mode

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

```
at+rsi_opermode=
<oper_mode>,<feature_bit_map>,<tcp_ip_feature_bit_map>,<custom_feature_bit_map>,<ext_cus
tom_feature_bit_map>,<bt_feature_bit_map>,<ext_tcp_ip_feature_bit_map>,<ble_feature_bit_
map>\r\n
```

Note:

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

at+rsi_opermode=<oper_mode>,<feature_bit_map>,<tcp_ip_feature_bit_map>,<custom_feature_bitmap><ext_custom_feature_bit_map>\r\n

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

at+rsi_opermode=<oper_mode>,<feature_bit_map>,<tcp_ip_feature_bit_map>,<custom_feature_bitmap><ext_tcp_ip_feature_bit_map>\r\n

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

at+rsi_opermode=<oper_mode>,<feature_bit_map>,<tcp_ip_feature_

bit_map>,<custom_feature_bitmap><ext_custom_feature_bit_map> <bt_feature_bit_map>\r\n

if BIT(31) is set to 1 in bt_feature_bit_map

at+rsi_opermode=<oper_mode>,<feature_bit_map>,<tcp_ip_feature_

bit_map>,<custom_feature_bitmap><ext_custom_feature_bit_map><bt_feature_bit_map><ext_tcp_ip_feature_bit_map><ble_feature_bit_map>\r\n

Binary Mode:
The structure of the payload is give below

```
typedef struct
{
uint32 oper_mode;
uint32 feature_bit_map;
uint32 tcp_ip_feature_bit_map;
uint32 custom_feature_bit_map;
uint32 ext_custom_feature_bit_map;
uint32 bt_feature_bit_map;
uint32 ext_tcp_ip_feature_bit_map;
uint32 ble_feature_bit_map;
 } operModeFrameSnd;
```

Command Parameters:

Oper_mode:
Sets the mode of operation. oper_mode contains two parts <wifi_oper_mode, coex_mode>. Lower two bytes represent wifi_oper_mode and higher two bytes represent coex_modes.
oper_mode = ((wifi_oper_mode) | (coex_mode << 16))

Wi-Fi_oper_mode values:

- 0 - Wi-Fi Client Mode. The module works as a normal client that can connect to an Access Point with different security modes other than enterprise security.
- 1 – Wi-Fi Direct™ or Autonomous GO**.** In this mode, the module either acts as a Wi-Fi Direct node or as an Autonomous GO (with intent value 16), depending on the inputs supplied for the command "Configure Wi-Fi Direct Peer-to-Peer Mode". In Autonomous GO and in Wi-Fi Direct GO mode, a maximum of 4 client devices are supported.
- 2 – Enterprise Security Client Mode. The module works as a client that can connect to an Access Point with WPA/WPA2-Enterprise security.
- 6 – Access Point mode**.** In this mode, the module acts as an Access Point, depending on the inputs supplied for the command "Configure AP Mode". In Access Point mode, a Maximum of 8 clients can connect based on the bits set in custom feature select in opermode.
- 8 - PER Mode. This mode is used for calculating packet error rate and mostly used during RF certification tests.
- 9 – Cocurrent mode. This mode is used to run module in concurrent mode. In concurrent mode, host can connect to a AP and can create AP simultaneously.

Note:

In concurrent mode

1. AP MAC address's last byte will differ and it will be one plus the station mode MAC last byte.
2. In TCP/IP non bypass mode, Broadcast / Multicast packet will go to first created interface (e.g. if Station mode connects first the broadcast / multicast packet will go to the network belonging to station mode).
3. IPV6 support is not present in the current release.
4. In Concurrent mode, aggregration is not supported.

coex_mode bit values: enables respective protocol
BIT 0 : Enable/Disable WLAN mode.
0 – Disable WLAN mode
1 – Enable WLAN mode
BIT 1 : Enable/Disable ZigBee mode.
0 – Disable ZigBee mode

1 – Enable ZigBee mode
BIT 2 : Enable/Disable BT mode.
0 – Disable BT mode
1 – Enable BT mode
BIT 3 : Enable/Disable BTLE mode.
0 – Disable BTLE mode
1 – Enable BTLE mode

Note:
In BTLE mode, need to enable BT mode also.
Following table represents possible coex modes supported:

1	WLAN
2	ZigBee
3	WLAN + ZigBee
4	Bluetooth
5	WLAN + Bluetooth
6	Bluetooth + Zigbee co-existence*
7	WLAN + Bluetooth + ZigBee co-existence*
8	Dual Mode (Bluetooth and BLE)
9	WLAN + Dual Mode
10	Dual Mode + ZigBee co-existence*
11	WLAN + Dual Mode + ZigBee co-existence*
12	BLE mode
13	WLAN + BLE
14	BLE and ZigBee co-existence*
15	WLAN + BLE + ZigBee co-existence*

* Will be supported in future releases

Table 12: Coex Modes Supported

Note:

Following CoeX mode is supported in RS9116W

1. WLAN STA+BT (Only TCP/IP Bypass mode)
2. WLAN STA + BLE
3. WLAN STA + ZB
4. BLE + ZB
5. WLAN AP + BT (Only support is present in TCP/IP Bypass mode)
6. WLAN AP + BLE
7. WLAN AP + ZB
8. WLAN STA + BT (with TCP/IP stack)
9. WLAN STA + BT + BLE
10. WLAN AP + BT (with TCP/IP stack)
11. WLAN AP + BT + BLE

To select proper CoeX mode please refer **WiSeConnect_TCPIP_Feature_Selection_v1.4.0.xlsx** given in the release package.

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

BT card ready frame is described in **RS9116-WiseConnect-BT-Classic-Software-PRM-API-Guide-v1.6.0.pdf**, BLE card ready frame is described in **RS9116-WiseConnect-BLE-Software-PRM-API-Guide-v1.6.0.pdf** and ZigBee card ready frame is described in **RS9116-WiseConnect-ZigBee-Software-PRM-API-Guide-v1.6.0.pdf**

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

feature_bit_map[0]- To enable open mode
0 - Open Mode Disabled
1- Open Mode enabled (No Security)

feature_bit_map[1]- To enable PSK security
0 - PSK security disabled
1 - PSK security enabled

feature_bit_map[2]-To enable Aggregation in station mode
0-Aggregation disabled
1-Aggregation enabled

feature_bit_map[3]-To enable LP GPIO hand shake
0 – LP GPIO hand shake disabled
1 – LP GPIO hand shake enabled

feature_bit_map[4]-To enable ULP GPIO hand shake
0 – ULP GPIO hand shake disabled
1 – ULP GPIO hand shake enabled

feature_bit_map[5]- Reserved

feature_bit_map[6]- Reserved

feature_bit_map[7]-To disable WPS support
0 – WPS enable
1 - WPS disable in AP mode and station Mode

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

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

tcp_ip_feature_bit_map: To enable TCP/IP related features.

tcp_ip_feature_bit_map[0]- To enable TCP/IP bypass
0 - TCP/IP bypass mode disabled
1 - TCP/IP bypass mode enabled

tcp_ip_feature_bit_map[1]- To enable http server
0 - HTTP server disabled
1 - HTTP server enabled

tcp_ip_feature_bit_map[2]- To enable DHCPv4 client
0 - DHCPv4 client disabled
1 - DHCPv4 client enabled

tcp_ip_feature_bit_map[3]- To enable DHCPv6 client
0 - DHCPv6 client disabled
1 - DHCPv6 client enabled

tcp_ip_feature_bit_map[4]- To enable DHCPv4 server
0 - DHCPv4 server disabled
1 - DHCPv4 server enabled

tcp_ip_feature_bit_map[5]- To enable DHCPv6 server
0 - DHCPv6 server disabled
1 - DHCPv6 server enabled

tcp_ip_feature_bit_map[6]- To enable dynamic update of web pages (JSON objects)
0 - JSON objects disabled
1 - JSON objects enabled

tcp_ip_feature_bit_map[7]- To enable HTTP client
0 - To disable HTTP client
1 - To enable HTTP client

tcp_ip_feature_bit_map[8]- To enable DNS client
0 - To disable DNS client
1 - To enable DNS client

tcp_ip_feature_bit_map[9]- To enable SNMP agent
0 - To disable SNMP agent
1 - To enable SNMP agent

tcp_ip_feature_bit_map[10]- To enable SSL
0 - To disable SSL
1 - To enable SSL

tcp_ip_feature_bit_map[11]- To enable PING from module(ICMP)
0 - To disable ICMP
1 - To enable ICMP

tcp_ip_feature_bit_map[12]- To enable HTTPS Server
0 - To disable HTTPS Server
1 - To enable HTTPS Server

tcp_ip_feature_bit_map[14]- To send configuration details to host on submitting configurations on wireless configuration page
0 - Do not send configuration details to host
1 - Send configuration details to host

tcp_ip_feature_bit_map[15]- To enable FTP client
0 - To disable FTP client
1 - To enable FTP client

tcp_ip_feature_bit_map[16]- To enable SNTP client
0 - To disable SNTP client
1 - To enable SNTP client

tcp_ip_feature_bit_map[17]- To enable IPv6 mode
0 - To disable IPv6 mode
1 - To enable IPv6 mode
IPv6 will also get enabled if DHCP v6 client/DHCP v6 server is enabled irrespective of tcp_ip_feature_bit_map[17].

tcp_ip_feature_bit_map[19]- To MDNS and DNS-SD
0 - To disable MDNS and DNS-SD
1 - To Enable MDNS and DNS-SD

tcp_ip_feature_bit_map[20]- To enable SMTP client
0 - To disable SMTP client
1 - To Enable SMTP client

tcp_ip_feature_bit_map[21 - 24]- To select no of sockets
possible values are 1 to 10 . If User tried to select more than 10 sockets it will be reset to 10 sockets only . Default no of sockets is 10, if this selection is not done by the user.

tcp_ip_feature_bit_map[25]- To select Single SSL socket
0 – selecting single socket is Disabled

1 – Selecting single socket is enabled

NOTE: By default two SSL sockets are supported

tcp_ip_feature_bit_map[26]- To allow loading Private & Public certificates
0 – Disable loading private & public certificates

1 – Allow loading private & public certificates

Note:

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

Or

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

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

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

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

tcp_ip_feature_bit_map[29]- To enable POP3 client

0 - To disable POP3 client
1 - To Enable POP3 client

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

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

1 – To enable Extended tcp_ip feature bitmap

0 – To disable Extended tcp_ip feature bitmap

tcp_ip_feature_bit_map[13] set to '0'.

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

Note:
Feature selection utility is provided in the package. WiSeConnect device supports the selected features combination only if it is feasible as per the **WiSeConnect_TCPIP_Feature_Selection_v1.4.0.xlsx**

custom_feature_bit_map:
This bitmap is used to enable following custom features:

BIT[2]: If this bit is set to '1', the DHCP server behavior, when the module is in AP mode, changes. The DHCP server, when it assigns IP addresses to the client nodes, does not send out a Gateway address, and sends only the assigned IP and Subnet values to the client. It is highly recommended to keep this value at '0' as the changed behavior is required in only very specialised use cases and not in normal AP functionality. The default value of this bit is '0'.

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

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

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

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

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

BIT[10]: Used to enable/disable **Asynchronous messages** to host to indicate the module state.
1- Enable asynchronous message to host

0-Disable asynchronous message to host

BIT[11] : To enable/disable packet pending (**Wake on Wireless**) indication in UART mode
1 – Enable packet pending indication

0– Disable packet pending indication

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

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

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

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

BIT[18]: Reserved

BIT[19]: Reserved

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

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

BIT[23] : To enable/disable HTTP/HTTPs authentication
1 - Enable
0 – Disable

BIT[24]: To enable/disable higher clock frequency in module to improve throughput
1 - Enable
0 – Disable

BIT[25]: To give HTTP server credentials to host in get configuration command
1 – To include HTTP server credentials in get configuration command response
0 – To exclude HTTP server credentials in get configuration command response

BIT[26]: To accept or reject new connection request when maximum clients are connected in case of LTCP.
1 - Reject
0 – Accept

By default this bit value is zero.
When BIT[26] is zero: For a LTCP socket when maximum clients are connected if a new connection request is received, then this connection request will not be rejected. Instead module will maintain this connection request in LTCP pending list.
This request will be served when any of the connected client is disconnected.
When BIT[26] is set: For a LTCP socket when maximum clients are connected if a new connection request is received, then this connection request will be rejected immediately. Module will not maintain this connection request in LTCP pending list.

BIT[27]: To enable dual band roaming and vcsafd feature this bit is used.
1 - Enable dual band roaming and rejoin
0 – Disable dual band roaming and rejoin.

BIT[28]: To enable real time clock from host
1 - Enable real time clock feature given by host
0 – Disable real time clock feature
BIT[29]: To Enable IAP support in BT mode

1 - Enable
0 – Disable

BIT[31]: This bit is used to validate extended custom feature bitmap.
1 – Extended feature bitmap valid
0 – Extended feature bitmap is invalid

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

Note:

For UART / USB-CDC in AT mode:

When user does not give any tcp_ip_feature_bit_map value then default settings for client mode, Enterprise client mode, WiFi-Direct mode are as follows:
HTTP server, DHCPv4 client, DHCPv6 client and JSON objects are enabled.

When user does not give any tcp_ip_feature_bit_map value then default settings for Access point mode are:
HTTP server, DHCPv4 server, DHCPv6 server and JSON objects are enabled.

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

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

ext_custom_feature_bit_map:

This feature bitmap is an extention of custom feature bitmap and is valid only if BIT[31] of custom feature bitmap is set. This enables the following feature.
BIT[0]: To enable antenna diversity feaute.
1 – Enable antenna diversity feature
0 – Disble antenna diversity feature

BIT[1]:This bit is used to enable 4096 bit RSA key support
1 – Enable 4096 bit RSA key support
0 – Disable 4096 bit RSA key support

Note:

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

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

1 – Enable 4096 bit key support for SSL sockets

0 – Disable 4096 bit key support for SSL sockets

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

packet 1 – Enable sending broadcast without waiting for DTIM

0 – AP sends broadcast packet at DTIM only

Note:

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

BIT[5]: This bit is used to enable Pre authentication Support.

1 – Enable Pre authentication Support

0 – Disable Pre authentication Support

BIT[6]: This bit is used to enable 40MHZ Support

1 – Enable 40MHZ Support

0 – Disable 40MHZ Support

BIT[9]: EXT_HTTP_SKIP_DEFAULT_LEADING_CHARACTER

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

BIT[11]: This bit is used to enable 802.11R Over the Air Roaming Support

1 – Enable 802.11R OTA support

0 – Disable 802.11R OTA support

Note:

1.Resource Request Support is not Present.

2.If both BIT[10] and BIT[11] are Enabled then it will select only 802.11R OTA Roaming Protocol.

3.If both BIT[10] and BIT[11] are not enabled then it will select as Legacy Roaming.

BIT[12]: This bit is used to enable 802.11J support.

1 – Enable 802.11J support

0 – Disable 802.11J support

Note: If this bit is enable , set region command is mandatory and band value must be 1.

BIT[13]: This bit is used to enable 802.11W support.

1 – Enable 802.11W support

0 – Disable 802.11W support

BIT[14]: This bit is used to set TLS Multiple versions Support in SSL

1 – Enable TLS Multiple Version Support

0 – Disable TLS Multiple Version Support

BIT[15]: This bit is used to Support 16 Stations in AP Mode.

1 – Enable 16 Stations support in AP mode

0 – Disable 16 Stations support in AP mode

Note: If this bit is enable then 16 stations can connect in AP mode otherwise Maximum of 8 stations can connect.

BIT[16]: This bit is used to enable 802.11R Over the Distributed System Roaming Support

1 – Enable 802.11R ODS support

0 – Disable 802.11R ODS support

BIT[19]:This bit is used to enable low power mode in GPIO based ULP power save to achieve low power in BLE Different states like advertising, scanning and connected state.

1 - enable

0 - disable

BIT[20]:This bit is used to configure 320k mode .

1 - enable

0 - disable

BIT[21] - This bit is enabled to configure 256k mode when 192k is enabled by default.

1 - enable

0 - disable

(BIT[20] | BIT[21])-This bit is used to configure 384k mode.

Note : This is applicable only in WiSeConnect product mode

1- enable

0-disable

ext_tcp_ip_feature_bit_map:

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

1- Enable DHCP USER CLASS

0- Disable DHCP USER CLASS

BIT[2]: This bit is used to enable HTTP server root path bypass enabled

1- Enable HTTP server root path bypass

0- Disable HTTP server root path bypass

bt_feature_bit_map:

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

BIT[0:1] – HP/LP Chain selection in bt classic

0 - BDR/EDR HP chain

1- BDR HP chain

2- BDR LP chain

BIT[2:14] – reserved

BIT[15] – HFP profile bit enable

1- enable the HFP profile

0- disable the HFP profile

BIT[16:19] – reserved for future use

BIT[20:22] – number of slaves supported by BT

Maximum no of bt slaves: 2

BIT [23] – A2DP profile bit enable

1- enable the A2DP profile

0- disable the A2DP profile

BIT [24] – A2DP profile role selection

1- A2DP source

0- A2DP sink

BIT [25] – A2DP accelerated mode selection

1- enable accelerated mode

0- disable accelerated mode

BIT [26] – A2DP i2s mode selection

1- enable i2s mode

0- disable i2s mode

BIT [27:29] – reserved

BIT[30] – RF Type selection

1 - Internal Rf Type selection

0 - External Rf Type selection

BIT[31] - Validate ble feature bit map.

1 - valid ble feature bit map

0 - Ignore ble feature bit map

ble_feature_bit_map:

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

BIT [0:7] – BLE nbr of attributes,

Maximum No of Ble attributes = 80, Please refer **NOTE** given below for more info

BIT[8:11] – BLE Nbr of GATT services

maximum no services - 10, Please refer **NOTE** given below for more info

BIT [12:15] – BLE Nbr of slaves

Maximum No of Ble slaves = 8, Please refer **NOTE** given below for more info

BIT[16:23] – BLE tx powersave index

Give 31 as ble tx power index (eg: 31<<16)

This variable is used to select the ble tx power index value.The following are the possible values.

Default Value for BLE Tx Power Index is 31
Range for the BLE Tx Power Index is 1 to 75 (0, 32 index is invalid)

1 - 31 BLE -0DBM Mode

33 - 63 BLE- 10DBM Mode

64- 75 BLE - HP Mode.

BIT[24:26] – BLE powersave options

BLE_DUTY_CYCLING BIT(24)

BLR_DUTY_CYCLING BIT(25)

BLE_4X_PWR_SAVE_MODE BIT(26)

BIT [27:28] - BLE Nbr of masters

Maximum No od BLE Masters = 2, Please refer **NOTE** given below for more info

BIT[29] - Reserved

BIT[30] - To ensure the RS9113 - RS9116 compatible features

1 - enable the 9116 compatible features

0 - enable the 9113 compatible features

BIT[31] - Reserved

NOTE:
If bit bt_feature_bit_map[31] is set:

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

If bit bt_feature_bit_map is not set:

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

Response:
AT Mode:

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

Binary Mode:
There is no response payload for this command.

Example 1:
To enable wlan and ble operating mode with ble power save index as 31 and 9116 compatible feature enabled.

AT Mode:

```
at+rsi_opermode=851968,0,1,2147483648,2149580800,3221225472,0,1075773440\r\n
```

Response:
OK\r\n

bt_loaded\r\n

Example 2:
To enable 7 ble slaves , 5 services and 25 attributes, 30 ble power save index

Command in AT Mode:

```
at+rsi_opermode=851968,0,1,2147483648,2149580800,3221225472,0,1996057\r\n
```

Response:
OK\r\n

bt_loaded\r\n

6.1.2 Query RSSI

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[RSI_BT_BD_ADDR_LEN];
}QueryRssiFrameSnd;
UINT08 uQueryRSSIBuf[RSI_BT_BD_ADDR_LEN];
} RSI_BT_CMD_QUERY_RSSI;
```

AT command format:

```
at+rsibt_getrssi=<BDAddress>?\r\n
Parameters:
BDAddress - BT Address of the connected remote device.
Response Payload:
typedef struct rsi_bt_resp_query_rssi {
UINT08 RSSI;
} RSI_BT_RESP_QUERY_RSSI;
```

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

Response parameters:
RSSI – RSSI value of the connected remote device.

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

Response:
OK 130\r\n

6.1.3 Query Local BD Address

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

Payload Structure:
No Payload required.

AT command format:

```
at+rsibt_getlocalbdaddr?\r\n
```

Response Payload:

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

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

Response Parameters:
BDAddress - BT Address of the local device

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

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

6.2 BLE Core commands

6.2.1 Advertise Local Device

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

Payload Structure:

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

AT Command format:

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

Note:

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

Parameters:
Status – To enable/disable Advertising.
1 – Enable Advertising
0 – Disable Advertising

AdvertiseType –

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

FilterType –

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

DirectAddrType –

0 – Public address
1 – Random address

DirectAddr – Remote device BD Address
adv_int_min-

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

adv_int_max-

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

own_add_type-

Value	Parameter Description
0x00	Public Device Address (default)

Value	Parameter Description
0x01	Random Device Address
0x02 – 0xFF	Reserved for future use

adv_channel_map-

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

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

AT command Ex:
at+rsibt_advertise=1,128,0,0,0,32,32,0,7\r\n(enable)

Response:
OK\r\n

Note:

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

6.2.2 Scan

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

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

AT Command format:

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

Note:
All Parameters should be in Decimal.

Parameters:
Status – To enable/disable Scanning
1 – Enable Scanning
0 – Disable Scanning
Scantype –

Scan type	Description
0	Passive Scanning
1	Active Scanning

FilterType –

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

Scan_int-

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

scan_win-

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

own_add_type-

Value	Parameter Description
0x00	Public Device Address (default)

Value	Parameter Description
0x01	Random Device Address
0x02 – 0xFF	Reserved for future use

Response Payload:

There is no response payload for this command.

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

Response: OK\r\n

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

Response: OK\r\n

Note:

In parameters Scan Interval must be greater than scan window.

6.2.3 Connect

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

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

AT Command format:

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

Parameters:
AddressType – Specifies the type of the address mentioned in BDAdress
0 – Public Address
1 – Random Address

BDAdress – BD Address of the remote device

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

LE_Scan_Interval:

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

LE_Scan_Window:

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

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

Conn_Interval_Min:

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

Conn_Interval_Max:

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

Conn_Latency:

The conn_latency parameter defines the maximum allowed connection Latency.

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

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

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

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

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

Response: OK\r\n

Note:
Reception of the response doesn't mean that Connection with the remote device is completed.
Connection is said to complete after it receives **Connection Complete Event**. The user is recommended not to give further commands before receiving the above Event. But the user is allowed to give **Disconnect** even before **Connection Complete Event** is received.

6.2.4 Disconnect

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

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

AT Command format*:*

```
at+rsibt_disconnect=<BDAddress>\r\n
```

Parameters:
BDAdress – BD Address of the remote device
Response Payload:
There is no response payload for this command.
AT command Ex: at+rsibt_disconnect=53-41-CC-FF-91-2A\r\n
Response: OK\r\n

6.2.5 Query Device State

Description: This is used to query state of the local device.
Payload Structure: No Payload required.
AT Command format:

```
at+rsibt_getdevstate?\r\n
```

Response Payload:

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

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

Response Parameters:
DeviceState –

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

AT command Ex: at+rsibt_getdevstate?\r\n
Response: OK 8\r\n

6.2.6 Start Encryption

Description: This is used to initiate the Encryption procedure.
Payload Structure:

```
struct {
UINT08 BDAddress[RSI_BT_BD_ADDR_LEN];
UINT16  RemoteEDIV;
UINT08  RemoteRand[8];
UINT08  RemoteLTK[16];
} RSI_BLE_CMD_START_ENCRYPTION;
```

AT command Format: at+rsibt_startencrypt=<BDAddress>,<RemoteEDIV>,<RemoteRand>,<RemoteLTK>\r\n
Parameters:
BDAddress - Remote BD Address.

RemoteEDIV - Remote device Encryption Diversifier

RemoteRand - Remote device Random number

RemoteLTK - Remote device Long Term Key

Response Payload:
There is no response payload for this command.
AT command Ex: at+rsibt_startencrypt=B4-99-4C-64-BE-F5,253C,45,B6,53,7A,37,42,85,9D,65,BA,84,DA,CC,56,85,9D,3A,74,3C,45,23,78,4D,3A\r\n
Response: OK\r\n

6.2.7 SMP Pair Request

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

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

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

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

Response Payload:
There is no response payload for this command.
AT command Ex: at+rsibt_smpreq=B4-99-4C-64-BE-F5,1\r\n
Response: OK\r\n

6.2.8 SMP Response

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

```
typedef union {
struct {
UINT08 BDAddress[RSI_BT_BD_ADDR_LEN];
UINT08 IOCapability;
}SmpRespFrameSnd ;
UINT08 uSmpRespBuf[RSI_BT_BD_ADDR_LEN + 1];
} RSI_BLE_CMD_SMP_RESPONSE ;
AT Command format:at+rsibt_smpresp=<BDAddress>,<IOCapability>\r\n
```

Parameters:
IOCapability –

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

Response Payload:
There is no response payload for this command.
AT command Ex: at+rsibt_smpresp=74-04-2B-7A-93-EF,1\r\n
Response: OK\r\n

6.2.9 SMP Passkey

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

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

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

Parameters:
Passkey – Passkey to authenticate with the Remote device.

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

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

Response: OK\r\n

6.2.10 Initialize BLE module

Description: This is used to initialize the BLE module
AT Command format:

```
at+rsibt_btinit\r\n
```

Payload Structure:
No Payload required.

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

AT command Ex: at+rsibt_btinit\r\n

Response: OK\r\n

6.2.11 Deinitialize BLE module

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

```
at+rsibt_btdeinit\r\n
```

Payload Structure:
No Payload required.

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

AT command Ex: at+rsibt_btdeinit\r\n

Response: OK\r\n

6.2.12 BT Antenna Select

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

```
typedef struct rsi_bt_cmd_antenna_select{
UINT08 AntennaVal;
} RSI_BT_CMD_ANTENNA_SELECT;
```

AT Command format:

```
at+rsibt_btantennaselect=<AntennaVal>\r\n
```

Parameters:
AntennaVal – To select the internal or external antenna
0 – Internal Antenna.
1 – External Antenna.
Response Payload:
There is no response payload for this command.

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

6.2.13 BLE Set Advertise Data

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

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

AT Command format:

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

Parameters:
Length – data length. Max advertise data length is 31.
Data – Actual data
Response Payload:
There is no response payload for this command.
AT command Ex: at+rsibt_setadvertisedata=8,2,1,6,4,9,72,72,72\r\n
Response: OK\r\n

Note:
Name set in this command will be shown on remote device when remote device scan for advertising device(except for ios versions, refer NOTEs in section:2.3.1.1)
Advertise and Scan response data format

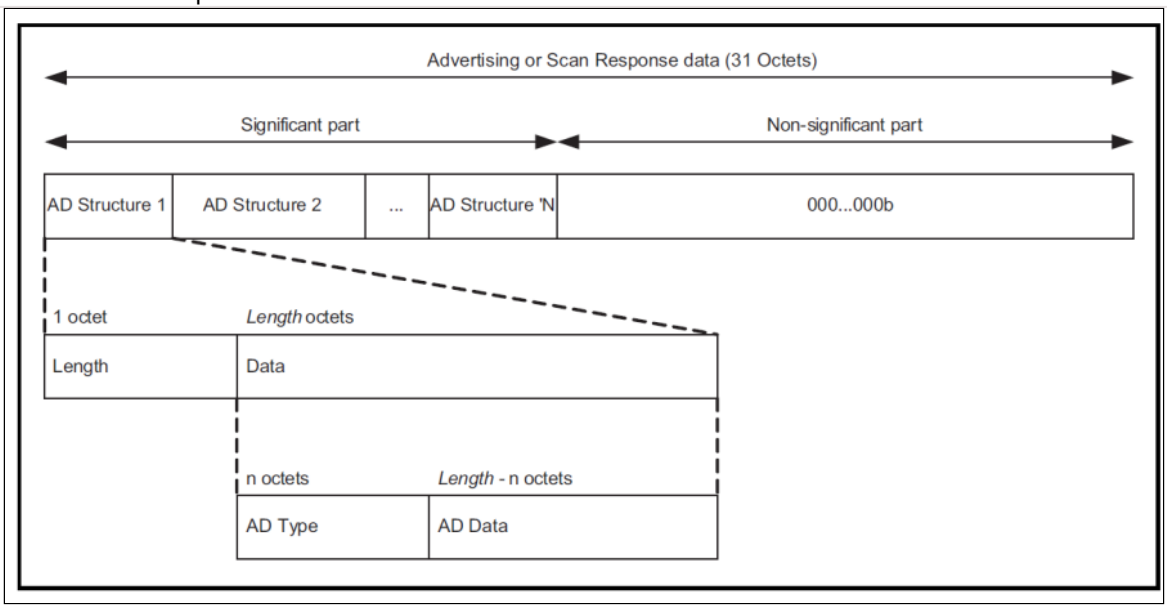


Figure 45: Advertising or scan response data

For more information on this advertising data with types, information is available at following link:
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

6.2.14 BLE Set Scan Response Data

Description:

This command is used to set the scan response data.

Payload Structure:

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

AT Command format:

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

Parameters:

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

Data – Actual data

Response Payload:

There is no response payload for this command.

AT command Ex:

at+rsibt_setscanrspdata=8,2,1,6,4,9,72,72,72\r\n

Response:

OK\r\n

Note:
Name set in this command will be shown on remote device when remote device scan for advertising device(except for ios versions, refer NOTEs in section:2.3.1.1)

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

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

Payload Structure:

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

AT command Format:

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

Parameters:
BDAddress: Remote BD Address.
Timeout

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

Response Payload:

There is no response payload for this command.

AT command Ex:

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

Response:

OK\r\n

6.2.15 BLE Get LE ping timeout. Currently Get LE ping is not supported

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

```
typedef struct rsi_ble_resp_get_ping_timeout{
UINT16 Timeout;
} RSI_BLE_RESP_GET_PING_TIMEOUT;
```

AT command Format:

```
at+rsibt_getlepingtimeout=<Connected Remote BD_ADDR>\r\n
```

Response Payload:

There is no response payload for this command.

AT command Ex:

at+rsibt_setlepingtimeout?\r\n

Response:

OK 30000\r\n

Description:

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

Payload Structure:

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

AT command Format:

```
at+rsibt_setrandadd=<BDAddress>\r\n
```

Parameters:
BDAddress:

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

Return Parameters:
Status:

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

Response Payload:

OK 0 \r\n

AT command Ex:

at+rsibt_setrandadd= B4-99-4C-64-BE-F5\r\n

Response:

OK 0\r\n

6.2.16 BLE Data Encrypt

Description: This command is used to encrypt the data.
Payload Structure:

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

AT command Format:
at+rsibt_leencrypt=<key>,<data>\r\n
Parameters:
key – key length is 16 Bytes.
Data – Actual data , length is 16 Bytes.
Response Payload:

typedef struct rsi_ble_resp_data_encrypt {

```
UINT08 EncData[16];
}RSI_BLE_RESP_DATA_ENCRYPT;
```

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

AT command Ex: at+rsibt_leencrypt=1,2,3,4,5,6,7,8,9,0,B,C,D,E,F,10, 1,2,3,4,5,6,7,8,9,0,B,C,D,E,F,10\r\n
Response: OK 10 93 c1 5a e4 a5 db fd 5e c2 4c 61 8a a3 11 28 \r\n

6.2.17 BLE Whitelist

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

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

AT command format: at+rsibt_lewhitelist=<Add/deletebit>,<BDAddress>,<BDAddressType>\r\n

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

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

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

Response Payload:
There is no response payload for this command

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

Response:

OK\r\n

6.2.18 BLE Set Phy command

Description: The LE_Set_PHY command is used to set the PHY preferences for the connection identified by the Connection_Handle.
Binary Payload Structure:

```
typedef struct ble_set_phy {
UINT08 AllPhys;
UINT08 TxPhy;
UINT08 RxPhy;
UINT16 PhyOptions;
} BLE_SET_PHY;
```

AT command format:
at+rsibt_setphy=<remotebdaddress>,<all_phy>,<tx_phy>,<rx_phy>,<phy_ options>

Parameters:

BDAddress – BDAddress of the remote device.

<All_phy>– The Host has no preference among the transmitter PHYs supported by the Controller.

< tx_phy >-
0 - The Host prefers to use the LE 1M transmitter PHY (possibly among others)
1 - The Host prefers to use the LE 2M transmitter PHY (possibly among others)
2 - The Host prefers to use the LE Coded transmitter PHY (possibly among others)
3 – 7 Reserved for future use

<rx_phy> -
0 - The Host prefers to use the LE 1M receiver PHY (possibly among others)
1 - The Host prefers to use the LE 2M receiver PHY (possibly among others)
2 - The Host prefers to use the LE Coded receiver PHY (possibly among others)
3 – 7 Reserved for future use

<phy_ options>
0 = the Host has no preferred coding when transmitting on the LE Coded

PHY
1 = the Host prefers that S=2 coding be used when transmitting on the LE

Coded PHY
2 = the Host prefers that S=8 coding be used when transmitting on the LE

Coded PHY
3 = Reserved for future use

Response Payload:
There is no response payload for this command

ATcommandEx:
at+rsibt_setphy=00-23-A7-00-00-0D,0,1,2,0

Response: OK\r\n

6.2.19 BLE Read Phy command

Description: The LE_Read_PHY command is used to read the current transmitter PHY and receiver PHY on the connection identified by the Connection_Handle.
Binary Payload Structure:

```
typedef struct rsi_ble_cmd_read_phy {
UINT08 BDAddress[6];
}RSI_BLE_CMD_READ_PHY;
```

AT command format: at+rsibt_readphy=<remotebdaddr>

Parameters:
BDAddress – BDAddress of the remote device.

Response Payload:
There is no response payload for this command

ATcommandEx:
eg:at+rsibt_readphy=00-23-A7-00-00-0D

Response:

OK\r\n

6.2.20 BLE Set Data Length Command

Description: The LE_Set_Data_Length command allows the Host to suggest maximum transmission packet size and maximum packet transmission time.
Binary Payload Structure:

```
typedef struct rsi_ble_cmd_set_data_length {
UINT08 BDAddress[6];
UINT16 TxOctets;
UINT16 TxTime;
} RSI_BLE_CMD_SET_DATA_LENGTH;
```

AT command format:

At+rsibt_setdatalength=<BDAddress>,<TXOctets><TXTime>\r\n

Parameters:
BDAddress – BDAddress of the remote device.
TXOctets– Preferred maximum number of payload octets that the local I Controller should include in a single Link Layer packet on this connection.
TXTime-Preferred maximum number of microseconds that the local Controller should use to transmit a single Link Layer packet on this connection.

Response Payload:
There is no response payload for this command

ATcommandEx:
at+rsibt_setdatalength =00-23-A7-80-6F-CD,40,330,\r\n

Response:

OK\r\n

6.2.21 BLE Read Maximum Data Length Command

Description: The LE_Read_Maximum_Data_Length command allows the Host to read the Controller's maximum supported payload octets and packet duration times for transmission and reception.

Binary Payload Structure:

```
typedef struct rsi_ble_resp_read_maximum_data_length {
UINT16 supportedMaxTxOctets;
UINT16 supportedMaxTxTime;
UINT16 supportedMaxRxOctets;
UINT16 supportedMaxRxTime;
} RSI_BLE_RESP_READ_MAXIMUM_DATA_LENGTH;
```

AT command format: At+rsibt_readdatalength?

Response Payload:
There is no response payload for this command

ATcommandEx:
at+rsibt_readdatalength?\r\n

Response:

OK\r\n

6.2.22 BLE_Resolvlist

Description: The Resolving_List command is used to add/remove/clear one deviceto the list of address translations used to resolve Resolvable Private Addresses in the Controller.
Binary Payload Structure:

```
typedef struct rsi_ble_cmd_resolving_list {
UINT08 Type;
UINT08 Peer_Identity_Address_Type;
UINT08 Peer_Identity_Address[6];
UINT08 Peer_IRK[16];
UINT08 Local_IRK[16];
} RSI_BLE_CMD_RESOLVING_LIST;
```

AT command format: at+rsibt_resolvlist=<Process_type>,<address_type>,<peer_address>,<peer_irk>,<local_irk>

Parameters:
<Process_type>
1 = Add
2=Remove
3=Clear Resolvlist.

<AddressType > =
0x00 Public Identity Address
0x01 Random (static) Identity Address
0x02 – 0xFF Reserved for Future Use

<peer_address> = Public or Random (static) Identity address of the peer device

< peer_irk> = IRK of the peer device.

<local_irk> = IRK of the local device.

Response Payload:
There is no response payload for this command

ATcommandEx:
at+rsibt_resolvlist=1,0,00-23-A7-00-00-0D,
01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,02,02,02,02,02,02,02,02,02,02,02,02,02,02,02,02
-->remove device from resolvlist
at+rsibt_resolvlist=2,<address_type>,<peer_address>

eg:at+rsibt_resolvlist=2,0,00-23-A7-00-00-0D
-->clear resolvlist
at+rsibt_resolvlist=3

Response:

OK\r\n

6.2.23 BLE GetResolvlist Size

Description: The BLE_Read_Resolving_List_Size command is used to read the total number of address translation entries in the resolving list that can be stored in the Controller.

```
Binary Payload Structure:
typedef struct rsi_ble_resp_read_resolving_list_size {
UINT08 size;
} RSI_BLE_RESP_READ_RESOLVING_LIST_SIZE;
```

AT command format:

At+rsibt_ getresolvlistsize?\r\n

Response Payload:
There is no response payload for this command

ATcommandEx:
at+rsibt_ getresolvlistsize?\r\n

Response:

OK\r\n

6.2.24 BLE SetResolution Enable

Description: The BLE_Set_Address_Resolution_Enable command is used to enable resolution of Resolvable Private Addresses in the Controller,The LE _Timeout command set the length of time the Controller uses a Resolvable Private Address before a new resolvable private address is generated and starts being used.This timeout applies to all addresses generated by the Controller.

Binary Payload Structure:

```
typedef struct rsi_ble_cmd_addr_resolution_enable {
UINT08 Enable;
UINT16 RPA_Timeout;
} RSI_BLE_CMD_ADDR_RESOLUTION_ENABLE;
```

AT command format:

at+rsibt_setresolutionenable=<enable>,<timeout>

Parameters:
<enable> =
0-disable
1-enable

<timeout>-RPA_Timeout measured in s

Range for N: 0x0001 – 0xA1B8 (1 s – approximately 11.5 hours)

Default: N= 0x0384 (900 s or 15 minutes)

Response Payload:
There is no response payload for this command

ATcommandEx:
eg:at+rsibt_setresolutionenable=1,60

Response:

OK\r\n

6.2.25 BLE SetPrivacy Mode

Description:

The HCI_LE_Set_Privacy_Mode command is used to allow the Host to specify the privacy mode to be used for a given entry on the resolving list.

Binary Payload Structure:

```
typedef struct rsi_ble_cmd_set_privacy_mode {
UINT08 Peer_Identity_Address_Type;
UINT08 Peer_Identity_Address[6];
UINT08 privacy_mode;
} RSI_BLE_CMD_SET_PRIVACY_MODE;
```

AT command format:
at+rsibt_setprivacymode=<peer_addr_type>,<peer_addr>,<privacy_mode>**Parameters:**

<peer_addr_type>-

Value Parameter Description
0x00 Public Identity Address
0x01 Random (static) Identity Address

All other values Reserved for future use

<peer_addr> - Public Identity Address or Random (static) Identity Address of the advertiser.

<privacy_mode> -
0x00 Use Network Privacy Mode for this peer device (default)
0x01 Use Device Privacy Mode for this peer device

All other values Reserved for future use

Response Payload:
There is no response payload for this command

ATcommandEx:
eg:at+rsibt_setprivacymode=0,00-23-A7-00-00-0D,1

Response: OK\r\n

6.2.26 BLE Connection Update Command

Description: The LE_UPDATE_PARAMS command is used to change the Link Layer connection parameters of a connection.

Binary Payload Structure:

```
typedef struct bt_le_connupdate_params {
  UINT16   MinInterval;
  UINT16   MaxInterval;
  UINT16   Latency;
  UINT16   Timeout;
  UINT16   Min_CE_Len;
  UINT16   Max_CE_Len;
}BLE_CONNPARAMS_UPDATE;
```

AT command format:
at+rsibt_updateparams=<bd_addr>,<Conn_Interval_Min>,<Conn_Interval_Max>,<Conn_Latency>,<Supervision_Timeout>

Parameters:

bd_addr – BDAddress of the remote device.

Conn_Interval_Min (in dec) – Minimum value for the connection interval. This shall be less than or equal to Conn_Interval_Max.

Conn_Interval_Max (in dec) – Maximum value for the connection interval. This shall be greater than or equal to Conn_Interval_Min.

Min and max interval Range : 6 to 3200 (Time = N * 1.25 ms, Time Range: 7.5 ms to 4 s.)

Conn_Latency (in dec) – Slave latency for the connection in number of connection events.

Range: 0 to 499

Supervision_Timeout (in dec) – Supervision timeout for the LE Link.

Range: 10 to 3200 (Time = N * 10 ms, Time Range: 100 ms to 32 s)

No need to pass Min_CE_Len and Max_CE_Len value.

All other values Reserved for future use

Response Payload:
There is no response payload for this command

ATcommandEx:

at+rsibt_updateparams=00-1A-7D-DA-71-13,12,12,0,500

Response: OK\r\n

6.3 BLE GATT Profile commands

6.3.1 Query profiles list

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

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

AT Command format: at+rsibt_getallprofiles=<BDAddress>,<StartHandle>,<EndHandle>\r\n

Parameters:

BDAddress – Remote BD Address.

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

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

Response Payload:

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

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

Response Parameters:

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

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

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

Response: OK 3\n
1,B,2,1800\n
C,F,2,1801\n
10,FFFF,2,180A\r\n

6.3.2 Query Profile

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[6];
UINT08 Reserved[2];
UUID_T ProfileUUID;
}QueryProfileDescFrameSnd ;
UINT08 uQueryProfileDescBuf[28];
} RSI_BLE_CMD_QUERY_PROFILE ;
```

AT command format: at+rsibt_getprofile=<BDAddress>,<size_uuid>,<ProfileUUID>\r\n

Parameters:

BDAddress – Remote BD Address.
ProfileUUID – UUID of the profile.

Response Payload:

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

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

Response Parameters:

ProfileDescriptor – PROFILE_DESCRIPTOR is explained above.

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

Response:

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

6.3.3 Query Characteristic Services

Description:

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

Payload Structure:

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

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

Parameters:

BDAddress – Remote BD Address

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

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

Response Payload:

```
typedef struct bt_le_char_serv {
UINT08 CharacterProperty;
UINT08 Reserved;
UINT16 CharacterHandle;
UUID_T CharacterUUID;
} BT_LE_CHAR_SERV;
typedef struct characteristic_service {
UINT16 Handle;
UINT08 Reserved[2];
BT_LE_CHAR_SERV CharServ;
}CHARACTERISTIC_SERVICE ;
typedef struct rsi_ble_resp_query_char_services {
UINT08 NumberOfCharServices;
UINT08 reserved[3];
CHARACTERISTIC_SERVICE CharacteristicService[5];
} RSI_BLE_RESP_QUERY_CHARACTERISTIC_SERVICES;
```

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

Response parameters:

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

CHARACTERISTIC_SERVICE Structure members	Size/type	Description
CharacterUUID	UUID	Characteristic UUID

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

AT COMMAND Ex:

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

Response:

OK 5, 2,2,3,2,2A00\n
4,2,5,2,2A01\n
6,2,7,2,2A02\n
8,8,9,2,2A03\n
A,2,B,2,2A04\r\n

6.3.4 Query Include Services

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[6];
UINT08 StartHandle[2];
UINT08 EndHandle[2];
}QueryIncludeServFrameSnd ;
UINT08 uQueryIncludeServBuf[10];
} RSI_BLE_CMD_QUERY_INCLUDE_SERVICES ;
```

AT Command format: at+rsibt_getincservices=<BDAddress>,<StartHandle>,<EndHandle>\r\n

Parameters:

BDAddress – Remote BD Address.

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

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

Response Payload:

```
typedef struct bt_le_inc_serv {
UINT16 IncludeStartHandle;
UINT16 IncludeEndHandle;
UUID IncludeUUID;
} BT_LE_INC_SERV;
typedef struct include_service {
UINT16 Handle;
UINT08 Reserved[2];
BT_LE_INC_SERV IncServ;
}INCLUDE_SERVICE ;
typedef struct rsi_ble_resp_query_include_service {
UINT08 NumberOfIncludeServices;
UINT08 reserved[3];
INCLUDE_SERVICE IncludeServices[BLE_MAX_RESP_LIST];
} RSI_BLE_RESP_QUERY_INCLUDE_SERVICE;
```

Response Parameters:

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

RSI_BLE_RESP_QUERY_INCLUDE_SERVICE Structure members	Size/type	Description
NumberOfIncludeServices	8 bit	No of Include services.
IncludeServices[5]	INCLUDE_SERVICE	Each Include Service details.

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

Response: OK 0,\r\n

6.3.5 Read Characteristic Value by UUID

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[6];
UINT08 StartHandle[2];
UINT08 EndHandle[2];
UINT08 Reserved[2];
UUID_T CharacterUUID;
}ReadCharValByUuidFrameSnd ;
UINT08 uReadCharValByUuidBuf[12 + sizeof(UUID_T)];
} RSI_BLE_CMD_READ_CHAR_VALUE_BY_UUID ;
```

AT Command format: at+rsibt_readbytype=<BDAddress>,<StartHandle>,<Endhandle>,<size>, <UUID>\r\n

Parameters:

BDAddress – Remote BD Address.

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

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

Reserved - Padding

CharacterUUID – UUID whose Attribute values are to be known.

Response Payload:

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

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

Response Parameters:

NumberOfValues – Number of valid bytes in the CharacterValue.

CharacterValue – Attribute value of the given CharacterUUID

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

Response:

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

6.3.6 Query Attribute

Description:

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[6];
UINT08 StartHandle[2];
UINT08 EndHandle[2];
}QueryAttFrameSnd ;
UINT08 uQueryAttBuf[10];
} RSI_BLE_CMD_QUERY_ATT_DESC ;
```

AT Command Format: at+rsibt_getdescriptors=<BDAddress>,<StartHandle>,<EndHandle>\r\n

Parameters:

BDAddress – Remote BD Address.

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

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

Response Payload:

```
typedef struct attribute_descriptor {
UINT16 Handle[2];
UINT08 reserved[2];
UUID_T AttributeTypeUUID;
} ATTRIBUTE_DESCRIPTOR;
typedef struct rsi_ble_resp_query_att_desc {
UINT08 NumberOfAttributes;
UINT08 reserved[3];
ATTRIBUTE_DESCRIPTOR AttributeDescriptor[BLE_MAX_RESP_LIST];
} RSI_BLE_RESP_QUERY_ATT_DESC;
```

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

Response Parameters:

Handle – Handle of each Attribute

AttributeTypeUUID – UUID of each Attribute

NumberOfAttributes – No of attributes

AttributeDescriptor – Descriptor of each Attribute

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

Response: OK 5
1,2,2800
2,2,2803
3,2,2A00
4,2,2803
5,2,2A01\r\n

6.3.7 Query Attribute Value

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[6];
UINT08 Handle[2];
}QueryAttValFrameSnd ;
UINT08 uQueryAttValBuf[8];
} RSI_BLE_CMD_QUERY_ATT_VALUE ;
```

AT Command format: at+rsibt_readvalue=<BDAddress>,<Handle>\r\n

Parameters:

BDAddress – Remote BD Address.

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

Response Payload:

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

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

Response Parameters:

NumberOfValues – No of valid bytes in the AttributeValues

AttributeValues – Attribute value of the specified Handle

AT command Ex:

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

Response:

OK 2,0,18\r\n

6.3.8 LE L2CAP Credit Based Flow Control Connection Request

Description:

This command is used to initiate new PSM connection with remote device.

Payload Structure:

```
typedef struct rsi_ble_cmd_psm_conn_req {

    UINT08  BDAddress[6];

    UINT16  PSM;

} RSI_BLE_CMD_PSM_CONN_REQ;
```

Command format:

at+rsibt_psmconnreq=<remote_addr>, <psm>\r\n

Parameters:

BDAddress – Remote BD Address.

PSM - Protocol/Service Multiplexer (PSM) This field helps to indicate the protocol.

Response Payload:

There is no response payload for this command

AT CommandEx:

Ex: at+rsibt_psmconnreq=00-23-A7-00-00-0A,23

Response: OK\r\n

6.3.9 LE L2CAP Credit Based Flow Control Data Trasnfer

Description: This command is used to send data to remote device through specific PSM connection.

Binary Payload Structure:

```
typedef struct rsi_ble_cmd_psm_data {

    UINT08  BDAddress[6];

    UINT16  Lcid;

    UINT16  Len;

    UINT08  Data[MAX_GATT_EVENT_DATA_LEN];

} RSI_BLE_CMD_PSM_DATA;
```

AT command format:

at+rsibt_sendpsmdata=<remote_addr>, <lcid>,<data_len>,<data>\r\n

Parameters:

Remote-BDAddress – BDAddress of the remote device

Lcid – local channel identifiervalue

Data_len – length of the data to be sent to remote device.

Data –Actual data that has to be sent

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_sendpsmdata=00-232-A7-00-00-09, 80,5,1,1,2,3,4 r\n

Response:

OK\r\n

6.3.10 LE L2CAP Credit Based Flow Control Connection Response

Description:

This command is used to accept or reject the remote device PSM connection request.

Binary Payload Structure:

```
binary commad staructure:

typedef struct rsi_ble_cmd_psm_conn_resp{

    UINT08  BDAddress[6];

    UINT16  LCID;

    UINT08  Result;

} RSI_BLE_CMD_PSM_CONN_RESP;
```

AT command format:

at+rsibt_psmconnresp=<remote_addr>,<lcid>,<resp>\r\n

Parameters:

Remote-BDAddress – BDAddress of the remote device.

Lcid – local channel identifiervalue

resp: 0 - Reject the PSM connection request

1 - Accept the PSM connection request

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_psmconnresp=00-232-A7-00-00-09,1 r\n

Response:

OK\r\n

6.3.11 LE L2CAP Credit Based Flow Control Disconnection

Description:

This command is used to disconnect the logical connection between the devices.

Binary Payload Structure:

```
binary commad staructure:

typedef struct rsi_ble_cmd_psm_disconn {

    UINT08  BDAddress[6];

    UINT16  Lcid;

} RSI_BLE_CMD_PSM_DISCONN;
```

AT command format:

at+rsibt_psmdisconn=<remote_addr>, <lcid>\r\n

Parameters:

Remote-BDAddress – BDAddress of the remote device.

Lcid – local channel identifiervalue

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_psmdisconn =00-232-A7-00-00-09,80r\n

Response:

OK\r\n

6.3.12 LE Enhanced Receiver Test Mode

Description:

This command is used to start a test where the DUT receives test reference packets at a fixed interval.

Binary Payload Structure:

```
binary commad staructure:

typedef struct rsi_ble_cmd_en_rx_test_mode {

    UINT08 RxChannel;

    UINT08 Phy;

    UINT08 Modulation;

} RSI_BLE_CMD_EN_RX_TEST_MODE;
```

AT command format: at+rsibt_enhancedrxtest=<RX_channel>, <phy>,<Modulation>\r\n

Parameters:

<RX_channel> - Channel in which packet have to be received.

<phy>– 0x00 Reserved for future use

0x01 Receiver set to use the LE 1M PHY

0x02 Receiver set to use the LE 2M PHY

0x03 Receiver set to use the LE Coded PHY

0x04 – 0xFF Reserved for future use.

<Modulation> – 0x00 Assume transmitter will have a standard standard modulation index

0x01 Assume transmitter will have a stable modulation index

0x02 – 0xFF Reserved for future use

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_enhancedrxtest=30,1,1\ r\n

Response:

OK\r\n

6.3.13 LE Enhanced Transmitter Test Mode

Description: This command is used to start a test where the DUT generates test reference packets at a fixed interval.

Binary Payload Structure:

```
binary commad staructure:

typedef struct rsi_ble_cmd_en_tx_test_mode {

    UINT08 RxChannel;

    UINT08 Phy;

    UINT08 TxLen;

    UINT08 TxDataMode;

} RSI_BLE_CMD_EN_TX_TEST_MODE;
```

AT command format: at+rsibt_enhancedtxtest=<RX_channel>, <phy>,< TxLen>,< TxDataMode>\r\n

Parameters:

<TX_channel> - Channel in which packet have to be sent.

<phy>– 0x00 Reserved for future use

0x01 Receiver set to use the LE 1M PHY

0x02 Receiver set to use the LE 2M PHY

0x03 Receiver set to use the LE Coded PHY

0x04 – 0xFF Reserved for future use.

< TxLen> - Length in bytes of payload data in each packet.

< TxDataMode> - 0x00 PRBS9 sequence ‘11111111100000111101...

0x01 Repeated ‘11110000’

0x02 Repeated ‘10101010’

0x03 PRBS15

0x04 Repeated ‘11111111’

0x05 Repeated ‘00000000’

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_enhancedtxtest=30,1,2,0\ r\n

Response:

OK\r\n

6.3.14 LE Enhanced End Test Mode

Description:

This command is used to stop any test which is in progress.

Binary Payload Structure:

There is no payload structure.

AT command format:

at+rsibt_endtest\r\n

Return Parameters:

< NumOfPkts > - Number of RX packets received are displayed

Response Payload:

binary commad staructure:

```
typedef struct rsi_ble_rsp_en_end_test_mode {

    UINT16 NumOfPkts;

} RSI_BLE_RSP_EN_END_TEST_MODE;
```

ATcommandEx:

at+rsibt_entest\ r\n

Response:

10

6.3.15 **LE LTK Request Reply**

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

Binary Payload Structure:

```
typedef struct rsi_ble_cmd_le_ltkreqreply{

    UINT08    BDAddress[6];

    UINT08    ReplyType;

    UINT08    LocalLTK[16];

}RSI_BLE_CMD_LE_LTKREQREPLY;
```

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

Parameters:

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

Reply Type –

0- Negative reply

1 – Positive reply

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

Response Payload:

There is no response payload for this command

ATcommandEx:

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

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

Response:

OK\r\n

Description:

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[6];
UINT08 NumberOfHandles;
UINT08 Reserved;
UINT16 Handles[5];
}QueryMulAttValFrameSnd ;
UINT08 uQueryMulAttValBUF[18];
} RSI_BLE_CMD_QUERY_MULTIPLE_ATT_VALUES ;
```

AT Command format:
at+rsibt_readmultiple=<BDAddress>,<NumberofHandles>,<Handles>\r\n

Parameters:

BDAddress – Remote BD Address.

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

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

Response Payload:

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

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

Response Parameters:

NumberOfValues – No of valid bytes in the AttributeValues

AttributeValues – Attribute value of the specified Handles

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

Response:

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

6.3.16 Query Long Attribute Value

Description:

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[6];
UINT16 Handle;
UINT16 Offset;
}QueryLongAttValFrameSnd ;
UINT08 QueryLongAttValBuf[10];
} RSI_BLE_CMD_QUERY_LONG_ATT_VALUE ;
```

AT Command format: at+rsibt_longread=<BDAddress>,<Handle>,<Offset>\r\n

Parameters:

BDAddress – Remote BD Address.

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

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

Response Payload:

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

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

Response Parameters:

NumberOfValues – No of valid bytes in the LongAttValue.

LongAttValue – Attribute value of the specified Handle.

AT command Ex:

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

Response:

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

6.3.17 Set Attribute Value

Description:

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[6];
UINT08 Handle[2];
UINT08 Length;
UINT08 Value[25];
}SetAttValFrameSnd ;
UINT08 uSetAttValBuf[34];
} RSI_BLE_CMD_SET_ATT_VALUE ;
```

AT Command format:

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

Parameters:

BDAddress – Remote BD Address.

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

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

Value – Value to be set in the specified Handle

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

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

Response:

OK \r\n

6.3.18 Set Attribute Value no Ack

Description:

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

Payload Structure:

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

AT Command format:

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

Parameters:

BDAddress – Remote BD Address.

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

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

Value – Value to be set in the specified Handle

Response Payload:

There is no response payload for this command.

AT command Ex:

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

Response:

OK\r\n

6.3.19 Set Long Attribute Value

Description:

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

Payload Structure:

```
typedef union {
struct {
UINT08 BDAddress[6];
UINT08 Handle[2];
UINT08 Offset[2];
UINT08 Length;
UINT08 Value[40];
}SetLongAttValFrameSnd;
UINT08 uSetLongAttValBuf[51];
}RSI_BLE_CMD_SET_LONG_ATT_VALUE;
```

AT Command format:

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

Parameters:

BDAddress – Remote BD Address.

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

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

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

Value – Value to be set in the specified Handle

Response Payload:

There is no response payload for this command.

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

Response: OK\r\n

6.3.20 Set Prepare Long Attribute Value

Description:

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

Payload Structure:

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

AT Command format:

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

Parameters:

BDAddress – Remote BD Address.

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

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

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

Value – Value to be set in the specified Handle

Response Payload:

There is no response payload for this command.

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

Response:

OK \r\n (or)

ERROR, err_no\r\n

6.3.21 Execute Long Attribute Value

Description:

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

Payload Structure:

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

AT Command format:

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

Parameters:

BDAddress – Remote BD Address.

Flag – This parameter decides whether to set/not set the previously sent Prepare Long Attribute values.
0 – Don't set the values
1 – Set the values

Response Payload:

There is no response payload for this command.

AT command Ex:

at+rsibt_executewrite=C0-FF-EE-C0-FF-EE,1\r\n

Response:

OK\r\n

6.3.22 Read Response

Description :

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

Binary Payload Structure:

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

6.4 BLE Create New Service Commands

6.4.1 Add GATT Service Record

Description:

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

Payload Structure:

```
typedef union {
struct {
UUID_T ServiceUUID;
UINT16 NbrAttributes;
UINT16 MaxAttDataSize;
} AddServiceRecord;
UINT08 uAddServiceRecord[24];
} RSI_BLE_CMD_ADD_GATT_SERVICE;
```

AT Command format:

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

Parameters:

ServiceUUID – BLE supporting service UUID value.

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

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

Response Payload:

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

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

Result Code	Description
ERROR <Error_code>	Command Fail.

Response Parameters:

ServiceHndlerPtr – created GATT service record handle.

StartHndl - service record starting attribute handle value.

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

Response:

OK 157A8,A\r\n

6.4.2 Add Attribute Record

Description:

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

Payload Structure:

```
typedef union {
struct {
void *ServiceHndlerPtr;
UINT16 Hndl;
UINT16 Reserved;
UUID_T AttUUID;
UINT08 Prop;
UINT08 Data[31];
UINT16 DataLen;
} AddAttRecord;
UINT08 uAddAttRecord[62];
} RSI_BLE_CMD_ADD_GATT_ATTRIBUTE;
```

AT Command format:

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

Parameters:

ServiceHndlerPtr – service record handle.

Hndl – handle of the attribute record.

AttUUID – attribute record UUID.

Data – attribute record data value.

DataLen – attribute record data length.

Prop - property of the attribute.

Response Payload:

There is no response payload for this command.

AT command Ex:

at+rsibt_addattribute=157A8,A,2,2A02,1,1,FF\r\n

Response:

OK\r\n

6.4.3 Set Local Attribute Value

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

Payload Structure:

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

AT Command format: at+rsibt_setlocalattvalue=<Hndl>,<DataLen>,<Data>\r\n

Parameters:

Hndl – handle of the attribute record.

DataLen – attribute record data length.

Data – attribute record data value.

Response Payload:

There is no response payload for this command.

AT command Ex:

at+rsibt_setlocalattvalue=A,1,EE\r\n

Response:

OK\r\n

6.4.4 Get Local Attribute Value

Description:

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

Payload Structure:

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

Parameters:

Hndl – handle of the attribute record.

AT Command format:

at+rsibt_getlocalattvalue=<Hndl>\r\n

Response Payload:

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

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

Parameters:

Hndl – handle of the attribute record.

DataLen – attribute record data length.

Data – attribute record data value.

AT command Ex:

at+rsibt_getlocalattvalue=A\r\n

Response:

OK A,1,EE\r\n

6.4.5 Remove Service

Description:

This command is used to remove service from the profile.

Binary Payload Structure:

```
typedef rsi_ble_remove_service{
BT_LE_PROFILEREC *ServiceHndlerPtr;
} RSI_BLE_REMOVE_SERVICE;
```

AT command format: at+rsibt_removeservice=<Servicehandlerptr>

Parameters:

ServiceHndlerPtr – service record handle.

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_removeservice=157A8

Response:

OK\r\n

6.4.6 Remove Attribute

Description:

This command is used to remove an attribute.

Binary Payload Structure:

```
typedef rsi_ble_remove_attribute{
BT_LE_PROFILEREC *ServiceHndlerPtr;
UINT16 AttHandle;
} RSI_BLE_REMOVE_ATTRIBUTE;
```

AT command format: at+rsibt_removeattribute=<Servicehandlerptr>,<AttHandle>

Parameters:

ServiceHndlerPtr – service record handle.

AttHandle-Handle of the attribute which has to be removed.

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_removeservice=157A8,10

Response:

OK\r\n

6.5 BLE Core Events

6.5.1 Advertise Report Event

Description:

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

Payload Structure:

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

AT Event format: AT+RSIBT_ADVRTISE, <<addr_type >, <bd_addr >,<RSSI>,<Type>,<adv_data_len>,< adv_data >>\r\n

Parameters:
addr_type: Type of address of advertiser

1. Public address
2. Random address

addr: Advertiser address
RSSI: Signal strength indication between devices
adv_data_len: total raw advertisement data length
adv_data: advertisement data

type: 0→ Connectable undirected advertising (ADV_IND).

1→ Connectable directed advertising (ADV_DIRECT_IND)

2 → Scannable undirected advertising (ADV_SCAN_IND)

3 → Non connectable undirected advertising (ADV_NONCONN_IND)

4 → Scan Response (SCAN_RSP)

0x05-0xFF Reserved for future use

AT event Ex:
AT+RSIBT_ADVRTISE,addr_type:0,addr:B4-99-4C-64-BC-AF,RSSI:-31,Type:3,adv_data_len:3,adv_data:2,1,6
In the above example
addr_type = 0
addr = B4-99-4C-64-BC-AF
RSSI = -31
adv_data_len = 3
adv_data = 2,1,6

type:3

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

6.5.2 LE Connected Event

Description:

This event indicates either the Connection is Success or not, when 9113 compatible features BIT[30] is set from ble_feature_bit_map in opermode.

Payload Structure:

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

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

Parameters:
BDAddresstype – Address type of the connected device
0 – Public address
1 – Random address
BDAddress – BD Address of the connected device
Status – 0 - Success
Non-zero – Failure

AT event Ex:

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

6.5.3 Disconnected

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

Payload Structure:

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

AT Event Format:

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

Parameters:
BDAddress – BD address of the remote BT device.

AT event Ex:

AT+RSIBT_LE_DISCONNECTED 62-CB-12-9D-CA-F2,4E13\r\n

6.5.4 SMP Request Event

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

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

Payload Structure:

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

Event format: AT+RSIBT_SMP_REQUEST <bd_addr>\r\n

Parameters:

BDAddress – Remote BD Address.

AT event Ex:

AT+RSIBT_SMP_REQUEST AA-BB-CC-DD-EE-FF\r\n

6.5.5 SMP Response Event

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

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

Payload Structure:

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

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

Parameters:
BDAddress – Remote BD Address.

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

6.5.6 SMP Passkey Event

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

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

Payload Structure:

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

Event format*:*

AT+RSIBT_SMPPASSKEY <bd_addr>\r\n

Parameters:
BDAddress – Remote BD Address.

AT event Ex:

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

6.5.7 SMP Failed Event

Description:

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

Payload Structure:

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

Event format*:* AT+RSIBT_SMP_FAILED ,<bd_addr>,<error>\r\n

Parameters:

BDAddress – Remote BD Address.

Error – reason for SMP failing.

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

6.5.8 SMP Encrypt Enabled Event

Description:

This event is raised when the encryption gets started. If some problem occurs in starting the encryption, an error code will be sent in this event.

Payload Structure:

No Payload

Event format:

AT+RSIBT_ENCRYPTION_ENABLED 88-DA-1A-9E-81-87, Local EDIV: 1FFF Local

RAND: 71,80,0,BF,0,0,C0,1 Local LTK: 63,63,6D,45,62,55,81,9D,A4,22,8E,69,CA,77,7F,B0\r\n or

AT+RSIBT_ENCRYPTION_DISABLED\r\n

6.5.9 LE ping payload timeout Currently LE ping is not supported

Description:

This event is raised when the LE ping timeout exceeds.

Payload Structure:

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

Event format*:*

AT+RSIBT_LE_PING_TIMEOUT <bd_addr>\r\n

Parameters:

BDAddress – Remote BD Address.

Event Ex:

AT+RSIBT_LE_PING_TIMEOUT 88-DA-1A-16-E6-1C\r\n

6.5.10 LE MTU Size

Description:

This event is raised after LE connection.

Payload Structure:

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

Parameters:

BDAddress – Remote BD Address.

MTU_size - Size of MTU

Event Ex:

AT+RSIBT_LE_REMOTE_MTU_SIZE 00-1A-7D-34-54-66,64

6.5.11 SMP Passkey Display Event

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

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

Action:

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

Payload Structure:

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

Event format*:*

```
AT+RSIBT_SMPPASSKEY_DISPLAY <bd_addr>,<Passkey>\r\n
```

Parameters:

BDAddress – Remote BD Address.

Passkey - Passkey

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

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

6.5.12 Phy Update Event

Description:

The LE PHY Update Complete Event is used to indicate that the Controller has changed the transmitter PHY or receiver PHY in use.

Payload Structure:

```
typedef struct rsi_ble_event_phy_update_comp {
UINT08 BDAddress[6];
UINT08 Status;
UINT08 TxPhy;
UINT08 RxPhy;
} RSI_BLE_EVENT_PHY_UPDATE_COMP;
Event format:
AT+RSIBT_LE_PHY_UPDATE_COMPLETE = <bd_addr>, <status><Tx_phy> Rx_phy>\r\n
```

Parameters:

BDAddress – BD Address of the connected device

Status

0 - success

1 - Failure

< tx_phy >-

0 - The Host prefers to use the LE 1M transmitter PHY (possibly among others)

1 - The Host prefers to use the LE 2M transmitter PHY (possibly among others)

2 - The Host prefers to use the LE Coded transmitter PHY (possibly among others)

3 – 7 Reserved for future use

<rx_phy> -

0 - The Host prefers to use the LE 1M receiver PHY (possibly among others)

1 - The Host prefers to use the LE 2M receiver PHY (possibly among others)

2 - The Host prefers to use the LE Coded receiver PHY (possibly among others)

3 – 7 Reserved for future use

AT event Ex:

AT+RSIBT_ LE_PHY_UPDATE_COMPLETE =B4-99-4C-64-BE-F5,0,2,1\r\n

6.5.13 BLE Data Length Change Event

Description:
The LE Data Length Change event notifies the Host of a change to either the maximum Payload length or the maximum transmission time of packets in either direction.

Payload Structure:

```
struct bt_le_length_update {
UINT16 MaxTxOctets;
UINT16 MaxTxTime;
UINT16 MaxRxOctets;
UINT16 MaxRxTime;
};
```

Event format*:*

AT+RSIBT_LE_DATA_LENGTH_CHANGE_EVENT
<BDAddress><MaxTxOctets>,<MaxTxTime>,<MaxRxOctets><MaxRxTime>\r\n

Parameters:

BDAddress – Remote BD Address.

MaxTXOctets– The maximum number of payload octets in a Link Layer packet that the local Controller will send on this connection.

MaxTXTime- The maximum time that the local Controller will take to send a Link Layer packet on this connection.

MaxRXOctets– The maximum number of payload octets in a Link Layer packet that the local Controller expects to receive on this connection

MaxRXTime- The maximum time that the local Controller expects to take to receive a Link Layer packet on this connection

Event Ex:

AT+RSIBT_LE_DATA_LENGTH_UPDATE 00-23-A7-12-23-24,40,330,50,320 \r\n

6.5.14 SMP Secure Connection Passkey Event

Description:
This event is raised when the SMP Secure Connection Passkey comes from the connected remote device.
Action:
Upon reception of this event **SMP Passkey** command has to be given.
Payload Structure:

```
typedef struct {
UINT08 BDAddress[6];
UINT32 Passkey;
} RSI_BLE_SC_PASSKEY;
```

Event format: AT+RSIBT_LE_SC_PASSKEY = <bd_addr>\r\n

Parameters:

BDAddress – Remote BD Address.

Passkey- Passkey to authenticate with the Remote device.

AT event Ex:

AT+RSIBT_LE_SC_PASSKEY <remote_addr>,<passkey>\r\n

6.5.15 LE Directed Advertising Report Event

Description:

The LE Directed Advertising Report event indicates that directed advertisements have been received.

Payload Structure:

```
struct bt_le_directed_adv_report {
UINT08 Address_type;
STR Address[18];
UINT08 Direct_address_type;
STR Direct_address[18];
UINT08 Rssi;
};
```

Event format:
AT+RSIBT_LE_DIRECTED_ADV_REPORT<AddressType><Address>,<DirectedAddressType>,<DirectedAddress><Rssi>\r\n

Parameters:

<AddressType>- Specifies the type of the address mentioned in BDAdress
0 – Public Address
1 – Random Address
2- Public Identity Address.

<BDAddress> – Remote BD Address.

<DirectedAddressType> - Random Device Address.

<DirectedAddress> - Random Device Address.

<Rssi>-Received signal strength value.

Event Ex:

AT+RSIBT_LE_DATA_LENGTH_CHANGE_EVENT 1,00-23-A7-12-23-24,1, 00-23-A7-12-23-24,21\r\n

6.5.16 Enhanced Connection Complete Event

Description: Enhanced Connection Complete Event indicates both of the hosts forming a connection that a new connection has been established.
Payload Structure:

```
struct bt_le_enhanced_conn_hndl{
UINT08 AddrType;
STR RemoteAddr[6];
STR LocalResolvableAddr[6];
STR PeerResolvableAddr[6];
};
```

Event format: AT+RSIBT_ LE_DEVICE_ENHANCE_CONNECTED= <addr_type>,<bd_addr>, <status>,<LocalResolvableAddr >,< PeerResolvableAddr >\r\n

Parameters:

BDAddresstype – Address type of the connected device
0 – Public address
1 – Random address

BDAddress – BD Address of the connected device

LocalResolvableAddr-Resolvable address of the local device.

PeerResolvableAddr-Resolvable address of the remote device.
Status – 0 - Success
Non-zero – Failure

AT event Ex:

AT+RSIBT_ LE_DEVICE_ENHANCE_CONNECTED=0,B4-99-4C-64-BE-F5, 74-99-4C-64-BB-F5, 48-99-4C-64-BE-F5,0\r\n

6.5.17 L2cap Credit Based Flow Control Connection Request Event

Description:

This event is used to notify the PSM connection request to host

Payload Structure:

```
typedef struct rsi_ble_event_psm_conn_req {

UINT08  BDAddress[6];

UINT16  PSM;

UINT16  Lcid;

} RSI_BLE_EVENT_PSM_CONN_REQ;
```

Event format:

AT+RSIBT_LEPSMCONNREQ<remote_addr>,<status>,<psm>,<MTU>, <MPS>,<lcid>\r\n

Parameters:

Remote-BDAddress – BD Address of the remote device

Status :

0 – Success

Non-zero – Failure

Psm – Protocol/Service Multiplexer (PSM) This field helps to indicate the protocol.

Lcid – local device channel identifier

AT event Ex: AT+RSIBT_LEPSMCONNREQ B4-99-4C-64-BE-F5,0,23,17,17,80 \r\n

6.5.18 L2cap Credit Based Flow Control Connection Complete Event

Description:

This event is used to notify the PSM connection request to host

Payload Structure:

```
typedef struct rsi_ble_event_psm_conn_complete {

    UINT08  BDAddress[6];

    UINT16  PSM;

    UINT16  MTU;

    UINT16  MPS;

    UINT16  Lcid;

} RSI_BLE_EVENT_PSM_CONN_COMPLETE;
```

Event format:

AT+RSIBT_LEPSMCONNCOMPLETE <remote_addr>,<status>,<psm>,<MTU>, <MPS>,<lcid>\r\n

Parameters:

Remote-BDAddress – BD Address of the remote device

Status:

0 – Success

Non-zero – Failure

Psm – Protocol/Service Multiplexer (PSM) This field helps to indicate the protocol.

MTU – Maximum transmission unit that device can transmit.

MPS – Maximum payload size , this is the maximum size of l2cap packet that can be transmited.

Lcid – local device channel identifier

AT event Ex:

AT+RSIBT_LEPSMCONNCOMPLETE B4-99-4C-64-BE-F5,0,23,17, 17,80\r\n

6.5.19 L2cap Credit Based Flow Control RX Data Event

Description:

This event is used to notify the remote device specific PSM data has been transferd to Host

Payload Structure:

```
typedef struct rsi_ble_event_psm_rx_data {

  UINT08  BDAddress[6];

        UINT16  Lcid;

        UINT16  Len;

        UINT08  Data[MAX_GATT_EVENT_DATA_LEN];

} RSI_BLE_EVENT_PSM_RX_DATA;
```

Event format:

AT+RSIBT_PSMDATARX <remote_bdaddr>,<lcid>, <data_length>,<data>\r\n

Parameters:

Remote-BDAddress – BD Address of the remote device

Lcid – local device channel identifier

Data length – length of data transferred

Data – Actual data that got transfered

AT event Ex:

AT+RSIBT_PSMDATARX B4-99-4C-64-BE-F5,80, 5,1,2,3,4,5\r\n

6.5.20 L2cap Credit Based Flow Control Disconnection Event

Description:

This functions is used to notify the disconnect event to host

Payload Structure:

```
typedef struct rsi_ble_event_psm_disconn {

    UINT08  BDAddress[6];

    UINT16  Lcid;

} RSI_BLE_EVENT_PSM_DISCONN;
```

Event format: AT+RSIBT_PSMDISCONNECTED <remote_bdaddr>,<lcid>\r\n

Parameters:

Remote-BDAddress – BD Address of the remote device

Lcid – local device channel identifier

AT event Ex: AT+RSIBT_PSMDISCONNECTED B4-99-4C-64-BE-F5,80\r\n

6.5.21 PSM Conn Failed Event

Description:

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

Payload Structure:

No payload

Event format: AT+RSIBT_LE_PSMCONNFAILED ,<error>\r\n

Parameters:

Error – reason for PSM conn failing.

Event Ex:AT+RSIBT_LE_PSMCONNFAILED ,4C02 \r\n

6.5.22 LE LTK Request Event

Description:

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

Payload Structure:

```
typedef struct rsi_bt_event_le_ltk_request {

    UINT08  BDAddress[6];

    UINT16  LocalEDIV;

    UINT08  LocalRand[8];

} RSI_BT_EVENT_LE_LTK_REQUEST;
```

Event format:

AT+RSIBT_LTK_REQUEST <bd_addr>, < LocalEDIV >, < LocalRand >\r\n

Parameters:

BDAddress – Remote BD Address.

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

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

6.5.23 LE Security Keys Event

Description:

This event is raised after encryption enabled with remote device.

Payload Structure:

```
typedef struct rsi_ble_event_security_keys {
    UINT08 BDAddress[6];
    UINT08 LocalIRK[16];
    UINT08 RemoteIRK[16];
    UINT16 RemoteEDIV;
    UINT08 RemoteRand[16];
    UINT08 RemoteLTK[16];
        UINT08 IdentityAddressType;
    UINT08 IdentityAddress[6];
 } RSI_BLE_EVENT_SECURITY_KEYS;
```

Event format:

AT+RSIBT_SECURITY_KEYS <BDAddress>, <LocalIRK>, <RemoteIRK>, <RemoteEDIV>, <RemoteRand>, <RemoteLTK>, <Remote Identity Address Type>,<Remote Identity Address>\r\n

Parameters:

BDAddress – Remote BD Address.

LocalIRK - IRK of local device

RemoteIRK - IRK of remote device

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

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

RemoteLTK - Remote device Long Term Key

IdentityAddressType - Address type of the remote device Identity Address

IdentityAddress - Identity Address of the remote device

6.5.24 Conn Update Event

Description:

The LE CONN Update Complete Event is used to indicate that the Controller has changed the connection parameter.

Payload Structure:

```
typedef struct rsi_ble_event_conn_update_comp {
  UINT08  BDAddress[6];
  UINT16  ConnInterval;
  UINT16  ConnLatency;
  UINT16  Timeout;
} RSI_BLE_EVENT_CONN_UPDATE_COMP;
Event format:
AT+RSIBT_LE_CONN_UPDATE_COMPLETE <bd_addr>,<status>,<Conn_Interval>,<Conn_Latency>,<Supervision_Timeout>
```

Parameters:

bd_addr - BD Address of the connected device

status (in hex)

0x00 Connection_Update command successfully completed.

0x01 – 0xFF Connection_Update command failed to complete.

Conn_Interval (in hex) - Connection interval used on this connection.

Range: 0x0006 to 0x0C80 (Time = N * 1.25 ms, Time Range: 7.5 ms to 4000 ms.)

Conn_Latency (in hex) – Slave latency for the connection in number of connection events.

Range: 0x0000 to 0x01F3

Supervision_Timeout (in hex)– Supervision timeout for the LE Link.

Range: 0x000A to 0x0C80 (Time = N * 10 ms, Time Range: 100 ms to 32000 ms)

AT event Ex:

AT+RSIBT_LE_CONN_UPDATE_COMPLETE 00-1A-7D-DA-71-13,0,6,0,C80\r\n

6.6 BLE GATT Events

6.6.1 GATT Notification

Description:

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

Payload Structure:

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

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

Parameters:

BDAddress – Remote BD Address.

Handle – Handle related to the Value

Length – Number of valid bytes in the Value

Value – The contents of the received Notification packet

Event Ex:

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

6.6.2 GATT Indication

Description:

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

Payload Structure:

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

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

Parameters:

BDAddress – Remote BD Address.

Handle – Handle related to the Value

Length – Number of valid bytes in the Value

Value – The contents of the received Indication packet

Event Ex:

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

6.6.3 GATT Write

Description:

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

Payload Structure:

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

Event format:

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

Parameters:

BDAddress – Remote BD Address.

Handle – Handle related to the Value

Length – Number of valid bytes in the Value

Value – The contents of the received Notification packet

Event Ex:

AT+RSIBT_WRITE,C0-FF-EE-C0-FF-EE,5,1,2\r\n

6.6.4 GATT READ

Description:

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

Payload Structure:

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

Parameters:

BDAddress – Remote BD Address.

Handle – Handle related to the Value

Type - To indicate read type
0 – read request
1 – read blob request

Reserved – Reserved for future use

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

Note:

In BTLE mode, need to enable BT mode.

7 Embedded BLE Error Codes

7.1 Generic Error Codes

Following Table represents possible coex modes supported:

Error Code	Description
0x0103	Timeout
0x4E01	Unknown HCI command
0x4E02	Unknown Connection Identifier
0x4E03	Hardware failure
0x4E04	Page timeout
0x4E05	Authentication failure
0x4E06	Pin missing
0x4E07	Memory capacity exceeded
0x4E08	Connection timeout
0x4E09	Connection limit exceeded
0x4E0A	SCO limit exceeded
0x4E0B	ACL Connection already exists
0x4E0C	Command disallowed
0x4E0D	Connection rejected due to limited resources
0x4E0E	Connection rejected due to security reasons
0x4E0F	Connection rejected for BD address
0x4E10	Connection accept timeout
0x4E11	Unsupported feature or parameter
0x4E12	Invalid HCI command parameter
0x4E13	Remote user terminated connection
0x4E14	Remote device terminated connection due to low resources
0x4E15	Remote device terminated connection due to power off
0x4E16	Local device terminated connection
0x4E17	Repeated attempts
0x4E18	Pairing not allowed
0x4E19	Unknown LMP PDU
0x4E1A	Unsupported remote feature
0x4E1B	SCO offset rejected

Error Code	Description
0x4E1C	SCO interval rejected
0x4E1D	SCO Air mode rejected
0x4E1E	Invalid LMP parameters
0x4E1F	Unspecified
0x4E20	Unsupported LMP Parameter
0x4E21	Role change not allowed
0x4E22	LMP response timeout
0x4E23	LMP transaction collision
0x4E24	LMP PDU not allowed
0x4E25	Encryption mode not acceptable
0x4E26	Link key cannot change
0x4E27	Requested QOS not supported
0x4E28	Instant passed
0x4E29	Pairing with unit key not supported
0x4E2A	Different transaction collision
0x4E2B	Reserved 1
0x4E2C	QOS parameter not acceptable
0x4E2D	QOS rejected
0x4E2E	Channel classification not supported
0x4E2F	Insufficient security
0x4E30	Parameter out of mandatory range
0x4E31	Reserved 2
0x4E32	Role switch pending
0x4E33	Reserved 3
0x4E34	Reserved slot violation
0x4E35	Role switch failed
0x4E36	Extended Inquiry Response too large
0x4E37	Extended SSP not supported
0X4E38	Host busy pairing
0X4E3C	Directed Advertising Timeout
0X4E60	Invalid Handle Range

Table 7-1: Bluetooth Generic Error Codes

7.2 Mode Error Codes

Error Code	Description
0x4A01	Invalid Handle
0x4A02	Read not permitted
0x4A03	Write not permitted
0x4A04	Invalid PDU
0x4A05	Insufficient authentication
0x4A06	Request not supported
0x4A07	Invalid offset
0x4A08	Insufficient authorization
0x4A09	Prepare queue full
0x4A0A	Attribute not found
0x4A0B	Attribute not Long
0x4A0C	Insufficient encryption key size
0x4A0D	Invalid attribute value length
0x4A0E	Unlikely error
0x4A0F	Insufficient encryption
0x4A10	Unsupported group type
0x4A11	Insufficient resources
0x4B01	SMP Passkey entry failed
0x4B02	SMP OOB not available
0x4B03	SMP Authentication Requirements
0x4B04	SMP confirm value failed
0x4B05	SMP Pairing not supported
0x4B06	SMP Encryption key size insufficient
0x4B07	SMP command not supported
0x4B08	SMP pairing failed
0x4B09	SMP repeated attempts
0x4B0C	SMP Failed
0x4C02	PSM Conn Failed
0x4D00	BLE Remote device found
0x4D01	BLE Remote device not found
0x4D02	BLE Remote device structure full

Error Code	Description
0x4D03	Unable to change state
0x4D04	BLE not Connected
0x4D05	BLE socket not available.
0x4D06	Attribute record not found
0x4D07	Attribute entry not found
0x4D08	Profile record full
0x4D09	Attribute record full
0x4D0A	BLE profile not found (profile handler invalid)
0x4E3E	Connection Failed to be Established

Table 7-2: BLE Error Codes

8 Embedded BLE Power Save Operation

Description:
This feature explains the configuration of **Power Save** modes of the module.These can be issued at any time after Opermode command. By default, Power Save is in disable state.
There are five different modes of Power Save. They are outlined below.

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

Note:
1. Power Save modes 2 and 8 are not supported in USB / USB-CDC interface. Instead, they are supported in UART / SPI interfaces.
2. In SPI interface, when Power Save mode is enabled, after wakeup from sleep, the host has to re-initialize SPI interface of the module.
3. When co-ex mode is enable, power save mode is not applicable for WLAN also.

8.1 Power Save Operations

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

8.1.1 Power Save Mode 0

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

8.1.2 Power Save Mode 2 (GPIO based mode)

For detail description for power save mode 2, please refer to the section **8.16.1.2** in **RS9116-WiseConnect-Software-PRM-v1.X.X.pdf**.

8.1.3 Power Save Mode 3 (Message based mode)

For detail description for power save mode 3, please refer to the section **8.16.1.3** in **RS9116-WiseConnect-Software-PRM-v1.X.X.pdf**.
In BLE, Power Save mode 2 and 3 can be used during Advertise /Scan /Connected states. Operational behavior is as below depending on the state.

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

For detail description for power save mode 8, please refer to the section **8.16.1.4** in **RS9116-WiseConnect-Software-PRM-v1.X.X.pdf**.

For detail description for power save mode 9, please refer to the section **8.16.1.4** in **RS9116-WiseConnect-Software-PRM-v1.X.X.pdf**.

Note:

Power save disable command has to be given before changing the state from standby to the remaining states and wise-versa. Suppose if Power Save is enabled in standby state, so, in order to move to Scanning state, first Power Save disable command need to be issued before giving Scan command.

When the module is configured in a co-ex mode and WLAN is in INIT_DONE stare, powersave mode 2 & 3 are valid after association in the WLAN. Where as in BT & BLE alone modes, it will enter into power save mode (2 & 3) in all states (except in standby state).

9 Embedded BLE API Library

9.1 API File Organization

Bluetooth APIs are organized into following directory structure

	Path(Within *RS9116.WC.GENR.x.x.x* folder)
BLE APIs	host/apis/ble/core/
Interface Specific APIs	host/apis/intf/
HAL APIs	host/apis/hal/
BLE Reference Applications	host/apis/ble/ref_apps/
BLE Linux Application	RS9116.WC.GENR.xxx/host/reference_projects/LINUX/Application/ble/src
Linux USB Driver	host/reference_projects/LINUX/Driver/usb/src

9.2 BLE API Generic Prototypes

9.2.1 Set Local name

```
INT16 rsi_bt_set_local_name(RSI_BT_CMD_SET_LOCAL_NAME *SetLocalName);
```

9.2.2 Query Local name

```
INT16 rsi_bt_query_local_name(void);
```

9.2.3 Query RSSI

```
INT16 rsi_bt_query_rssi(RSI_BT_CMD_QUERY_RSSI *GetRSSI);
```

9.2.4 Query Local BD Address

```
INT16 rsi_bt_query_local_bd_address(void);
```

9.3 BLE Core Prototypes

9.3.1 Advertise Local Device

```
INT16 rsi_ble_advertise(RSI_BLE_CMD_ADVERTISE *LEAdv);
Scan
```

```
INT16 rsi_ble_scan(RSI_BLE_CMD_SCAN *LEScan);
```

9.3.2 Connect

```
INT16 rsi_ble_connect(RSI_BLE_CMD_CONNECT *LEConn);
```

9.3.3 Disconnect

```
INT16 rsi_ble_disconnect(RSI_BLE_CMD_DISCONNECT *uLEDisconnect);
```

9.3.4 Query Device STAte

```
INT16 rsi_ble_query_device_state(void);
```

9.3.5 STArt Encryption

```
INT16 rsi_ble_start_encryption(RSI_BLE_CMD_ENCRYPTTION *uLEEncrypt);
```

9.3.6 SMP Pair Request

```
INT16 rsi_ble_smp_pair_request(RSI_BLE_CMD_SMP_PAIR_REQUEST *SMPPairReq);
```

9.3.7 SMP Response

```
INT16 rsi_ble_smp_response(RSI_BLE_CMD_SMP_RESPONSE *SMPResp);
```

9.3.8 SMP Passkey

```
INT16 rsi_ble_smp_passkey(RSI_BLE_CMD_SMP_PASSKEY *SMPPasskey);
```

9.3.9 BLE Init

```
INT16 rsi_ble_device_init();
```

9.3.10 BLE Deinit

```
INT16 rsi_ble_device_deinit();
```

9.3.11 BT Antenna Select

```
INT16 rsi_bt_antenna_select(RSI_BT_CMD_ANTENNA_SELECT *uAntennaSelect);
```

9.3.12 Set Advertise Data Payload

```
INT16 rsi_ble_device_SeTAddvertiseData(RSI_BLE_CMD_SET_ADVERTISE_DATA *uLEAdvertiseData);
```

9.3.13 Set LE Ping Timeout Currently LE ping is not supported

```
INT16 rsi_ble_device_SetLePingTimeout(RSI_BLE_CMD_SET_PING_TIMEOUT * uSetLePingTimeout);
```

9.3.14 Get LE Ping Timeout Currently LE ping is not supported

```
INT16 rsi_ble_device_GetLePingTimeout(void);
```

9.3.15 Set Random Address

```
INT16 rsi_ble_set_random_address(RSI_BLE_CMD_SET_RANDOM_ADDRESS *uSetRandAdd);
```

9.3.16 LE Data Encrypt

```
INT16 rsi_ble_data_encrypt(RSI_BLE_CMD_DATA_ENCRYPT *uDataEncrypt);
```

9.4 BLE GATT Prototypes

9.4.1 Query profiles list

```
INT16 rsi_ble_query_profiles_list(RSI_BLE_CMD_PROFILE_LIST *uGetProfileList);
```

9.4.2 Query Profile

```
INT16 rsi_ble_query_profile(RSI_BLE_CMD_QUERY_PROFILE *GetProf);
```

9.4.3 Query Characteristic Services

```
INT16 rsi_ble_query_characteristic_services(RSI_BLE_CMD_QUERY_CHARACTERISTIC_SERVICES *GetCharServ);
```

9.4.4 Query Include Services

```
INT16 rsi_ble_query_include_service(RSI_BLE_CMD_QUERY_INCLUDE_SERVICES *GetIncludeServ);
```

9.4.5 Read Characteristic Value by UUID

```
INT16 rsi_ble_read_char_value_by_UUID(RSI_BLE_CMD_READ_CHAR_VALUE_BY_UUID *ReadCharValByUUID);
```

9.4.6 Query Attribute

```
INT16 rsi_ble_query_att(RSI_BLE_CMD_QUERY_ATT_DESC *GeTAtt);
```

9.4.7 Query Attribute Value

```
INT16 rsi_ble_query_att_value(RSI_BLE_CMD_QUERY_ATT_VALUE *GetattVal);
```

9.4.8 Query Multiple Attribute Values

```
INT16 rsi_ble_query_multi_att_values(RSI_BLE_CMD_QUERY_MULTIPLE_ATT_VALUES *GetMulAttVal);
```

9.4.9 Query Long Attribute Value

```
INT16 rsi_ble_query_long_att_value(RSI_BLE_CMD_QUERY_LONG_ATT_VALUE *GetLongAttVal);
```

9.4.10 Set Attribute Value

```
INT16 rsi_ble_set_att_value(RSI_BLE_CMD_SET_ATT_VALUE *SetattVal);
```

9.4.11 Set Attribute Value no Ack

```
INT16 rsi_ble_set_att_value_no_ack(RSI_BLE_CMD_SET_ATT_VALUE_NO_ACK *SeTAttValNoAck);
```

9.4.12 Set Long Attribute Value

```
INT16 rsi_ble_set_long_att_value(RSI_BLE_CMD_SET_LONG_ATT_VALUE *SetLongAttVal);
```

9.4.13 Set Prepare Long Attribute Value

```
INT16 rsi_ble_set_prep_long_att_value(RSI_BLE_CMD_SET_PREPARE_LONG_ATT_VALUE * uLePrepareAttVal);
```

9.4.14 Execute Long Attribute Value

```
INT16 rsi_ble_execute_long_att_value(RSI_BLE_CMD_EXECUTE_LONG_ATT_VALUE_WRITE *ExeLongAttVal);
```

9.4.15 Add GATT Service Record

```
INT16 rsi_ble_device_AddService(RSI_BLE_CMD_ADD_GATT_SERVICE *uLEAddService);
```

9.4.16 Add Attribute Record

```
INT16 rsi_ble_device_AddServiceAttribute(RSI_BLE_CMD_ADD_GATT_ATTRIBUTE *uLEAddAttribute);
```

9.4.17 Set Local Attribute Record value

```
INT16 rsi_ble_device_SetLocalAttValue(RSI_BLE_CMD_SET_LOCAL_ATT_VALUE *uLESetLocalAttVal);
```

9.4.18 Get Local Attribute Record value

```
INT16 rsi_ble_device_GetLocalAttValue(RSI_BLE_CMD_GET_LOCAL_ATT_VALUE *uLEGetLocalAttVal);
```

9.4.19 BLE Whitelist

```
INT16 rsi_ble_white_list(RSI_BLE_CMD_WHITE_LIST *uBleWhiteListl);
```

9.4.20 BLE Notify value

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

9.5 Application

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

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

To facilitate Application development we have defined a data structure named RSI_BLE_API as described below.This structure is initialized by the application using rsi_ble_init_struct API of the rsi_ble_config_init.c file (application layer file). The user may change the values assigned to the macros without worrying about understanding the contents of the structure.

The contents of this structure are explained in brief below, using the declaration of the structure in rsi_ble_global.h file (which is also an application layer file and is placed in core APIs include folder).

```
Typedef union{
RSI_BT_CMD_SET_LOCAL_NAME uSetLocalName;
RSI_BT_CMD_SET_LOCAL_COD uSetLocalCOD;
RSI_BT_CMD_QUERY_RSSI uQryRssi;
RSI_BT_CMD_QUERY_LINK_QUALITY uQryLinkQuality;
RSI_BT_CMD_ANTENNA_SELECT uAntennaSelect;
RSI_BLE_CMD_ADVERTISE uLeAdvertise;
RSI_BLE_CMD_SCAN uLeScan;
RSI_BLE_CMD_CONNECT uLeConnect;
RSI_BLE_CMD_DISCONNECT uLeDisConnect;
RSI_BLE_CMD_ENCRYPTION uLeSmpEncrypt;
RSI_BLE_CMD_SMP_PAIR_REQUEST uLeSmpReq;
RSI_BLE_CMD_SMP_RESPONSE uLeSmpResp;
RSI_BLE_CMD_SMP_PASSKEY uLeSmppasskey;
RSI_BLE_CMD_SET_PING_TIMEOUT uLeSetPingTimeout;
RSI_BLE_CMD_SET_ADVERTISE_DATA uLeSetadvertiseDaTA;
RSI_BLE_CMD_QUERY_PROFILE uLeSev;
RSI_BLE_CMD_PROFILE_LIST uLeAllServ;
RSI_BLE_CMD_QUERY_CHARACTERISTIC_SERVICES uLeCharServ;
RSI_BLE_CMD_QUERY_INCLUDE_SERVICES uLeIncServ;
RSI_BLE_CMD_READ_CHAR_VALUE_BY_UUID uLeCharVal;
RSI_BLE_CMD_QUERY_ATT_DESC uLeAttDesc;
RSI_BLE_CMD_QUERY_ATT_VALUE uLeAttVal;
RSI_BLE_CMD_QUERY_MULTIPLE_ATT_VALUES uLeMulAttVals;
RSI_BLE_CMD_QUERY_LONG_ATT_VALUE uLeLongAttVal;
RSI_BLE_CMD_SET_ATT_VALUE uLeSetattVal;
RSI_BLE_CMD_SET_ATT_VALUE_NO_ACK uLeSetCmdAttVal;
RSI_BLE_CMD_SET_LONG_ATT_VALUE uLeSetLongAttVal;
RSI_BLE_CMD_SET_PREPARE_LONG_ATT_VALUE uLePrepareAttVal;
RSI_BLE_CMD_EXECUTE_LONG_ATT_VALUE_WRITE uLeExecuteWrite;
RSI_BLE_CMD_ADD_GATT_SERVICE uLeAddService;
RSI_BLE_CMD_ADD_GATT_ATTRIBUTE uLeAddAttribute;
RSI_BLE_CMD_SET_LOCAL_ATT_VALUE uLeSetLocalAttVal;
RSI_BLE_CMD_GET_LOCAL_ATT_VALUE uLeGetLocalAttVal;
RSI_BLE_CMD_SET_RANDOM_ADDRESS uLeRandAdd;
}RSI_BLE_API;
```

10 Appendix A : Sample Flows

10.1 Configure BLE device in Central Mode

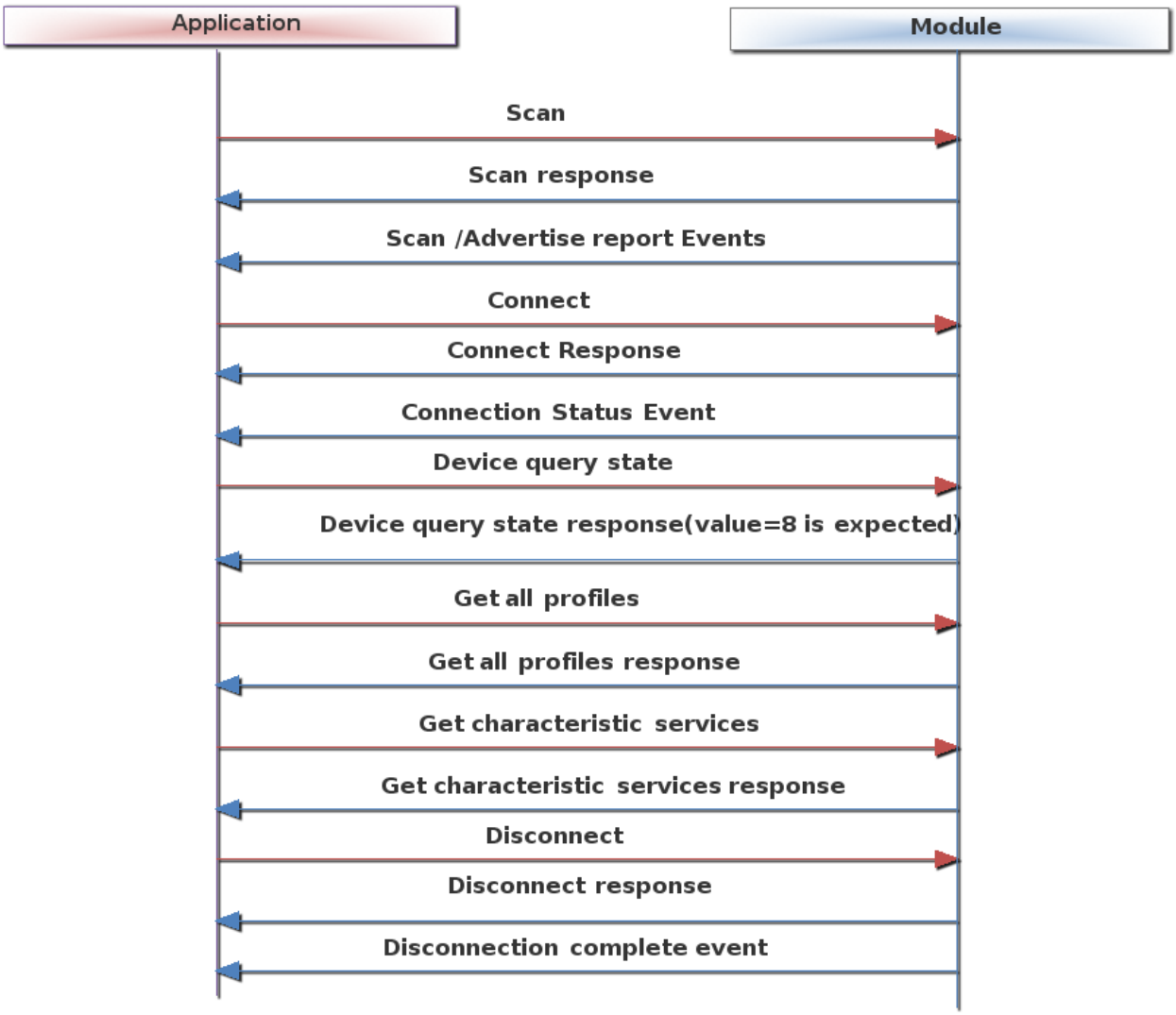


Figure 46 : Sample command sequence of BLE Central Mode

10.2 Configure BLE device in Peripheral Mode

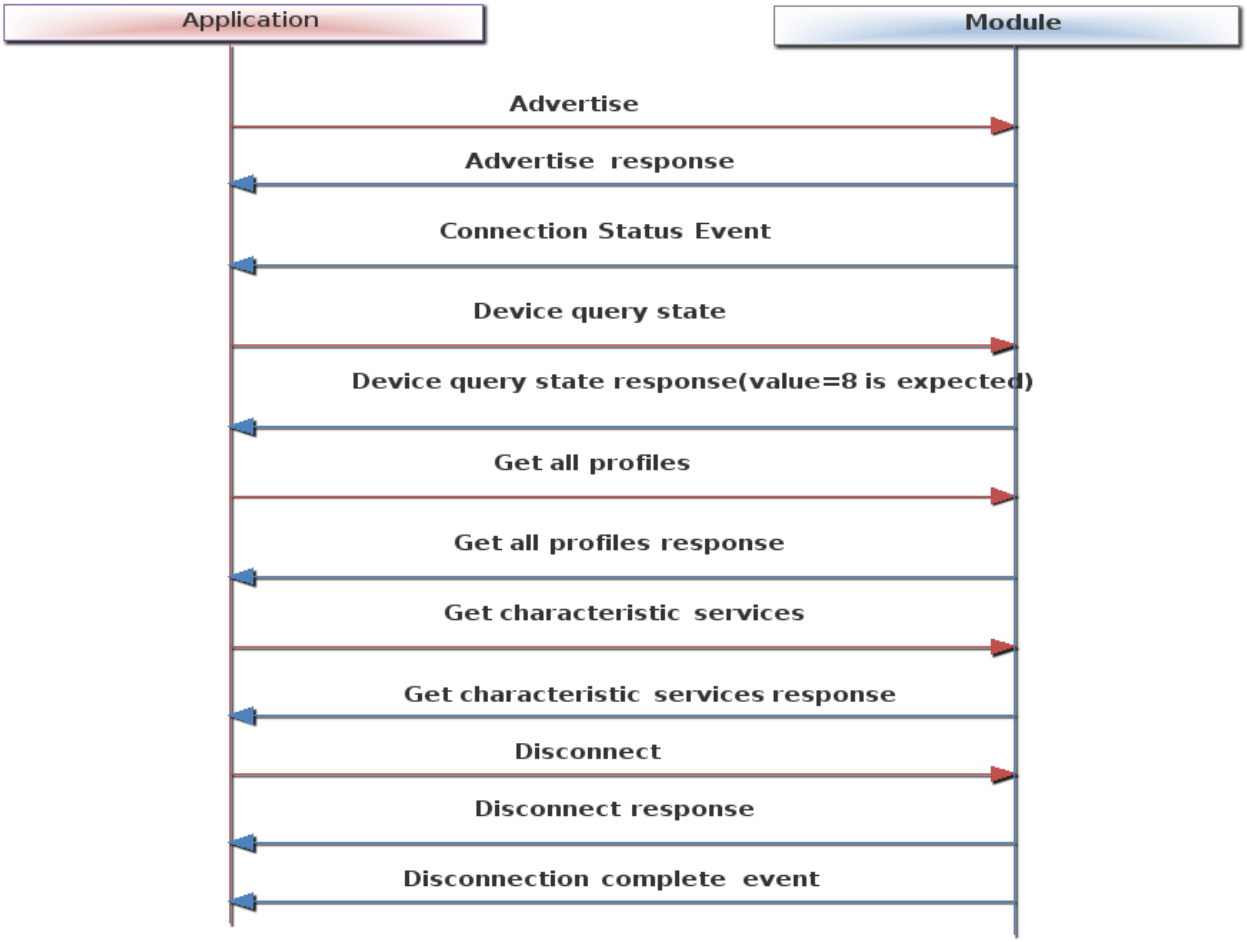


Figure 47 : Sample command sequence of BLE Peripheral Mode

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

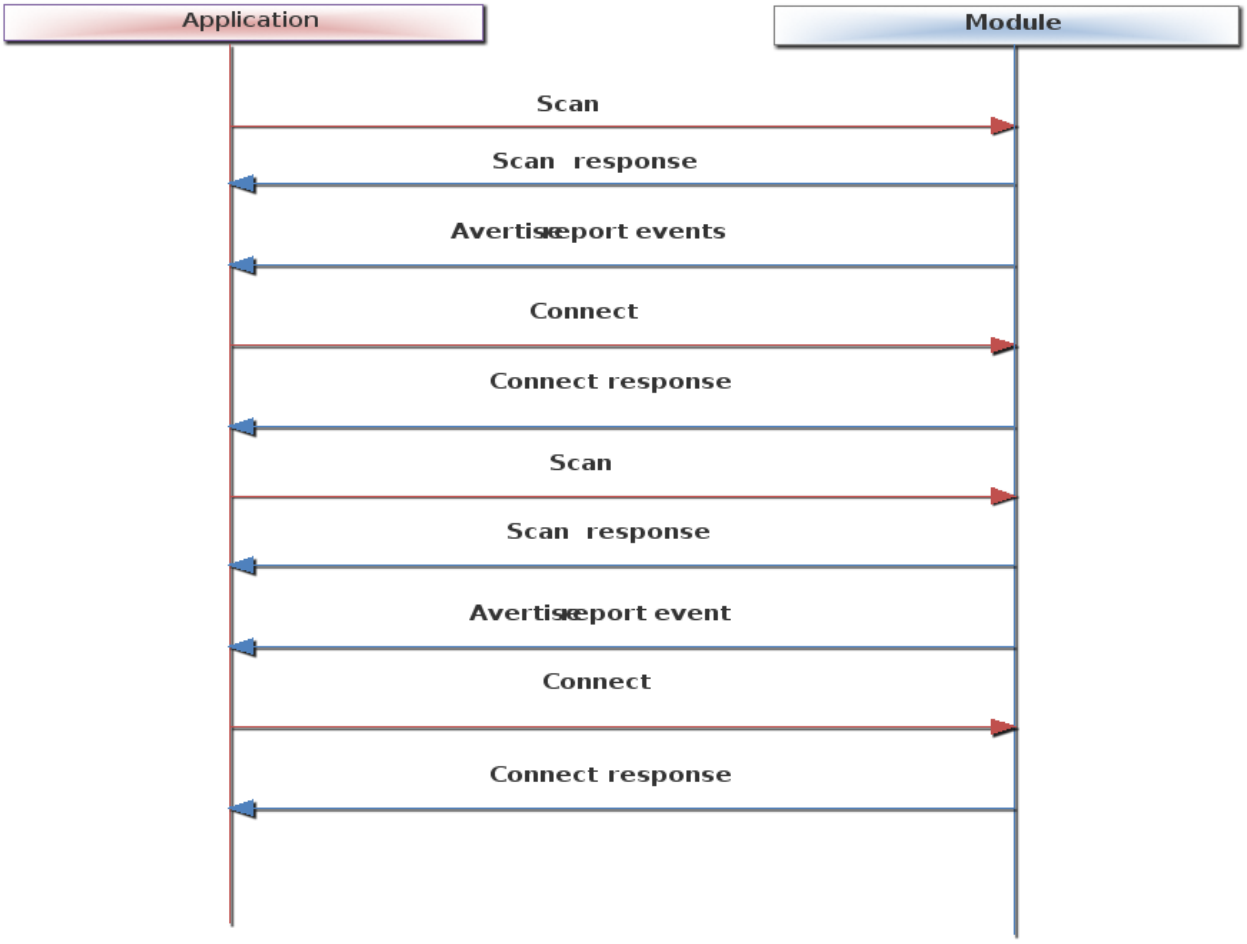


Figure 48 : Sample command sequence of BLE multiple slaves

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

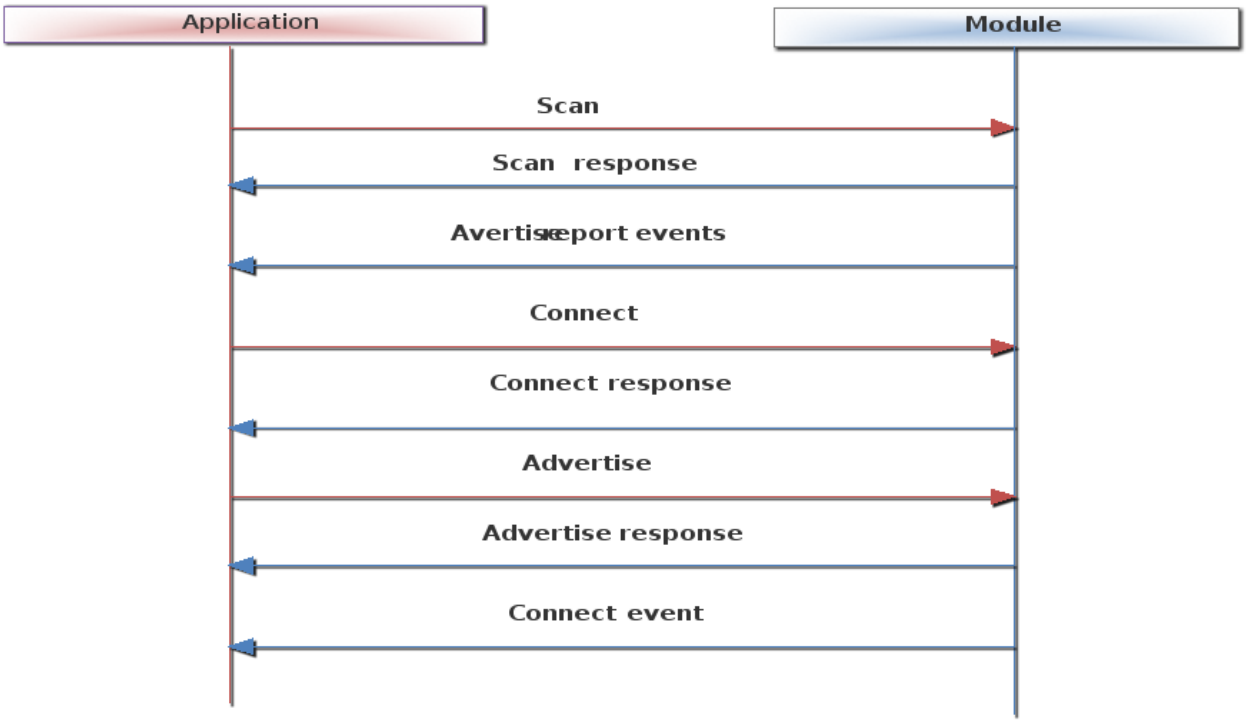
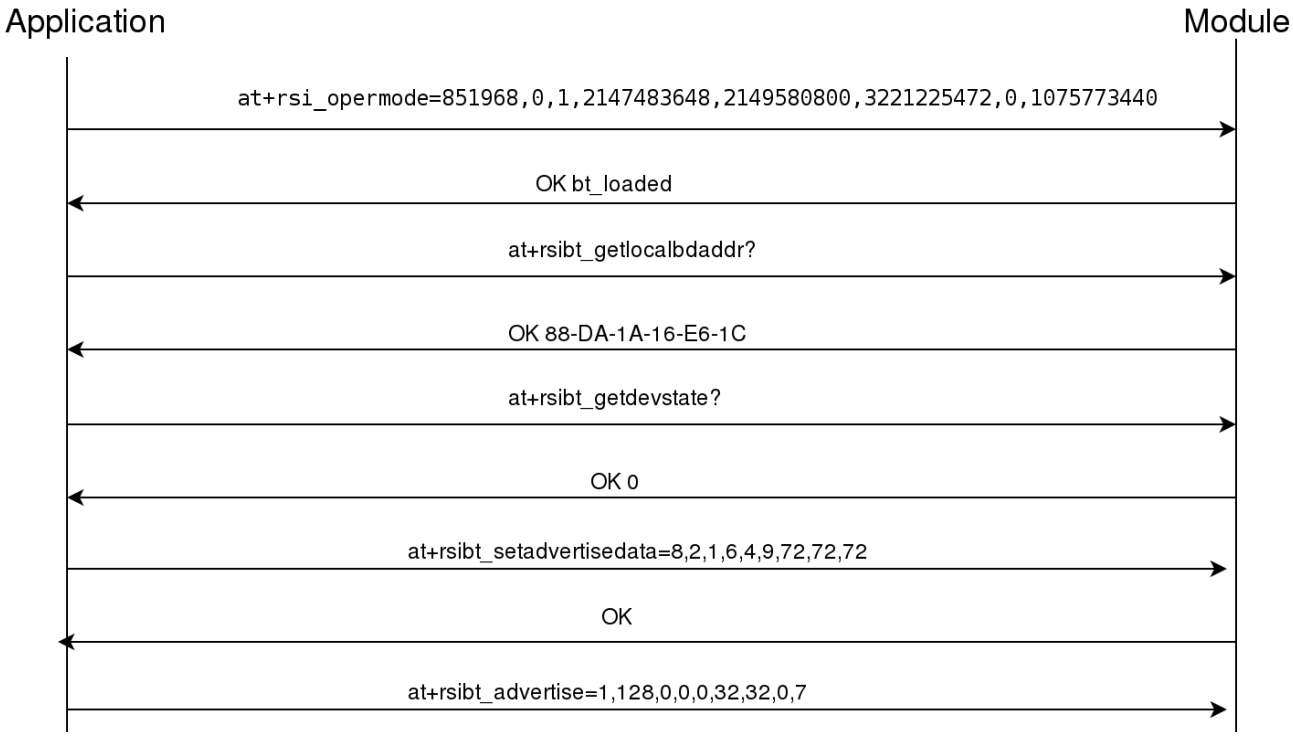


Figure 49 : Sample command sequence of BLE dual role

10.5 AT command flow to configure BLE device in Peripheral Mode



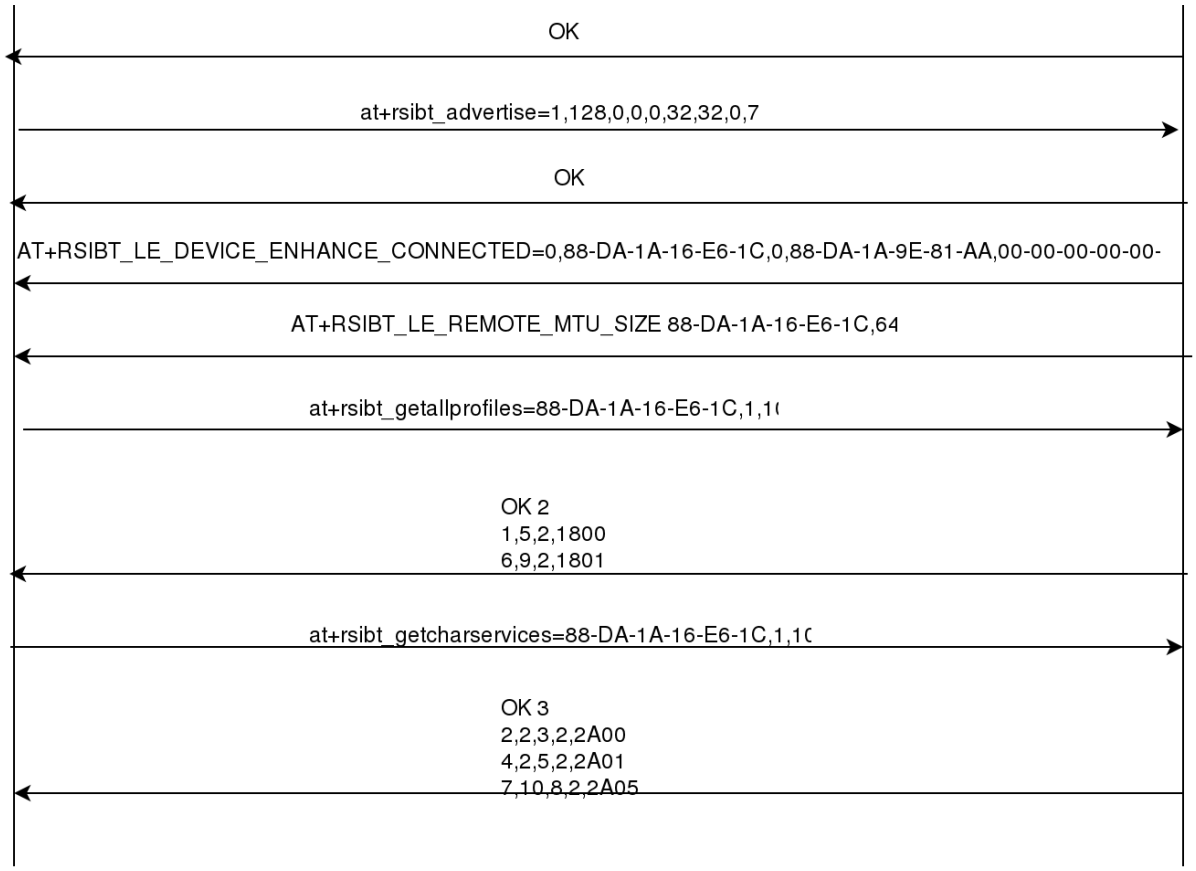
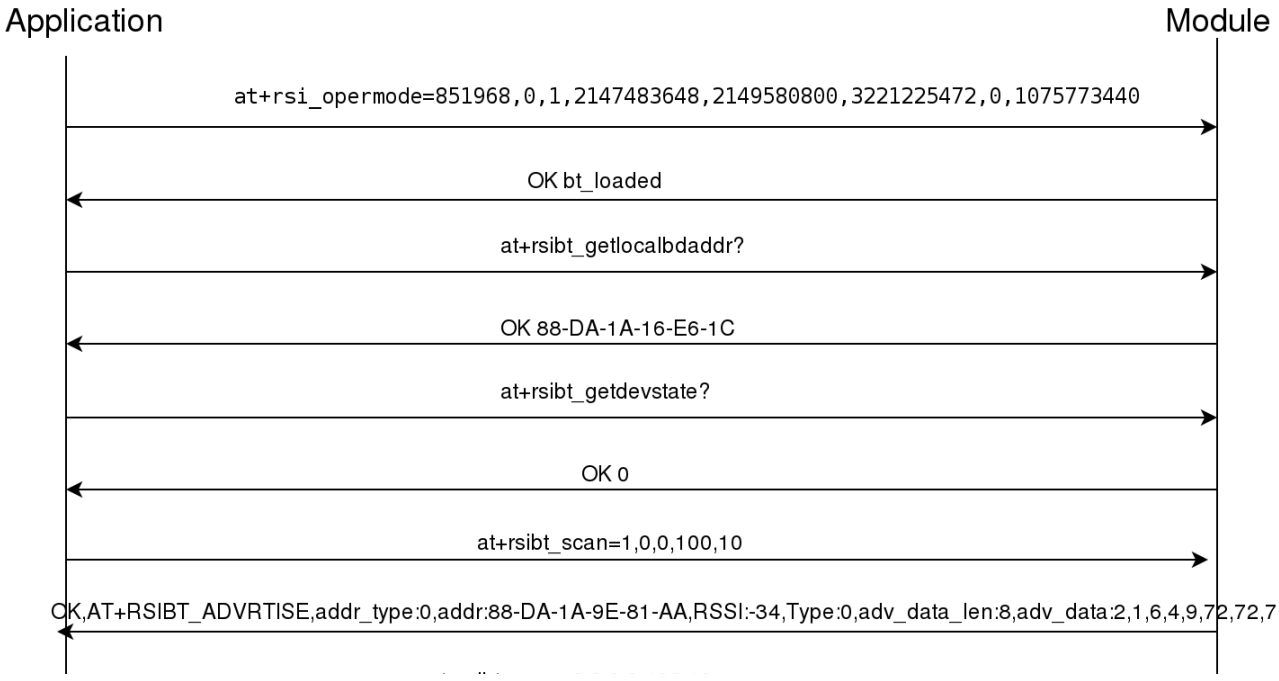


Figure 50 : Sample AT command sequence of BLE Peripheral Mode

10.6 AT command flow to configure BLE device in Central Mode



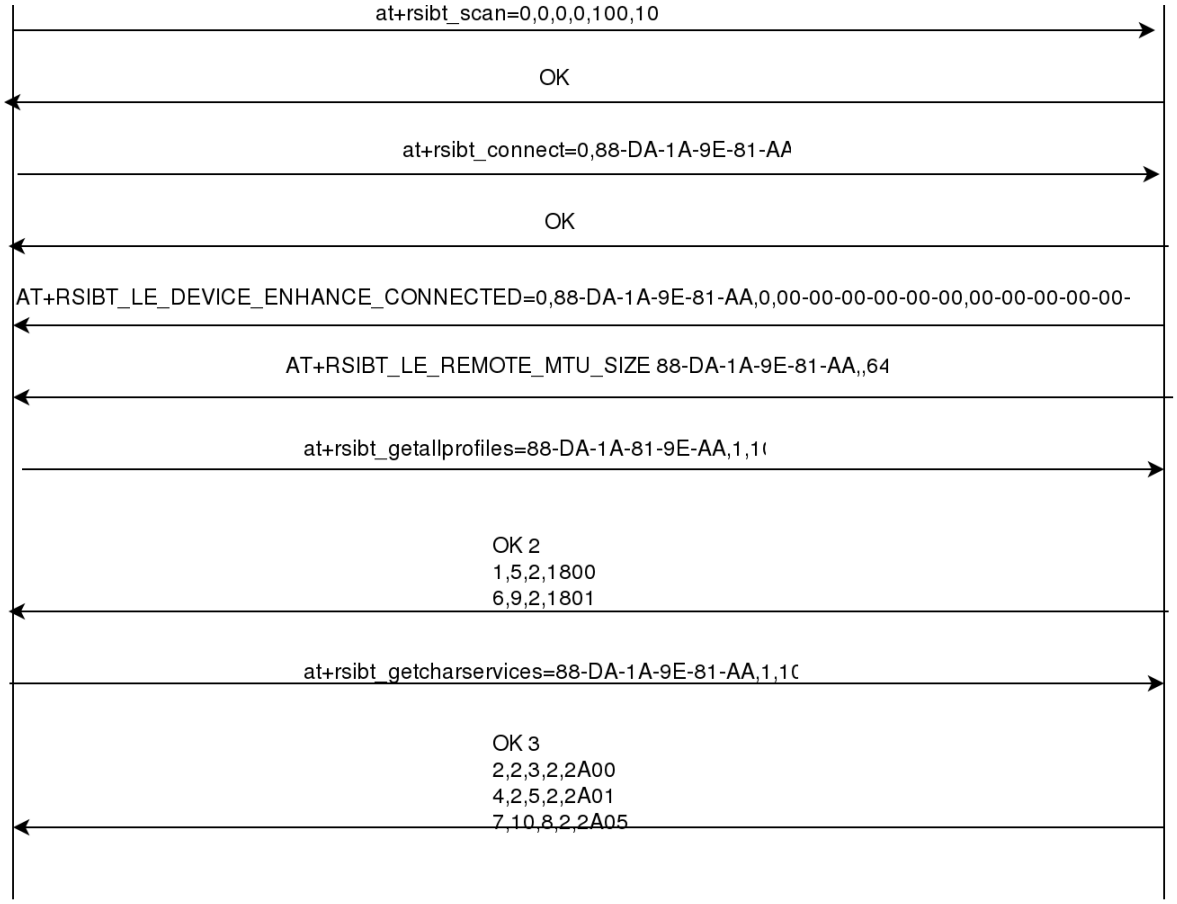


Figure 51 : Sample AT command sequence of BLE Central Mode

10.7 Security Management Protocol (SMP) in Slave mode

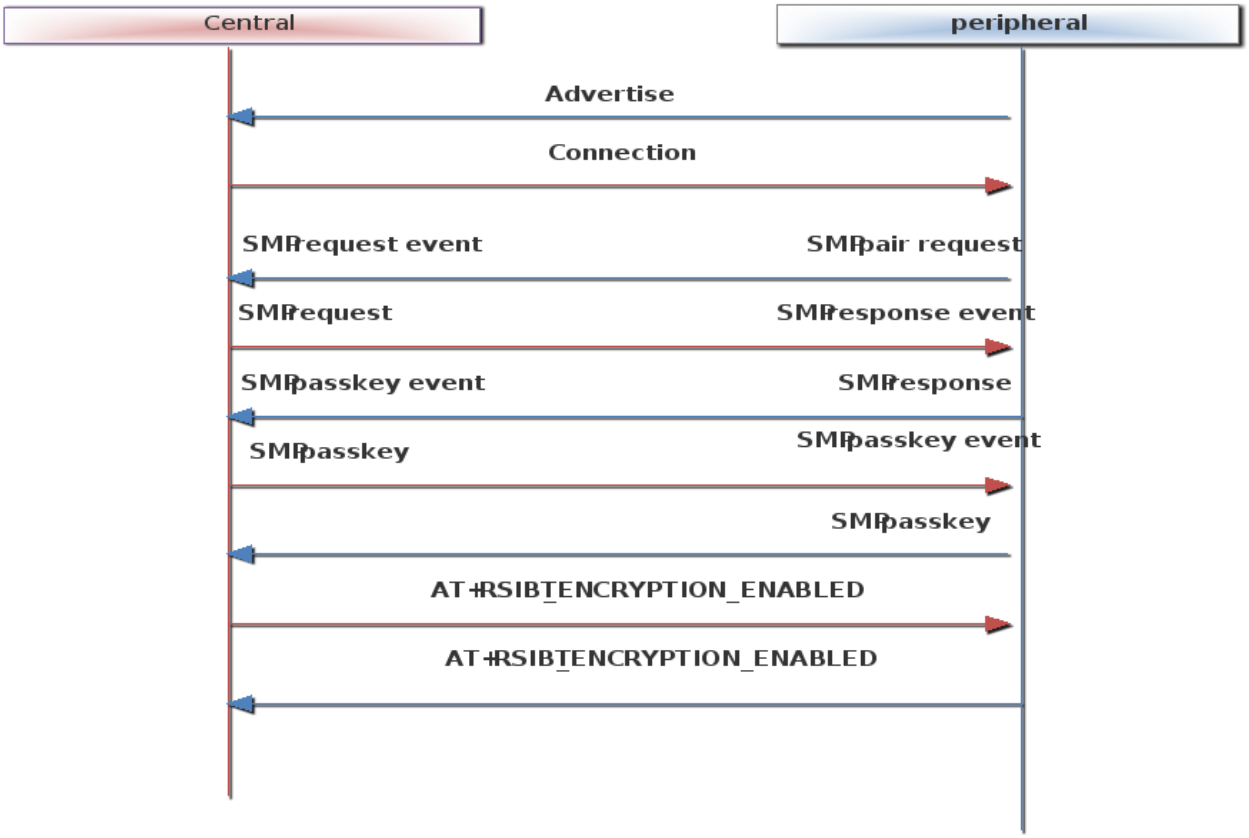


Figure 52 : SMP command sequence of BLE Slave (peripheral) Mode

10.8 Security Management Protocol(SMP) in Master mode

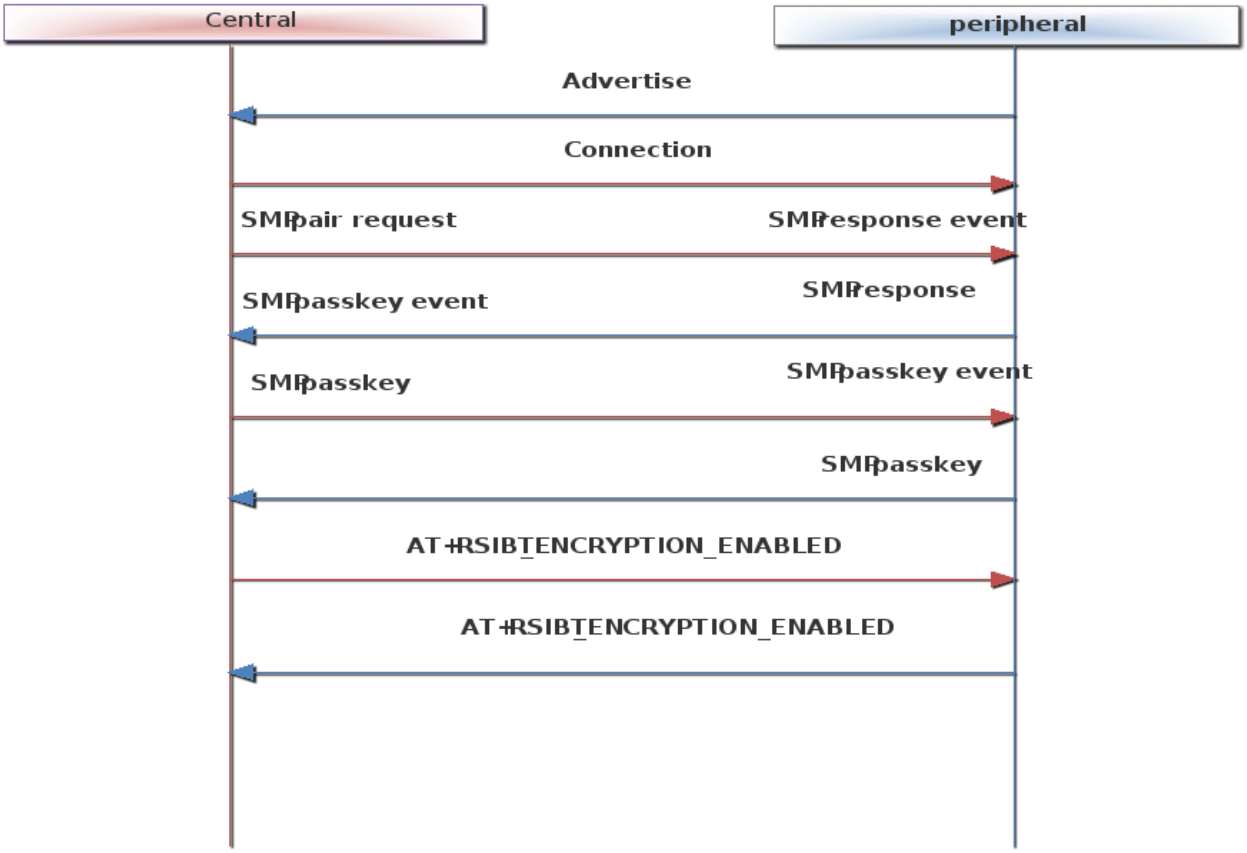


Figure 53 : SMP command sequence of BLE Master (Central) Mode

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

In order to run the Wi-Fi client and BT-LE coexistence mode, user has to issue the operating mode as first command with coex parameters.
After operating mode command module will operate in both Wi-Fi STA mode and BT LE mode. So, user can issue Wi-Fi commands as well as BT LE commands in parallel on host interface.

Common Command:

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

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

Wifi_oper_mode = 0 (to operate wifi in STA mode)

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

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

Tcp_ip_feature_bit_map = 1 (TCP/IP Bypass mode)

Custom_feature_bit_map=(1<<31)

ext_custom_feature_bit_map=(1<<31)

bt_custom_feature_bit_map

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

bt_custom_feature_bit_map i.e, 6th parameter.

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

Wi-Fi Command Sequence to Associate with Access Point:

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

Please refer RS9116-Wiseconnect-Software-PRM-vx.x.x.pdf for Wi-Fi commands description.

BT LE Command Sequence:

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

After successful Wi-Fi and BT LE connection user can send Wi-Fi raw data packets into air and also can issue GATT commands.

Wi-Fi+BT LE coex Rx flow:

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

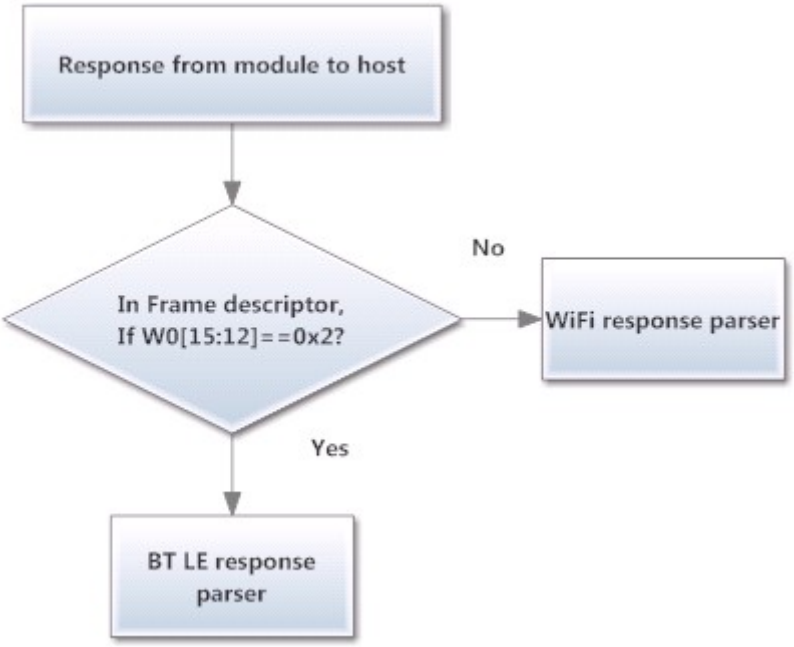


Figure 54 : Sample flow for WiFi + BT LE response

11 Revision History

Revision No.	Version No.	Date	Changes
1	v1.0	November 2017	Advance version
2	v1.1	Feb 2018	Formatting changes
3	v1.2	April 2018	Generalized for WiSeConnect and WiSeMCU
4	v1.3	June 2018	Replaced the old commands with updated ones.
5	v1.4	July 2018	1. BLE few commands modified 2. BLE Appendix changes
6	v1.5	August 2018	Added ext_custom_feat_bitmap field description and modified feature bitmap description in opermode command