REYAX
TECHNOLOGY CORPORATION, LTD

04-JUL-2019 56312E30

RYWB116

Embedded BT Classic Software Programming Reference Manual (PRM)

Disclaimer

The information in this document pertains to information related to REYAX TECHNOLOGY CO., LTD. products. This information is provided as a service to our customers, and may be used for information purposes only.

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

Table of Contents

1 Embedded BT Classic Overview 10
1.1 Architecture 10
1.1.1 Bluetooth Classic Architecture 11
1.2 Application 11
1.3 Profiles 11
1.4 Bluetooth Core 11
1.5 OS Abstraction Layer 12
1.5.1 Host-Module Interface 12
2 Embedded Bootloader and Interfaces 13
2.1 Bootloader 13
2.1.1 Features 13
2.2 Host Interaction Mode 13
2.2.1 Host Interaction Mode in UART / USB-CDC 13
2.2.2 Loading selected Wireless Firmware in the Module 19
2.3 Host Interaction Mode in SPI / USB 24
2.3.1 SPI Startup Operations 24
2.3.2 SPI Startup Messages on Powerup 26
2.4 Loading Default Wireless Firmware in the Module 26
2.4.1 Load Default Wireless Firmware 26
2.5 Loading Selected Wireless Firmware in the Module 27
2.5.1 Load Selected Wireless Firmware 27
2.6 Upgrading Wireless Firmware in the Module 27
2.6.1 Upgrading Wireless Firmware 27
2.7 GPIO Based Bootloader Bypass Mode for RYWB116 28
2.8 Bypass Mode in UART / USB-CDC 28
2.8.1 Making Default Wireless Firmware Selection 28
2.8.2 Enable/Disable GPIO Based Bypass Option 29
2.8.3 Check Integrity of the Selected Image 30
2.9 Bypass Mode in SPI / USB 31
2.9.1 Selecting the Default Image 31
2.9.2 Enable / Disable GPIO Based Bootloader Bypass Mode 31
2.10 Other Operations 32
2.11 Update KEY 32
2.12 JTAG Selection 32
2.13 SPI Interface 32
2.14 Features 32
2.15 Hardware Interface 32
2.15.1 SPI Signals 32
2.15.2 Interrupt 33
2.15.3 SPI Interface Initialization 33
2.15.4 Module SPI Interface Initialization 36
2.15.5 Register Summary 45
2.16 Software Protocol 46
2.17 Commands 50
2.18 Features 50
2.19 Hardware Interface 51
2.20 Software Protocol 51
2.20.1 AT+ command mode 51
2.20.2 Binary command mode 51
2.21 Commands 52
2.22 USB Interface 52
2.23 Features 53

2.24 Hardware Interface 53
2.25 Software Protocol 53
2.25.1 USB Mode 53
2.25.2 USB CDC-ACM Mode 54
2.26 Commands 54
3 Embedded BT Classic Command Mode Selection 55
4 Embedded BT Classic Command Format 56
5 Embedded BT Classic Commands 69
5.1 Generic commands 69
5.1.1 Set Operating Mode 69
5.1.2 Set Local name 72
5.1.3 Query Local name 73
5.1.4 Set Local COD 74
5.1.5 Query Local COD 74
5.1.6 Query RSSI 75
5.1.7 Query Link Quality This command is not currently supported. 76
5.1.8 Query Local BD Address 77
5.1.9 Initialize BT module 77
5.1.10 Deinitialize BT module 78
5.1.11 BT Antenna Select 78
5.1.12 Set Feature Bitmap 79
5.1.13 Set Antenna Tx power level 79
5.2 Core commands 80
5.2.1 Set Profile Mode Present only SPP profile is supported. 80
5.2.2 Set Device Discovery mode 81
5.2.3 Get Device Discovery mode 82
5.2.4 Set Connectability mode 83
5.2.5 Get Connectablility mode 83
5.2.6 Set Pair mode 84
5.2.7 Get Pair mode 84
5.2.8 Remote Name Request 85
5.2.9 Remote Name Request Cancel 86
5.2.10 Inquiry 86
5.2.11 Inquiry Cancel 87
5.2.12 Extended Inquiry Response Data 87
5.2.13 Bond or Create Connection 89
5.2.14 Bond Cancel or Create Connection Cancel 89
5.2.15 UnBond or Disconnect 90
5.2.16 Set Pin type This command is not currently supported. 90
5.2.17 Get Pin type This command is not currently supported. 91
5.2.18 User confirmation 92
5.2.19 Pass key Request Reply 92
5.2.20 Pincode Request Reply 93
5.2.21 Get Local Device Role 94
5.2.22 Set Local Device Role or switch the role 95
5.2.23 Get Service List 95
5.2.24 Search Service 96
5.2.25 Linkkey Reply 97
5.2.26 Set SSP mode 98
5.2.27 Sniff Mode 98
5.2.28 Sniff Exit 99
5.2.29 Sniff Subrating Currently, the sniff subrating command is not supported. 100
5.3 SPP commands 100
5.3.1 SPP Connect 101
5.3.2 SPP Disconnect 101
5.3.3 SPP Transfer 102

5.4 A2DP commands 102
5.4.1 A2DP Connect 102
5.4.2 A2DP Disconnect 103
5.5 AVRCP Commands 104
5.5.1 AVRCP Connect 104
5.5.2 AVRCP Disconnect 104
5.5.3 AVRCP Play 105
5.5.4 AVRCP Pause 105
5.5.5 AVRCP stop 106
5.5.6 AVRCP next 107
5.5.7 AVRCP previous 107
5.6 Power Save 108
5.6.1 Power save Operations 108
5.7 IAP commands 109
5.7.1 IAP connect 110
5.7.2 IAP Disconnect 110
5.7.3 IAP Set Accessory Information 111
5.7.4 IAP Find Protocol Type 112
5.7.5 IAP Set Protocol Type 112
5.7.6 IAP Set Application Protocol Information 113
5.7.7 IAP1 Identification 114
5.7.8 IAP1 Apple Device Authentication 115
5.7.9 Set Assistive Touch 115
5.7.10 Set Voice Over 116
5.7.11 AP1 Get Ipod Info (Name, SW version, serial number) 116
5.7.12 IAP1 Set Extended Interface Mode (ON/OFF) 117
5.7.13 IAP1 Get Lingo Protocol Version 118
5.7.14 IAP1 Set Ipod Preferences 118
5.7.15 IAP1 Get Ipod Preferences 120
5.7.16 IAP1 Set UI Mode 120
5.7.17 IAP1 Get UI Mode 121
5.7.18 IAP1 Set Event Notification 121
5.7.19 IAP1 Get Event Notification 122
5.7.20 IAP1 Get Supported Event Notification 123
5.7.21 IAP1 Launch Application 124
5.7.22 IAP1 Get Localization Info 125
5.7.23 IAP1 Application Data Session Acknowledgment 125
5.7.24 IAP1 Application Accessory Data Transfer 126
5.7.25 IAP1 Get Voice Over Parameter 127
5.7.26 IAP1 Set VoiceOver Parameter 127
5.7.27 IAP1 Set VoiceOver Context 128
5.7.28 IAP1 VoiceOver Event 129
5.7.29 IAP1 VoiceOver Text Event 130
5.7.30 IAP1 VoiceOver Touch Event 131
5.7.31 IAP1 Current VoiceOver Value 131
5.7.32 IAP1 Current VoiceOver Hint 132
5.7.33 IAP1 Current VoiceOver Trait 132
5.7.34 IAP1 iPod Out Button 134
5.7.35 IAP1 Video Button 135
5.7.36 IAP1 Audio Button 136
5.7.37 IAP1 Context Button 137
5.7.38 IAP1 Radio Button 138
5.7.39 IAP1 Camera Button 139
5.7.40 IAP1 Rotation Input 140
5.7.41 IAP1 Register HID Report Descriptor 140
5.7.42 IAP1 Send HID Report 141

5.7.43 IAP1 Unregister HID Report Descriptor 142
5.8 PER Commands 143
5.8.1 PER Transmit 143
5.8.2 Per receive 144
5.8.3 Per cw mode 145
5.8.4 Per stats 146
5.8.5 FH_MAP 146
5.9 Role change status 147
5.9.1 Unbond or Disconnect status 147
5.9.2 Bond Response 148
5.9.3 Inquiry response 148
5.9.4 Remote device name 149
5.9.5 Disconnected 150
5.9.6 User confirmation Request 150
5.9.7 User passkey display 151
5.9.8 User pincode request 151
5.9.9 User passkey request 152
5.9.10 Inquiry complete 152
5.9.11 Auth complete 152
5.9.12 User linkkey Request 153
5.9.13 User linkkey save 153
5.9.14 SSP Enable 154
5.9.15 Mode change 154
5.9.16 Sniff subrating Currently, Sniff subrating mode is not supported. 155
5.10 SPP events 155
5.10.1 SPP Receive 155
5.10.2 SPP connected 156
5.10.3 SPP Disconnected 156
5.11 A2DP events 157
5.11.1 A2DP Connected 157
5.11.2 A2DP Disconnected 157
5.11.3 A2DP Configured 158
5.11.4 A2DP Stream Open 158
5.11.5 A2DP Started 159
5.11.6 A2DP Suspend 159
5.11.7 A2DP Abort 160
5.11.8 A2DP Close 160
5.11.9 A2DP Stream Data 161
5.12 AVRCP Events 161
5.12.1 AVRCP Connect 161
5.12.2 AVRCP Disconnect 162
5.12.3 AVRCP Play 162
5.12.4 AVRCP Pause 163
5.12.5 AVRCP Stop 163
5.12.6 AVRCP Next 164
5.12.7 AVRCP Previous 164
5.13 Apple IAP1 Events IAP Connected 164
5.13.1 IAP Disconnected 165
5.13.2 IAP1 Accessory Authentication Started 165
5.13.3 IAP1 Accessory Authentication Failed 165
5.13.4 IAP1 Accessory Authentication Completed 166
5.13.5 IAP1 Now Playing Application Bundle Name 166
5.13.6 IAP1 Now Playing Application Display Name 166
5.13.7 IAP1 Assistive Touch Status 167
5.13.8 IAP1 IPodOut Status 167
5.13.9 IAP1 Flow Control Status 167

5.13.10 IAP1 Radio Tagging Status 168
5.13.11 IAP1 Camera status 168
5.13.12 IAP1 Database changed status 169
5.13.13 IAP1 Session Space Available Notification 169
5.13.14 IAP1 Bluetooth Status 169
5.13.15 IAP1 Voiceover Parameter Changed status 170
5.13.16 IAP1 Application Data Session Opened 170
5.13.17 IAP1 Application Data Session Closed 170
5.13.18 IAP1 Ipod Data Received 171
5.13.19 IAP1 Accessory HID Report 171
5.13.20 AFH Channel Classification 172
5.13.21 A2DP Disconnect 173
6 Embedded BT Classic Error Codes 174
6.1 Generic Error Codes 174
6.2 Core Error Codes 176
7 Embedded BT Classic API Library 182
7.1 Bluetooth Classic API File Organization 182
7.2 Bluetooth Classic Application 182
7.3 Bluetooth Classic API Prototypes 183
7.3.1 Set Local name 183
7.3.2 Query Local name 183
7.3.3 Set Local COD 184
7.3.4 Query Local COD 184
7.3.5 Query RSSI 184
7.3.6 Query Link Quality 184
7.3.7 Query Local BD Address 184
7.3.8 Initialize BT Module 184
7.3.9 Deinitialize BT Module 184
7.3.10 BT Antenna Select 184
7.3.11 Set Feature Bitmap 184
7.4 BT Classic Core Prototypes 185
7.4.1 Set Profile mode 185
7.4.2 Set Discovery mode 185
7.4.3 Get Discovery mode 185
7.4.4 Set Connectability mode 185
7.4.5 Get Connectability mode 185
7.4.6 Set Pair Mode 185
7.5 Get Pair Mode 185
7.5.1 Remote Name Request 185
7.5.2 Remote Name Request Cancel 185
7.5.3 Inquiry 186
7.5.4 Inquiry Cancel 186
7.5.5 Set EIR data 186
7.5.6 Bond or Create Connection 186
7.5.7 Bond Cancel or Create Connection Cancel 186
7.5.8 Unbond or Disconnect 186
7.5.9 Set Pin Type 186
7.5.10 Get Pin Type 186
7.5.11 User Confirmation 186
7.5.12 Passkey Request Reply 187
7.5.13 Pincode Reply 187
7.5.14 Get Local Device Role 187
7.5.15 Set Local Device Role 187
7.5.16 Get Service List 187
7.5.17 Search Service 187
7.5.18 Linkkey Reply 187

7.5.19 Enable SSP mode 187
7.5.20 Accept SSP confirm 187
7.5.21 Reject SSP confirm 188
7.5.22 Start sniff mode 188
7.5.23 Exit sniff mode 188
7.5.24 Sniff subrating mode Currently, Sniff Subrating mode is not supported 188
7.6 BT SPP Prototypes 188
7.6.1 SPP connect 188
7.6.2 SPP Disconnect 188
7.6.3 SPP Transfer 188
7.7 BT A2DP Prototypes 188
7.7.1 A2DP Init 188
7.7.2 A2DP Burst mode 189
7.7.3 A2DP Connect 189
7.7.4 Send PCM or MP3 Data 189
7.7.5 Send SBC Data 189
7.7.6 A2DP Disconnect 189
7.8 BT AVRCP Prototypes 189
7.8.1 AVRCP Init 189
7.8.2 AVRCP Connect 189
7.8.3 AVRCP Disconnect 189
7.8.4 AVRCP Play 190
7.8.5 AVRCP Pause 190
7.8.6 AVRCP Notify 190
7.8.7 AVRCP Notify response 190
7.8.8 AVRCP Play status response 190
7.8.9 AVRCP Get capcabilites response 190
7.8.10 AVRCP Get attribute list response 190
7.8.11 AVRCP Get attribute value list response 190
7.8.12 AVRCP Element attribute response 190
7.9 PER Commands Prototypes 191
7.9.1 PER Transmit 191
7.9.2 PER Receive 191
7.9.3 PER CW Mode 191
7.9.4 PER Stats 191
7.9.5 Afh map 191
8 Embedded BT Classic Appendix A :Sample flows 192
8.1 Configure BT device in Master mode 192
8.2 Configure BT device in Slave mode 194
8.3 Configure BT device in Master Mode and do SPP Tx 195
8.4 Configure BT device in Slave Mode and do SPP Tx 196
8.5 AT command sequence to perform SPP data transfer in BT Master mode 196
8.6 AT command sequence to perform SPP data transfer in BT Slave mode 198

About the Document

This document describes the commands to operate RYWB116 in Bluetooth. The parameters in the commands and their valid values with the expected responses from the modules are also described. The document should be used by the developer to write software on the Host MCU to control and operate the module.

1 Embedded BT Classic Overview

This section describes the commands to operate RYWB116. 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

RYWB116 APIs are designed in layers, where each Layer is independent and uses the service of underlying layers.

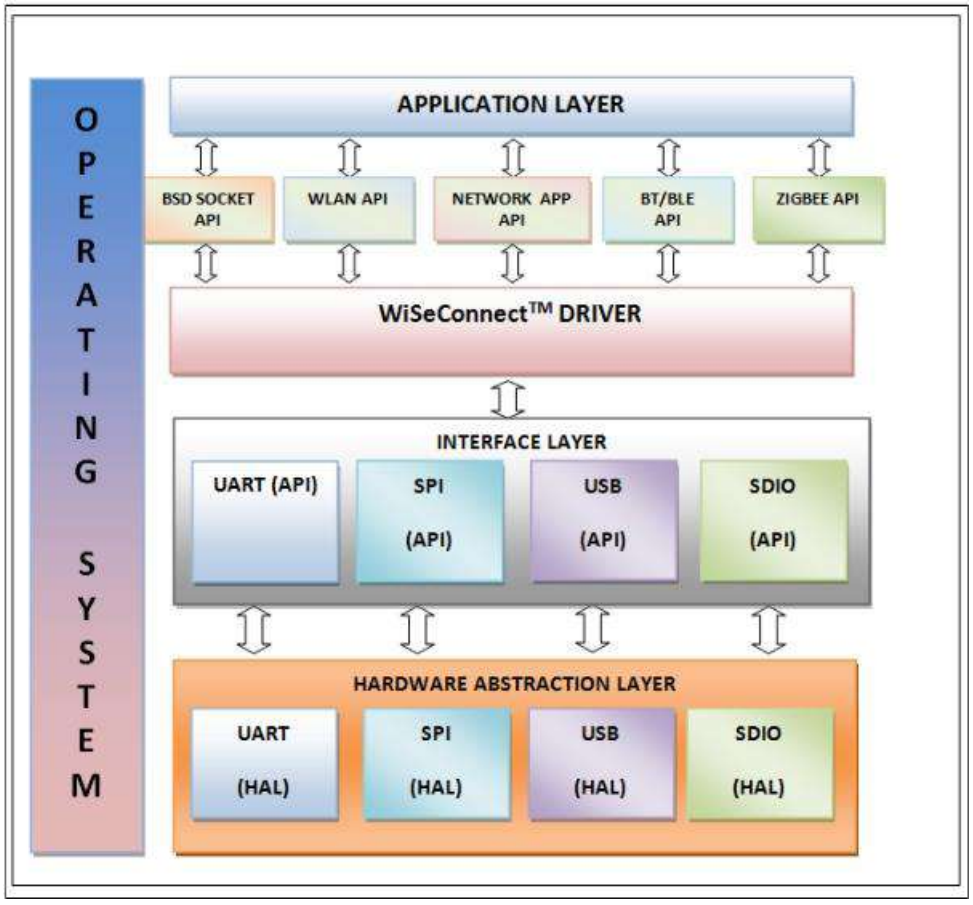


Figure 1: Architecture Overview

1.1.1 Bluetooth Classic Architecture

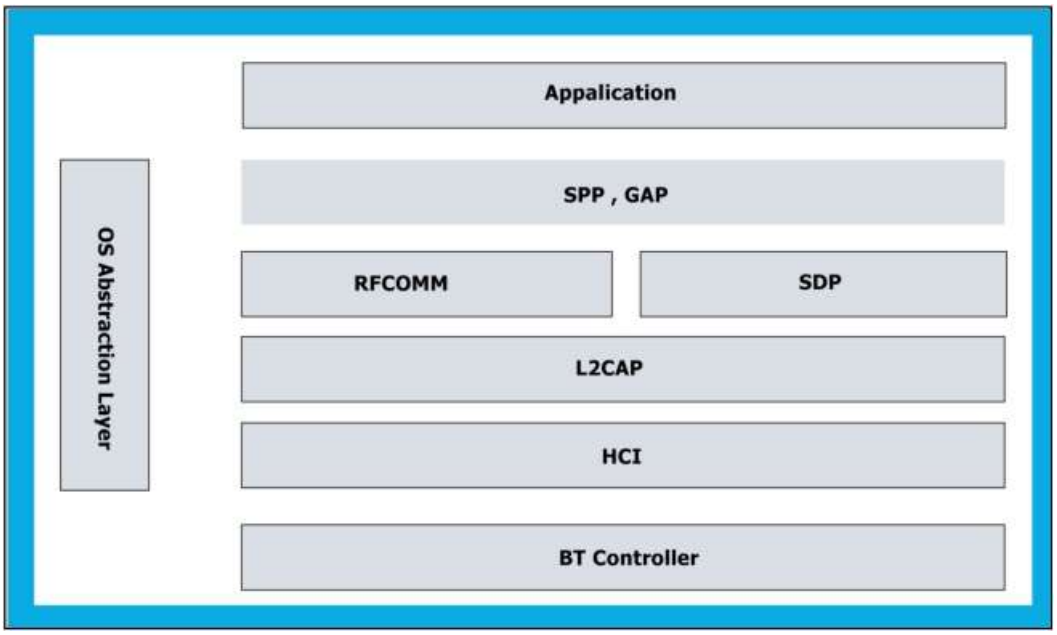


Figure 2: Bluetooth Software Architecture

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.

1.3 Profiles

There are number of Bluetooth profiles defined in the Bluetooth specification. We currently supports profiles including Serial Port Profile (SPP). We provide framework to develop new profiles very easily. We will continue to add new profiles.

1.4 Bluetooth Core

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

- RFCOMM
- SDP
- L2CAP
- HCI Generic Driver
- HCI BUS Driver

RFCOMM is a transport protocol based on L2CAP. It emulates RS-232 serial ports. The RFCOMM protocol supports up to 60 simultaneous connections between two BT devices. RFCOMM provides data stream interface for higher level applications and profiles.
SDP (Service Discovery Protocol) provides a means for applications to discover which services are available and to determine the characteristics of those available services. SDP uses an existing L2CAP connection. Further connection to Bluetooth devices can be established using information obtained via SDP.
L2CAP (Logical Link Control and Adaptation Protocol) provides connection-oriented and connectionless data services to upper layer protocols with data packet size up to 64 KB in length. L2CAP performs the segmentation and reassemble of I/O packets from the baseband controller.
HCI Generic Driver – This driver implements the HCI Interface standardized by Bluetooth SIG. It establishes the communication between the Stack and the HCI Firmware in the Bluetooth hardware. It communicates with the Bluetooth controller hardware via the HCI Bus driver.
HCI Transport Layer Driver – The Bluetooth controllers are connected to the host using interface like UART, USB,

SDIO, SPI, USB-CDC etc. The HCI Transport Layer Driver provides hardware abstraction to the rest of the Bluetooth stack software. This driver makes it possible to use Bluetooth stack with different hardware interfaces.

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.

1.5.1 Host-Module Interface

This document is primarily concerned with the host-module interface. The host can interface with RYWB116 using following list of interfaces to configure and send/receive data.

- SPI
- UART
- USB

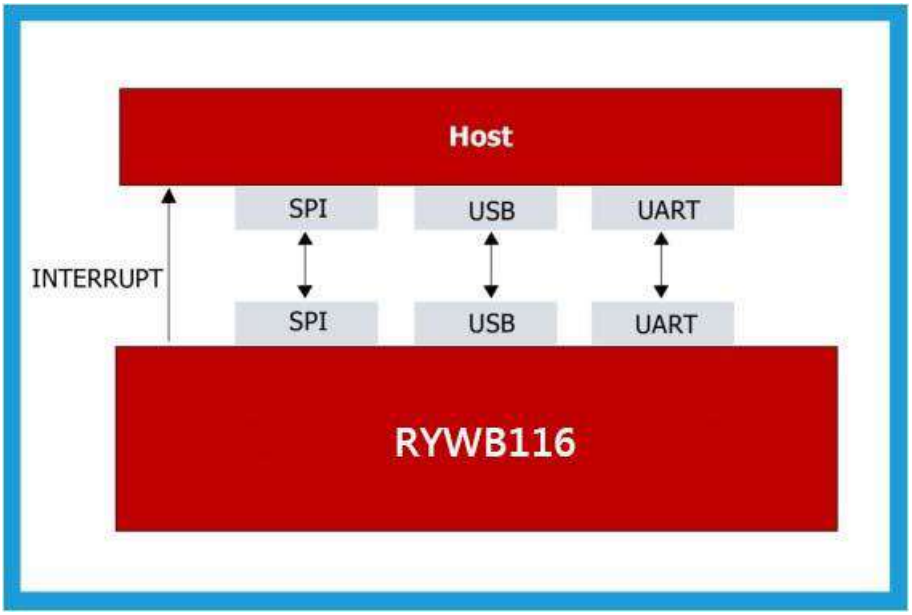


Figure 3: Host Interface Block Diagram

Note:
USB-CDC/UART/USB/SPI are the host interfaces for RYWB116.

2 Embedded Bootloader and Interfaces

2.1 Bootloader

2.1.1 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 RYWB116 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.

2.2 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.

2.2.1 Host Interaction Mode in UART / USB-CDC

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

2.2.1.1 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 CHECKSUM FAILED"** is displayed when the backup integrity also fails before displaying the bootup options.

Hyper Terminal Configuration

RYWB116 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 in the figure below.

Note:
Default baud rate of the module is 115200.

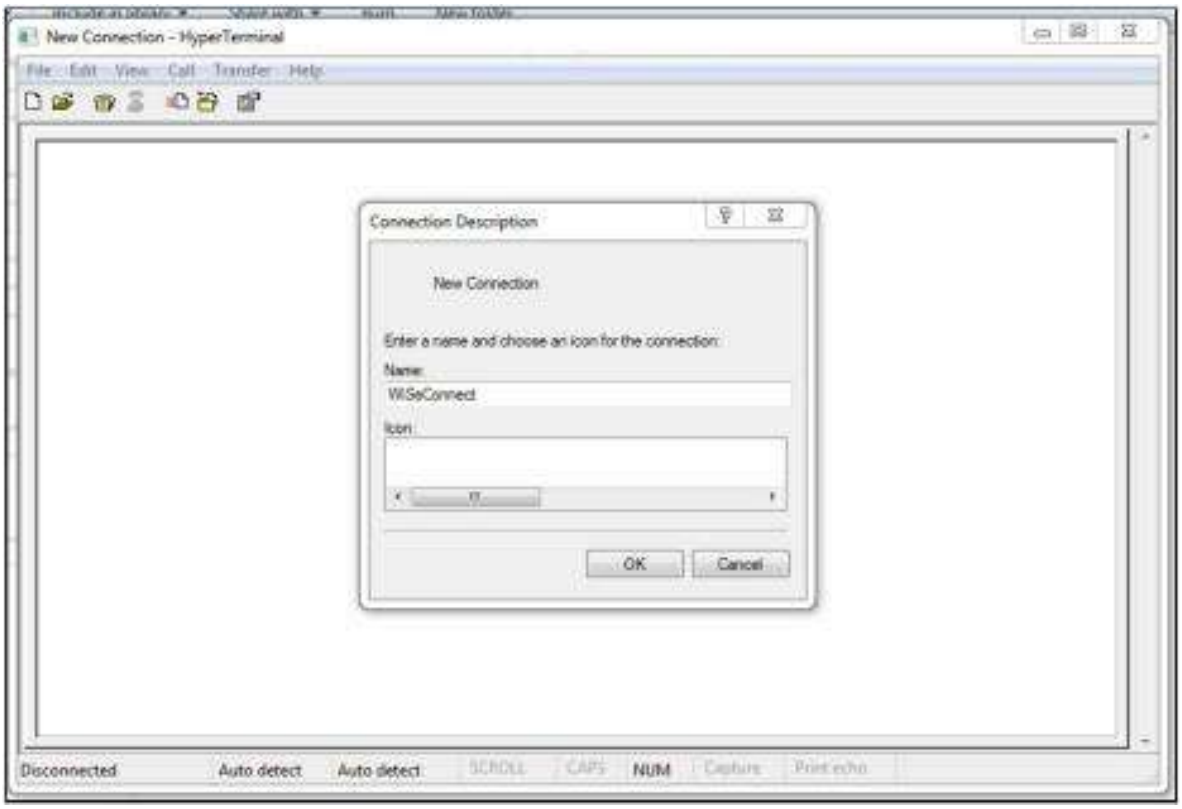


Figure 4: HyperTerminal Name field Configuration

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

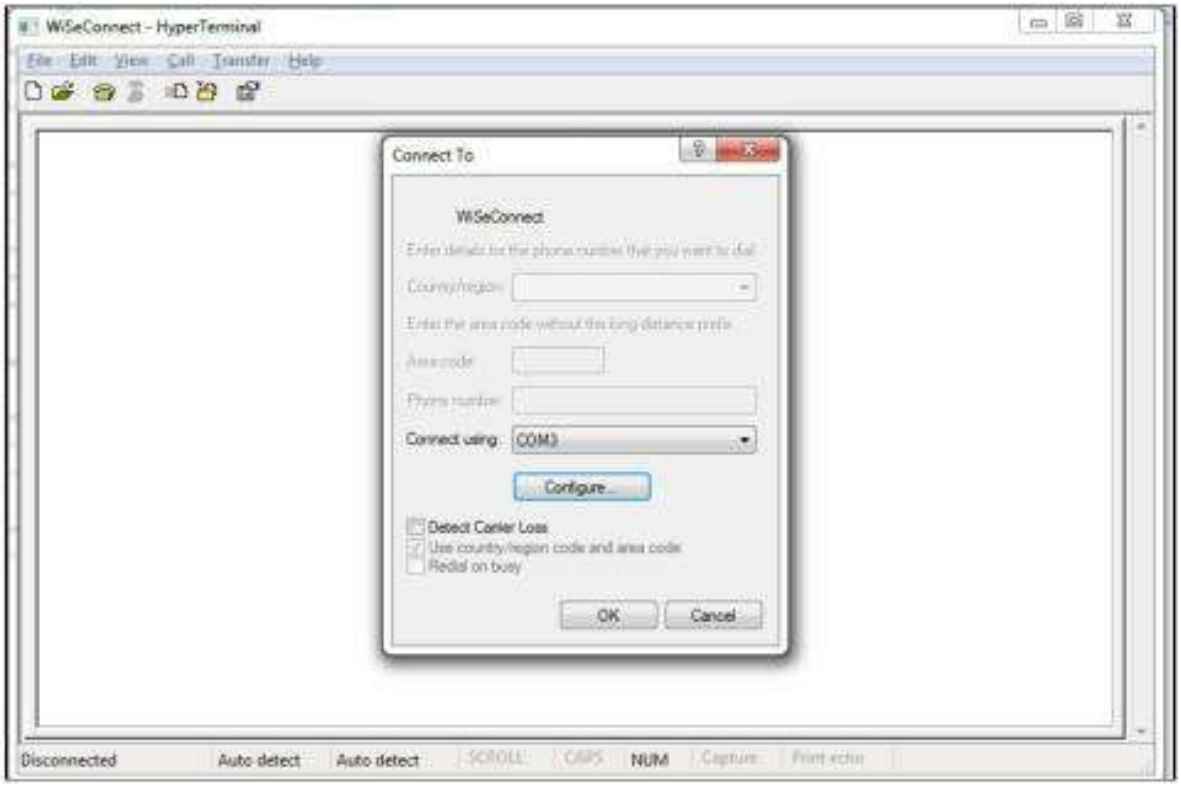


Figure 5: HyperTerminal COM port field Configuration

- 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.

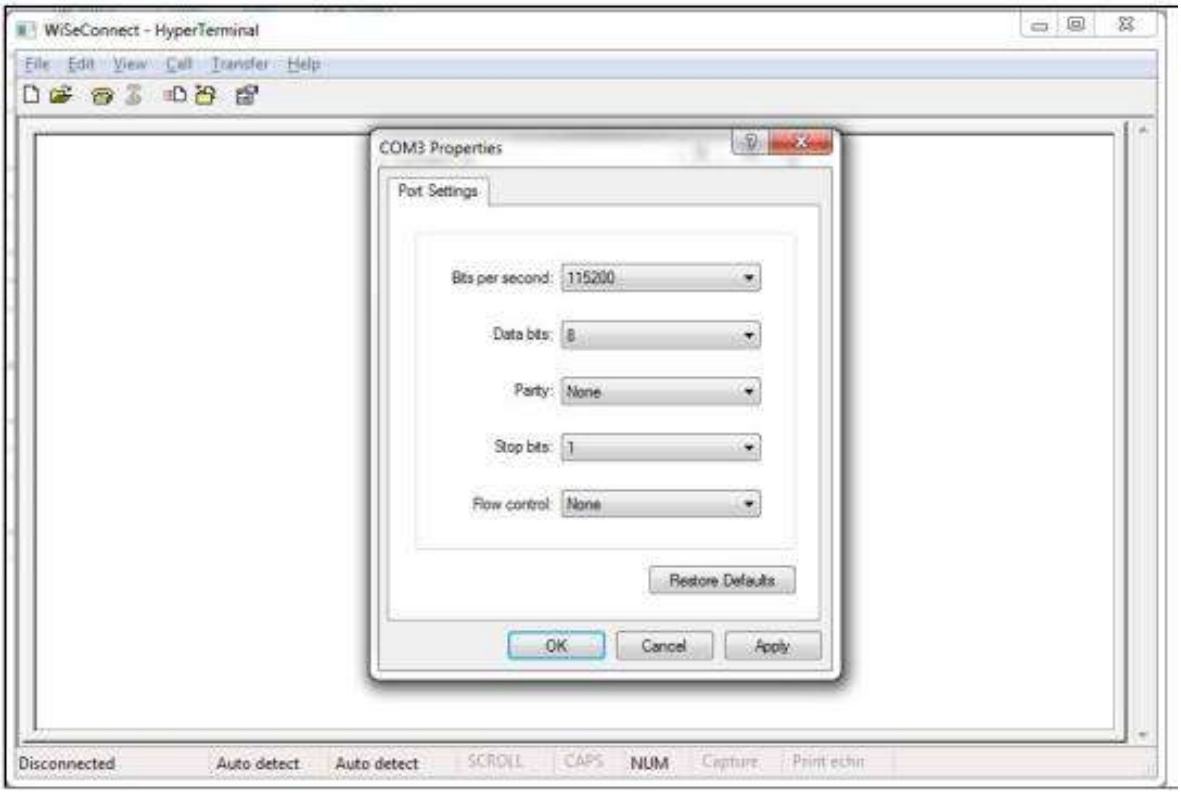


Figure 6: HyperTerminal Baud rate field Configuration

Set the following values for the fields shown in the Figure 5.

- 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 RYWB116 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.
RYWB116 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 RYWB116, 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 RYWB116.
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.

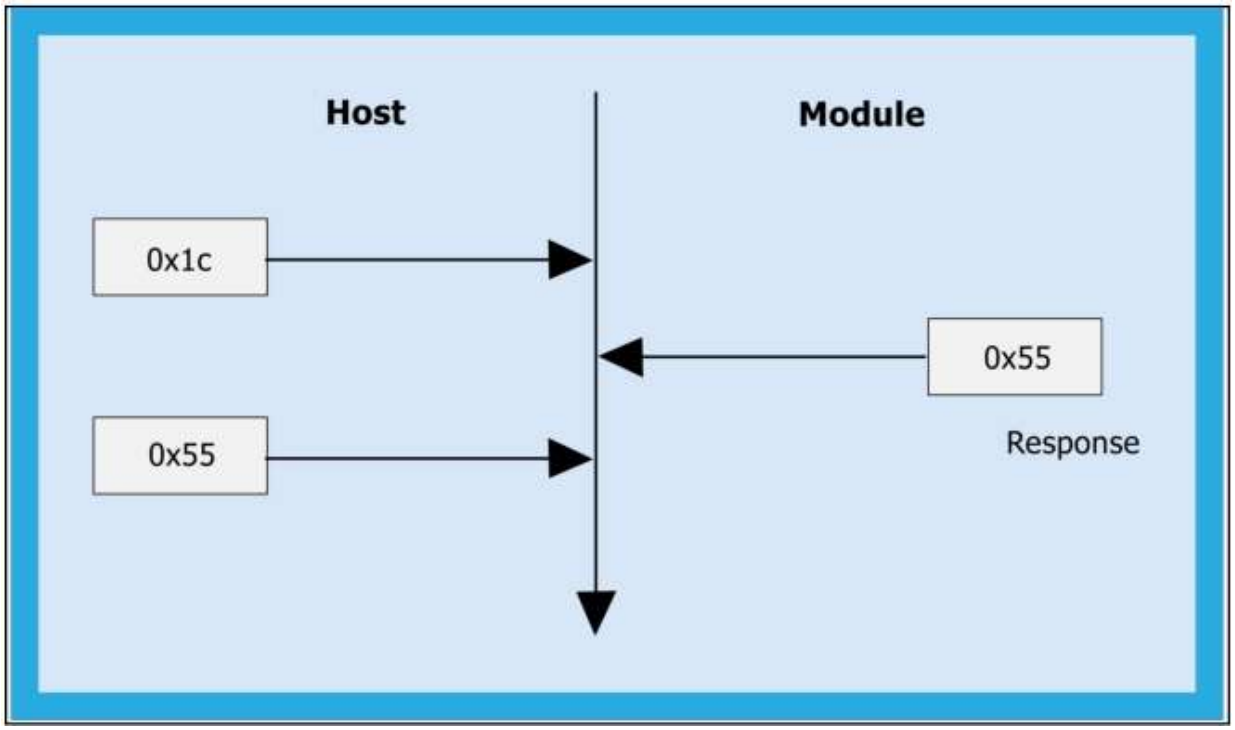


Figure 7: ABRD exchange between Host and module

Below are the bootup options, Firmware upgrade and Firmware loading procedure for RYWB116.

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

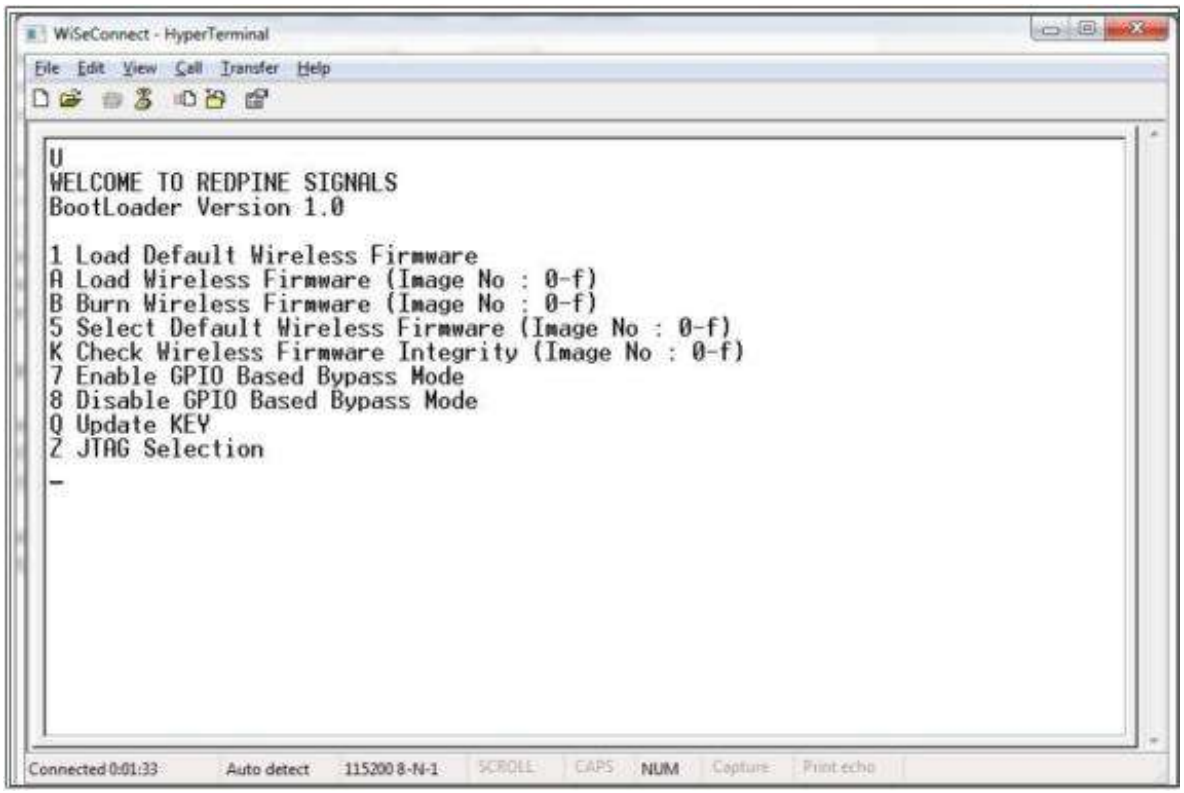


Figure 8: RYWB116 Module UART/USB-CDC Welcome Message

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

2.2.1.3 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.

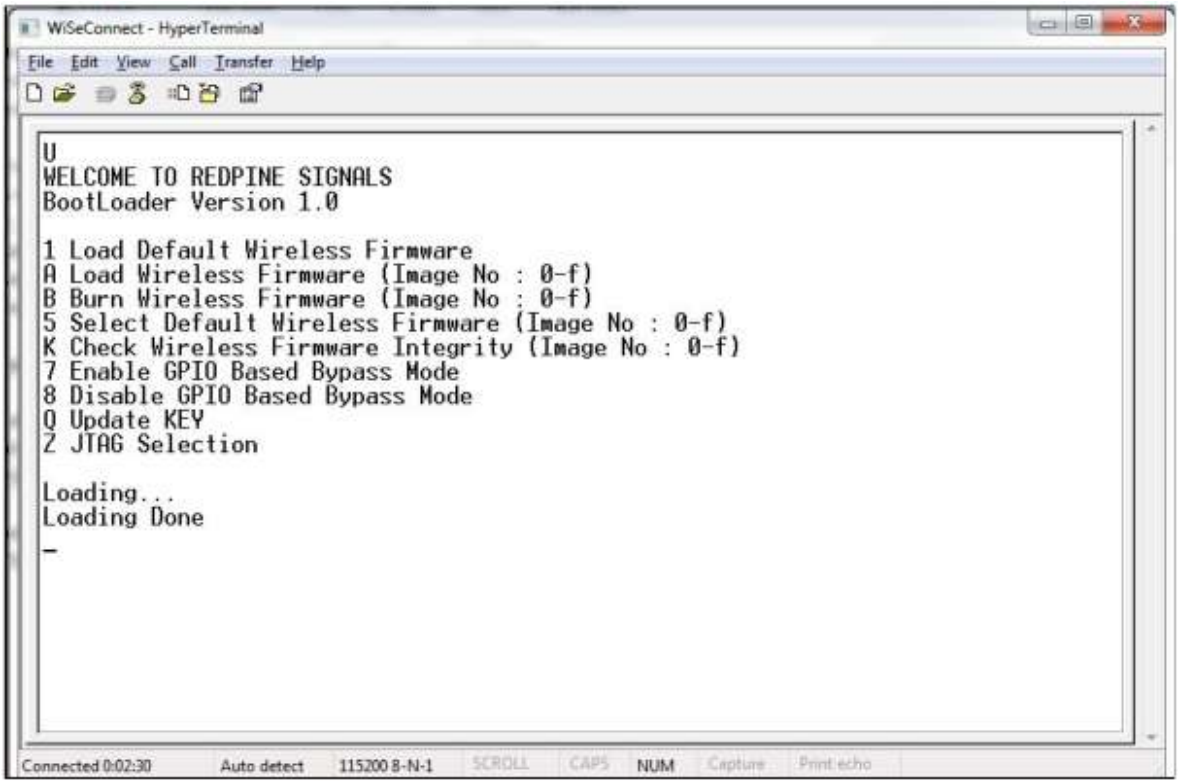


Figure 9: RYWB116 Module UART / USB-CDC Default Firmware Loaded

2.2.2 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)" .

2.2.2.1 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".

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.

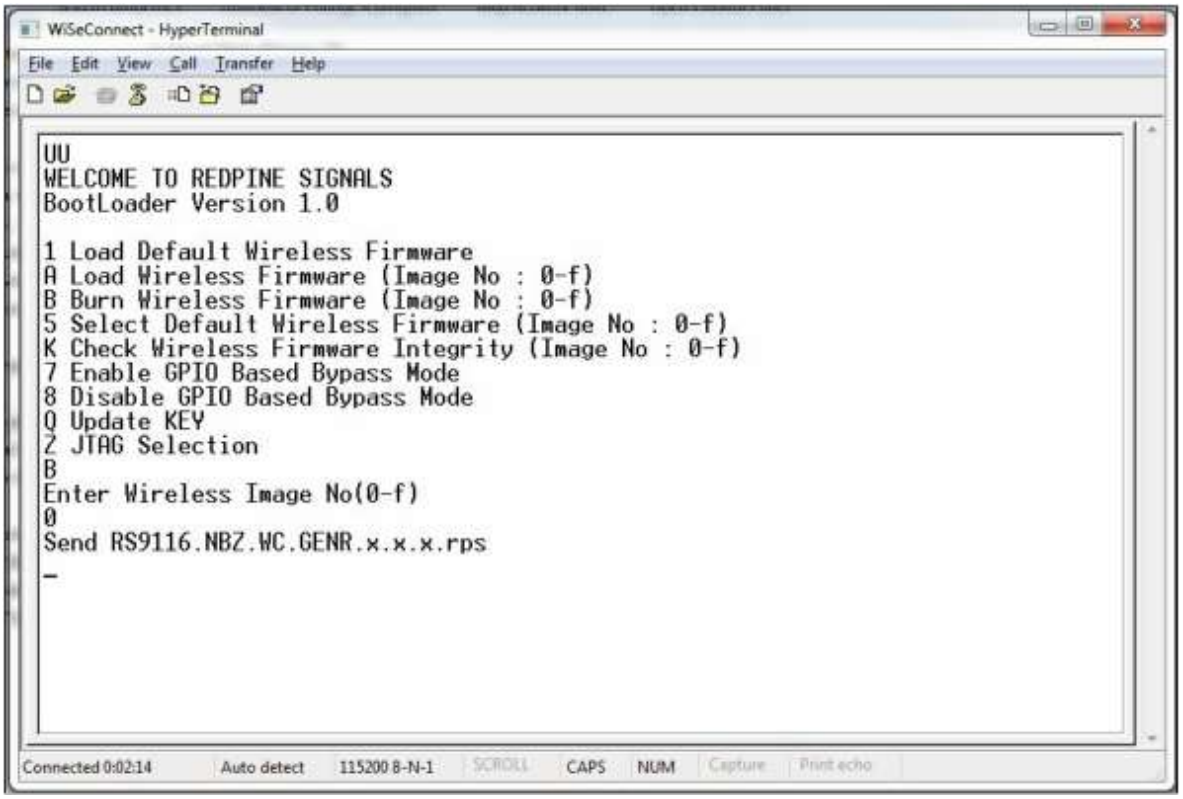


Figure 10: RYWB116 Module Firmware Upgrade File Prompt Message

- 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.

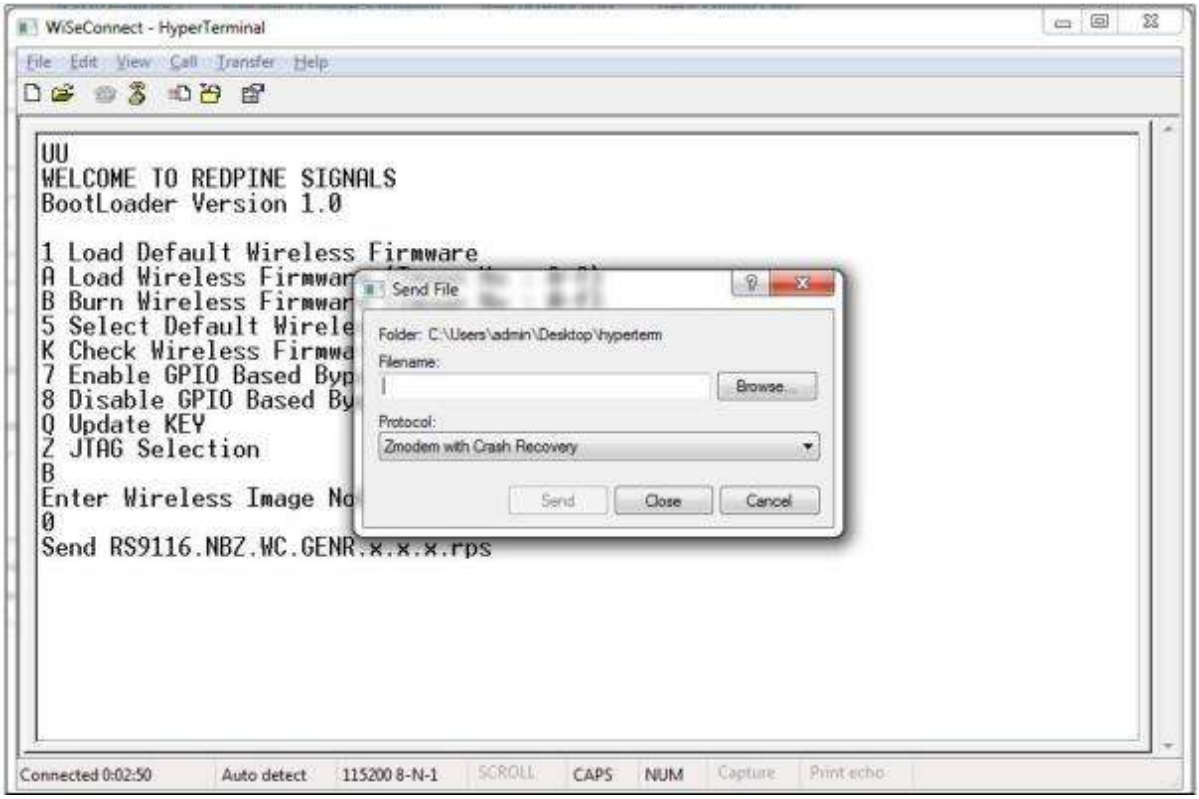


Figure 11: RYWB116 Module Firmware Upgrade File Selection Message

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

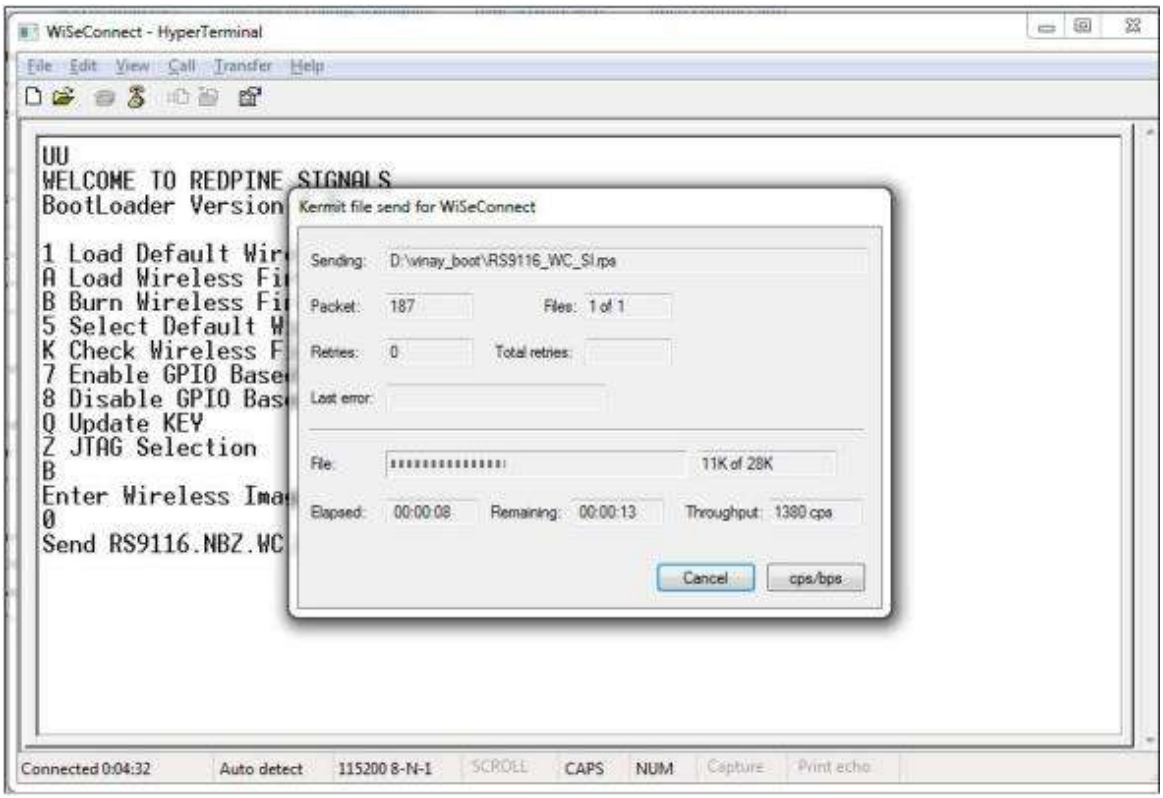


Figure 12: RYWB116 Module Firmware Upgrade File transfer Message

- 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.

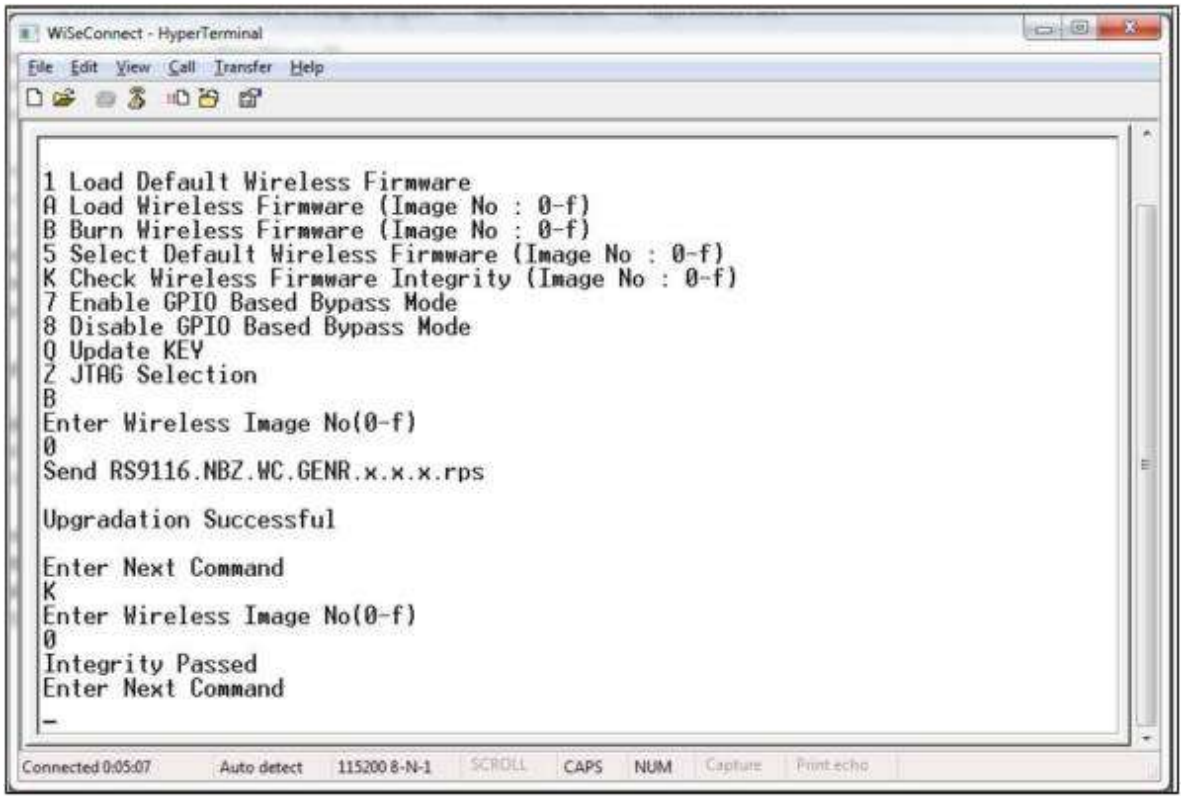


Figure 13: RYWB116 Module Firmware Upgrade Completion Message

- 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 **Loading the Default Wireless Firmware in the Module** to load the firmware from flash, select Option 1 from the above the **Figure**.
- The module is ready to accept commands from the Host.

2.3 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.

2.3.1 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 1: HOST_INTF_REG_IN Register values

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

Table 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: 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 4: Bootloader Response Codes

2.3.2 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.

2.4 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.

2.4.1 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_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_DEFAULT_TA_FW) to load default wireless Image

2.5 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.

2.5.1 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

2.6 Upgrading Wireless Firmware in the Module

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

2.6.1 Upgrading Wireless Firmware

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

1. After reading the valid BOARD_READY (i.e. HOST_INTERACT_REG_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 4 Kbytes 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.

HOST
Module
Module writes BOARD_READY(HOST_INTERACT_REG_OUT_VALID | BOOTLOADER VERSION) in HOST_INTF_REG_OUT
Host writes RSI_UPGRADE_IMAGE_1_FW(HOST_INTERACT_REG_IN_VALID | BURN_TA_FW | (IMAGE_NO << 8) in HOST_INTF_REG_IN
Polling HOST_INTF_REG_OUT to read SEND_RPS_FILE(0xAB32)
Polling HOST_INTF_REG_OUT to read BOARD_READY(HOST_INTERACT_REG_OUT_VALID | BOOTLOADER VERSION)
Module writes RSI_SEND_RPS_FILE(0xAB32) in HOST_INTF_REG_OUT
Master write to PING_LOCATION
Host writes PING_VALID(0xAB49) in HOST_INTF_REG_IN
Polling HOST_INTF_REG_OUT to read PONG_AVAIL (0xAB4f)
Module writes PONG_AVAIL (0xAB4f) in HOST_INTF_REG_OUT
Master write to PONG_LOCATION
Host writes PONG_VALID(0xAB4f) in HOST_INTF_REG_IN
Polling HOST_INTF_REG_OUT to read PING AVAIL (0xAB4f)
Host writes EOF_REACHED(0xAB45) in HOST_INTF_REG_IN
Module writes FWUP_SUCCESSFUL (0xAB53) in HOST_INTF_REG_OUT
(or) FWUP_CRC_FAILURE(0xABCC) in HOST_INTF_REG_OUT
Figure 14: Wireless Firmware upgradation through SPI/USB

2.7 GPIO Based Bootloader Bypass Mode for RYWB116

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.
2. Enable Bootloader bypass mode.
3. Assert LP_WAKEUP to Bypass Bootloader on power up.

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.

2.8 Bypass Mode in UART / USB-CDC

2.8.1 Making Default Wireless Firmware Selection

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

2.8.1.1 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".

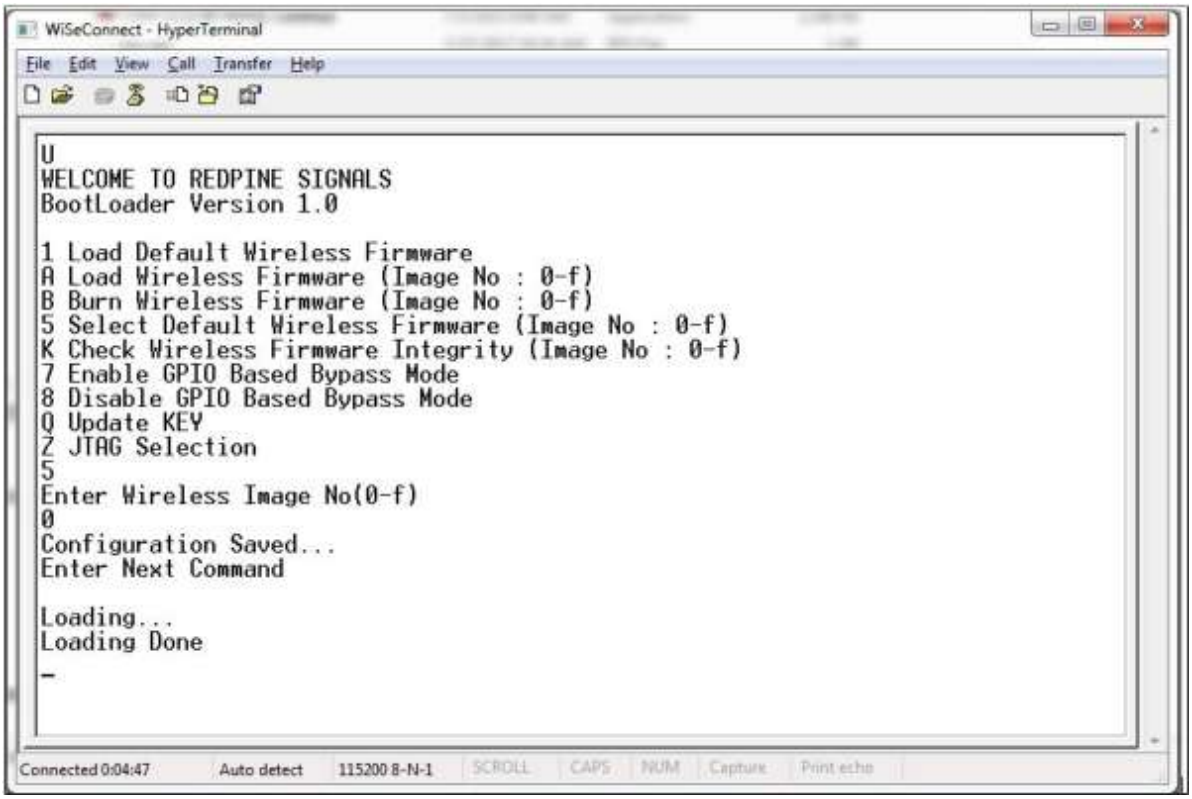


Figure 15: Making Image no - 0 as default image

2.8.2 Enable/Disable GPIO Based Bypass Option

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

2.8.2.1 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. "Default image invalid" will be displayed.
- Once enabled, from next bootup, Bootloader will latch the value of GPIO-15. If asserted, it will bypass the whole boot loading process and will load the default firmware image selected.
- 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".
- Then select option 7 "Enable GPIO Based Bypass Mode"
- 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.

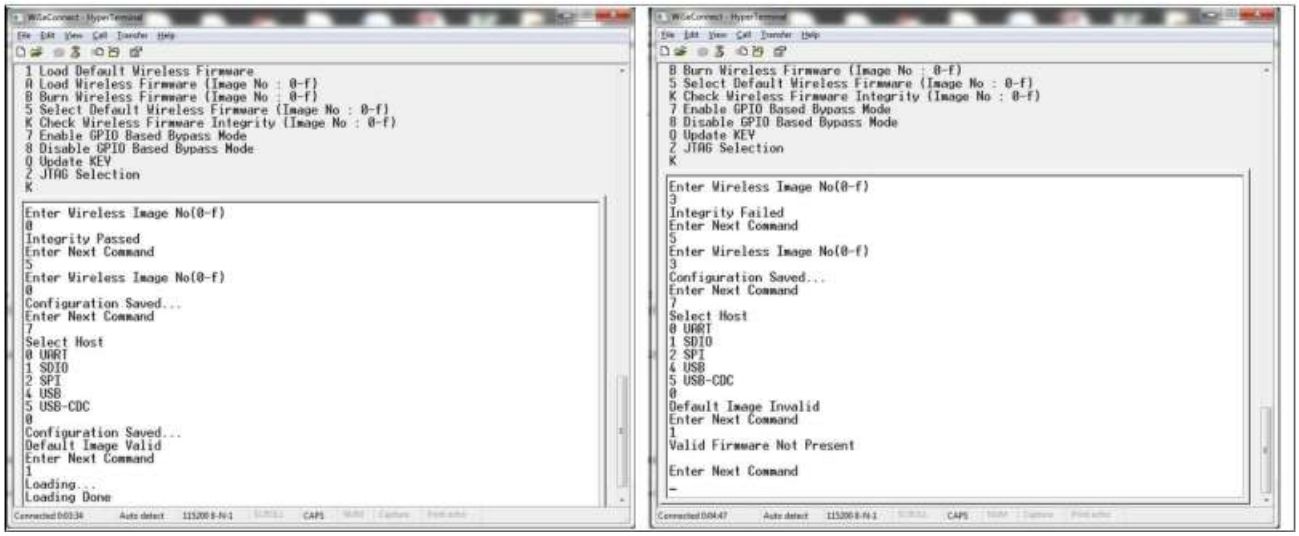


Figure 16: Enabling the GPIO based bypass mode a) Valid Default Firmware b) Invalid Firmware

2.8.2.2 Disabling the GPIO Based Bypass Mode

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

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.

2.8.3 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.

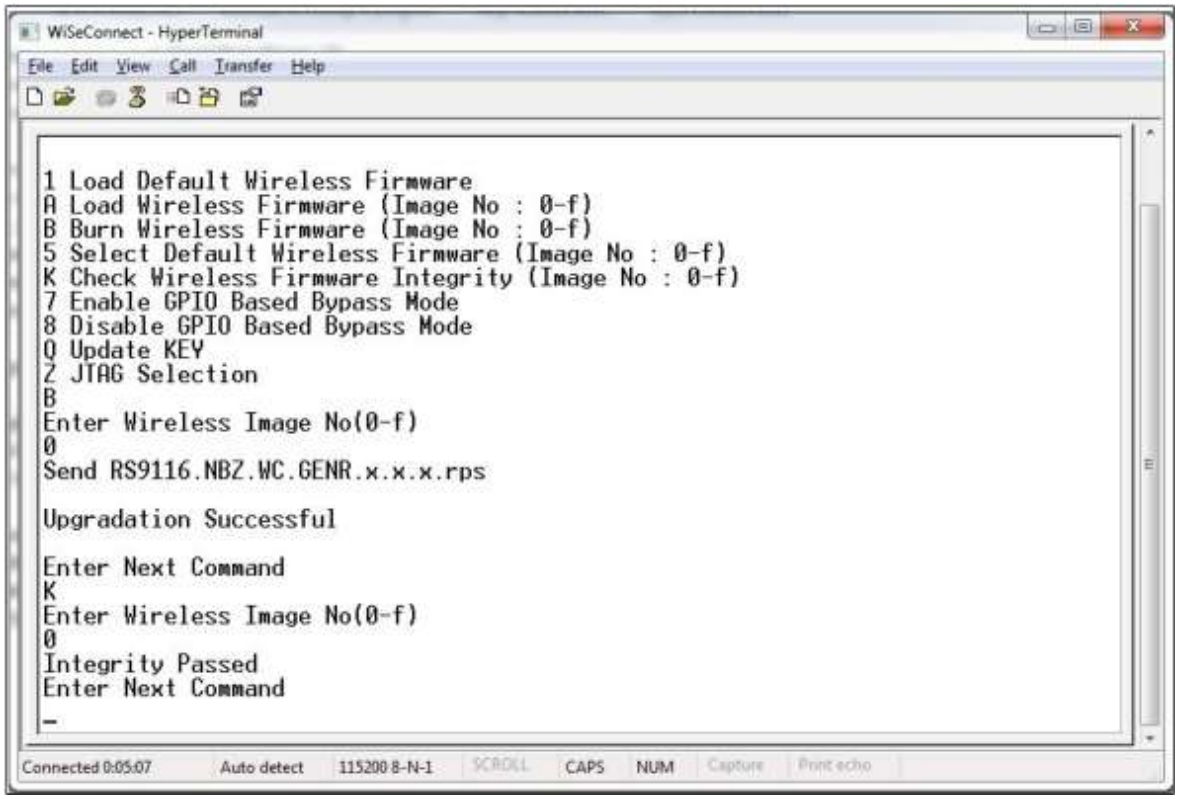


Figure 17: Integrity Check passed

2.9 Bypass Mode in SPI / USB

2.9.1 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

2.9.2 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.

2.10 Other Operations

This section contains additional, less frequently used bootloader options.

2.11 Update KEY

NOTE:
This feature is not enabled in current release.

2.12 JTAG Selection

NOTE:
This feature is not enabled in current release.

2.13 SPI Interface

This section describes RYWB116 SPI interface, including the commands and processes to operate the module via SPI.

2.14 Features

The features are as follows:

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

2.15 Hardware Interface

The SPI interface on the RYWB116 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 RYWB116 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).

2.15.1 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

2.15.2 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.

2.15.3 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).

2.15.3.1 Operations through SPI

The RYWB116 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.

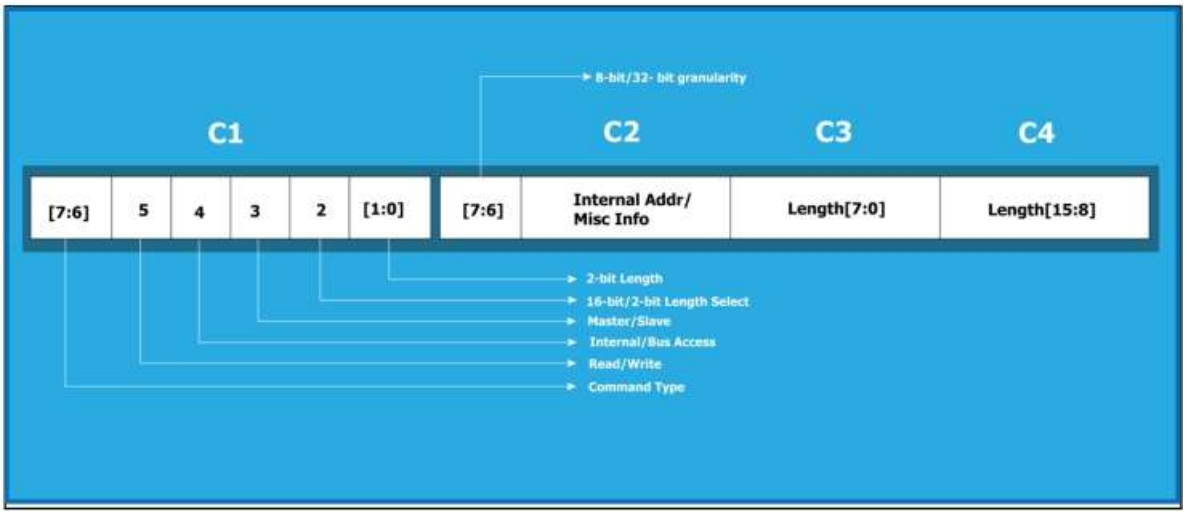


Figure 20: SPI Command Description

The command descriptions are as follows:

C1	[7:6]	Command Type "00"- Initialization Command "01"- Read/Write Command "10", "11"- Reserved for future use

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

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

2.15.3.2 Module Response

The RYWB116 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

2.15.3.3 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.

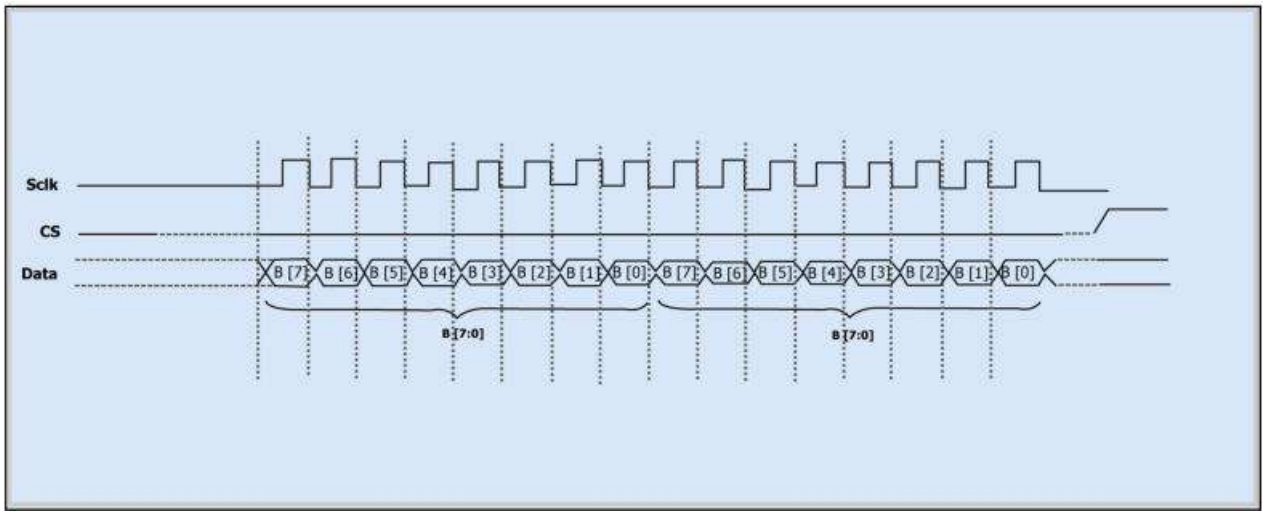


Figure 21: 8-bit Mode

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.

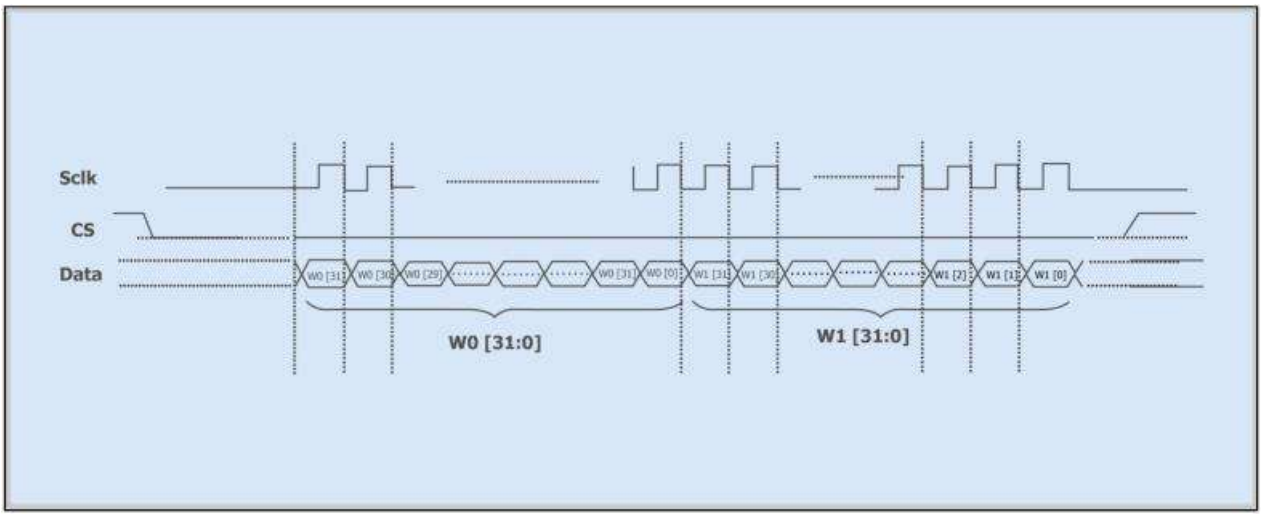


Figure 22: 32-bit Mode

Bit Ordering of Module Response

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

2.15.4 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.

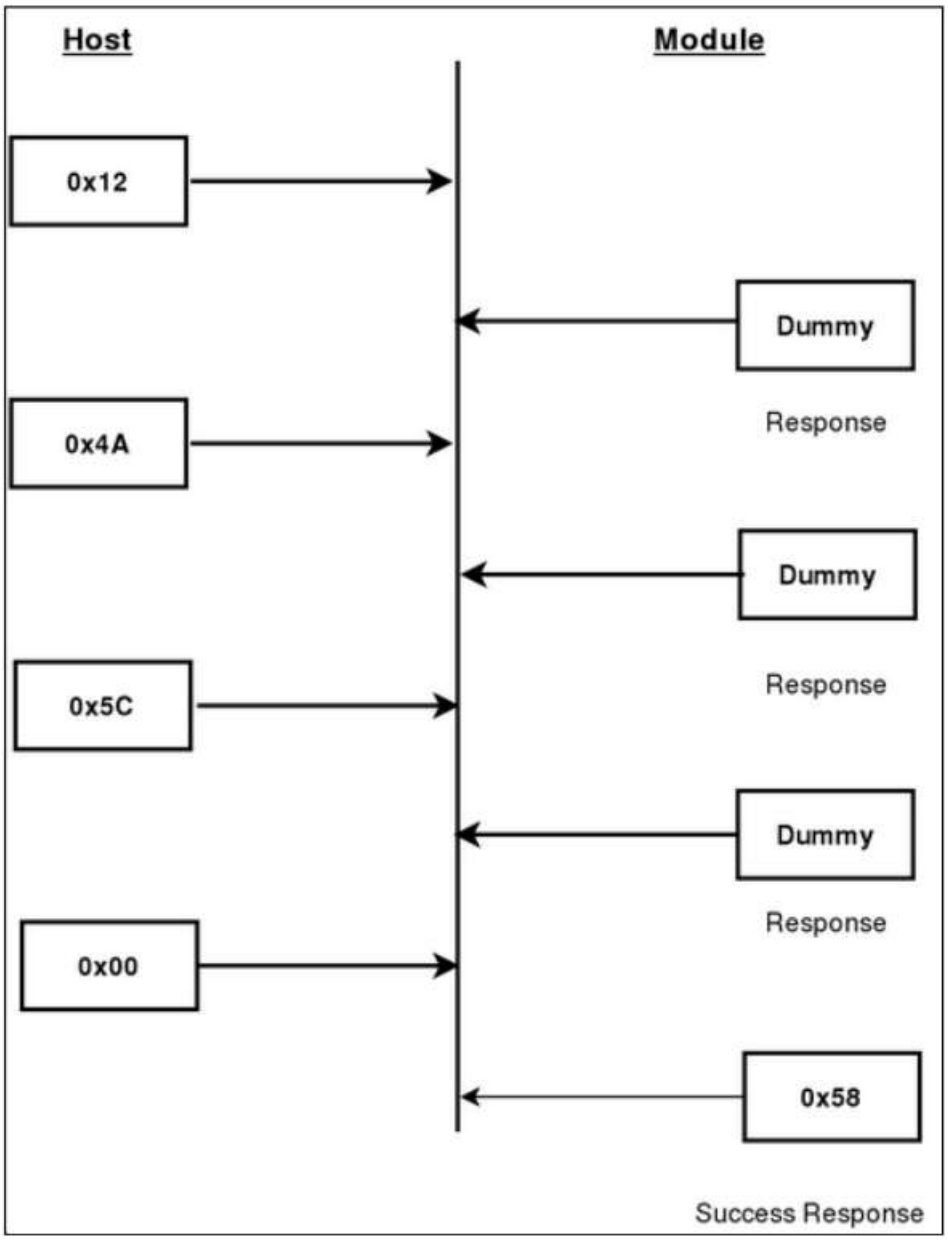


Figure 23: SPI Initialization exchanges between Host and Module

Host Interactions Using SPI Command

This section describes the procedures to be followed by the Host to interact with the RYWB116 using SPI commands.
The Host interactions to the module can be categorized as below.

Error rendering macro 'captioneditem' : Plugin license error: EXPIRED

2.15.4.1 Memory Type

Host need to access the memory / registers of the RYWB116 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.

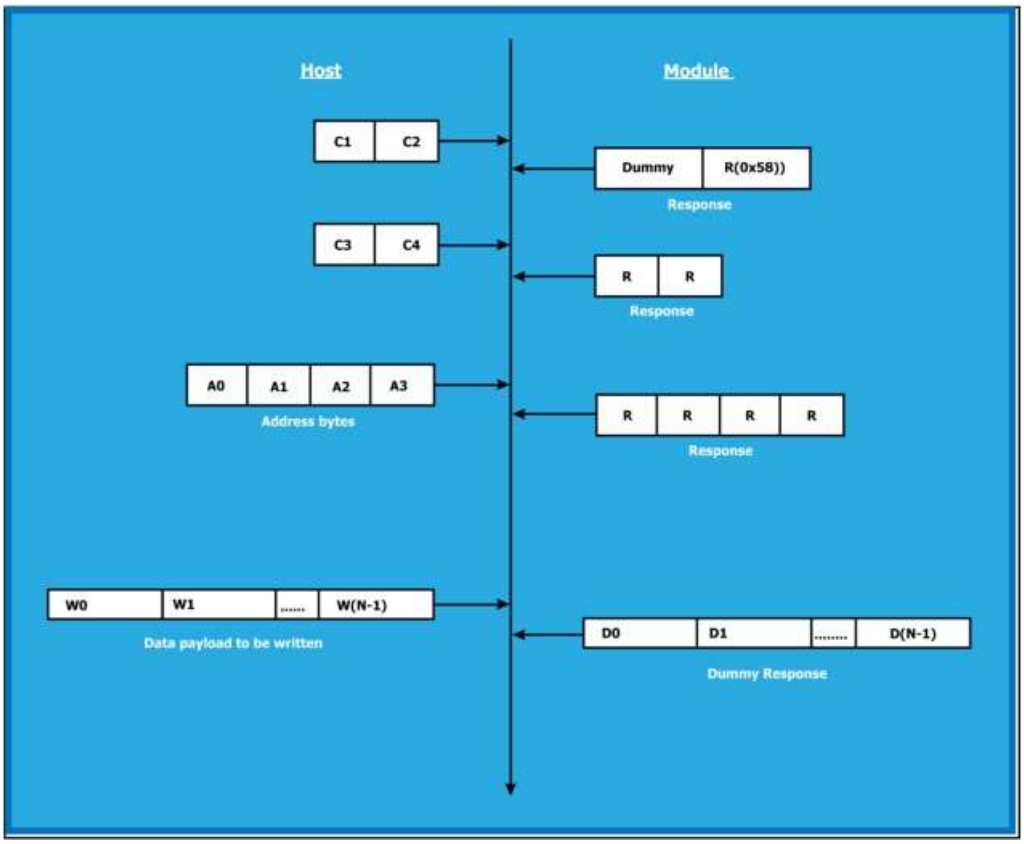


Figure 24: Memory Write

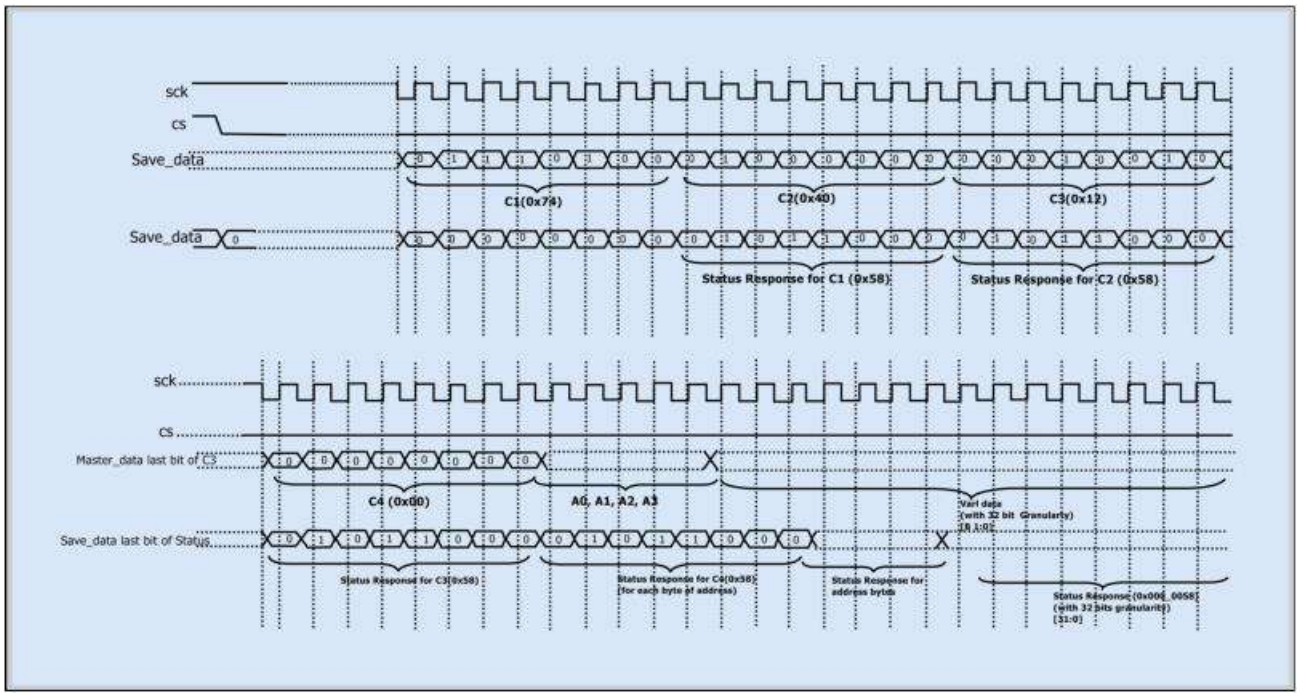


Figure 25: Interactions in the physical interface

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 RYWB116 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**).

2.15.4.2 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.

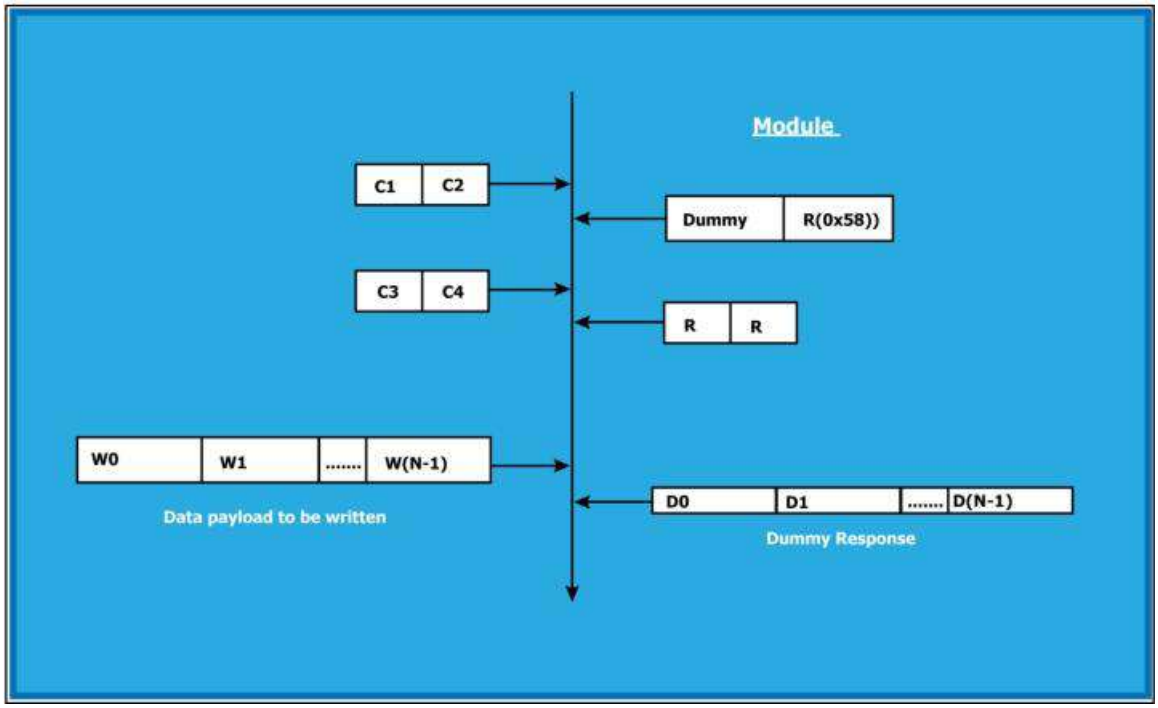


Figure 26: Frame Write

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 (RYWB116).
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**).

2.15.4.3 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.

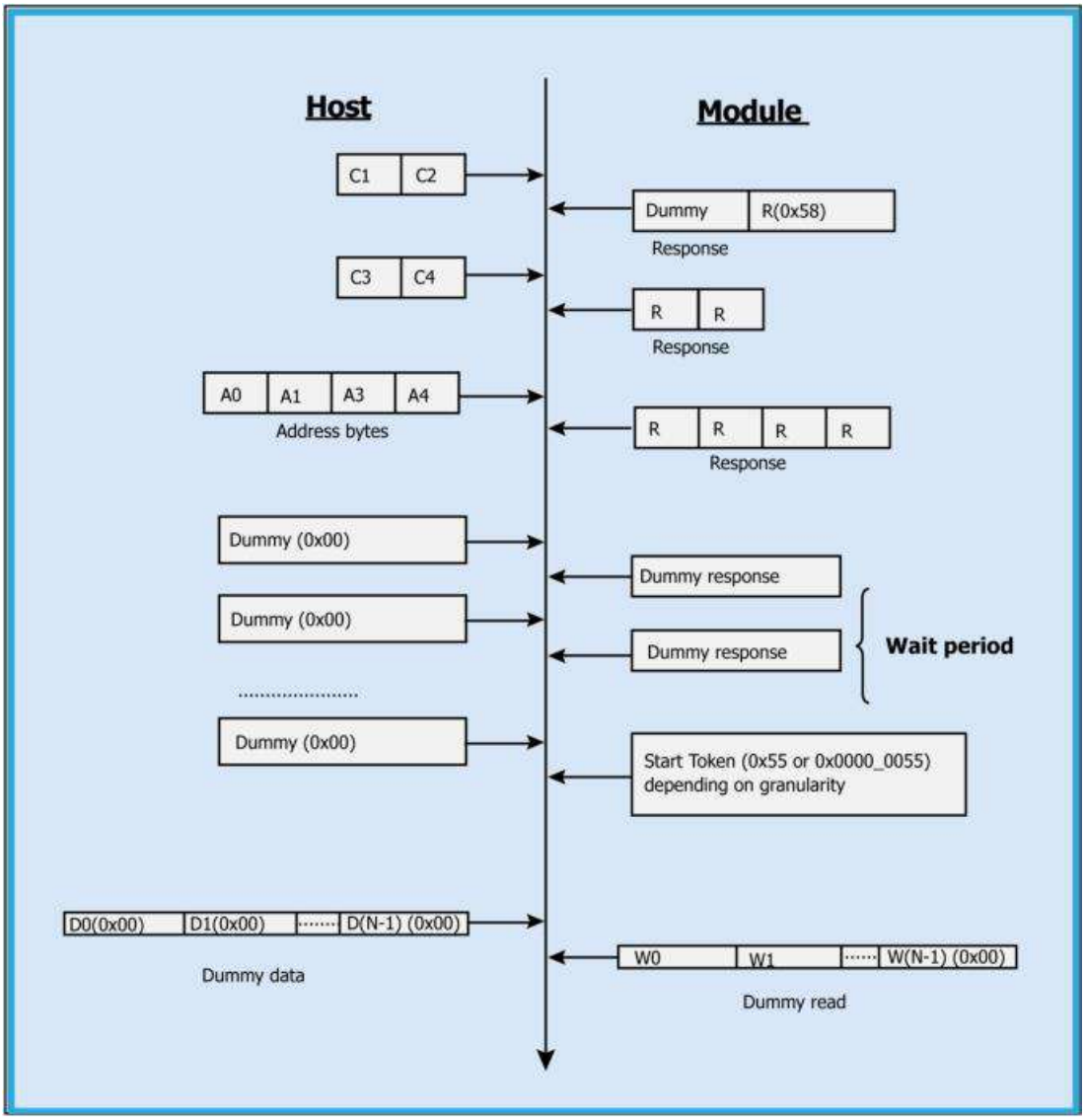


Figure 27: Memory read

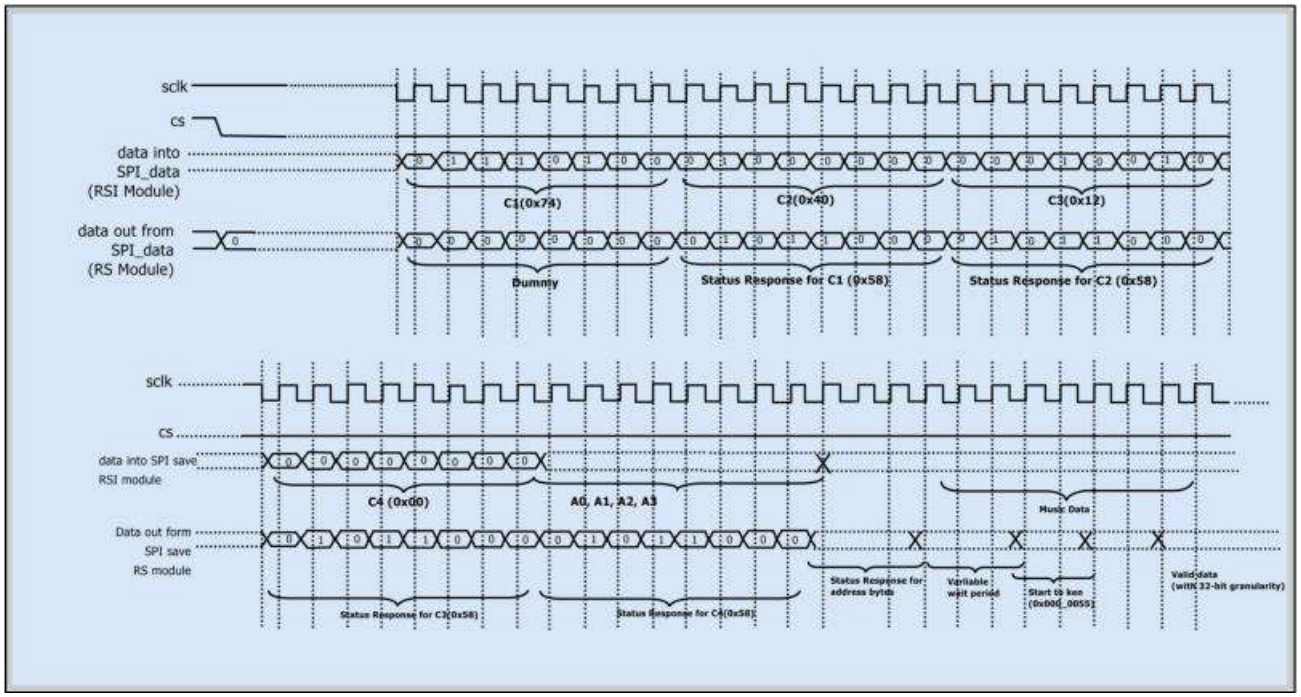


Figure 28: Memory Read at Physical Interface

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 RYWB116.
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]

2.15.4.4 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.

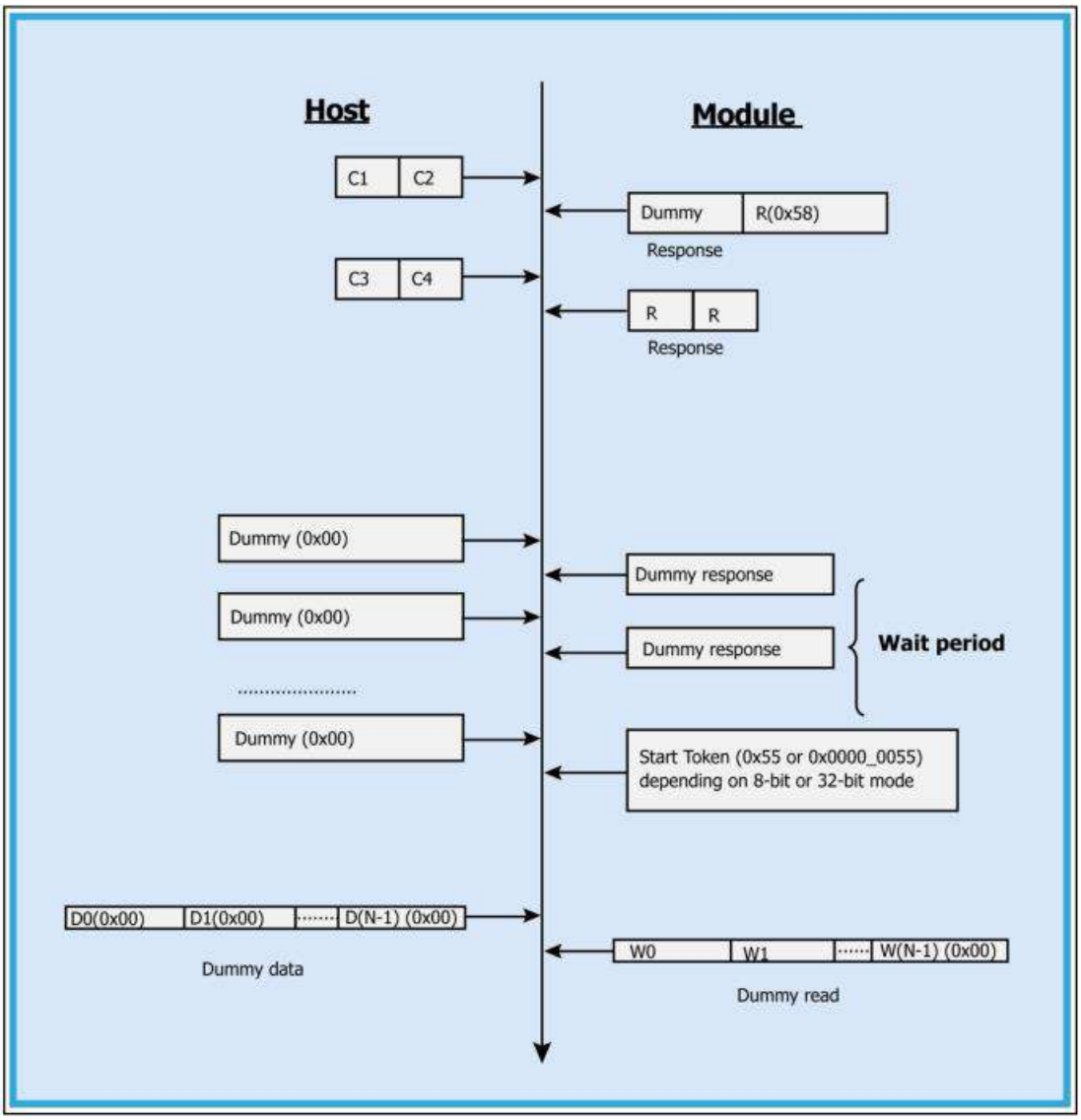


Figure 29: Frame Read

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 RYWB116.
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]

2.15.4.5 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:

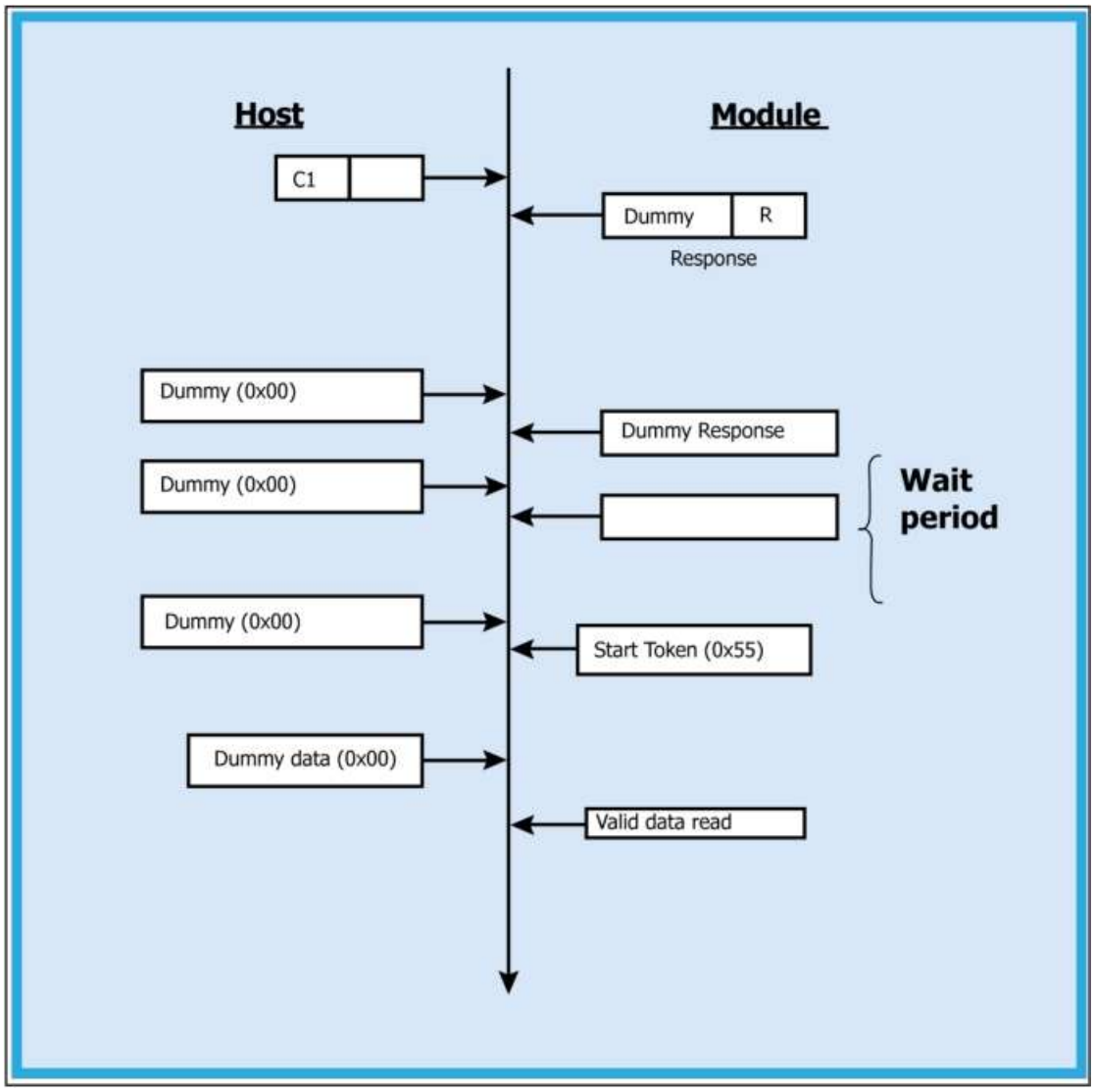


Figure 30: Register Read

2.15.4.6 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:

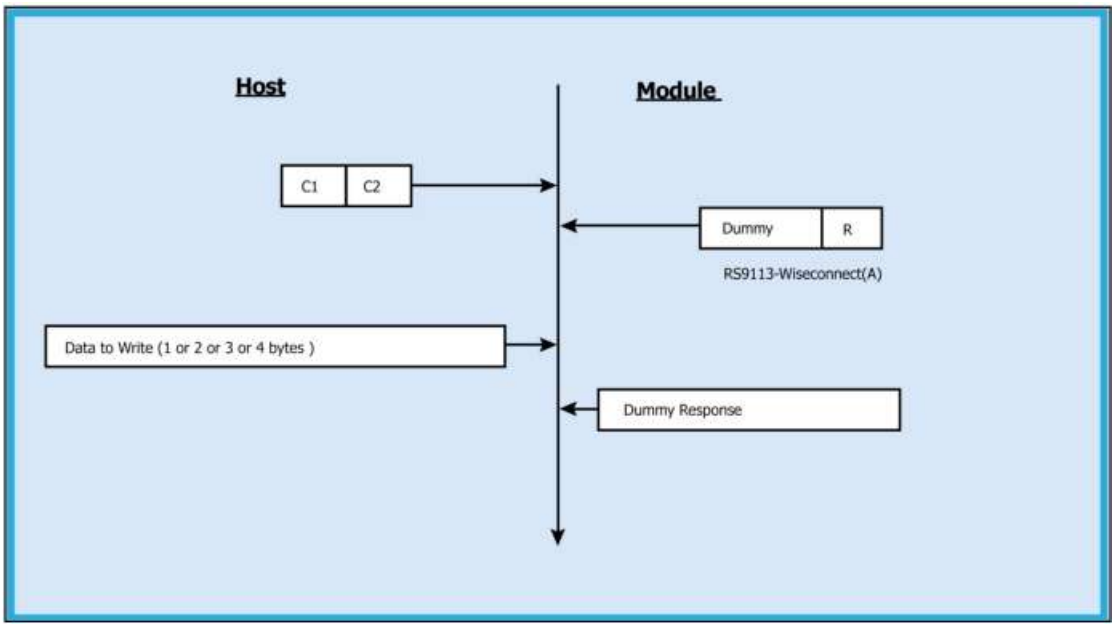


Figure 31: Register Write

2.15.5 Register Summary

Register	Address
SPI_HOST_INTR	0x00

Table 6: RYWB116 Module Register Description

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

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 7: SPI Host Interrupt Register

2.16 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.

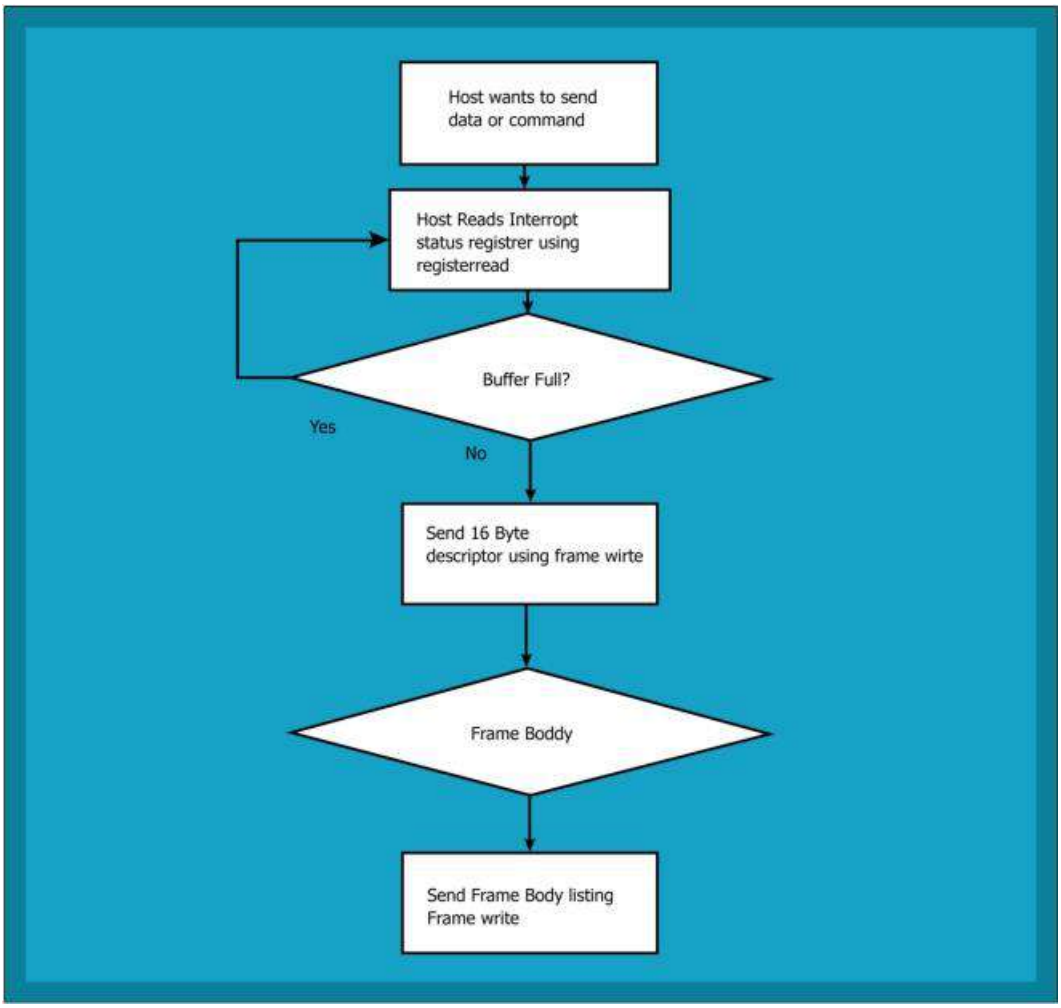


Figure 32: Tx From Host to Module

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

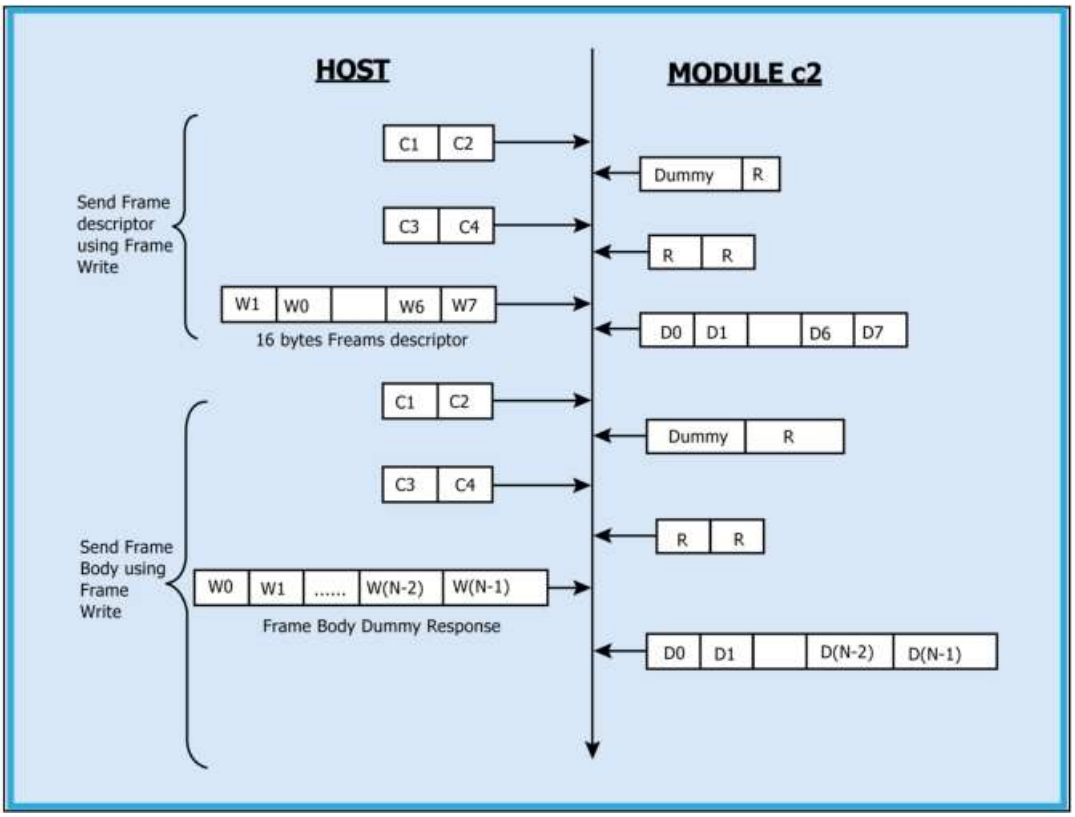


Figure 33: Exchanges between Host and Module for Tx operation

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:

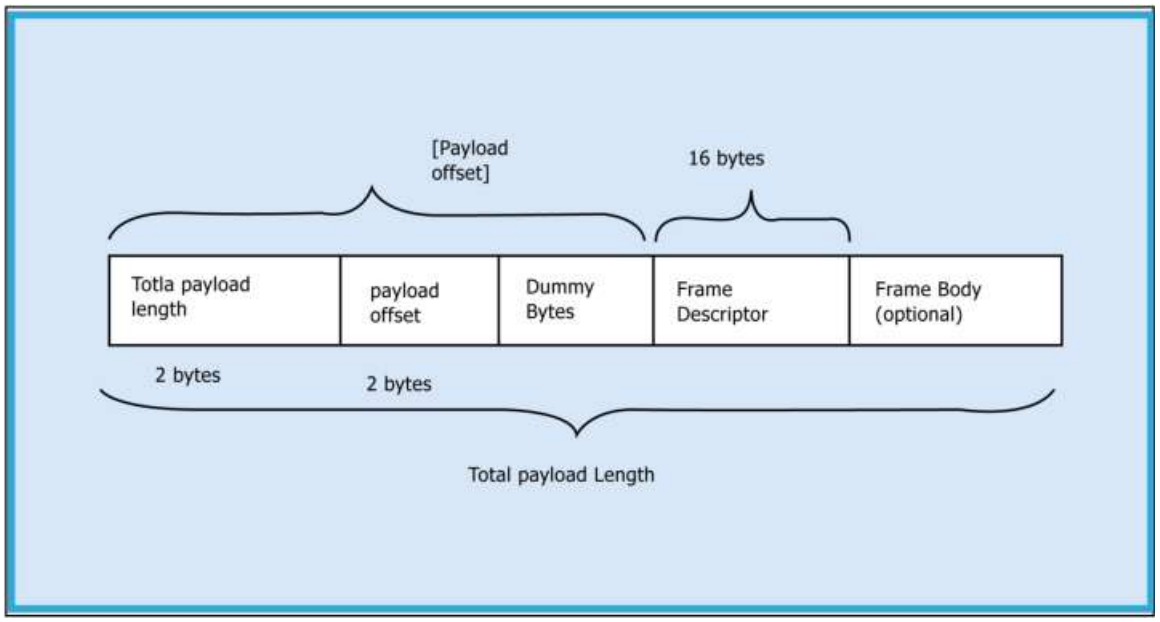


Figure 34: RX Frame Format

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.

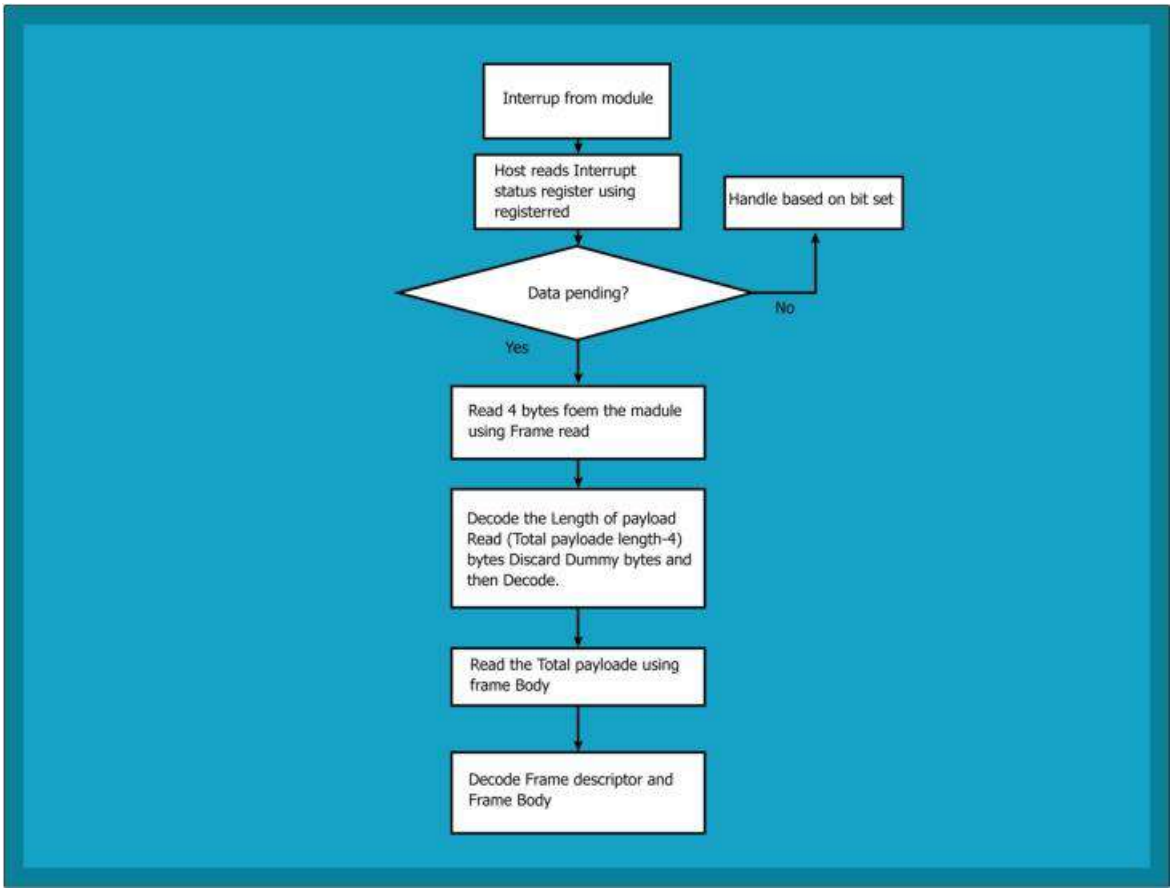


Figure 35: RX Operation from Module to Host

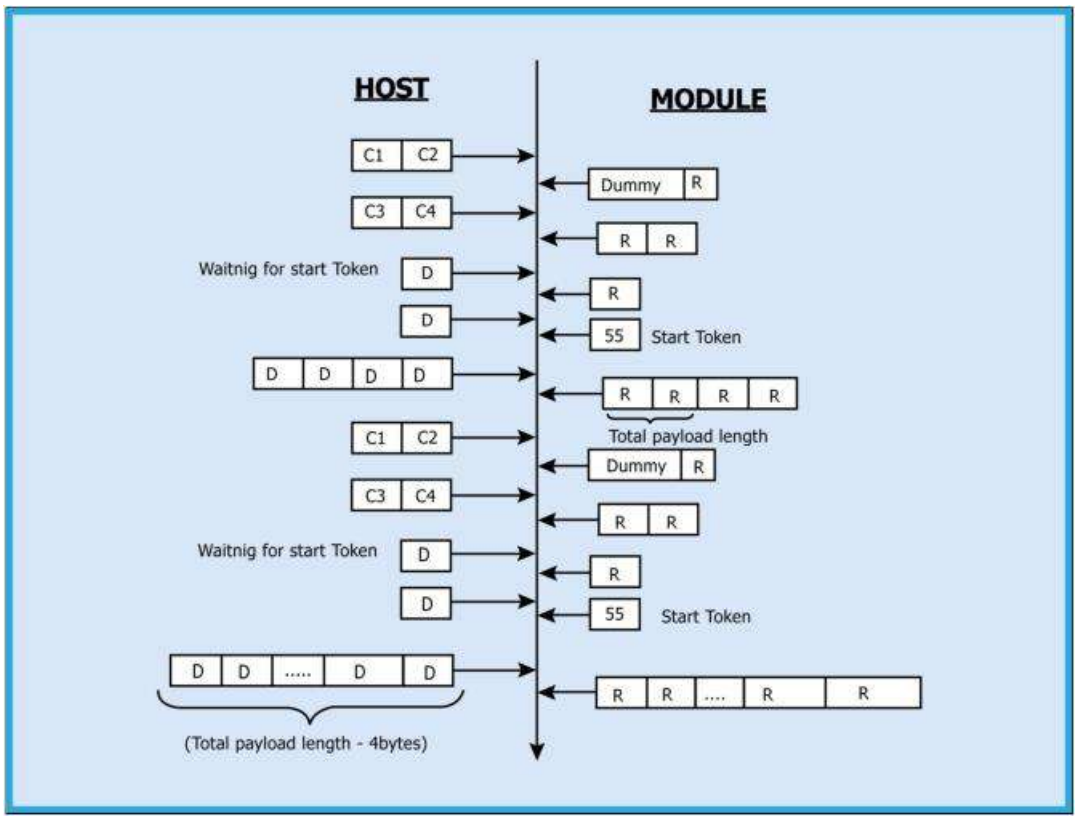


Figure 36: Message exchanges between Host and Module for Rx operation

2.17 Commands

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

The UART on the RYWB116 is used as a host interface to configure the module to send data and also to receive data.

2.18 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.

2.19 Hardware Interface

The UART interface on the RYWB116 transmits / receives data to / from the Host in UART mode. The RYWB116 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

2.20 Software Protocol

2.20.1 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 RYWB116 responds with either an 'OK' or 'ERROR' string, for Management or Data frames along with a result or error code.

2.20.2 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 RYWB116 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 informations from the RYWB116

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

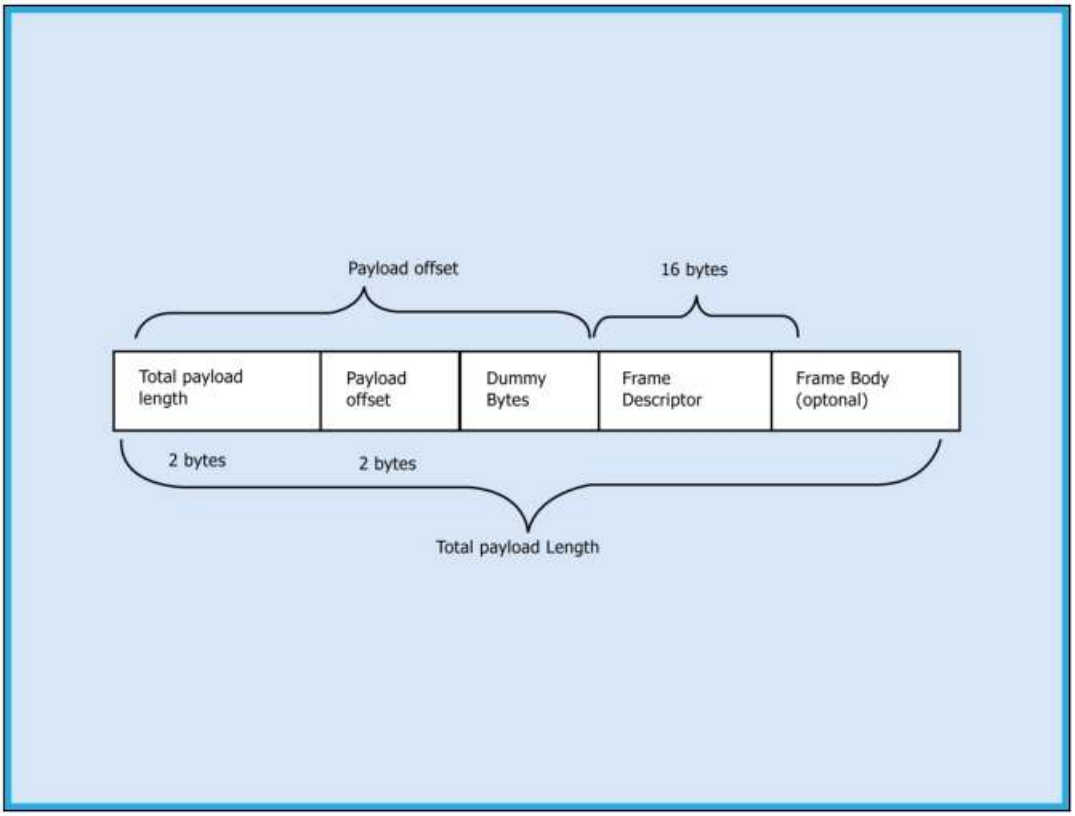


Figure 37: RX Frame format

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.

2.21 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.

2.22 USB Interface

RYWB116 supports USB interface which allows the host to configure and send / receive data through the module via USB.

2.23 Features

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

2.24 Hardware Interface

USB mode uses the standard USB 2.0 interface.

2.25 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.

RYWB116 supports two modes by using USB interface.

- USB mode
- USB CDC mode

2.25.1 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.
RYWB116 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:

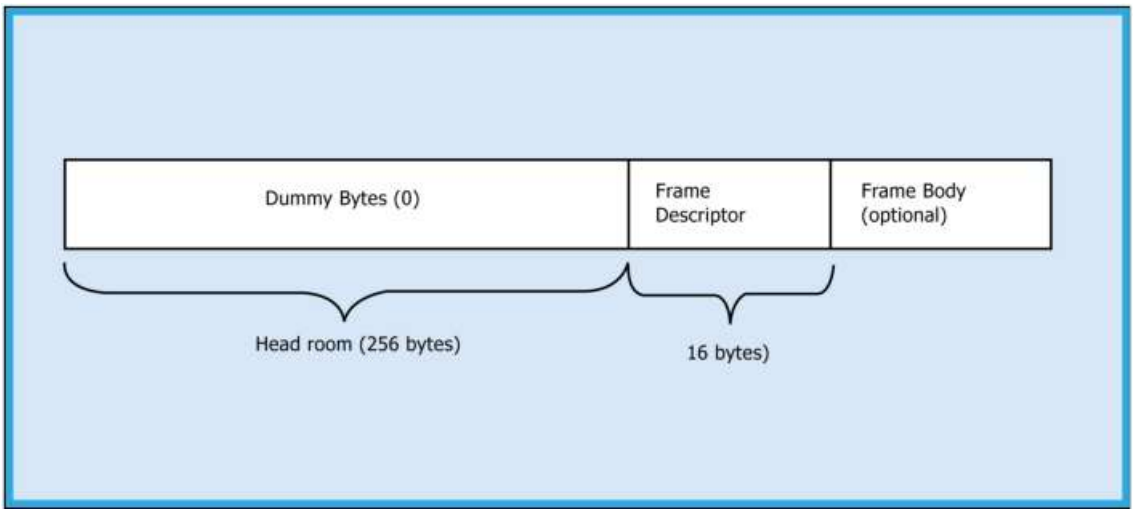


Figure 38: Command/Data Packet format from host to module in USB mode

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.

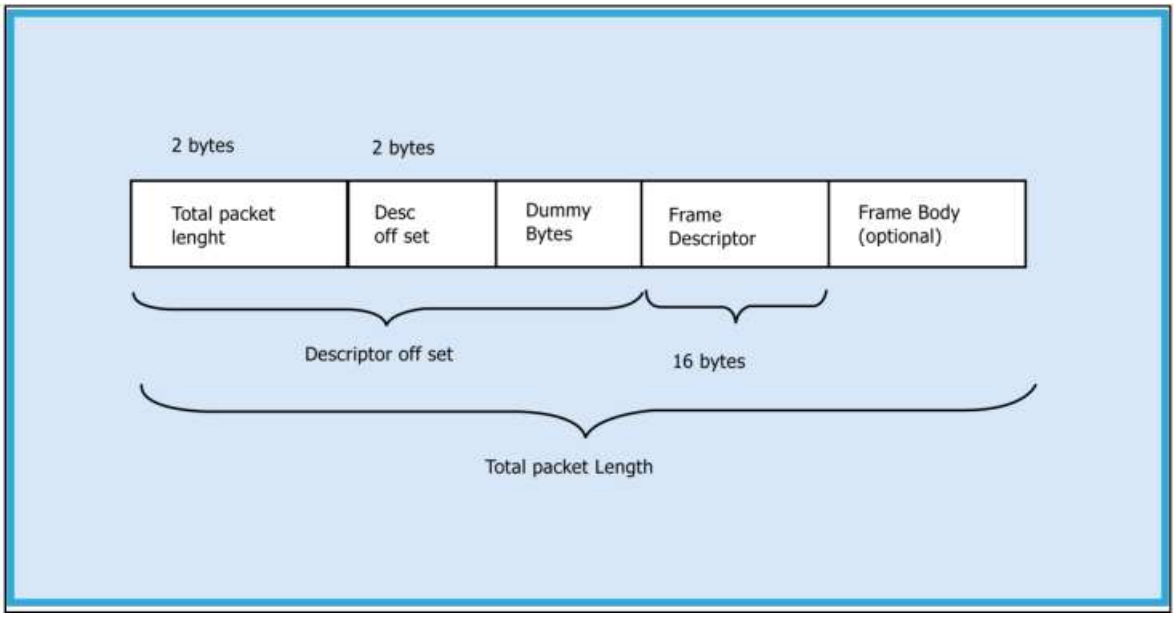


Figure 39: Command/Data Packet format from module to host in USB mode

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 54: 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.

2.25.2 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"**.

2.26 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**.

3 Embedded BT Classic 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.

4 Embedded BT Classic Command Format

This section explains the general command format. The commands should be sent to the Module in the specified format.
The commands are sent to the module and the responses are read from the module using frame write/frame read (as mentioned in the preceding sections). 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)

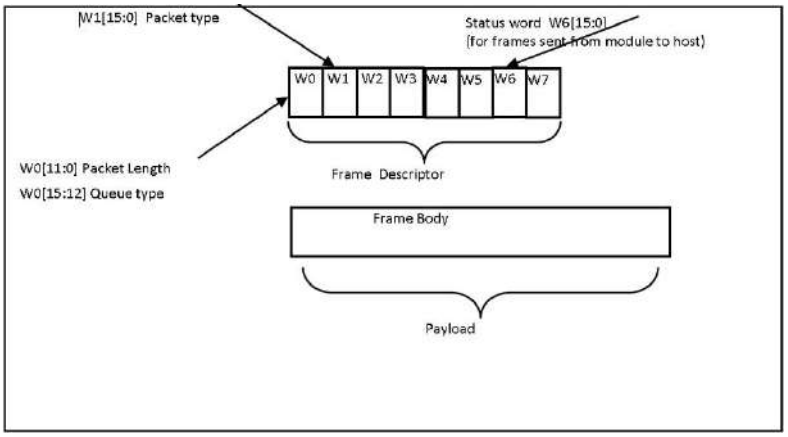


Figure 40: Command frame format

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

Command	Command ID
Set Local Name	0x0001
Query Local Name	0x0002
Set Local COD	0x0003
Query Local COD	0x0004
Query RSSI	0x0005
Query Link Quality	0x0006
Query Local BD Address	0x0007
Set Profile Mode	0x0008
Set Device Discover Mode	0x0009
Get Device Discover Mode	0x000A
Set Connection mode	0x000B
Get Connection mode	0x000C
Set Pair mode	0x000D
Get Pair mode	0x000E
Remote Name Request	0x000F
Remote Name Request Cancel	0x0010
Inquiry	0x0011
Inquiry Cancel	0x0012
Bond or Create Connection	0x0013
Bond Cancel or Create Connection Cancel	0x0014
Unbond or Disconnect	0x0015
Set Pin Type	0x0016
Get Pin Type	0x0017
User Confirmation	0x0018
Passkey Reply	0x0019
Pincode Reply	0x001A
Get Local Device Role	0x001B
Set Local Device Role	0x001C
Get Service List	0X001D

Command	Command ID
Search Service	0X001E
SPP connect	0X001F
SPP Disconnect	0X0020
SPP Transfer	0X0021
Initialize BLE module	0x008D
Deinitialize BLE module	0x008E
Antenna Select	0x008F
Linkkey Reply	0x0091
PER Transmit	0x0098
PER Receive	0x0099
PER Stats	0x009A
PER CW mode	0x009B
Sniff Mode	0x009D
Sniff Exit	0x009E
Sniff Subrating	0x009F
Feature Bit map	0x00A6
Set antenna Tx power level	0x00A7
AFH channel Classification	0x00D2
Set SSP mode	0x00A0
Set EIR data	0X00A9
A2DP connect	0x0022
A2DP disconnect	0x0023
A2DP Start	0x00CE
A2DP Suspend	0x00CF
A2DP PCM data request	0x00D0
A2DP SBC data request	0x00D1
AVRCP Connect	0X0024
AVRCP Disconnect	0X0025
AVRCP Play	0X0026
AVRCP Pause	0X0027
AVRCP Stop	0X0028
AVRCP Next	0X0029
AVRCP Previous	0X002A
HFP Connect	0x002D
HFP Disconnect	0x002E
HFP Phone operator	0x002F
HFP Call accept	0x0030
HFP Call reject	0x0031
HFP Dial number	0x0032
HFP Dial member	0x0033
HFP Redial	0x0034

Command	Command ID
HFP Voice Recognition active	0x0035
HFP Voice Recognition deactive	0x0036
HFP Speak gain	0x0037
HFP Mic gain	0x0038
HFP Get calls	0x0039
HFP Audio	0x003A
PBAP Connect	0x003B
PBAP Disconnect	0x003C
PBAP Contacts	0x003D
Set AFH Channel Classification	0x00D2
AVRCP Get Capabilities	0x00D3
AVRCP Get Attributes List	0x00D4
AVRCP Get Attributes Values List	0x00D5
AVRCP Get Current Attribute Value	0x00D6
AVRCP Set Current Attribute Value	0x00D7
AVRCP Get Element Attributes	0x00D8
AVRCP Get Play Status	0x00D9
AVRCP Get Register Notification	0x00DA
AVRCP Get Version	0x00DB
AVRCP Get Attribute Text	0x00DC
AVRCP Get Attribute Value Text	0x00DD
AVRCP Get Battery Status	0x00DE
AVRCP Get Character Sets	0x00DF
AVRCP Capabilities Response	0x00E0
AVRCP Attributes List Response	0x00E1
AVRCP Attributes Values List Response	0x00E2
AVRCP Get Current Attributes Values List	0x00E3
AVRCP Set Current Attributes Values List	0x00E4
AVRCP Get Element Attributes Response	0x00E5
AVRCP Get Play Status Response	0x00E6
AVRCP Get Register Notification Response	0x00E7
AVRCP Get Attribute Text Response	0x00E8
AVRCP Get Attribute Value Text Response	0x00E9
AVRCP Get Battery Status Response	0x00EA
AVRCP Get Character Sets Response	0x00EB
AVRCP Notficiation	0x00EC
AVRCP CMD Reject	0x00ED
Add Device ID	0x00EE
A2DP Get Config	0x00FE
A2DP Set Config	0x00FF

Table 9: Command IDs in BT Classic mode

Response	Response ID
Set Local Name	0x0001
Query Local Name	0x0002
Set Local COD	0x0003
Query Local COD	0x0004
Query RSSI	0x0005
Query Link Quality	0x0006
Query Local BD Address	0x0007
Set Profile Mode	0x0008
Set Device Discover Mode	0x0009
Get Device Discover Mode	0x000A
Set Connection Mode	0x000B
Get Connection Mode	0x000C
Set Pair Mode	0x000D
Get Pair Mode	0x000E
Remote Name Request	0x000F
Remote Name Request Cancel	0x0010
Inquiry	0x0011
Inquiry Cancel	0x0012
Bond or Create Connection	0x0013
Bond Cancel or Create Connection Cancel	0x0014
Unbond or Disconnect	0x0015
Set Pin Type	0x0016
Get Pin Type	0x0017
User Confirmation	0x0018
Passkey Reply	0x0019
Pincode Reply	0x001A
Get Local Device Role	0x001B
Set Local Device Role	0x001C
Get Service List	0X001D
Search Service	0X001E
SPP connect	0X001F
SPP Disconnect	0X0020
SPP Transfer	0X0021
Initialize BT Module	0x008D
Deinitialize BT Module	0x008E
Antenna Select	0x008F
Linkkey Reply	0x0091
PER Transmit	0x0098
PER Receive	0x0099

Response	Response ID
PER Stats	0x009A
PER CW mode	0x009B
Sniff Mode	0x009D
Sniff Exit	0x009E
Sniff Subrating	0x009F
Feature Bit map	0x00A6
Set Antenna Tx Power Level	0x00A7
AFH channel Classification	0x00D2
Set SSP mode	0x00A0
Set EIR data	0X00A9
A2DP Connect	0x0022
A2DP Disconnect	0x0023
A2DP Start	0x00CE
A2DP Suspend	0x00CF
A2DP PCM Data	0x00D0
A2DP SBC Data	0x00D1
AVRCP Connect	0X0024
AVRCP Disconnect	0X0025
AVRCP Play	0X0026
AVRCP Pause	0X0027
AVRCP Stop	0X0028
AVRCP Next	0X0029
AVRCP Previous	0X002A
HFP Connect	0x002D
HFP Disconnect	0x002E
HFP Phone operator	0x002F
HFP Call accept	0x0030
HFP Call reject	0x0031
HFP Dial number	0x0032
HFP Dial member	0x0033
HFP Redial	0x0034
HFP Voice Recognition Active	0x0035
HFP Voice Recognition Deactive	0x0036
HFP Speak Gain	0x0037
HFP Mic Gain	0x0038
HFP Get Calls	0x0039
HFP Audio	0x003A
PBAP Connect	0x003B
PBAP Disconnect	0x003C
PBAP Contacts	0x003D

Response	Response ID
Set AFH Channel Classification	0x00D2
AVRCP Get Capabilities	0x00D3
AVRCP Get Attributes List	0x00D4
AVRCP Get Attributes Values List	0x00D5
AVRCP Get Current Attribute Value	0x00D6
AVRCP Set Current Attribute Value	0x00D7
AVRCP Get Element Attributes	0x00D8
AVRCP Get Play Status	0x00D9
AVRCP Get Register Notification	0x00DA
AVRCP Get Version	0x00DB
AVRCP Get Attribute Text	0x00DC
AVRCP Get Attribute Value Text	0x00DD
AVRCP Get Battery Status	0x00DE
AVRCP Get Character Sets	0x00DF
AVRCP Capabilities Response	0x00E0
AVRCP Attributes List Response	0x00E1
AVRCP Attributes Values List Response	0x00E2
AVRCP Get Current Attributes Values List	0x00E3
AVRCP Set Current Attributes Values List	0x00E4
AVRCP Get Element Attributes Response	0x00E5
AVRCP Get Play Status Response	0x00E6
AVRCP Get Register Notification Response	0x00E7
AVRCP Get Attribute Text Response	0x00E8
AVRCP Get Attribute Value Text Response	0x00E9
AVRCP Get Battery Status Response	0x00EA
AVRCP Get Character Sets Response	0x00EB
AVRCP Notficiation	0x00EC
AVRCP CMD Reject	0x00ED
Add Device ID	0x00EE
A2DP Get Config	0x00FE
A2DP Set Config	0x00FF

Table 10: Response IDs in BT Classic mode

Event	Event ID
Role Change Status	0x1000
Unbond or Disconnect	0x1001
Bond Response	0x1002
Inquiry response	0x1003
Remote Device Name	0x1004

Event	Event ID
Remote Name Request cancelled	0x1005
Disconnected	0x1006
User Confirmation Request	0x1007
User Passkey Display	0x1008
User Pincode Request	0x1009
User Passkey Request	0x100A
Inquiry Complete	0x100B
Authentication Complete	0x100C
User Linkkey Request	0x100D
User Linkkey Save	0x100E
SSP Complete	0x100F
BT Mode Changed	0x1010
BT Sniff Subrating Changed	0x1011
BT User Passkey Notify	0x1012
SPP Receive Data	0x1100
SPP Connected	0x1101
SPP Disconnected	0x1102
A2DP Connected	0x1200
A2DP Disconnected	0x1201
A2DP Configured	0x1202
A2DP Open	0x1203
A2DP Start	0x1204
A2DP Suspend	0x1205
A2DP Abort	0x1206
A2DP Close	0x1207
A2DP Encoded data	0x1208
A2DP PCM data	0x1209
A2DP More data request	0x120A
A2DP Codec Config	0x120B
AVRCP Connected	0x1300
AVRCP Disconnected	0x1301
AVRCP Play	0x1302
AVRCP Pause	0x1303
AVRCP Next	0x1304
AVRCP Previous	0x1305
AVRCP Stop	0x1306
AVRCP Notify	0x1310
AVRCP Capabilities Request	0x1311
AVRCP Attributes List Request	0x1312
AVRCP Values List Request	0x1313

Event	Event ID
AVRCP Current Attribute Value Request	0x1314
AVRCP Set Attribute Value Request	0x1315
AVRCP Attribute Text Request	0x1316
AVRCP Value Text Request	0x1317
AVRCP Character Set Request	0x1318
AVRCP Battery Status Request	0x1319
AVRCP Element Attribute Request	0x131A
AVRCP Player Status Request	0x131B
AVRCP Register Notification	0x131C
HFP Connected	0x1400
HFP Disconnected	0x1401
HFP Ring	0x1402
HFP Call caller id	0x1403
HFP Audio Connected	0x1404
HFP Audio Disonnected	0x1405
HFP Dial complete	0x1406
HFP answer complete	0x1407
HFP Hang up complete	0x1408
HFP Send DTMF Complete	0x1409
HFP Call wait	0x140A
HFP Voice recognition deactivated	0x140B
HFP Voice recognition activated	0x140C
HFP Service not found	0x140D
HFP Call status	0x140E
HFP Signal strength	0x140F
HFP Battery level	0x1410
HFP Phone service	0x1411
HFP Roaming status	0x1412
HFP Call setup	0x1413
HFP Call held status	0x1414
PBAP Connected	0x1450
PBAP Disconnected	0x1451
PBAP Data	0x1452

Table 11: Event IDs in BT Classic mode

5 Embedded BT Classic Commands

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

Note:
A command should not be issued by the Host before receiving the response of a previously issued command from the module.

5.1 Generic commands

5.1.1 Set Operating Mode

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

```
at+rsi_opermode=
<oper_mode>,<feature_bit_map>,<tcp_ip_feature_bit_map>,<custom_feature_bit_map>,<ext_custom_feature_bit_map
>,<bt_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))

Note:
Please refer to RYWB116_Embedded_WLAN_Software_Programming_Reference_Manual.pdf for more details on WLAN and co-existence of other protocols with WLAN. In BTLE mode, need to enable BT mode also.

Following table represents BT coex modes supported :

Coex_mode	Description
4	Bluetooth
8	Dual Mode (Bluetooth and BLE)

Table 12: Coex Modes of BT Supported

Note:
1. If coex mode enabled in opermode command, then BT / BLE protocol will start and give corresponding card ready in parallel with opermode command response (which will be handled by corresponding application).
2. BT card ready frame is described in RYWB116_Embedded_BT_Classic_Software_Programming_Reference_Manual.pdf, BLE card ready frame is described in RYWB116_Embedded_BLE_Software_Programming_Reference_Manual.pdf
3. Feature selection utility is provided in the package. RYWB116 supports the selected features combination only if it is feasible as per the RSXXXXX_TCPIP_Feature_Selection_vX.xlsx

custom_feature_bit_map:
This bitmap is used to enable following BT/BLE custom features:

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:
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.

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:**
> 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[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 RYWB116 product mode

1-enable
0-disable

bt_feature_bit_map:

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

BIT[0:14] – 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. For classic opermode we can ignore validating ble feature bit map in BIT(31)

1 - valid ble feature bit map
0 - Ignore ble feature bit map

Response:

AT Mode:

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

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

Example:

AT Mode:

```
at+rsi_opermode=327680,0,1,2147483648,2149580800,1073741824\r\n
```

Response:

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

5.1.2 Set Local name

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

Binary Payload Structure:

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

AT command format:

```
at+rsibt_setlocalname=<NameLength>,<Name>\r\n
```

Parameters:
NameLength – Length of the name of the local device.
Name – Name of the local device.

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

AT command Ex:

```
at+rsibt_setlocalname=8,reyax\r\n
```

Response:

```
OK\r\n
```

5.1.3 Query Local name

Description:

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

Binary Payload Structure:

No Payload required.

AT command format:

```
at+rsibt_getlocalname?\r\n
```

Response Payload:

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

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

Response Parameters:
NameLength – Length of the name of the local device.
Name – Name of the local device.

AT command Ex:

```
at+rsibt_getlocalname?\r\n
```

Response:

OK 8,reyax\r\n

5.1.4 Set Local COD

Description:

This is used to indicate the capabilities of the local device to other devices. It is a parameter received during the device discovery procedure on the BR/EDR physical transport, indicating the type of device. The Class of Device parameter is only used on BR/EDR and BR/EDR/LE devices using the BR/EDR physical transport.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_local_cod {
UINT32  LocalCOD;
}  RSI_BT_CMD_SET_LOCAL_COD;
```

AT command format:

```
at+rsibt_setlocalcod=<local_device_class>\r\n
```

Parameters:
Local COD – Class of the Device of the local device

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

AT command Ex:

```
at+rsibt_setlocalcod=7A020C\r\n
```

Response:

```
OK\r\n
```

5.1.5 Query Local COD

Description:

This is used to query Class of Device of the local device.

Binary Payload Structure:

No Payload required.

AT command format:

```
at+rsibt_getlocalcod?\r\n
```

Response Payload:

```
typedef struct rsi_bt_resp_query_local_cod {
UINT32  LocalCOD;
}  RSI_BT_RESP_QUERY_LOCAL_COD;
```

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

Response Parameters:
LocalCOD – Class of the Device of the local device
AT command Ex:

```
at+rsibt_getlocalcod?\r\n
```

Response:

```
OK  7A020C\r\n
```

5.1.6 Query RSSI

Description:

This is used to query RSSI of the connected remote BT Device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_query_rssi {
UINT08 BDAddress[6];
}  RSI_BT_CMD_QUERY_RSSI;
```

AT command format:

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

Parameters:
BDAddress – BD 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 230\r\n
```

5.1.7 Query Link Quality This command is not currently supported.

Description:
This is used to query the link quality between the local device and the connected remote device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_query_link_quality {
UINT08 BDAddress[6];
} RSI_BT_CMD_QUERY_LINK_QUALITY;
```

AT command format:

```
at+rsibt_getlinkqlty=<BDAddress>?\r\n
```

Parameters:
BDAddress – BD Address of the connected remote device

Response Payload:

```
typedef struct rsi_bt_resp_query_link_quality {
UINT08 LinkQuality;
} RSI_BT_RESP_QUERY_LINK_QUALITY;
```

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

Response parameters:
LinkQuality – Link quality value.

AT command Ex:

```
at+rsibt_getlinkqlty=AA-BB-CC-DD-EE-FF?\r\n
```

Response:

```
OK 123\r\n
```

5.1.8 Query Local BD Address

Description:

This is used to query the BD address of the local device.

Binary 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[6];
} 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 - BD Address of the local device
AT command Ex:

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

Response:

```
OK AA-BB-CC-DD-EE-FF\r\n
```

5.1.9 Initialize BT module

Description:
This is used to initialize the BT module.
Binary Payload Structure:
No Payload required
AT command format:.

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

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

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

Response:

```
OK\r\n
```

5.1.10 Deinitialize BT module

Description:

This is used to de-initialize the BT module.To again initialize the module **Initialize BT module** command is used.

Binary Payload Structure:

No Payload required

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
```

5.1.11 BT Antenna Select

Description:

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

Binary Payload Structure:

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

AT command format:

```
at+rsibt_btantennaselect=<antenna_val>\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
```

5.1.12 Set Feature Bitmap

Description:

This is used to enable/disable the features.

Binary Payload Structure:

```
typedef union
{
struct {
UINT32 Feature_BitMap;
} BitMapFrameSend;
UINT08 uFeatureBitMapBuf[2];
}  RSI_BT_CMD_FEATURE_BIT_MAP
```

AT command format:

```
at+rsibt_setfeaturebitmap=<featurebitmap>\r\n
```

Parameters:
featurebitmap –
1 – Enable BT security
0 – Disable BT security

Response Payload:
There is no response payload for this command

ATcommandEx:

```
at+rsibt_setfeaturebitmap =1\r\
```

Response:

```
OK\r\n
```

5.1.13 Set Antenna Tx power level

Description:

This is used to set the Bluetooth antenna transmit power level.This command serves for selecting the maximum power to be used for the device.

Binary Payload Structure:

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

AT command format:

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

Parameters:
protocol_mode –
1 –BT Classic
Power_level - range of the power levels used, in terms of dbm
Minimum value – 1
Maximum value - 14

Response Payload:
There is no response payload for this command

ATcommandEx:

```
at+rsibt_setantennatxpowerlevel =1,10\r\n
```

Response:

```
OK\r\n
```

5.2 Core commands

5.2.1 Set Profile Mode Present only SPP profile is supported.

Description:

This is used to initialize the particular profiles in Bluetooth embedded host

stack.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_profile_mode{
UINT08 ProfileMode;
}RSI_BT_CMD_SET_PROFILE_MODE;
```

AT command format:

```
at+rsibt_setprofilemode=<ProfileMode>\r\n
```

Parameters:
Profile Mode – Set specific bits to enable the profiles.

Bit No	Description
0	SPP Profile
1	A2DP Profile
2	AVRCP Profile
3	HFP Profile
4	PBAP Profile
5	IAP Profile

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

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

Response:

```
OK\r\n
```

NOTE: According to profile requirements, need to give the bit numbers. For example if you required spp profile + A2DP Profile then u have to give value 3.

5.2.2 Set Device Discovery mode

Description:

This is used to set the BT module in any of the three Discovery modes. We have to use time out for only limited discovering.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_discv_mode {
UINT08  Mode;
UINT08 Reserved[3];
INT32 Timeout;
} RSI_BT_CMD_SET_DISCV_MODE;
```

AT command format:

```
at+rsibt_setdiscvmode=<mode>,<timeout>\r\n
```

Parameters:
Mode – To enable/disable discovering
0 – disable discovering
1 – enable discovering
2 – limited discovering
TimeOut – time out value in milli seconds.

Note: Better to use below 1 hour(i.e.. >3600000ms)

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

AT command Ex:

```
at+rsibt_setdiscvmode=2,10000\r\n
```

Response:

```
OK\r\n
```

5.2.3 Get Device Discovery mode

Description:

This is used to get the discovery mode of the BT module, currently the BT module was set.

Binary Payload Structure:
There is no payload for this command.

AT command format:

```
at+rsibt_getdiscvmode?\r\n
```

Response Payload:
typedef struct rsi_bt_resp_query_discovery_mode {
UINT08 DiscoveryMode;
} RSI_BT_RESP_QUERY_DISCOVERY_MODE;

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

Response Parameters:
DiscoveryMode – enabled/disabled discovering
0 – Disabled device discover
1 – Enabled device discover

AT command Ex:

```
at+rsibt_getdiscvmode?\r\n
```

Response:

```
OK 1\r\n
```

5.2.4 Set Connectability mode

Description:

This is used to set the BT module in one of the two Connectability modes.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_connection_mode {
UINT08  ConnMode;
}RSI_BT_CMD_SET_CONN_MODE;
```

AT command format:

```
at+rsibt_setconnmode=<ConnMode>\r\n
```

Parameters:
ConnMode – To enable/disable connectability
0 – disable connection mode
1 – enable connection mode

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

AT command Ex:

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

Response:

```
OK\r\n
```

5.2.5 Get Connectablility mode

Description:
This is used to get the connectable mode, currently the BT module was set.

Binary Payload Structure:
There is no payload for this command

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

Response Payload:
typedef struct rsi_bt_resp_query_conn_mode {
UINT08 ConnMode;
} RSI_BT_RESP_QUERY_CONN_MODE;

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

Response Parameters:
ConnMode – enabled/disabled connection mode
0 – Disabled connection mode
1 – Enabled connection mode

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

Response:

```
OK 1\r\n
```

5.2.6 Set Pair mode

Description:

This will enable or disable the Pairing mode of the BT module.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_pair_mode {
UINT08  PairMode;
}RSI_BT_CMD_SET_PAIR_MODE;
```

AT command format:
at+rsibt_setpairmode=<PairMode>\r\n

Parameters:
PairMode – To enable/disable Authentication
0 – disable Authentication mode
1 – enable Authentication mode

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

AT command Ex:

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

Response:

```
OK\r\n
```

5.2.7 Get Pair mode

Description:

This will retrieve the current pairing mode of the BT module.

Binary Payload Structure:

There is no payload for this command.

AT command format:

```
at+rsibt_getpairmode?\r\n
```

Response Payload:

```
typedef struct
rsi_bt_resp_query_pair_mode {
UINT08  PairMode;
}  RSI_BT_RESP_QUERY_PAIR_MODE;
```

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

Response Parameters:
PairMode – enabled/disabled Authentication mode
0 – Disabled Authentication mode
1 – Enabled Authentication mode

AT command Ex:

```
at+rsibt getpairmode?\r\n
```

Response:

```
OK 0\r\n
```

5.2.8 Remote Name Request

Description:

This is used to know the name of the remote BT device, using its BD address. The response to this command containing the remote BT device name will be sent to the host through **"RMTDEVNAME"** event.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_remote_name_req {
UINT08 BDAddress[6];
}RSI_BT_CMD_REMOTE_NAME_REQUEST;
```

AT command format:

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

Parameters:
BDAddress – remote device BD Address

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

AT command Ex:

```
at+rsibt_rmtnamereq= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.2.9 Remote Name Request Cancel

Description:

This will cancel the request served by "Remote Name Request" command. The cancellation will be confirmed through "Remote Name Request Cancelled" event.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_remote_name_req_cancel {
UINT08 BDAddress[6];
}RSI_BT_CMD_REMOTE_NAME_REQUEST_CANCEL;
```

AT command format:

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

Parameters:

BDAddress – remote device BD Address

Response Payload:

There is no response payload for this command.

AT command Ex:

```
at+rsibt_rmtnamereqcancel= AA-BB-CC-DD-EE-FF \r\n
```

Response:

```
OK\r\n
```

5.2.10 Inquiry

Description:

This will perform an inquiry scan to find any BT devices in the vicinity. The response is sent using **"INQRESP"** event.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_inquiry {
UINT08 InquiryType;
UINT08 Reserved[3];
UINT32 Duration;
UINT08  MaxNbrdev;
UINT08 Reserved[3];
}  RSI_BT_CMD_INQUIRY;
```

AT command format:

```
at+rsibt_inquiry=<InquiryType>,<Duration>,<MaxNbrdev>\r\n
```

Parameters:
InquiryType –
0 -Standard Inquiry
1 - Inquiry with RSSI
2 - Extended Inquiry
Duration – Extended Time in milliseconds (up to 10000ms)
MaxNbrdev – maximum number of devices to scan (from 1 to 10)

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

AT command Ex:

```
at+rsibt_inquiry=1,10000,10\r\n
```

Response:

```
OK\r\n
```

5.2.11 Inquiry Cancel

Description:
This will cancel the inquiry scan which was already in the process, served by **"Inquiry"** command.

Binary Payload Structure:
There is no payload Structure for this command.

AT command format:

```
at+rsibt_inquirycancel\r\n
```

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

AT command Ex:

```
at+rsibt_inquirycancel\r\n
```

Response:

```
OK\r\n
```

5.2.12 Extended Inquiry Response Data

Description:

This command is used to set the Extended Inquiry Response data.

Payload Structure:

```
typedef union {
struct {
UINT08  FECRequired;
UINT08 DataLength;
UINT08 EIRData[200];
} EIRDatasnd;
UINT08 uEIRDataBuf[2];
}  RSI_BT_CMD_SET_EIR_DATA;
```

AT Command format:

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

Parameters:

Length – data length. Max EIR data length is 200 Bytes.
Data – Actual data

Response Payload:

There is no response payload for this command.
Note:
Data should be in hex format

AT command Ex:

```
at+rsibt_seteir=8,2,1,0,4,9,72,72,72\r\n
```

Response:

```
OK\r\n
```

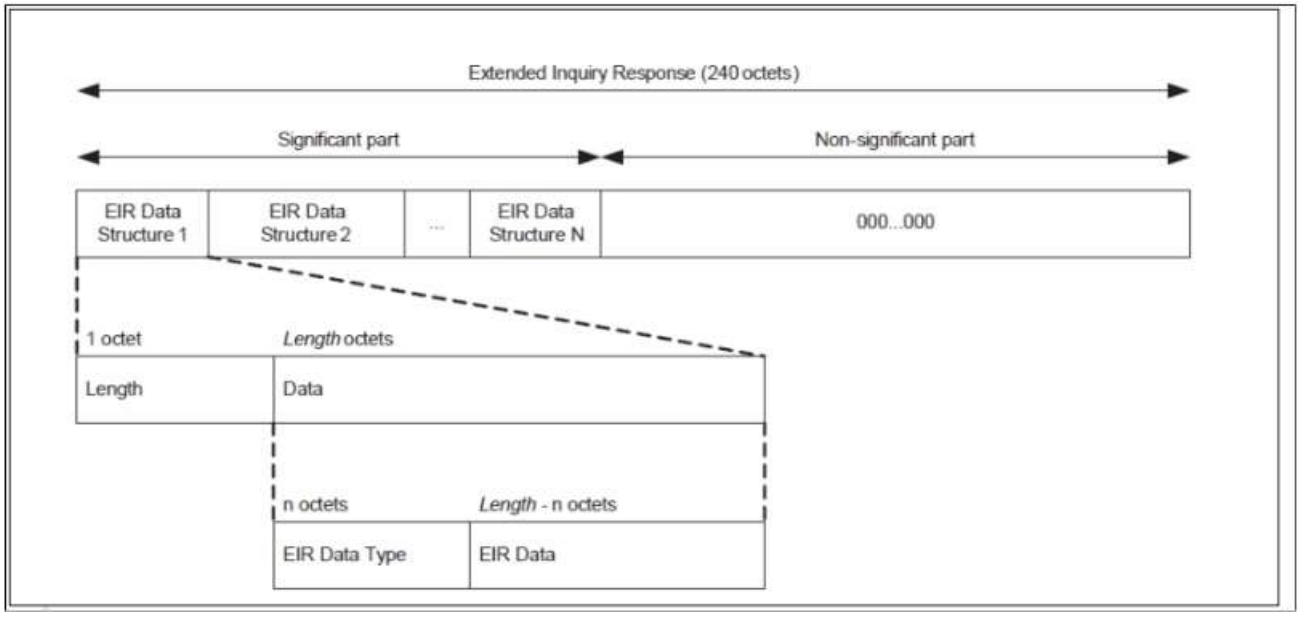


Figure 41: Extented inquiry response

5.2.13 Bond or Create Connection

Description:

This will creates bonding (connection) between the BT module and the remote BT device based on BD address along with security.

Binary Payload Structure:

AT command format:

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

Parameters:
BDAddress – remote device BD Address.

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

AT command Ex:

```
at+rsibt_bond= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.2.14 Bond Cancel or Create Connection Cancel

Description:

This will disconnect the connection between the BT module and the remote BT device, only while the bonding is in progress.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_bond_cancel {
UINT08 BDAddress[6];
}RSI_BT_CMD_BOND_CANCEL;
```

AT command format:

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

Parameters:

BDAddress – remote device BD Address

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

AT command Ex:

```
at+rsibt_bondcancel = AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.2.15 UnBond or Disconnect

Description:

This un-bonds the device, which was already bonded, based on BD address of the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_unbond {
UINT08 BDAddress[6];
}RSI_BT_CMD_UNBOND;
```

AT command format:

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

Parameters:
BDAddress – remote device BD Address

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

AT command Ex

```
at+rsibt_unbond= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.2.16 Set Pin type This command is not currently supported.

Description:

This is used to set the PIN code or pass key of the local BT module.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_pin_type {
UINT08  PINType;
}RSI_BT_CMD_SET_PIN_TYPE;
```

AT command format:

```
at+rsibt_setpintype=<PINType>\r\n
```

Parameters:
PINType 0 – variable pin
1 – fixed pin
Response Payload:
There is no response payload for this command.
AT command Ex:

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

Response:

```
OK\r\n
```

5.2.17 Get Pin type This command is not currently supported.

Description:
This is used to get the PIN code or pass key of the local BT module.
Binary Payload Structure:
There is no response payload for this command.
AT command format:

```
at+rsibt_getpintype?\r\n
```

Response Payload:

```
typedef struct rsi_bt_resp_query_pin_type {
UINT08  PINType;
}  RSI_BT_RESP_QUERY_PIN_TYPE;
```

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

Response Parameters:
PINType –
0 – variable pin
1 – fixed pin
AT command Ex:

```
at+rsibt_getpintype?\r\n
```

Response:

```
OK 1\r\n
```

5.2.18 User confirmation

Description:

This will give the confirmation for the values sent by remote BT devices at the time of bonding.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_user_confirmation {
UINT08 BDAddress[6];
UINT08 Confirmation;
} RSI_BT_CMD_USER_CONFIRMATION;
```

AT command format:

```
at+rsibt_usrconfirmation=<BDAddress>,<Confirmation>\r\n
```

Parameters:
bd_addr: BD address of the remote BT device which send connection request.
confirmation:
0- NO. If both remote and local values are not same.
1- YES. If both remote and local values are same.

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

AT command Ex:

```
at+rsibt_usrconfirmation= AA-BB-CC-DD-EE-FF,1\r\n
```

Response:

```
OK\r\n
```

5.2.19 Pass key Request Reply

Description:
The user passkey entry is used to respond on a user passkey entry request (UPER).

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_passkey_reply {
UINT08 BDAddress[6];
UINT08  ReplyType;
UINT08 Reserved;
UINT32 Passkey;
} RSI_BT_CMD_PASSKEY_REPLY;
```

AT command format:

```
at+rsibt_usrpasskey=<BDAddress>,<ReplyType>,<Passkey>\r\n
```

Parameters:
BDAddress – Remote BD Address.
ReplyType –
0 – negitive reply
1 – positive reply
Passkey – Entered Passkey number in decimanl (range from 0 to 999999).

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

AT command Ex:

```
at+rsibt_usrpasskey= AA-BB-CC-DD-EE-FF,1,123456\r\n
```

Response:

```
OK\r\n
```

5.2.20 Pincode Request Reply

Description:

The user pincode entry is used to respond on a user pin code entry request (UPER). If we want to connect with remote device then we can respond with positive reply. Else send negative reply.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_pincode_reply {
UINT08 BDAddress[6];
UINT08  ReplyType;
UINT08 Reserved;
UINT08  Pincode[MAX_PINCODE_REPLY_SIZE];
}  RSI_BT_CMD_PINCODE_REPLY;
```

AT command format:

```
at+rsibt_usrpincode=<BDAddress>,<ReplyType>,<Pincode>\r\n
```

Parameters:
BDAddress – Remote BD Address.
ReplyType –
0 – negative reply
1 – positive reply
Reserved - Padding
Pincode – Entered Pincode number(must be in string format max string length is 16 bytes).

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

AT command Ex:

```
at+rsibt_usrpincode= AA-BB-CC-DD-EE-FF,1,1234\r\n
```

Response:

```
OK\r\n
```

5.2.21 Get Local Device Role

Description:

This gets the role of the local BT module when connected with a particular remote BT device, based on BD address of the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_query_role {
UINT08 BDAddress[6];
}  RSI_BT_CMD_QUERY_ROLE;
```

AT command format:

```
at+rsibt_getmasterslaverole=<BDAddress>?\r\n
```

Parameters:
BDAddress – Remote BD Address

Response Payload:
typedef struct rsi_bt_resp_query_role {
UINT08 Role;
} RSI_BT_RESP_QUERY_ROLE;

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

Response Parameters:

Role –
0 – Master role
1 – slave role

AT command Ex:

```
at+rsibt_getmasterslaverole= AA-BB-CC-DD-EE-FF?\r\n
```

Response:

```
OK 1\r\n
```

5.2.22 Set Local Device Role or switch the role

Description:

This is used to change the current role of the local BT module, with respect to the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_role {
UINT08 BDAddress[6];
UINT08  Role;
} RSI_BT_CMD_SET_ROLE;
```

AT command format:

```
at+rsibt_setmasterslaverole=<BDAddress>,<Role>\r\n
```

Parameters:
BDAddress – Remote BD Address
Role - 0 – Master role
1 – Slave role
Response Payload:
There is no response payload for this command.
AT command Ex:

```
at+rsibt _setmasterslaverole=AA-BB-CC-DD-EE-FF,1\r\n
```

Response:

```
OK\r\n
```

5.2.23 Get Service List

Description:

This is used to search for the services supported by the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_query_services {
UINT08 BDAddress[6];
} RSI_BT_CMD_QUERY_SERVICES;
```

AT command format:

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

Parameters:
BDAddress – Remote BD Address

Response Payload:

```
typedef struct rsi_bt_resp_query_services {
UINT08 NumberOfServices;
UINT08 Reserved[3];
UINT32 ServiceUUIDs[32];
} RSI_BT_RESP_QUERY_SERVICES;
```

Result Code	Description
OK <bd_addr>,<nbr_srvs_found>,<srv_uuid_1>,<srv_uuid_2>,…….	Command Success with valid response.
ERROR <Error_code>	Command Fail.

Parameters:
NumberOfServices – Number of services in the list
ServiceUUIDs – list of serivce UUID's
NOTE: it will Display only 32 bit UUID's

AT command Ex:

```
at+rsibt_getsrvs= AA-BB-CC-DD-EE-FF \r\n
```

Response:

```
OK\r\n
```

5.2.24 Search Service

Description:

This is used to find whether a particular service is supported by the remote BT device.

Binary Payload Structure:

```
typedef struct {
UINT08 BDAddress[6];
UINT08 Reserved[2];
UINT32  ServiceUUID;
} RSI_BT_CMD_SEARCH_SERVICE;
```

AT command format:
at+rsibt_searchsrv=<BDAddress>,<ServiceUUID>\r\n

Parameters:
BDAddress – Remote BD Address
ServiceUUID – 16 bit or 32 bit UUID.

Response Payload:

```
typedef struct {
UINT08 SearchStatus;
} RSI_BT_RESP_SEARCH_SERVICE;
```

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

Response parameters:
Search status: 1-Yes, 0-No

AT command Ex:

```
at+rsibt_searchsrv= AA-BB-CC-DD-EE-FF,1105\r\n
```

Response:

```
OK 1\r\n
```

5.2.25 Linkkey Reply

Description:

The link key reply is used to respond on a link key request event. If we have previous link key of connecting device then we can respond with positive reply. Else send negative reply.

Binary Payload Structure:

```
typedef struct {
UINT08 BDAddress[6];
UINT08  ReplyType;
UINT08 Reserved;
UINT08 LinkKey[16];
} RSI_BT_CMD_LINKKEY_REPLY;
```

AT command format:

```
at+rsibt_usrlinkkey=<BDAddress>,<ReplyType>,<LinkKey>\r\n
```

Parameters:
BDAddress – Remote BD Address.
ReplyType –
0 – negitive reply
1 – positive reply
LinkKey – Link key saved for the remote BD address in host.

Response Payload:
There is no response payload for this command

ATcommandEx:

```
at+rsibt_usrlinkkey=AA-BB-CC-DD-EE-FF,1,3C,A5,50,25,DC,D0,B0,AB,B7,C3,4F,4D,9,79,2C,5C\r\n
```

Response:

```
OK\r\n
```

5.2.26 Set SSP mode

Description:

Set SSP mode is used to enable Simple Secure Pair mode and also used to select the IOCapability for SSP mode.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_ssp_mode {
UINT08  PairMode;
UINT08 IOCapability;
}  RSI_BT_CMD_SET_SSP_MODE;
```

AT command format:

```
at+rsibt_setsspmode=<PairMode>,<IOCapability>\r\n
```

Parameters:
PairMode–
0 – Disable
1 - Enable
IOCapability –
0x00 - DisplayOnly
0x01 - DisplayYesNo
0x02 - KeyboardOnly
0x03 - NoInputNoOutput
Response Payload:
There is no response payload for this command

ATcommandEx:

```
at+rsibt_setsspmode=1,1\r\n
```

Response:

```
OK\r\n
```

5.2.27 Sniff Mode

Description:

Enables the Host to support a low-power policy and allows the devices to enter Inquiry Scan, Page Scan, and a number of other possible actions.
The local device will return the actual sniff interval in the Interval parameter of the Mode Change event, if the command is successful.

Binary Payload Structure:

```
typedef struct {
UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
UINT16 SniffMaxIntr;
UINT16 SniffMinIntr;
UINT16 SniffAttempt;
UINT16 SniffTimeout;
}RSI_BT_CMD_SNIFF_MODE;
```

AT command format:

```
at+rsibt_sniffmode=<BDAddress>,<SniffMaxIntr>,<SniffMinIntr>,<SniffAttempt>,<sniffTimeout>\r\n
```

Parameters:
bd_addr- Remote BD Address.
SniffMaxIntr & SniffMinIntrv- The Sniff_Max_Interval and Sniff_Min_Interval command parameters are used to specify the requested acceptable maximum and minimum periods in the Sniff Mode.
SniffAttempt- Master shall poll the slave at least once in the sniff attempt transmit slots starting at each sniff anchor point.
SniffTimeout- Timeout after which device enter sniff subrating mode.

Response Payload:
There is no response payload for this command

ATcommandEx:

```
at+rsibt_sniffmode= AA-BB-CC-DD-EE-FF ,192,160,4,2\r\n
```

Response:

```
OK\r\n
```

5.2.28 Sniff Exit

Description:

To end the Sniff mode.

Binary Payload Structure:

```
typedef struct {
UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
}RSI_BT_CMD_SNIFF_EXIT;
```

AT command format:

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

Parameters:
BDAddress- Remote BD Address.

Response Payload:
There is no response payload for this command

ATcommandEx:

```
at+rsibt_sniffexit= AA-BB-CC-DD-EE-FF \r\n
```

Response:

OK\r\n

5.2.29 Sniff Subrating Currently, the sniff subrating command is not supported.

Description:

When the sniff mode timeout has expired a device shall enter sniff subrating mode. Sniff subrating mode allows a device to use a reduced number of sniff anchor points.

Binary Payload Structure:

```
typedef struct {
UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
UINT16  MaxLatency;
UINT16  MinRemoteTimeout;
UINT16 MinLocalTimeout;
}RSI_BT_CMD_SNIFF_SUBRATING;
```

AT command format:

```
at+rsibt_sniffsubrating=< BDAddress >,< MaxLatency>,<MinRemoteTimeout >,< MinLocalTimeout >\r\n
```

Parameters:
BDAddress- Remote BD Address.
maximum_latency- Maximum allowed sniff subrate of the remote device.
minimum_remote_timeout- Minimum base sniff subrate timeout that the remote device may use
minimum_local_timeout- Minimum base sniff subrate timeout that the local device may use.

Response Payload:
There is no response payload for this command

ATcommandEx:

```
at+rsibt_sniffsubrating= AA-BB-CC-DD-EE-FF , 192,1000,1000\r\n
```

Response:

```
OK\r\n
```

5.3 SPP commands

Note:
SPP profile will not connect without pair process or authentication.

5.3.1 SPP Connect

Description:

This is used to establish SPP connection with the remote BT device, specified by the BD address.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_spp_connect {
UINT08 BDAddress[6];
} RSI_BT_CMD_SPP_CONNECT;
```

AT command format:

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

Parameters:
BDAddress – Remote BD address.

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

AT command Ex:

```
at+rsibt_sppconn= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.3.2 SPP Disconnect

Description:
This is used to disconnect the SPP connection with the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_spp_disconnect {
UINT08 BDAddress[6];
} RSI_BT_CMD_SPP_DISCONNECT;
```

AT command format:

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

Parameters:
BDAddress – Remote BD address

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

AT command Ex:

```
at+rsibt_sppdisconn= AA-BB-CC-DD-EE-FF \r\n
```

Response:

```
OK\r\n
```

5.3.3 SPP Transfer

Description:

This is used to send data to the remote BT device using SPP profile. This command contains a data length field, which tells the BT module about the length of data in bytes user want to send from the application.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_spp_transfer {
UINT16 DataLength;
UINT08 Data[200];
}  RSI_BT_CMD_SPP_TRANSFER;
```

AT command format:

```
at+rsibt_spptx=<DataLength>,<Data>\r\n
```

Parameters:
DataLength – SPP data length (range of Data length is 1 to 200 Bytes).
Data – SPP data.

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

AT command Ex:

```
at+rsibt_spptx=5,iiiii\r\n
```

Response:

```
OK\r\n
```

5.4 A2DP commands

5.4.1 A2DP Connect

Description:

This is used to establish A2DP connection with the remote BT device, specified by the BD address.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_a2dp_connect {
UINT08   BDAddress[6];
} RSI_BT_CMD_A2DP_CONNECT;
```

AT command format:

```
at+rsibt_a2dpconn=<BDAddress>\r\n
```

Parameters:
BDAddress – Remote BD address

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

AT command Ex:

```
at+rsibt_a2dpconn= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.4.2 A2DP Disconnect

Description:

This is used to disconnect the A2DP connection with the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_a2dp_disconnect {
UINT08   BDAddress[6];
}
RSI_BT_CMD_A2DP_DISCONNECT;
```

AT command format:

```
at+rsibt_a2dpdisconn=<BDAddress>\r\n
```

Parameters:

BDAddress – Remote BD address

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

AT command Ex:

```
at+rsibt_a2dpdisconn= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.5 AVRCP Commands

5.5.1 AVRCP Connect

Description:

This is used to establish AVRCP connection with the remote BT device, specified by the BD address.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_avrcp_connect {
UINT08   BDAddress[6];
} RSI_BT_CMD_AVRCP_CONNECT;
```

AT command format:

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

Parameters:

BDAddress – Remote BD address

Response Payload:

There is no response payload for this command.

AT command Ex:

```
at+rsibt_avrcpconn= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.5.2 AVRCP Disconnect

Description:

This is used to disconnect the AVRCP connection with the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_avrcp_disconnect {
UINT08   BDAddress[6];
} RSI_BT_CMD_AVRCP_DISCONNECT;
```

AT command format:

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

Parameters:
BDAddress – Remote BD address

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

AT command Ex:

```
at+rsibt_avrcpdisconn= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.5.3 AVRCP Play

Description:
This is used to play audio in the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_avrcp_play
{
UINT08   BDAddress[6];
} RSI_BT_CMD_AVRCP_PLAY;
```

AT command format:

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

Parameters:
BDAddress – Remote BD address

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

AT command Ex:

```
at+rsibt_avrcpplay= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.5.4 AVRCP Pause

Description:
This is used to pause audio in the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_avrcp_pause
{
UINT08   BDAddress[6];
} RSI_BT_CMD_AVRCP_PAUSE;
```

AT command format:

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

Parameters:
BDAddress – Remote BD address

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

AT command Ex:

```
at+rsibt_avrcppause= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.5.5 AVRCP stop

Description:
This is used to stop audio in the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_avrcp_stop
{
UINT08   BDAddress[6];
} RSI_BT_CMD_AVRCP_STOP;
```

AT command format:

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

Parameters:
BDAddress – Remote BD address

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

AT command Ex:

```
at+rsibt_avrcpstop= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.5.6 AVRCP next

Description:
This is used to go to the next audio file in the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_avrcp_next
{
UINT08   BDAddress[6];
} RSI_BT_CMD_AVRCP_NEXT;
```

AT command format:

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

Parameters:

BDAddress – Remote BD address

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

AT command Ex:

```
at+rsibt_avrcpnext= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.5.7 AVRCP previous

Description:
This is used to go to the previous audio file in the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_avrcp_previous
{
UINT08   BDAddress[6];
} RSI_BT_CMD_AVRCP_PREVIOUS;
```

AT command format:

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

Parameters:
BDAddress – Remote BD address

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

AT command Ex:

```
at+rsibt_avrcpprev= AA-BB-CC-DD-EE-FF\r\n
```

Response:

```
OK\r\n
```

5.6 Power Save

Description
This feature explains the configuration of **Power Save** modes of the module. These can be issued at any time after Opermode command. By default, Power Save is in disable state..
There are five different modes of Power Save. They are outlined below.

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

Note:

1. Power Save modes 2 and 8 are not supported in USB / USB-CDC interface. Instead, they are supported in UART / SPI interfaces.
2. In SPI interface, when Power Save mode is enabled, after wakeup from sleep, the host has to re-initialize SPI interface of the module.
3. When co-ex mode is enable, power save mode is not applicable for WLAN also.

5.6.1 Power save Operations

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

5.6.1.1 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.

5.6.1.2 Power Save Mode 2 (GPIO based mode):

For detail description for power save mode 2, please refer to RYWB116_Embedded_WLAN_Software_Programming_Reference_Manual.pdf.

5.6.1.3 Power Save Mode 3 (Message based mode):

For detail description for power save mode 3, please refer to RYWB116_Embedded_WLAN_Software_Programming_Reference_Manual.pdf.

Usage in BT-Classic Mode:

In Classic, Power Save mode 2 and 3 can be used during Discoverable / Connectable / Connected sniff states. Each of these states is explained below:

- **Discoverable Mode State:** In this state, the module is awake during Inquiry Scan window duration and sleeps till Inquiry Scan interval.

The default inquiry scan window value is 11.25 msec, and inquiry scan interval is 1.28 sec.

- **Connectable Mode State:** In this state, the module is awake during Page Scan window duration and sleeps till Page Scan interval.

The default page scan window value is 11.25 msec, and page scan interval is 1.28 msec.

- **Connected Sniff State:** While the module is in connected state as a master or slave, once the module has configured with Power Save mode with GPIO based or message based, the module will goes into power save mode in connected state. This will work when the module and peer device supports sniff feature. And also the module should configure with sniff command after a successful connection, before configuring it with power save command.

The module will goes into power save after serving a sniff anchor point and will wake up before starting a sniff anchor point.
Sniff connection anchor point may vary based on the remote device t_sniff value.

5.6.1.4 Power Save Mode 8 (GPIO based mode):

For detail description for power save mode 8, please refer to RYWB116_Embedded_WLAN_Software_Programming_Reference_Manual.pdf.

5.6.1.5 Power Save Mode 9 (Message based mode):

For detail description for power save mode 9, please refer to RYWB116_Embedded_WLAN_Software_Programming_Reference_Manual.pdf.

Note:

1. Power save disable command has to be given before changing the state from standby to the remaining states and wise-versa.
2. For Page scan, Inquiry scan, sniff parameters related information, please refer Bluetooth protocol specification document.
3. When the module is configured in a co-ex mode and WLAN is in INIT_DONE state, power save mode 2 & 3 are valid after association in the WLAN. Where as in BT alone modes, it will enter into power save mode (2 & 3) in all the states (except in standby state).
4. In BT connected state, power save will work only when the module is in sniff mode.
5. If BT is in connected state and it is in Active mode even though power save command is issued module will not enter into the power save.

5.7 IAP commands

This profile is used for remote controlling of apple devices. It is having IAP1 and IAp2 protocols. Currently, we are supporting only IAP1. We are supporting two lingos are General lingo and Simple remote lingo.

5.7.1 IAP connect

Description:

This command is used to establish an IAP connection with the remote Apple device, specified by the BD address.

Command:

```
at+rsibt_iapconn=<bd_addr>\r\n
```

Parameters:
bd_addr- BD address of the remote Apple device, with which the IAP connection has to be established.

Response:

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

AT command Ex:
at+rsibt_iapconn=11-22-33-44-55-66\r\n

Response:

```
OK\r\n
```

5.7.2 IAP Disconnect

Description:
This command is used to disconnect the Apple device which was connected using IAP.

Command:

```
at+rsibt_iapdisconn=<bd_addr>\r\n
```

Parameters:
bd_addr- BD address of the remote Apple device, with which the IAP connection has to be released.

Response:

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

AT command Ex:

```
at+rsibt_iapdisconn=11-22-33-44-55-66\r\n
```

Response:

```
OK\r\n
```

5.7.3 IAP Set Accessory Information

Description:

This command is used to set the accessory information like name, manufacturer, model number, serial number, firmware and hardware version, supported languages by the accessory and currently used language in the accessory. This information should be sent to the accessory before starting the identification procedure without fail.When the user wants to change these values, the accessory must go through the identification procedure again, which will update the accessory information at Apple device side. This command is common for both IAP1 and IAP2.

Command:

```
at+rsibt_iapsetaccessoryinfo=<info_type>,<data_len>,<info_data>\r\n
```

Parameters:

nbr_lang_supp- This parameter is used only for Info type 8, which indicates the number of languages supported by the accessory.

Info_Type	Info_Data
1	**Accessory Name.** This should be <= 63 characters.
2	**Accessory Manufacturer.** This should be <= 63 characters.
3	**Accessory Model Number.** This should be <= 63 characters.
4	**Accessory Serial Number.** This should be <= 63 characters.
5	**Accessory Firmware Version.** Ex: Major version, Minor version, Revision version.
6	**Accessory Hardware Version.** Ex: Major version, Minor version, Revision version.
7	**Accessory Current Language.**
8	**Accessory Supported Languages.** These values should be according the following format mentioned in the below link. * http://www.loc.gov/standards/iso639-2/php/English_list.php*

Response:

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

AT command Ex:

```
at+rsibt_iapsetaccessoryinfo=1,Reyax_BT_Accessory\r\n
at+rsibt_iapsetaccessoryinfo=5,1,2,3\r\n
at+rsibt_iapsetaccessoryinfo=7,1,en\r\n
at+rsibt_iapsetaccessoryinfo=8,3,en,fr,hi\r\n
```

Response:

```
OK\r\n
```

5.7.4 IAP Find Protocol Type

Description:

This command is used to find the type of protocol(IAP1 or IAP2)supported by the connected Apple device. If the connected device supports IAP1, the identification procedure will start automatically. If it doesn't start, start IAP1 identification manually, by sending the IAP1 Identification command. If the device supports IAP2, the IAP2 Identification procedure should be started manually by sending IAP2 Identification command.

Command:

```
at+rsibt_iapfindprotocoltype\r\n
```

Parameters:
None

Response:

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

AT command Ex:

```
at+rsibt_iapfindprotocoltype\r\n
```

Response:

```
OK 2\r\n
```

5.7.5 IAP Set Protocol Type

Description:

This command is used to Set the protocol type of accessory to IAP1. This command is used when the user wants to use only IAP1 to communicate with the device, even though the device supports IAP2. This command forces the accessory to use only IAP1. IAP1 Identification should be done manually. This command is used to set only IAP1 in the accessory, but not used to set IAP2.

Note:

when this command is used, IAP Find Protocol Type command should not be used prior to this command. If IAP Find Protocol Type command is already used, the Bluetooth connection has to be re-established, to use this (IAP Set Protocol Type) command.

Command:
at+rsibt_iapsetprotocoltype=1\r\n

Parameters:
Type: 1 – IAP1
2 – IAP2

Response:

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

AT command Ex:

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

Response:

```
OK\r\n
```

5.7.6 IAP Set Application Protocol Information

Description:

The Application Protocol Information is needed only when the accessory wants to communicate with an application in the Apple device. Using this command, the accessory will intimate the Apple device about the details of application. This must be done before identification procedure. Using this command the user can disable the support of communicating with the application in Apple device. This command is common for both IAP1 and IAP2.

Command:

```
at+rsibt_iapsetappprotocolinfo=<mode>,<protocol_index>,<protocol_str_len>,<protocol_str>,<bundle_seed>,<met
a_data>\r\n
```

Parameters:
Mode - 0 – Disable application support in Accessory.
1 – Enable application support in Accessory.
If this value is "0", the accessory will neglect the remaining fields in the command and removes the application support in accessory.
protocol_index - Index of the protocol assigned by the accessory. 1 to 255 can be used.
protocol_str_len - Length of the protocol string.
protocol_str- Actual protocol string. This is a reverse DNS name like **"com.apple.Music"**
bundle_seed- Bundle seed ID string allocated by Apple.Inc, to the accessory application developer.

meta_data	Details
0	The Apple device doesn't try to find a matching app. It doesn't display a "Find App For This Accessory" button in the Settings > General .About >Accessory Name screen.
1	The Apple device tries to find a matching app. It doesn't display a "Find App For This Accessory" button in the Settings > General .About >Accessory Name screen.
3	The Apple device doesn't try to find a matching app, but it displays a "Find App For This Accessory" button in the Settings > General .About >Accessory Name screen so the user can find one.

Response:

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

AT command Ex:

```
at+rsibt_iapsetappprotocolinfo\r\n
```

Response:

```
OK\r\n
```

5.7.7 IAP1 Identification

Description:
This command is used to start the Identification procedure of an accessory with the Apple device. The Identification will be followed by the Authentication procedure, which will be initiated by the Apple device. Identification is the mandatory step to be followed by the accessory. User can use remaining commands only after receiving the notification
"AT+RSIBT_IAP1_ACCESSORY_AUTH_COMPLETED"

Command:

```
at+rsibt_iap1identification\r\n
```

Parameters:
NONE

Response:

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

AT command Ex:

```
at+rsibt_iap1identification\r\n
```

Response:

```
OK\r\n
```

5.7.8 IAP1 Apple Device Authentication

Description:

This command is used to authenticate the connected Apple device. This procedure will be initiated by the accessory and it is not mandatory.

Command:

```
at+rsibt_iap1deviceauthentication\r\n
```

Parameters:
NONE

Response:

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

AT command Ex:

```
at+rsibt_iap1deviceauthentication\r\n
```

Response:

```
OK\r\n
```

5.7.9 Set Assistive Touch

Description:

This command is used to enable or disable the assistive touch feature in the Apple device. The restore on exit parameter indicates, whether the Apple device has to restore the previous (before connected with the accessory) settings of the assistive touch feature, after the accessory gets disconnected with the Apple device.

Command:

```
at+rsibt_iap1setassistivetouch=<mode>,<restore_on_exit>\r\n
```

Parameters:
Mode- Assistive Touch Mode.
0 – OFF.
1 – ON.
restore_on_exit- 1 – Apple device restores the original settings when the accessory is disconnected.
0 – Apple device doesn't perform the restore.

Response:

Result Code	Description
OK	Command Success.

Result Code	Description
ERROR <Error_code>	Command Fail.

AT command Ex:

```
at+rsibt_iap1setassistivetouch=1,1\r\n
```

Response:

```
OK\r\n
```

5.7.10 Set Voice Over

Description:

This command is used to enable or disable the voiceover feature in the Apple device. The restore on exit parameter indicates, whether the Apple device has to restore the previous (before connected with the accessory) settings of the voiceover feature, after the accessory gets disconnected with the Apple device.

Command:

```
at+rsibt_iap1setvoiceover=<mode>,<restore_on_exit>\r\n
```

Parameters:
Mode- Voice over Mode.
0 – OFF.
1 – ON.
restore_on_exit - 1 – Apple device restores the original settings when the accessory is disconnected.
0 – Apple device doesn't perform the restore.

Response:

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

AT command Ex:

```
at+rsibt_iap1setvoiceover=1,1\r\n
```

Response:

```
OK\r\n
```

5.7.11 AP1 Get Ipod Info (Name, SW version, serial number)

Description:

This command is used to get the information (name, software version, serial number) of the connected Apple device.

Command:

```
at+rsibt_iap1getipodinfo\r\n
```

Parameters:
Name- Name of the Apple device connected. It is null terminated character array.
Software Version- Software version of the connected Apple device. Major, Minor and Revision version numbers separated by "-".
Serial Number- Serial Number of the connected Apple Device. It is null terminated character array.

Response:

Result Code	Description
OK <Name(null terminated array)>,<software version>,<serial number>	Command Success.
ERROR <Error_code>	Command Fail.

AT command Ex:

```
at+rsibt_iap1getipodinfo\r\n
```

Response:

```
OK Apple_iPhone,6-1-3,6Q105EN0A4S\r\n
```

5.7.12 IAP1 Set Extended Interface Mode (ON/OFF)

Description:

This command is used to set the Extended Interface Mode ON and OFF in the connected Apple device. This command works only when the Accessory supports Extended Interface Mode Lingo.

Command:

```
at+rsibt_iap1setextendedintfmode=<mode>\r\n
```

Parameters:
Mode- Sets the mode of Extended Interface.
0 – Extended Interface Mode OFF
1 – Extended Interface Mode ON.

Response:

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

AT command Ex:

```
at+rsibt_iap1setextendedintfmode=1\r\n
```

Response:

```
OK\r\n
```

5.7.13 IAP1 Get Lingo Protocol Version

Description:

This command is used to get the protocol version of a particular lingo, supported by the Apple device.

Command:

```
at+rsibt_iap1getlingoprotocolversion=<lingo_id>\r\n
```

Parameters:
lingo_id- Lingo ID for which protocol version has to be known.
Protocol version- Protocol version of the lingo. Major and minor version will be separated by "-".

Response:

Result Code	Description
OK <lingo_id>,<protocol version>	Command Success.
ERROR <Error_code>	Command Fail.

AT command Ex:

```
at+rsibt_iap1getlingoprotocolversion=0\r\n
```

Response:

```
OK 0,1-3\r\n
```

5.7.14 IAP1 Set Ipod Preferences

Description:

This command is used to set the IPOD class of preference, like video out settings, screen configuration etc.

Command:

```
at+rsibt_iap1setipodpreferences=<class_id>,<setting_id>,<restore_on_exit>\r\n
```

Parameters:
class_id- Class ID of the preference.
setting_id- Settings ID of the preference.
restore_on_exit- 1 – Restore original settings on exit (disconnection).
0 – Keep current settings even after exit.

Class_id	Preference	Settings	Settings_id
0	Video out	OFF	**0**
		ON	**1**
1	Screen Configuration	Fill entire screen	**0**
		Fit to screen edge	**1**
2	Video Signal Format	NTSC	**0**
		PAL	**1**
3	Line-out usage	Not used	**0**
		Used	**1**
8	Video out connection	Composite	**1**
		S-Video	**2**
		Component	**3**
9	Closed captioning	OFF	**0**
		ON	**1**
10	Video monitor aspect ratio	4:3 (full screen)	**0**
		16:9 (wide screen)	**1**
12	Subtitles	OFF	**0**
		ON	**1**
13	Video alternate audio channel	OFF	**0**
		ON	**1**
15	Pause on power removal	OFF	**0**
		ON	**1**

Response:

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

AT command Ex:

```
at+rsibt_iap1setipodpreferences=0,1,1\r\n
```

Response:

```
OK\r\n
```

5.7.15 IAP1 Get Ipod Preferences

Description:

This command is used for Identification of an accessory with the Apple device.

Command:

```
at+rsibt_iap1getipodpreferences=<class_id>\r\n
```

Parameters:
Class_id- Class ID of the preference.
Setting_id- Settings ID of the preference.

Response:

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

AT command Ex:
at+rsibt_iap1getipodpreferences=0\r\n

Response:

```
OK 0,1\r\n
```

5.7.16 IAP1 Set UI Mode

Description:

This command is used to set the UI mode of the Apple device.

Command:

```
at+rsibt_iap1setuimode=<mode>\r\n
```

Parameters:
Mode - UI mode options.
0 – Standard Apple device operating mode
1 – Extended interface mode
2 – ipod out full screen mode
3 – ipod out action safe mode.

Response:

Result Code	Description
OK	Command Success.

Result Code	Description
ERROR <Error_code>	Command Fail.

AT command Ex:

```
at+rsibt_iap1setuimode=0\r\n
```

Response:

```
OK\r\n
```

5.7.17 IAP1 Get UI Mode

Description:

This command is used to get the current UI mode set in the Apple device.

Command:

```
at+rsibt_iap1getuimode\r\n
```

Parameters:
Mode - UI mode options.
0 – Standard Apple device operating mode
1 – Extended interface mode
2 – ipod out full screen mode
3 – ipod out action safe mode.

Response:

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

AT command Ex:

```
at+rsibt_iap1getuimode\r\n
```

Response:

```
OK 0\r\n
```

5.7.18 IAP1 Set Event Notification

Description:

This command is used to set selected notifications from the Apple device. These notifications are asynchronous events sent by the Apple device on occurrence of a particular event in Apple device.

Command:

```
at+rsibt_iap1seteventnotification=<nbr_notifications>,<1>,<2>,....<n>\r\n
```

Parameters:
nbr_notifications- Number of notifications to be set.

Notification	Details
2	Flow Control.
3	Radio tagging status.
4	Camera status.
5	Charging Info.
9	Database Changed.
10	Now Playing Application Bundle Name Status.
11	Session Space Available.
15	Ipod out mode status.
17	Bluetooth connection status
19	Now playing application display name
20	Assistive touch status.

Response:

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

AT command Ex:

```
at+rsibt_iap1seteventnotification=6,2,3,4,5,9,10\r\n
```

Response:

```
OK\r\n
```

5.7.19 IAP1 Get Event Notification

Description:

This command is used to get the list of notification enabled in the Apple device.

Command:

```
at+rsibt_iap1geteventnotification\r\n
```

Parameters:
nbr_notifications - Number of notifications already set in the Apple device.

Notification	Details
2	Flow Control.
3	Radio tagging status.
4	Camera status.
5	Charging Info.
9	Database Changed.
10	Now Playing Application Bundle Name Status.
11	Session Space Available.
15	Ipod out mode status.
17	Bluetooth connection status
19	Now playing application display name
20	Assistive touch status.

Response:

Result Code	Description
OK <nbr_notifications>,<notification1>,<2>,<3>,….	Command Success.
ERROR <Error_code>	Command Fail.

AT command Ex:

```
at+rsibt_iap1geteventnotification\r\n
```

Response:

```
OK 6,2,3,4,5,9,10\r\n
```

5.7.20 IAP1 Get Supported Event Notification

Description:

This command is used to get the list of notifications supported by Apple device.

Command:

```
at+rsibt_iap1suppeventnotifications\r\n
```

Parameters:
nbr_notifications- Number of notifications supported by Apple device.

Notification	Details
2	Flow Control.
3	Radio tagging status.
4	Camera status.
5	Charging Info.
9	Database Changed.
10	Now Playing Application Bundle Name Status.
11	Session Space Available.
15	Ipod out mode status.
17	Bluetooth connection status
19	Now playing application display name
20	Assistive touch status.

Response:

Result Code	Description
OK <nbr_notifications>,<notification1>,<2>,<3>,….	Command Success.
ERROR <Error_code>	Command Fail.

AT command Ex:

```
at+rsibt_iap1suppeventnotifications\r\n
```

Response:

```
OK 6,2,3,4,5,9,10\r\n
```

5.7.21 IAP1 Launch Application

Description:

This command is used to launch an application in the Apple device.

Command:

```
at+rsibt_iap1launchapplication=<app_name>\r\n
```

Parameters:
app_name- Name of the application to be launched. It is a null terminated character array.

Response:

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

AT command Ex:

```
at+rsibt_iap1launchapplication=com.apple.Music\r\n
```

Response:

```
OK\r\n
```

5.7.22 IAP1 Get Localization Info

Description:

This command is used to get the localization information (language and region) currently set in the Apple device.

Command:

```
at+rsibt_iap1getlocalizationinfo\r\n
```

Parameters:
Language- Current language used in the Apple device. It is null terminated character array.
Region- Region settings in the Apple device. It is null terminated character array.

Response:

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

AT command Ex:

```
at+rsibt_iap1getlocalizationinfo\r\n
```

Response:

```
OK en-GB,en-IN\r\n
```

5.7.23 IAP1 Application Data Session Acknowledgment

Description:

This command is used to acknowledge the requests (open and close data session) related to the data session and to acknowledge the received data packet from the Apple device.

Command:

```
at+rsibt_iap1appdatasessionack=<recv_cmd_type>,<status>\r\n
```

Parameters:
recv_cmd_type- Acknowledgment for the event received.
1 – Open Data Session.
2 – Close Data Session.
3 – IPod Data Transfer.
Status- 0 – Success.
4 – Bad Parameter. This error should be set only when the Apple device sends Ipod Data without opening a session.

Response:

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

AT command Ex:

```
at+rsibt_iap1appdatasessionack=1,0\r\n
```

Response:

```
OK\r\n
```

5.7.24 IAP1 Application Accessory Data Transfer

Description:

This command is used to transfer data to the Apple device. If an Apple device unable to handle data, it will return an error **"ERR_IAP1_DROPPED_DATA"**. At that time the accessory has to wait for certain amount of time until it receives the event notification **"AT+RSIBT_IAP1_SESSION_SPACE_AVAILABLE"**.

Command:

```
at+rsibt_iap1appaccessorydatatransfer=<session_id>,<data_len>,<data>\r\n
```

Parameters:
session_id- Session ID sent by the Apple device at the time of opening the data session.
data_len- Length of data sent by the Accessory to the Apple device.
data- Actual data sent by the Accessory.
bytes_dropped- Number of bytes dropped by the Apple device without handling.

Response:

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

AT command Ex:

```
at+rsibt_iap1appaccessorydatatransfer=1,7,welcome\r\n
```

Response:

```
OK\r\n
ERROR 8017,50\r\n
```

5.7.25 IAP1 Get Voice Over Parameter

Description:

This command is used to get the current value of a particular voice over parameters, like volume, speaking rate.

Command:

```
at+rsibt_iap1getvoiceoverparameter=<param_type>\r\n
```

Parameters:
param_type- Type of the parameter to be set.
param_value- Parameter value corresponding to the parameter type.

Param_Type	Param_Value
0	Voiceover volume. 0(mute) to 255(full volume).
1	Speaking Rate. 0(slow) to 255(fast).

Response:

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

AT command Ex:

```
at+rsibt_iap1getvoiceoverparameter=0\r\n
```

Response:

```
OK 0,150\r\n
```

5.7.26 IAP1 Set VoiceOver Parameter

Description:

This command is used to set the value of a particular voice over parameters, like volume, speaking rate.

Command:

```
at+rsibt_iap1setvoiceoverparameter<param_type>,<param_value>\r\n
```

Parameters:
param_type- Type of the parameter to be set.
param_value- Parameter value corresponding to the parameter type.

Param_Type	Param_Value
0	Voiceover volume. 0(mute) to 255(full volume).
1	Speaking Rate. 0(slow) to 255(fast).

Response:

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

AT command Ex:

```
at+rsibt_iap1setvoiceoverparameter=0,150\r\n
```

Response:

```
OK\r\n
```

5.7.27 IAP1 Set VoiceOver Context

Description:

This command is used to set the context of voiceover in which the user is currently working.

Command:

```
at+rsibt_iap1setvoiceovercontext=<param_type>\r\n
```

Parameters:
param_type- The user interface context to be set.

Param_type	Details
0	None. Exit from or disable any context.
1	Header.
2	Link.
3	Form.
4	Cursor.
5	Vertical Navigation. Sets Move Next/Move Previous to move down/up to the next item vertically
6	Value Adjustment. Sets Move Next/Move Previous to increment/decrement the value of the current item by 5%.
7	Zoom Adjustment. Sets Move Next/Move Previous to zoom the current item in/out by 1 increment.

Response:

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

AT command Ex:

```
at+rsibt_iap1setvoiceovercontext=1\r\n
```

Response:

```
OK\r\n
```

5.7.28 IAP1 VoiceOver Event

Description:

This command is used to send the voice over events, like Move to first, move to last, scroll page up/down etc to control the Apple device.

Command:

```
at+rsibt_iap1voiceoverevent=<event_type>\r\n
```

Parameters:

event_type	Details
1	Move to First.
2	Move to Last.
3	Move to Next. When option 5 is set in the "Set Voiceover Context" command, Move to Next and Move to Previous are used to navigate up and down respectively.
4	Move to Previous.
5	Scroll Left Page.
6	Scroll Right Page.
7	Scroll Up Page.
8	Scroll Down Page.
11	Cut.
12	Copy.

13	Paste.
14	Home
18	Pause Speaking.
19	Resume Speaking.
20	Read all text from current point.
21	Read all text from top.
23	Escape. Voice over exit from a modal view, such as an alert or popover.

Response:

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

AT command Ex:

```
at+rsibt_iap1voiceoverevent=1\r\n
```

Response:

```
OK\r\n
```

5.7.29 IAP1 VoiceOver Text Event

Description:

This command is used to send text to Apple device, which can be used in filling a form, entering URL, and also used as text-to-speech, which can be pronounced by the Apple device.

Command:

```
at+rsibt_iap1voiceovertextevent=<event_type>,<data_len>,<data>\r\n
```

Parameters:
event_type- Type of the event to be set.
10 – Text in this type is used to fill the forms, links etc.
22 – Text in this type is used to be pronounced by the Apple device like text-to-speech.
data_len- Length of the event data.
data- Event data corresponding to the event type.

Response:

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

AT command Ex:

```
at+rsibt_iap1voiceovertextevent=22,29,Welcome to Reyax.\r\n
at+rsibt_iap1voiceovertextevent=10,14,www.google.com\r\n
```

Response:

```
OK\r\n
```

5.7.30 IAP1 VoiceOver Touch Event

Description:

This command is used to tap a selected item (select an item) in the Apple device.

Command:

```
at+rsibt_iap1voiceovertouchevent=<event_type>\r\n
```

Parameters:
event_type- Type of the event to be set. Set 0 as event type.

Response:

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

AT command Ex:

```
at+rsibt_iap1voiceovertouchevent=0\r\n
```

Response:

```
OK\r\n
```

5.7.31 IAP1 Current VoiceOver Value

Description:

This command is used to get the value of current voiceover item like the percentage of volume set.

Command:

```
at+rsibt_iap1currvoiceovervalue\r\n
```

Parameters:
Value- Value like volume %.

Response:

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

AT command Ex:

```
at+rsibt_iap1currvoiceovervalue\r\n
```

Response:

```
OK  62%\r\n
```

5.7.32 IAP1 Current VoiceOver Hint

Description:

This command is used to get the hint of current voiceover item like the percentage of volume set.

Command:

```
at+rsibt_iap1currvoiceoverhint\r\n
```

Parameters:
None

Response:

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

AT command Ex:

```
at+rsibt_iap1currvoiceoverhint\r\n
```

Response:

```
OK\r\n
```

5.7.33 IAP1 Current VoiceOver Trait

Description:

This command is used to get the type of item currently pointed by voice over.

Command:

```
at+rsibt_iap1currvoiceovertrait\r\n
```

Parameters:

Trait_type	Details
0	Button
1	Link
2	Search
3	Image
4	Selected
5	Sound
6	Keyboard Key
7	Static Text
8	Summary Element
9	Not Enabled
10	Updates Frequently
11	Starts Media Session
12	Adjustable
13	Back Button
14	Maps
15	Delete Key

Response:

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

AT command Ex:

```
at+rsibt_iap1currvoiceovertrait\r\n
```

Response:

```
OK 0,13\r\n
```

5.7.34 IAP1 iPod Out Button

Description:

This command is used to set the voice over events, like Move to first, move to last, scroll page up/down etc.

Command:

```
at+rsibt_iap1ipodoutbutton=<button_src>,<button_type>\r\n
```

Parameters:

button_src	Description
0	Car center console
1	Steering wheel
2	Car dashboard

button_type	Description
0	Select event
1	Left event
2	Right event
3	Up event
4	Down event
5	Menu button event

Response:

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

AT command Ex:

```
at+rsibt_iap1ipodoutbutton=1,1\r\n
```

Response:

```
OK\r\n
```

5.7.35 IAP1 Video Button

Description:

This command is used to access Video controls in the Apple device.

Command:

```
at+rsibt_iap1videobutton=<button_type>\r\n
```

Parameters:
button_type- Type of button pressed among the buttons mentioned below

Button_Type	Details
0	Play/Pause
1	Next Video
2	Previous Video
3	Stop
4	Play/Resume
5	Pause
6	Begin FF
7	Begin REW
8	Next Chapter
9	Previous Chapter
10	Next Frame
11	Previous Frame

Response:

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

AT command Ex:

```
at+rsibt_iap1videobutton=1\r\n
```

Response:

```
OK\r\n
```

5.7.36 IAP1 Audio Button

Description:

This command is used to access Audio controls in the Apple device.

Command:

```
at+rsibt_iap1audiobutton=<button_type>\r\n
```

Parameters:
button_type- Type of button pressed among the buttons mentioned below.

Button_Type	Details
0	Play/Pause
3	Next Track
4	Previous Track
5	Next Album
6	Previous Album
7	Stop
8	Play/Resume
9	Pause
11	Next Chapter
12	Previous Chapter
13	Next Playlist
14	Previous Playlist
15	Shuffle Setting Advance
16	Repeat Setting Advance
17	Begin Fast Forward
18	Begin Rewind
19	Menu

Response:

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

AT command Ex:

```
at+rsibt_iap1audiobutton=4\r\n
```

Response:

```
OK\r\n
```

5.7.37 IAP1 Context Button

Description:

This command is used to access the media controls according to the current type of media running in the Apple device. For example, when audio is playing in the Apple device, this command can be able to control the audio player. Like this it can control the video player, app store player as well.

Command:

```
at+rsibt_iap1contextbutton=<button_type>\r\n
```

Parameters:
button_type- Type of button pressed among the buttons mentioned below.

Button_Type	Details
0	Play/Pause
3	Next Track
4	Previous Track
5	Next Album
6	Previous Album
7	Stop
8	Play/Resume
9	Pause
11	Next Chapter
12	Previous Chapter
13	Next Playlist
14	Previous Playlist
15	Shuffle Setting Advance
16	Repeat Setting Advance
17	Power ON
18	Power OFF
19	Backlight for 30 Seconds
20	Begin Fast Forward

Button_Type	Details
21	Begin Rewind
22	Menu
23	Select
24	Up Arrow
25	Down Arrow
26	Backlight OFF

Response:

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

AT command Ex:

```
at+rsibt_iap1contextbutton=0\r\n
```

Response:

```
OK\r\n
```

5.7.38 IAP1 Radio Button

Description:

This command is used to initiate tagging action by iPod's Radio Application. This is supported only in 5G nano iPod.

Command:

```
at+rsibt_iap1radiobutton=<button_status>\r\n
```

Parameters:

button_status	Description
0	Radio button pushed to tag current song.
2	Button released.

Response:

Result Code	Description
OK	Command Success.

Result Code	Description
ERROR <Error_code>	Command Fail.

AT command Ex:

```
at+rsibt_iap1radiobutton=2\r\n
```

Response:

```
OK\r\n
```

5.7.39 IAP1 Camera Button

Description:

This command is used to control the camera in the Apple device. This command can be used only when the accessory get connected with ipod 5G.

Command:

```
at+rsibt_iap1camerabutton=<button_status>\r\n
```

Parameters:

button_status	Description
0	Button up
1	Button down

Response:

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

AT command Ex:

```
at+rsibt_iap1camerabutton=1\r\n
```

Response:

```
OK\r\n
```

5.7.40 IAP1 Rotation Input

Description:

This command is used to set the voice over events, like Move to first, move to last, scroll page up/down etc.

Command:

```
at+rsibt_iap1rotationinput=<usr_action_dur>,<src>,<dir>,<action>,<type>,<nbr_moves>,<total_moves>\r\n
```

Parameters:
usr_action_dur - Number of milliseconds since the start of the current user action.
Src - Location of the wheel.
0 – Car center console.
1 – Steering wheel.
2 – Car dashboard.
Dir - 0 – Counter clockwise.
1 – Clockwise.
Action - 0 – Rotation action completed; user has released control.
1 – Rotation in progress; wheel turning.
2 – Rotation repeat; the user has not advanced the wheel from the position reported in the last RotationInputStatus packet.
Type - 0 – Wheel has detents.
1 – Wheel reports angular degrees.
Nbr_moves - Number of detents or degrees the user has moved the wheel since the last RotationInputStatus packet.
Total_moves - Constant number of detents or degrees in a full turn of the wheel (max 360).

Response:

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

AT command Ex:

```
at+rsibt_iap1rotationinput=100,0,0,1,0,5,20\r\n
```

Response:

```
OK\r\n
```

5.7.41 IAP1 Register HID Report Descriptor

Description:

This command is used to register a HID Report descriptor with the Apple device. The format of the descriptor is same that is defined in the USB specification.

Command:

```
at+rsibt_iap1reghiddescriptor=<desc_index>,<vendor_id>,<product_id>,<country_code>,<len>,<descriptor>\r\n
```

Parameters:
desc_index: HID descriptor index.
vendor_id: Vendor ID of the accessory.
product_id: Product ID of an accessory.
country_code: country code. Country codes are listed in the table below.
len: Length of the descriptor bytes.
descriptor: Report descriptor. The descriptor values are not the ASCII values, but the numerical values.

Response:

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

AT command Ex:
Here is the example for mouse report descriptor.
at+rsibt_iap1reghiddescriptor=0,1452,4626,20,50, 05 01 09 02 A1 01 09 01 A1 00 05 09 19 01 29 03 15 00 25 01 95 03 75 01 81 02 95 01 75 05 81 01 05 01 09 30 09 31 15 81 25 7F 75 08 95 02 81 06 C0 C0 \r\n

Response:

```
OK\r\n
```

Apple also provides some customer usage page (0x0C), to access some of the controls in the Apple devices. The control codes are listed below.

HID Usage ID	HID Usage Name	iOS Function
0x0030	Power	Lock
0x0040	Menu	Home
0x00B5	Scan Next Track	Transport Right
0x00B6	Scan Previous Track	Transport Left
0x00CD	Play/Pause	Play/Pause
0x00E2	Mute	Mute
0x00E9	Volume increment	Louder
0x00EA	Volume decrement	Softer
0x01AE	AL keyboard layout	Toggle onscreen keyboard
0x01B1	AL screen saver	Picture frame
0x0221	AC search	Spotlight

5.7.42 IAP1 Send HID Report

Description: This command is used to send a report to the Apple device based on the descriptor registered with the Apple device.

Command:

```
at+rsibt_iap1sendhidreport=<desc_index>,<report_type>,<len>,<report>\r\n
```

Parameters:
desc_index- HID descriptor index, already registered with the Apple device.
report_type- Input report.
Len- Length of the report in bytes.
Report- Device report. The report values are not the ASCII values, but the numerical values.

Response:

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

AT command Ex:
Here is the example for mouse report.
at+rsibt_iap1sendhidreport=0,0,3, 01 00 00 \r\n
Response:

```
OK\r\n
```

5.7.43 IAP1 Unregister HID Report Descriptor

Description:

This command is used to unregister a HID Report descriptor with the Apple device.

Command:

```
at+rsibt_iap1unreghiddescriptor=<desc_index>\r\n
```

Parameters:
desc_index: HID descriptor index, already registered with the Apple device.

Response:

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

AT command Ex:

```
at+rsibt_iap1unreghiddescriptor=0\r\n
```

Response:

```
OK\r\n
```

5.8 PER Commands

5.8.1 PER Transmit

Description:

This will allow the configuration of the following parameters and starts the transmission of packets.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_per_transmit{
UINT08 enable;
UINT08 BDAddress[6];
UINT08 pkttype;
UINT16 pktlength;
UINT08 linktype;
UINT08 rate;
UINT08 rx_channel;
UINT08 tx_channel;
UINT08 scramblerseed;
UINT32 nbrofpkts;
UINT08 payloadtype;
UINT08 protocolmode;
UINT08 lechanneltype;
UINT08 powerindex;
UINT08 transmitmode;
UINT08 freqhopenable;
UINT08 antennaselect;
}RSI_BT_CMD_PER_TRANSMIT;
```

Command:

```
at+rsibt_pertransmit=<per_enable/
disablebit>,<dev_addr>,<pkt_type>,<pkt_length>,<link_type>,<br_edr_mode>,<rx_channel_index>,<tx_channel_ind
ex>,<scrambler_seed>,<no_of_packets>,<payload_type>,<classic_le_mode>,<le_channel_type>,<tx_power>,<tx_mode
>,<hopping_type>,<ant_sel>
\r\n
```

Parameters:
<enable>: This parameter indicates that this structure belongs to permode. „0?-disable, „1?-enable <BDAddress>: This is the device address in Classic mode and Access Code in LE mode. It is a 48-bit address in hexadecimal format with colon separation. e.g., 00:23:A7:01:02:03.
<pkttype>: This is the type of packet to be transmitted, as per the Bluetooth standard. Range is from 1 to 15.
<pktlength>: This is the length of the packet, in bytes, to be transmitted. Range is from 1 to 1021.
<linktype>: This parameter indicates the Link Type – ACL, SCO, eSCO. This parameter is valid only in the Classic mode and invalid in LE mode. „0? – SCO
, „1? – ACL , „2? – eSCO.
<rate>: This parameter decides whether the transmission has to happen in Basic Data Rate or Enhanced Data Rate in Classic mode. This parameter is invalid in LE mode. „1? – Basic Data Rate , „2? or „3? – Enhanced Data Rate
<rx_channel>: This parameter indicates the Receive channel index, as per the Bluetooth standard.
<tx_channel>: This parameter indicates the Transmit channel index, as per the Bluetooth standard.
<scramblerseed>: This parameter is the initial seed to be used for whitening. It should be set to „0? to disable whitening.
<nbrofpkts>: This is the number of packets to be transmitted. This is valid only when the <tx_mode> is set to Burst mode (0).
<payloadtype>: This parameter indicates the type of payload to be transmitted.

„0? – Payload consists of zeros.
„1? – Payload consists of 0xFF?s.
„2? – Payload consists of 0x55?s
„3? – Payload consists of 0xF0?s.
„4? – Payload consists of PN9 sequence.
<protocolmode>: This parameter is used to choose between Bluetooth Classic and LE modes for the packet transmission. „1? – Classic mode , „2? – LE Mode
<lechanneltype>: This parameter indicates the channel type in LE mode.
„0? – Advertising channel , „1? – Data channel
<powerindex>: This is the transmit power (in dBm) to be used by the module. The value should be between 0 and 18.
<transmitmode>: This parameter is used to choose between Burst and Continuous modes of transmission. „0? – Burst mode , „1? – Continuous mode
<freqhopenable>: This parameter is used to choose the hopping pattern
„0? – No hopping , „1? – Fixed hopping , „2? – Random hopping
<antennaselect>: This parameter is used to select one of the two RF ports connecting to antennas. For the modules without integrated antenna, it is used to select between pins RF_OUT_1 and RF_OUT_2. For the modules with integrated antenna and U.FL connector, it used to select between the two.
„2? – RF_OUT_2/Antenna , „3? – RF_OUT_1/U.FL

AT command Ex:

```
at+rsibt_transmit = 1,00-23-a7-01-02-03,4,20,1,1,10,10,0,0,1,1,0,10,0,0,2
```

Response:

```
OK\r\n
```

5.8.2 Per receive

Description:
This will allow the configuration of the following parameters and starts the reception of packets.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_per_receive{ UINT08 type;
UINT08 enable;
UINT08 BDAddress[6];
UINT08 linktype;
UINT08 pkttype;
UINT16 pktlength;
UINT08 scramblerseed;
UINT08 rate;
UINT08 rx_channel;
UINT08 tx_channel;
UINT08 protocolmode; UINT08 lechanneltype; UINT08 freqhopenable; UINT08 antennaselect;
}RSI_BT_CMD_PER_RECEIVE;
```

Command:

```
at+rsibt_perreceive=<perenable/
disable>,<dev_addr>,<link_type>,<pkt_type>,<pkt_length>,<scrambler_seed>,<br_edr_mode>,<rx_channel_index>,<
tx_channel_index>,<classic_le_mode>,<le_channel_type>,<hopping_type>,<ant_sel>
```

Parameters:
<enable>: This parameter is used to enable or disable the permode.
„0?-disable, „1?-enable
<BDAddress>: This is the device address in Classic mode and Access Code in LE mode. It is a 48-bit address in hexadecimal format with colon separation. e.g., 00:23:A7:01:02:03.
<linktype>: This parameter indicates the Link Type – ACL, SCO, eSCO. This parameter is valid only in the Classic mode and invalid in LE mode.
„0? – SCO , „1? – ACL , „2? – eSCO
<pkttype>: This is the type of packet to be received, as per the Bluetooth standard. Range is from 1 to 15.
<pktlength>: This is the length of the packet, in bytes, to be received. Range is from 1 to 1021.
<scramblerseed>: This parameter is the initial seed to be used for whitening. It should be set to „0? to disable whitening.
<rate>: This parameter decides whether the reception has to happen in Basic Data Rate or Enhanced Data Rate in Classic mode. This parameter is invalid in LE mode. „1? – Basic Data Rate , „2? or „3? – Enhanced Data Rate
<rx_channel>: This parameter indicates the Receive channel index, as per the Bluetooth standard.
<tx_channel>: This parameter indicates the Transmit channel index, as per the Bluetooth standard.
<protocolmode>: This parameter is used to choose between Bluetooth Classic and LE modes for the packet reception. „1? – Classic mode , „2? – LE Mode
<lechanneltype>: This parameter indicates the channel type in LE mode.
„0? – Advertising channel , „1? – Data channel
<freqhopenable>: This parameter is used to choose the hopping pattern.
„0? – No hopping , „1? – Fixed hopping , „2? – Random hopping
<antennaselect>: This parameter is used to select one of the two RF ports connecting to antennas. For the modules without integrated antenna, it is used to select between pins RF_OUT_1 and RF_OUT_2. For the modules with integrated antenna and U.FL connector, it used to select between the two.
„2? – RF_OUT_2/Antenna , „3? – RF_OUT_1/U.FL

AT command Ex:

```
at+rsibt_perreceive=1,00-23-a7-01-02-03,1,4,20,0,1,10,10,1,0,0,2 \
```

Response:

```
OK\r\n
```

5.8.3 Per cw mode

Description:
This command mode is used to configure the device to transmit a continuous wave.

Binary Payload Structure

```
typedef struct rsi_bt_cmd_cw_mode {
UINT08 channel; UINT08 cw_mode; UINT08 ant_sel;
} RSI_BT_CMD_CW_MODE;
```

Command:
at+rsibt_percwmode=<channel_index>,<start/stop>,<ant_sel>
Parameters:
<channel>: This parameter indicates the channel index, as per the Bluetooth standard. The range is from 0 to 78.
< cw_mode >: This parameter is used to Start or Stop the Continuous Wave mode transmission. Values are „0?-start , „2?-stop.
<ant_sel>: This parameter is used to select one of the two RF ports connecting to antennas. For the modules without integrated antenna, it is used to select between pins RF_OUT_1 and RF_OUT_2. For the modules with integrated antenna and U.FL connector, it used to select between the two. „2? – RF_OUT_2/Antenna, „3? – RF_OUT_1/U.FL

AT command Ex:

```
at+rsibt_percwmode=10,0,3
```

Response:

```
OK\r\n
```

5.8.4 Per stats

Description:
The following statistics are returned.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_per_stats{ UINT16 crc_pass_count;
UINT16 crc_fail_count;
UINT16  RSSI;
}RSI_BT_CMD_PER_STATS;
```

Command:

```
at+rsibt_perstats
```

Parameters:
Crc_pass_count: The number of packets received which passed CRC check. Crc_fail_count: The number of packets received which failed CRC check.
RSSI: The RSSI value of the last received packet. .

AT command Ex:

```
at+rsibt_perstats
```

Response:

5.8.5 FH_MAP

Description:
This command mode is used to configure the following parameters..

Binary Payload Structure

```
typedef struct rsi_bt_afhmap {
UINT08 startchannel; UINT08 endchannel;
}  RSI_BT_AFH_MAP;
```

Command:

```
at+rsibt_afhmap=<startingchannel>,<endingchannel>
```

Parameters:
<startchannel>: This parameter indicates the starting channel index, as per the Bluetooth standard. The range is from 0 to 78.
<endchannel>: This parameter indicates the ending channel index, as per the Bluetooth standard. The range is from 0 to 78.

AT command Ex:

```
at+rsibt_afhmap=12,30
```

Response:

```
OK\r\n
```

5.9 Role change status

Description: This event tells about the status of the command "Change Master Slave Role".

Binary Payload Structure:

```
typedef struct rsi_bt_event_role_change_status {
UINT08 BDAddress[6];
UINT08  Role;
}  RSI_BT_EVENT_ROLE_CHANGE_STATUS;
```

AT Event Format:

```
AT+RSIBT_  ROLECHANGESTAT  <BDAddress>< RoleChangeStatus  >\r\n
```

Parameters:
BDAddress – Remote BD Address
RoleChangeStatus –
0 – Master role
1 – Slave role

AT event Ex:

```
AT+RSIBT_  ROLECHANGESTAT  AA-BB-CC-DD-EE-FF,1\r\n
```

5.9.1 Unbond or Disconnect status

Description:
This event tells about the status of unbonding between the BT module and remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_unbond_status {
UINT08 BDAddress[6];
}  RSI_BT_EVENT_UNBOND_STATUS;
```

AT Event Format:

```
AT+RSIBT_UNBONDRESP  <BDAddress><unbond_status>\r\n
bond_status :
0 – Success
1 - Failure
```

Parameters:
BDAddress: BD address of the remote BT device in the vicinity of the BT module.

AT event Ex:

```
AT+RSIBT_UNBONDRESP  AA-BB-CC-DD-EE-FF,1\r\n
```

5.9.2 Bond Response

Description:
This event will be sent to the host, containing the status for "Bond" command.

Binary Payload Structure:

```
typedef struct rsi_bt_event_bond_response {
UINT08 BDAddress[6];
}  RSI_BT_EVENT_BOND_RESPONSE;
```

AT Event Format:

```
AT+RSIBT_BONDRESP  <BDAddress><bond_status><error>\r\n
bond_status :
0 – Success
1 - Failure
```

Parameters:
BDAddress: BD address of the remote BT device in the vicinity of the BT module.

AT event Ex:

```
AT+RSIBT_BONDRESP  AA-BB-CC-DD-EE-FF,0\r\n
```

5.9.3 Inquiry response

Description:
This event will be sent in response to the "Inquiry" command. This event contains the details of the BT device (like device name, BD address, COD, RSSI value) in the vicinity of the BT module.

Binary Payload Structure:

```
typedef struct rsi_bt_event_inquiry_response {
UINT08 InquiryType;
UINT08 BDAddress[6];
UINT08  NameLength;
INT08 RemoteDeviceName[50];
UINT08  COD[3];
UINT08  RSSI;
}  RSI_BT_EVENT_INQUIRY_RESPONSE;
```

AT Event Format:

```
AT+RSIBT_INQRESP <InquiryType>,<BDAddress>,<NameLength>,<RemoteDeviceName>\r\n (for Standard Inquiry).
AT+RSIBT_INQRESP
< InquiryType >,< InquiryType NameLength>,<RemoteDeviceName>,<Rssi>,<cod>\r\n (for normal & extended
inquiry types with RSSI).
```

Parameters:
InquiryType:
0- Standard Inquiry.
1- Inquiry with RSSI.
2- Extended inquiry with RSSI.
BDAddress: BD address of the remote BT device in the vicinity of the BT module. This parameter will be sent by host for all inquiry types.
NameLength: Length of the remote device name.
RemoteDeviceName: Name of the remote BT device, with corresponding BD address. This parameter will be sent by host for all inquiry types. This parameter will be present only if the **"rmt_name_len"** value is non-zero.
COD: Class of the remote BT device. This parameter will be sent by host for all inquiry types.
RSSI: RSSI value between the BT module and the remote BT device. This parameter will be sent only for inquiry types 1and 2.

AT event Ex:

```
AT+RSIBT_INQRESP 2,AA-BB-CC-DD-EE-FF,7,reyax,142,7a020c\r\n
```

5.9.4 Remote device name

Description: This event will be sent to the host in response to the "Remote Name Request" command. This event contains the name of the remote BT device, which was requested by the host.

Binary Payload Structure:

```
typedef struct rsi_bt_event_remote_device_name {
UINT08 BDAddress[6];
UINT08  NameLength;
INT08 RemoteDeviceName[50];
} RSI_BT_EVENT_REMOTE_DEVICE_NAME;
```

AT Event Format:

```
AT+RSIBT_RMTDEVNAME  <  BDAddress>,<  NameLength  >,<  RemoteDeviceName  /  ERROR  >\r\n
```

Parameters:
BDAddress: BD address of the remote BT device in the vicinity of the BT module.
NameLength: Length of the remote device name. If remote device name is not found this parameter will be zero and the **"rmt_dev_name"** contains the error code.
RemoteDeviceName: Name of the remote BT device, with corresponding BD address. If the remote device name is not available or remote device is not connected, this parameter contains the corresponding error code.

AT event Ex:

```
AT+RSIBT_RMTDEVNAME  AA-BB-CC-DD-EE-FF,7,reyax\r\n
```

5.9.5 Disconnected

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

Binary Payload Structure:

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

AT Event Format:

```
AT+RSIBT_DISCONNECTED  <  BDAddress>,<reason>\r\n
```

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

AT event Ex:

```
AT+RSIBT_CLASSIC_DISCONNECTED  AA-BB-CC-DD-EE-FF,0\r\n
```

5.9.6 User confirmation Request

Description:
This event is raised when User Confirmation Request comes from the remote BT Device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_user_confirmation_request {
UINT08 BDAddress[6];
UINT32 ConfirmationValue;
}  RSI_BT_EVENT_USER_CONFIRMATION_REQUEST;
```

AT Event Format:

```
AT+RSIBT_USRCONFIRMREQ=<  BDAddress  >,<confirmation_value>\r\n
```

Parameters:
BDAddress: The remote BT device BD address.
confirmationValue: Range from 0 to 999999 in decimal.

AT event Ex:

```
AT+RSIBT_USRCONFIRMREQ AA-BB-CC-DD-EE-FF,1234\r\n
```

5.9.7 User passkey display

Description:

This event is raised when User Passkey comes from the module.

Binary Payload Structure:

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

AT Event Format:

```
AT+RSIBT_USRPASSKEYDISP < BDAddress >,<Passkey>\r\n
```

Parameters:
BDAddress: The remote BT device BD address.
passkey: Range from 0 to 999999 in decimal.

AT event Ex:

```
AT+RSIBT_USRPASSKEYDISP AA-BB-CC-DD-EE-FF,12345\r\n
```

5.9.8 User pincode request

Description:

This event is raised when User Pincode is invoked by the remote BT Device. In such a case, user shall respond with **Pincode Reply** Command.

Binary Payload Structure:

```
typedef struct rsi_bt_event_user_pincode_request {
UINT08 BDAddress[6];
} RSI_BT_EVENT_USER_PINCODE_REQUEST;
```

AT Event Format:

```
AT+RSIBT_USRPINCODEREQ < BDAddress >\r\n
```

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

AT event Ex:

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

5.9.9 User passkey request

Description:

This event is raised when User passkey request comes from the remote BT Device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_user_passkey_request {
UINT08 BDAddress[6];
} RSI_BT_EVENT_USER_PASSKEY_REQUEST;
```

AT Event Format:

```
AT+RSIBT_USRPASSKEYREQ < BDAddress >\r\n
```

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

5.9.10 Inquiry complete

Description:
This event is raised after the Inquiry is completed.

Binary Payload Structure:

There is no Payload.

AT Event Format:

```
AT+RSIBT_INQCOMPLETE\r\n
```

5.9.11 Auth complete

Description:

This event tells about the status of authentication process.

Binary Payload Structure:

```
typedef struct rsi_bt_event_auth_complete {
UINT08 BDAddress[6];
} RSI_BT_EVENT_AUTH_COMPLETE;
```

AT Event Format:

```
AT+RSIBT_AUTHENTICATION_STATUS < BDAddress >, <STATUS>, <ERROR>\r\n
```

Parameters:
BDAddress – BD Address of the remote device.

5.9.12 User linkkey Request

Description:

This event is raised when user linkkey request comes from the remote BT Device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_user_linkkey_request {
UINT08 BDAddress[6];
} RSI_BT_EVENT_USER_LINKKEY_REQUEST;
```

AT Event Format:

```
AT+RSIBT_USRLNKKEYREQ < BDAddress >\r\n
```

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

AT event Ex:

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

5.9.13 User linkkey save

Description:

This event is raised when a device is paired and linkkey for remote BT Device is given to host.

Binary Payload Structure:

```
typedef struct rsi_bt_event_user_linkkey_save {
UINT08 BDAddress[6];
UINT08 LinkKey[16];
} RSI_BT_EVENT_USER_LINKKEY_SAVE;
```

AT Event Format:

```
AT+RSIBT_USRLNKKEYSAVE < BDAddress ><LinkKey>\r\n
```

Parameters:
BDAddress – BD Address of the remote BT device.
LinkKey – Link key for the remote BT device.

AT event Ex:

```
AT+RSIBT_USRLINKKEYSAVE AA-BB-CC-DD-EE-FF,3C,A5,50,25,DC,D0,B0,AB,B7,C3,4F,4D,9,79,2C,5C\r\n
```

5.9.14 SSP Enable

Description:

This event is raised when a device pairing mode set to Simple Secure pairing mechanism.

Binary Payload Structure:

```
typedef struct rsi_bt_event_ssp_complete
{
UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
UINT08 Status;
} RSI_BT_EVENT_SSP_COMPLETE;
```

AT Event Format:

```
AT+RSIBT_SIMPLEPAIRINGCOMPLETED < BDAddress >,<status>\r\n
```

Parameters:
BDAddress – BD Address of the remote BT device.
Status-

1. Simple Pairing succeeded.
2. Simple pairing failed.

AT event Ex:

```
AT+RSIBT_SIMPLEPAIRINGCOMPLETED AA-BB-CC-DD-EE-FF,0\r\n
```

5.9.15 Mode change

Description:

This event occurs when sniff mode is enable by either remote device or local device and it is raised to indicate whenever the device changes between Active mode and Sniff mode.

Binary Payload Structure:

```
typedef struct rsi_bt_event_mode_change
{
UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
UINT08  CurrentMode;
UINT08 Reserved;
UINT16 ModeInterval;
} RSI_BT_EVENT_MODE_CHANGE;
```

AT Event Format:

```
AT+RSIBT_MODECHANGED < BDAddress >,<current mode>,< ModeInterval >\r\n
```

Parameters:
BDAddress – BD Address of the remote BT device.
CurrentMode - State the connection is currently in.

0 - Active mode

1 - Hold mode (curerntly Not supported)

2 - Sniff mode
Mode Interval - Specify a time amount specific to each state. Time Range: 2-65534

AT event Ex:

```
AT+RSIBT_MODECHANGED  AA-BB-CC-DD-EE-FF,1,192\r\n
```

5.9.16 Sniff subrating Currently, Sniff subrating mode is not supported.

Description:

This event is raised to indicate that the device has either enabled sniff subrating or the sniff subrating parameters have been renegotiated by the link manager.

Binary Payload Structure:

```
typedef struct rsi_bt_event_sniff_subrating
{
UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
UINT16  MaxTxLatency;
UINT16  MinRemoteTimeout;
UINT16 MinLocalTimeout;
} RSI_BT_EVENT_SNIFF_SUBRATING;
```

AT Event Format:

```
AT+RSIBT_SNIFFSUBRATING  <  BDAddress  >,<MaxTxlatency>,  <  MinRemoteTimeout  >,<MinLocalTimeout>\r\n
```

Parameters:
BDAddress – BD Address of the remote BT device.
MaxTxLatency- Maximum latency for data being transmitted from the local device to the remote device.
MinRemoteTimeout – The base sniff subrate timeout in baseband slots that the remote device shall use.
MinLocalTimeout- The base sniff subrate timeout in baseband slots that the local device will use.
AT event Ex:

```
AT+RSIBT_SNIFFSUBRATING  AA-BB-CC-DD-EE-FF,192,1000,1000\r\n
```

5.10 SPP events

5.10.1 SPP Receive

Description:

This event is sent to the host when data is received from remote BT device through SPP. This data_len field contains the length of data to be send to the host.

Binary Payload Structure:

```
typedef struct {
UINT16 DataLength;
UINT08 Data[200];
} RSI_BT_EVENT_SPP_RECEIVE;
```

AT Event Format:

```
AT+RSIBT_SPPRX < DataLength >< Data >\r\n
```

Parameters:

Datalength – length of the data (from 1 byte to 200 bytes).

AT event Ex:

```
AT+RSIBT_SPPRX 4,rtrt\r\n
```

5.10.2 SPP connected

Description:

This event will be sent to the host, when the connection is established between BT Module and the remote BT device based on SPP profile.

Binary Payload Structure:

```
typedef struct rsi_bt_event_spp_connected {
UINT08 BDAddress[6];
} RSI_BT_EVENT_SPP_CONNECTED;
```

AT Event Format:

```
AT+RSIBT_SPPCONNECTED < BDAddress >\r\n
```

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

AT event Ex:

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

5.10.3 SPP Disconnected

Description:

This event will be sent to the host, when the existing SPP connection of the BT Module with the remote BT device is disconnected.

Binary Payload Structure:

```
typedef struct rsi_bt_event_spp_disconnected {
UINT08 BDAddress[6];
} RSI_BT_EVENT_SPP_DISCONNECTED;
```

AT Event Format:

```
AT+RSIBT_SPPDISCONNECTED < BDAddress >\r\n
```

Parameters:
BDAddress – BD Address of the disconnected device

AT event Ex:

```
AT+RSIBT_SPPDISCONNECTEDAA-BB-CC-DD-EE-FF\r\n
```

5.11 A2DP events

5.11.1 A2DP Connected

Description:

This event is send to the host when A2DP is Connected.

Binary Payload Structure:

```
typedef struct rsi_bt_event_a2dp_connected {
UINT08 BDAddress[6];
} RSI_BT_EVENT_A2DP_CONNECTED;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_A2DP_CONNECTED<BD ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_A2DP_CONNECTED 00-23-A7-01-01-01\r\n
```

5.11.2 A2DP Disconnected

Description:

This event is send to the host when A2DP is Disconnected.

Binary Payload Structure:

```
typedef struct rsi_bt_event_a2dp_disconnected {
UINT08   BDAddress[6];
} RSI_BT_EVENT_A2DP_DISCONNECTED;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_A2DP_DISCONNECTED<BD  ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.

AT event Ex:

```
AT+RSIBT_HOST_EVENT_A2DP_DISCONNECTED  00-23-A7-01-01-01\r\n
```

5.11.3 A2DP Configured

Description:

This event is send to the host when A2DP is Configured.

Binary Payload Structure:

```
typedef struct rsi_bt_event_a2dp_configured {
UINT08   BDAddress[6];
} RSI_BT_EVENT_A2DP_CONFIGURED;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_A2DP_CONFIGURED<BD  ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.

AT event Ex:

```
AT+RSIBT_HOST_EVENT_A2DP_CONFIGURED  00-23-A7-01-01-01\r\n
```

5.11.4 A2DP Stream Open

Description:

This event is send to the host when A2DP stream is Opened.

Binary Payload Structure:

```
typedef struct rsi_bt_event_a2dp_stream_open {
UINT08   BDAddress[6];
} RSI_BT_EVENT_A2DP_STREAM_OPEN;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_A2DP_OPEN<BD  ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_A2DP_OPEN  00-23-A7-01-01-01\r\n
```

5.11.5 A2DP Started

Description:

This event is send to the host when A2DP is Started.

Binary Payload Structure:

```
typedef struct rsi_bt_event_a2dp_stream_start {
UINT08   BDAddress[6];
}  RSI_BT_EVENT_A2DP_STREAM_START;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_A2DP_START<BD  ADDRESS>  <SampleFreq>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_A2DP_DISCONNECTED  00-23-A7-01-01-01  44  \r\n
```

5.11.6 A2DP Suspend

Description:

This event is send to the host when A2DP is Suspended.

Binary Payload Structure:

```
typedef struct rsi_bt_event_a2dp_stream_suspend {
UINT08   BDAddress[6];
}  RSI_BT_EVENT_A2DP_STREAM_SUSPEND;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_A2DP_SUSPEND<BD  ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_A2DP_SUSPEND  00-23-A7-01-01-01\r\n
```

5.11.7 A2DP Abort

Description:

This event is send to the host when A2DP is Aborted.

Binary Payload Structure:

```
typedef struct rsi_bt_event_a2dp_stream_abort {
UINT08   BDAddress[6];
}  RSI_BT_EVENT_A2DP_STREAM_ABORT;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_A2DP_ABORT<BD  ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_A2DP_ABORT  00-23-A7-01-01-01\r\n
```

5.11.8 A2DP Close

Description:

This event is send to the host when A2DP is Closed.

Binary Payload Structure:

```
typedef struct rsi_bt_event_a2dp_stream_close {
UINT08   BDAddress[6];
}  RSI_BT_EVENT_A2DP_STREAM_CLOSE;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_A2DP_CLOSE<BD  ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_A2DP_CLOSE  00-23-A7-01-01-01\r\n
```

5.11.9 A2DP Stream Data

Description:

This event is send to the host when A2DP Audio data is received from remote device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_a2dp_stream_data {
UINT08   BDAddress[6];
UINT16       SBCDataLen;
UINT08       SBCData[BT_CFG_ACL_MAX_PKT_LEN];
} RSI_BT_EVENT_A2DP_STREAM_DATA;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_A2DP_STREAM_DATA <BD ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.

AT event Ex:

```
AT+RSIBT_A2DPSBCDATA
s132,\0x80`\0x03\0xe8\0x00\0x01q\0x80\0x00\0x00\0x00\0x00\0x01\0x9c\0xbd5}\0x00\0x00\0x00\0x00\0x00\0x00\0x
00\0x00\0x00wv\0xdbn\0xed\0xb6\0xdb\0xbb\0xb6\0xdbwm\0xb6\0xdd\
0xdd\0xb6\0xdb\0xbbm\0xb6\0xee\0xed\0xb6\0xdd\0xdbm\0xb7wm\0xb6\0xee\0xdbm\0xbb\0xbbm\0xb7v\0xdbm\0xdd\0xdb
m\0xbb\0xb6\0xdbn\0xee\0xdbm\0xdd\0xb6\0xdbwv\0xdbn\0xed\0xb6\0xdb\0xbb\
0xb6\0xdbwm\0xb6\0xdd\0xdd\0xb6\0xdb\0xbbm\0xb6\0xee\0xed\0xb6\0xdd\0xdbm\0xb7wm\0xb6\0xee\0xdbm\0xbb\0xbbm
\0xb7v\0xdbm\0xdd\0xdbm\0xbb\0xb6\0xdbn\0xee\0xdbm\0xdd\0xb6\0xdb \r\n
```

5.12 AVRCP Events

5.12.1 AVRCP Connect

Description:

This event is send to the host when AVRCP is Connected.

Binary Payload Structure:

```
typedef struct rsi_bt_event_avrcp_connected {
UINT08   BDAddress[6];
} RSI_BT_EVENT_AVRCP_CONNECTED;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_AVRCPCONNECTED <BD ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_AVRCPCONNECTED  00-23-A7-01-01-01\r\n
```

5.12.2 AVRCP Disconnect

Description:

This event is send to the host when AVRCP is Disconnected.

Binary Payload Structure:

```
typedef struct rsi_bt_event_avrcp_disconnected {
UINT08   BDAddress[6];
}  RSI_BT_EVENT_AVRCP_DISCONNECTED;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_AVRCPDISCONNECTED  <BD  ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_AVRCPDISCONNECTED  00-23-A7-01-01-01\r\n
```

5.12.3 AVRCP Play

Description:

This event is send to the host when play command is given from the remote device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_avrcp_play {
UINT08   BDAddress[6];
}  RSI_BT_EVENT_AVRCP_PLAY;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_AVRCPPLAY  <BD  ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.

AT event Ex:

```
AT+RSIBT_HOST_EVENT_AVRCPPLAY 00-23-A7-01-01-01\r\n
```

5.12.4 AVRCP Pause

Description:

This event is send to the host when pause command is given from the remote device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_avrcp_pause {
UINT08 BDAddress[6];
} RSI_BT_EVENT_AVRCP_PAUSE;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_AVRCPPAUSE <BD ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.

AT event Ex:

```
AT+RSIBT_HOST_EVENT_AVRCPPAUSE 00-23-A7-01-01-01\r\n
```

5.12.5 AVRCP Stop

Description:

This event is send to the host when stop command is given from the remote device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_avrcp_stop {
UINT08 BDAddress[6];
} RSI_BT_EVENT_AVRCP_STOP;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_AVRCPSTOP <BD ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.

AT event Ex:

```
AT+RSIBT_HOST_EVENT_AVRCPSTOP 00-23-A7-01-01-01\r\n
```

5.12.6 AVRCP Next

Description:

This event is send to the host when next command is given from the remote device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_avrcp_next {
UINT08   BDAddress[6];
} RSI_BT_EVENT_AVRCP_NEXT;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_AVRCPNEXT <BD ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_AVRCPNEXT 00-23-A7-01-01-01\r\n
```

5.12.7 AVRCP Previous

Description:

This event is send to the host when previous command is given from the remote device.

Binary Payload Structure:

```
typedef struct rsi_bt_event_avrcp_previous {
UINT08   BDAddress[6];
} RSI_BT_EVENT_AVRCP_PREVIOUS;
```

AT Event Format:

```
AT+RSIBT_HOST_EVENT_AVRCPPREVIOUS <BD ADDRESS>\r\n
```

Parameters:

BDaddress – BD address of remote device.
AT event Ex:

```
AT+RSIBT_HOST_EVENT_AVRCPPREVIOUS 00-23-A7-01-01-01\r\n
```

5.13 Apple IAP1 Events IAP Connected

Description:

This event indicates that the connection was established between an Apple device and the accessory.

AT Event Format:
AT+RSIBT_IAP_CONNECTED < BDAddress >,<IAP_Protocol_Ver>\r\n

Parameters:
bd_addr: BD address of the remote BT device.
IAP_Protocol_Version: Version of the IAP supported by the Apple device, which was connected. 1 for IAP1 and 2 for IAP2.

AT event Ex:
AT+RSIBT_IAP_CONNECTED AA-BB-CC-DD-EE-FF,1\r\n

5.13.1 IAP Disconnected

Description:

This event will be sent to the host, when the connection between the accessory and an Apple device get disconnected.

AT Event Format:

```
AT+RSIBT_IAP_DISCONNECTED <BDAddress >,<IAP_Protocol_Ver>\r\n
```

Parameters:
bd_addr: BD address of the remote BT device.
IAP_Protocol_Version: Version of the IAP supported by the Apple device, which was connected. 1 for IAP1 and 2 for IAP2.

AT event Ex:

```
AT+RSIBT_IAP_DISCONNECTED AA-BB-CC-DD-EE-FF,1\r\n
```

5.13.2 IAP1 Accessory Authentication Started

Description:

This indicates the start of the Accessory Authentication procedure by the Apple device.

AT Event Format:

```
AT+RSIBT_IAP1_ACCESSORY_AUTH_STARTED\r\n
```

Parameters:
None

AT event Ex:

```
AT+RSIBT_IAP1_ACCESSORY_AUTH_STARTED\r\n
```

5.13.3 IAP1 Accessory Authentication Failed

Description:

When the Accessory Authentication gets failed, it will be indicated by this event. The reason for failure is indicated by the error code.

AT Event Format:

```
AT+RSIBT_IAP1_ACCESSORY_AUTH_FAILED <error_code>\r\n
```

Parameters:
error_code: The reason for failure is indicated by this error code.

AT event Ex:

```
AT+RSIBT_IAP1_ACCESSORY_AUTH_FAILED  0x8002\r\n
```

5.13.4 IAP1 Accessory Authentication Completed

Description:

This event indicates the completion of the Authentication procedure by the Apple device.

AT Event Format:

```
AT+RSIBT_IAP1_ACCESSORY_AUTH_COMPLETED\r\n
```

Parameters: None

AT event Ex:

```
AT+RSIBT_IAP1_ACCESSORY_AUTH_COMPLETED\r\n
```

5.13.5 IAP1 Now Playing Application Bundle Name

Description:

This event contains the current playing application bundle name. This event will be generated when the music player starts playing, when the voice memo starts recording etc.

AT Event Format:

```
AT+RSIBT_IAP1_CURR_APP_BUNDLE_NAME  <name>\r\n
```

Parameters:
name: Name of the current playing application name.

AT event Ex:

```
AT+RSIBT_IAP1_CURR_APP_BUNDLE_NAME  com.apple.mobileipod\r\n
```

5.13.6 IAP1 Now Playing Application Display Name

Description:

This event contains the current playing application Display name. This event will be generated when the music player starts playing, when the voice memo starts recording etc.

AT Event Format:

```
AT+RSIBT_IAP1_CURR_APP_DISPLAY_NAME  <name>\r\n
```

Parameters:
name: Name of the current playing application name.

AT event Ex:

```
AT+RSIBT_IAP1_CURR_APP_DISPLAY_NAME  Music\r\n
```

5.13.7 IAP1 Assistive Touch Status

Description:

This event indicates a change in the status of the Assistive Touch. This event will be sent to the host, whenever there is a change (ON/OFF) in the status of Assistive Touch.

AT Event Format:

```
AT+RSIBT_IAP1_ASSISTIVE_TOUCH  <status>\r\n
```

Parameters:
status: Status of Assistive touch. 1 for ON and 0 for OFF.

AT event Ex:

```
AT+RSIBT_IAP1_ASSISTIVE_TOUCH  1\r\n
```

5.13.8 IAP1 IPodOut Status

Description:

This event denotes the status of Ipod out.

AT Event Format:

```
AT+RSIBT_IAP1_IPODOUT  <status>\r\n
```

Parameters:
status: Status of ipod out. 1 for ON and 0 for OFF.

AT event Ex:

```
AT+RSIBT_ IAP1IPODOUT  0\r\n
```

5.13.9 IAP1 Flow Control Status

Description:

This event will be sent to the host, when the connection is established between BT Module and the remote BT device based on SPP profile.

AT Event Format:

```
AT+RSIBT_IAP1_FLOW_CONTROL  <IAP_Protocol_Ver>\r\n
```

Parameters:
IAP_Protocol_Version: Version of the IAP supported by the Apple device, which was connected. 1 for IAP1 and 2 for IAP2.

AT event Ex:

```
AT+RSIBT_IAP1_FLOW_CONTROL\r\n
```

5.13.10 IAP1 Radio Tagging Status

Description:

This event indicates the radio tagging status.

AT Event Format:

```
AT+RSIBT_IAP1_RADIO_TAGGING <status>\r\n
```

Parameters:

Status	Description
0	Tagging operation successful
1	Tagging operation failed
2	Information available for tagging(all required RT)
3	Information not available for tagging

AT Event Ex:

```
AT+RSIBT_IAP1_RADIO_TAGGING 0\r\n
```

5.13.11 IAP1 Camera status

Description:

This event indicates the camera status.

AT Event Format:

```
AT+RSIBT_IAP1_CAMERA <status>\r\n
```

Parameters:

Status	Description
0	Camera App off

Status	Description
3	Preview
4	Recording

AT Event Ex:

```
AT+RSIBT_IAP1_CAMERA  3\r\n
```

5.13.12 IAP1 Database changed status

Description:

This event will be raised when there is a change in the Apple device database.

AT Event Format:

```
AT+RSIBT_IAP1_DATABASE_CHANGED\r\n
```

Parameters:
None

AT Event Ex:

```
AT+RSIBT_IAP1_DATABASE_CHANGED\r\n
```

5.13.13 IAP1 Session Space Available Notification

Description:

This event will be sent to the host, when the connection is established between BT Module and the remote BT device based on SPP profile.

AT Event Format:

```
AT+RSIBT_IAP1_SESSION_SPACE_AVAILABLE\r\n
```

Parameters:
None

AT Event Ex:

```
AT+RSIBT_IAP1_SESSION_SPACE_AVAILBLE\r\n
```

5.13.14 IAP1 Bluetooth Status

Description: This event will be sent to the host, when the connection is established between BT Module and the remote BT device based on SPP profile.

AT Event Format:

```
AT+RSIBT_IAP1_BLUETOOTH_STATUS  <IAP_Protocol_Ver>\r\n
```

Parameters :
IAP_Protocol_Version: Version of the IAP supported by the Apple device, which was connected. 1 for IAP1 and 2 for IAP2.

AT Event Ex:

```
AT+RSIBT_IAP1_BLUETOOTH_STATUS\r\
```

5.13.15 IAP1 Voiceover Parameter Changed status

Description:

This event occurs whenever there is a change in any of the voiceover parameter like voiceover volume, speech rate.

AT Event Format:

```
AT+RSIBT_IAP1_VOICEOVER_PARAM_CHANGED <param_type>,<changed_value>\r\n
```

Parameters:
param_type: Type of the parameter changed.
0 – Voiceover Volume.
1 – Voiceover speech rate.
changed_value: The changed value of the voiceover parameter.

AT Event Ex:

```
AT+RSIBT_IAP1_VOICEOVER_PARAM_CHANGED 0,150\r\n
```

5.13.16 IAP1 Application Data Session Opened

Description:

This event indicates that a session is opened by an Apple device. The details of the session was denoted by the session id and protocol index, which was assigned through the EAProtocoToken at the time of Identification Procedure. If the data session is accepted by the user, this must be acknowledgment by "" command.

AT Event Format:

```
AT+RSIBT_IAP1_OPEN_DATA_SESSION? <session_id>,<protocol_index>\r\n
```

Parameters:
session_id: ID of the session opened by the Apple device.
protocol_index: Protocol index assigned for an application at the time of Identification.

AT Event Ex:

```
AT+RSIBT_IAP1_OPEN_DATA_SESSION? 25,1\r\n
```

5.13.17 IAP1 Application Data Session Closed

Description:

This event indicates the closure of the data session by the Apple device. This must be acknowledged by "" command.

AT Event Format:

```
AT+RSIBT_IAP1_CLOSE_DATA_SESSION? <session_id>\r\n
```

Parameters:
session_id: Session ID of the application to be closed.

AT Event Ex:

```
AT+RSIBT_IAP1_CLOSE_DATA_SESSION? 25\r\n
```

5.13.18 IAP1 Ipod Data Received

Description:

This event indicates the data received from the Apple device. If the data is received correctly, an acknowledgment is sent to the Apple device.

AT Event Format:

```
AT+RSIBT_IAP1_IPOD_DATA_RECEIVED
<session_id>,<data_len>,<data>\r\n
```

Parameters:
session_id: Session ID
data_len: Length of the data received from the Apple device.
data: Actual data received from the Apple device.

AT Event Ex:

```
AT+RSIBT_IAP1_IPOD_DATA_RECEIVED 25,7,welcome\r\n
```

5.13.19 IAP1 Accessory HID Report

Description:

This event indicates the accessory HID report sent by the Apple device. For example an LED report sent to the keyboard accessory.

AT Event Format:

```
AT+RSIBT_IAP1_IPOD_DATA_RECEIVED <desc_index>,<report_type>,<len>,<report>\r\n
```

Parameters:
desc_index: HID descriptor index, already registered with the Apple device.
report_type: Input report.
len: Length of the report in bytes.
report: Device report. The report values are not the ASCII values, but the numerical values.

AT Event Ex:

```
AT+RSIBT_IAP1_ACCESSORY_HID_REPORT 0,1,3, 01 00 00 \r\n
```

5.13.20 AFH Channel Classification

Description:

This is used to read and write the channel assessment modes and to set the AFH channel classification.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_afh_channel_classification {
        UINT08      ChannelAssessmentMode;
        UINT08   Enable;
        UINT08   Channel_Classification[10];
} RSI_BT_CMD_SET_AFH_CHANNEL_CLASSIFICATION;
```

AT command format:

```
at+rsibt_afhchannelclassification=<read/write>,<enable/disable>,<channel_map>\r\n
```

Parameters:
ChannelAssessmentMode : Weather to read from or write to the controller.

Enable: Channel assessment mode Enable or Disable.

Channel_Classification: Channel map of 10 bytes.

Response Payload:
Response comes Only when we given **ChannelAssessmentMode** as 0.

```
typedef struct rsi_bt_resp_query_afh_channel_assessment_code {
   UINT08 Mode;
} RSI_BT_RESP_QUERY_AFH_CHANNEL_ASSESSMENT_MODE;
```

Resp	Description
0	ChannelAssessmentMode is Disabled
1	ChannelAssessmentMode is Enabled
All other values	Reserved for future use.

AT command Ex:

```
at+rsibt_afhchannelclassification=1,1,ff,ff,ff,ff,0,0,0,0,0,7f\r\n
```

Response:

OK\r\n

```
typedef struct rsi_bt_event_role_change_status {
   UINT08 BDAddress[6];                    /* BD address of the remote device (6 Bytes) */
   UINT08 Role;                            /* Role                  (1 Byte ) */
} RSI_BT_EVENT_ROLE_CHANGE_STATUS;
```

5.13.21 A2DP Disconnect

Description:

This is used to disconnect the A2DP connection with the remote BT device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_a2dp_disconnect {
UINT08   BDAddress[6];
}
RSI_BT_CMD_A2DP_DISCONNECT;
```

AT command format:

```
at+rsibt_a2dpdisconn=<BDAddress>\r\n
```

Parameters:

BDAddress – Remote BD address

Response Payload:

There is no response payload for this command.

AT command Ex:

```
at+rsibt_a2dpdisconn= AA-BB-CC-DD-EE-FF\r\n
```

Response:

OK\r\n

6 Embedded BT Classic Error Codes

6.1 Generic Error Codes

Error Code	Description
0x4C01	BT_A2DP_ERR_PKT_ALLOC_FAILED
0x4C02	BT_A2DP_ERR_DMA_BUSY
0x4C03	BT_A2DP_ERR_INVALIED_M4_BUF
0x4C04	BT_A2DP_ERR_PKT_ADDED_QUEUE_FULL
0x4C05	BT_A2DP_ERR_QUEUE_PKT_NULL
0x4E01	Unknown HCI command
0x4E02	Unknown Connection Identifier
0x4E03	Hardware failure
0x4E04	timeout
0x4E05	Authentication failure
0x4E06	missing
0x4E07	Memory capacity exceeded
0x4E08	Connection timeout
0x4E09	Connection limit exceeded
0x4E0A	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

Error Code	Description
0x4E1A	Unsupported remote feature
0x4E1B	SCO offset rejected
0x4E1C	SCO interval rejected
0x4E1D	SCO Air mode rejected
0x4E1E	Invalid LMP parameters
0x4E1F	Unspecified
0x4E20	Unsupported LMP Parameter
0x4E21	Role change not allowed
0x4E22	LMP response timeout
0x4E23	LMP transaction collision
0x4E24	LMP PDU not allowed
0x4E25	Encryption mode not acceptable
0x4E26	Link key cannot change
0x4E27	Requested QOS not supported
0x4E28	Instant passed
0x4E29	Pairing with unit key not supported
0x4E2A	Different transaction collision
0x4E2B	Reserved 1
0x4E2C	QOS parameter not acceptable
0x4E2D	QOS rejected
0x4E2E	Channel classification not supported
0x4E2F	Insufficient security
0x4E30	Parameter out of mandatory range
0x4E31	Reserved 2
0x4E32	Role switch pending
0x4E33	Reserved 3
0x4E34	Reserved slot violation
0x4E35	Role switch failed
0x4E36	Extended Inquiry Response too large
0x4E37	Extended SSP not supported
0X4E38	Host busy pairing
0x4E39	Wrong BD Address
0x4e3C	ADVERTISING TIMEOUT
0x4E3E	Connection Failed to be Established
0x4FF8	BT Invalid Command

Table 13: Bluetooth Generic Error Codes

6.2 Core Error Codes

Error Code	Description
0x4040	IO Fail
0x4041	Unknown
0x4042	HW Busy
0x4043	Max Sock
0x4044	Short Buf
0x4045	Max Name Size
0x4046	Invalid Args
0x4047	Sock open fail
0x4048	Timeout
0x4049	Socket state invalid
0x404A	Bad bd address
0x404B	Acl packet error
0x404C	Pool alloc fail
0x404D	Tx fail
0x404E	Connection refused
0x404F	Confirmation result
0x4050	Remote user disconnected
0x4051	Remote device not responding
0x4052	Invalid command
0x4053	Unsupported feature param value
0x4054	Thread create fail
0x4055	Sem wait fail
0x4056	Pool full
0x4057	Hw buffer overflow
0x4058	Tx buffer empty
0x4059	HCI connection fail
0x405A	Operation incomplete

Error Code	Description
0x405B	Operation cancel
0x405C	BSP error
0x4060	Sco connection fail
0x4061	No HCI connection
0x4062	Socket disconnected
0x4063	Socket timeout
0x4064	HCI connection encrypt fail
0x4065	Max acl packet buffer length
0x4066	Max nbr acl packets
0x4067	Invalid state
0x4069	Remote name fail
0x406A	Invalid response
0x4071	Invalid psm
0x4072	Psm in use
0x4073	Invalid hci connection handle
0x4074	Invalid cid
0x4075	Invalid pkt
0x4080	Scn is in use
0x4081	Max acl connections
0x4082	Sock already exists
0x4100	Invalid pdu
0x4101	Invalid pdu data element
0x4102	Sdp service not found
0x4103	Sdp attribute not found
0x4104	Sdp max service attribute
0x4200	Max RF communication channels
0x4201	RF communication disconnected
0x4202	RF communication channel not found
0x4203	RF communication invalid packet
0x4204	RF communication remote credits zero
0x4205	RF communication invalid state
0x4206	RF communication fcoff

Error Code	Description
0x4207	RF communication no service connection
0x4300	HCI connection already exists
0x4301	Max hci connection
0x4302	SCO invalid state
0x0102	Unknown
0x0103	Timeout
0x0104	Memory alloc fail
0x0106	Io fail
0x0108	Unsupported
0x0109	Short buf
0x010A	Buf overflow
0x010B	Too large buf
0x010C	Io abort
0x010D	File open fail
0x1010	Os task invalid prio
0x1011	Os task prio exists
0x1012	Os task not stopped
0x1020	Os sem max value
0x1021	Os sem not available
0x1022	Os sem reset
0x1030	Os mutex not owner
0x1031	Os mutex not locked
0x1032	Os mutex lock failed
0x1033	Os mutex try lock failed
0x1040	Os msg queue full
0x1041	Os message queue empty
0x1050	Pipe empty
0x1051	Pipe full
0x1052	Invalid len
0x1053	Pipe read in use
0x1054	Pipe write in use
0x1060	Os timer expired

Error Code	Description
0x1061	Os timer state running
0x1070	Os can not wait
0x1080	Os mem pool empty
0x1081	Os mem pool size short
0x4500	SPP not connected
0x4501	SPP not initialized
0x4FF9	Inquiry cancel command is given when device is not in Inquiry State
0x4604	SPP Tx FAIL

Table 14: BT Classic Error Codes

Error Code	Description
0x1090	Os Event queue full
0x1091	Os Event not available
0x1092	Os Event not created
0x1093	Os Event prio not created
0x1094	Os Event no event created

Table 14: BT Event Queue Error Codes

Error Code	Description
0x8000	ERR_IAP1_SUCCESS
0x8001	ERR_IAP1_UNKNOWN_DATABASE
0x8002	ERR_IAP1_COMMAND_FAILED
0x8003	ERR_IAP1_DEVICE_OUTOF_RESOURCE
0x8004	ERR_IAP1_BAD_PARAM
0x8005	ERR_IAP1_UNKNOWN_ID
0x8006	ERR_IAP1_COMMAND_PENDING
0x8007	ERR_IAP1_NOT_AUTHENTICATED
0x8008	ERR_IAP1_BAD_AUTHENTICATION_VERSION
0x8009	ERR_IAP1_ACCESSORY_PWR_REQ_FAILED
0x800A	ERR_IAP1_CERTIFICATE_INVALID

Error Code	Description
0x800B	ERR_IAP1_CERTIFICATE_PERMISSION_FAILED
0x800C	ERR_IAP1_FILE_IN_USE
0x800D	ERR_IAP1_INVALID_FILE_HANDLE
0x800E	ERR_IAP1_DIRECTORY_NOT_EMPTY
0x800F	ERR_IAP1_OPERATION_TIMED_OUT
0x8010	ERR_IAP1_COMMAND_UNAVAILABLE
0x8011	ERR_IAP1_ACC_DETECT_NOT_GROUNDED
0x8012	ERR_IAP1_SELECTION_NOT_GENIUS
0x8013	ERR_IAP1_MULTISECTION_DATA_RECV_SUCCESS
0x8014	ERR_IAP1_LINGO_BUSY
0x8015	ERR_IAP1_MAX_ACCESSORY_CONN_REACHED
0x8016	ERR_IAP1_HID_DESC_INDEX_INUSE
0x8017	ERR_IAP1_DROPPED_DATA
0x8018	ERR_IAP1_UNSUPPORTED_IPODOUT_VIDEO_SETTINGS
0x8100	ERR_IAP1_IDPS_SUCCESS
0x8101	ERR_IAP1_IDPS_TKN_FIELDS_REJECTED
0x8102	ERR_IAP1_IDPS_TKN_FIELDS_MISSING
0x8103	ERR_IAP1_IDPS_TKN_FIELDS_INCORRECT_RESEND
0x8104	ERR_IAP1_IDPS_ACCESSORY_MAY_RETRY
0x8105	ERR_IAP1_IDPS_TIMEOUT
0x8106	ERR_IAP1_IDPS_NOT_SUPPORTED
0x8107	ERR_IAP1_IDPS_INVALID_TKN_FIELDS
0x8300	ERR_IAP_CP_SUCCESS
0x8301	ERR_IAP_CP_SUCCESS
0x8302	ERR_IAP_CP_INVALID_READ_REGISTER
0x8303	ERR_IAP_CP_INVALID_WRITE_REGISTER
0x8304	ERR_IAP_CP_INVALID_SIGNATURE_LEN
0x8305	ERR_IAP_CP_INVALID_CHALLENGE_LEN
0x8306	ERR_IAP_CP_INVALID_CERTIFICATE_LEN
0x8307	ERR_IAP_CP_SIGNATURE_GENERATION
0x8308	ERR_IAP_CP_CHALLENGE_GENERATION
0x8309	ERR_IAP_CP_SIGNATURE_VERIFICATION
0x830A	ERR_IAP_CP_INVALID_PROCESS_CTRL
0x830B	ERR_IAP_CP_PROCESS_CTRL_OUTOF_SEQUENCE
0x83F0	ERR_IAP_CP_I2C_WRITE_FAILED

Table 15: BT IAP Error Codes

7 Embedded BT Classic API Library

7.1 Bluetooth Classic API File Organization

Bluetooth APIs are organized into following directory structure

	Path(Within RS9116.WC.GENR.x.x.xfolder)
BT APIs	host/apis/bt/core/
Interface Specific APIs	host/apis/intf/
HAL APIs	host/apis/hal/
BT Reference Applications	host/apis/bt/ref_apps/
BT Linux Application	RS9116.WC.GENR.xxx/host/reference_projects/LINUX/Application/bt/src
Linux USB Driver	host/reference_projects/LINUX/Driver/usb/src

7.2 Bluetooth Classic Application

he 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 BT device.

1. bt_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 inquiry, scan 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_bt_config_init API and debug prints.
3. rsi_bt_config.h and rsi_bt_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_BT_API as described below.This structure is initialized by the application using rsi_bt_init_struct API of the rsi_bt_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_bt_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_BT_CMD_SET_PROFILE_MODE uSetProfMode;
RSI_BT_CMD_SET_DISCV_MODE uSetDiscvMode;
RSI_BT_CMD_SET_CONN_MODE uSetConnMode;
RSI_BT_CMD_SET_PAIR_MODE uSetPairMode;
RSI_BT_CMD_REMOTE_NAME_REQUEST uRemNameReq;
RSI_BT_CMD_REMOTE_NAME_REQUEST_CANCEL uRemNameReqCancel;
RSI_BT_CMD_INQUIRY uInq;
RSI_BT_CMD_BOND uBond;
RSI_BT_CMD_BOND_CANCEL uBondCancel;
RSI_BT_CMD_UNBOND uUnbond;
RSI_BT_CMD_SET_PIN_TYPE uSetPinType;
RSI_BT_CMD_USER_CONFIRMATION uUserConf;
RSI_BT_CMD_PASSKEY_REPLY uPasskeyReply;
RSI_BT_CMD_PINCODE_REPLY uPincodeReply;
RSI_BT_CMD_QUERY_ROLE uGetRole;
RSI_BT_CMD_SET_ROLE uSetRole;
RSI_BT_CMD_QUERY_SERVICES uGetServ;
RSI_BT_CMD_SEARCH_SERVICE uSearchServ;
RSI_BT_CMD_LINKKEY_REPLY uLinkKeyReply;
RSI_BT_CMD_SET_SSP_MODE uSspmodeBuf;
RSI_BT_CMD_SPP_CONNECT uSPPConn;
RSI_BT_CMD_SPP_DISCONNECT uSPPDisConn;
RSI_BT_CMD_SPP_TRANSFER uSPPTransfer;
RSI_BT_CMD_SNIFF_MODE uSniffEnable;
RSI_BT_CMD_SNIFF_EXIT uSniffDisable;
RSI_BT_CMD_SNIFF_SUBRATING uSniffSubrating;
RSI_BT_CMD_PER_CW_MODE uPerCw;
RSI_BT_CMD_PER_TRANSMIT uPerTransmit;
RSI_BT_CMD_PER_RECEIVE uPerReceive;
RSI_BT_CMD_PER_STATS uPerStats;
RSI_BT_CMD_FEATURE_BIT_MAP uFeatureBitMap;
}RSI_BT_API;
```

7.3 Bluetooth Classic API Prototypes

7.3.1 Set Local name

```
INT16 rsi_bt_set_local_name(RSI_BT_CMD_SET_LOCAL_NAME *SetLocalName);
```

7.3.2 Query Local name

```
INT16 rsi_bt_query_local_name(void);
```

7.3.3 Set Local COD

```
INT16  rsi_bt_set_local_cod(RSI_BT_CMD_SET_LOCAL_COD  *SetLocalCOD);
```

7.3.4 Query Local COD

```
INT16 rsi_bt_query_local_cod(void);
```

7.3.5 Query RSSI

```
INT16  rsi_bt_query_rssi(RSI_BT_CMD_QUERY_RSSI  *GetRSSI);
```

7.3.6 Query Link Quality

```
INT16  rsi_bt_query_link_quality(RSI_BT_CMD_QUERY_LINK_QUALITY  *GetLinkQuality);
```

7.3.7 Query Local BD Address

```
INT16 rsi_bt_query_local_bd_address(void);
```

7.3.8 Initialize BT Module

```
INT16 rsi_bt_device_init(void);
```

7.3.9 Deinitialize BT Module

```
INT16 rsi_bt_device_deinit(void);
```

7.3.10 BT Antenna Select

```
INT16  rsi_bt_antenna_select(RSI_BT_CMD_ANTENNA_SELECT  *uAntennaSelect);
```

7.3.11 Set Feature Bitmap

```
INT16 rsi_bt_enable_set_feature_bitmap (RSI_BT_CMD_FEATURE_BIT_MAP *FeatureBitMap)
```

7.4 BT Classic Core Prototypes

7.4.1 Set Profile mode

```
INT16  rsi_bt_set_profle_mode(RSI_BT_CMD_SET_PROFILE_MODE  *SetProfMode  );
```

7.4.2 Set Discovery mode

```
INT16  rsi_bt_set_discover_mode(RSI_BT_CMD_SET_DISCV_MODE  *SetDiscvMode);
```

7.4.3 Get Discovery mode

```
INT16 rsi_bt_get_discover_mode(void);
```

7.4.4 Set Connectability mode

```
INT16  rsi_bt_set_conn_mode(RSI_BT_CMD_SET_CONN_MODE  *SetConnMode);
```

7.4.5 Get Connectability mode

```
INT16 rsi_bt_get_conn_mode(void);
```

7.4.6 Set Pair Mode

```
INT16  rsi_bt_set_pair_mode(RSI_BT_CMD_SET_PAIR_MODE  *SetPairMode);
```

7.5 Get Pair Mode

```
INT16 rsi_bt_query_pair_mode(void);
```

7.5.1 Remote Name Request

```
INT16  rsi_bt_remote_name_request(RSI_BT_CMD_REMOTE_NAME_REQUEST  *RemNameReq);
```

7.5.2 Remote Name Request Cancel

```
INT16  rsi_bt_remote_name_request_cancel(RSI_BT_CMD_REMOTE_NAME_REQUEST_CANCEL  *RemNameReqCancel);
```

7.5.3 Inquiry

```
INT16 rsi_bt_inquiry(RSI_BT_CMD_INQUIRY *Inq);
```

7.5.4 Inquiry Cancel

```
INT16 rsi_bt_inquiry_cancel(void);
```

7.5.5 Set EIR data

```
INT16 rsi_bt_eir_data_value(RSI_BT_CMD_SET_EIR_DATA *EirData);
```

7.5.6 Bond or Create Connection

```
INT16  rsi_bt_bond(RSI_BT_CMD_BOND  *Bond);
```

7.5.7 Bond Cancel or Create Connection Cancel

```
INT16  rsi_bt_bond_cancel(RSI_BT_CMD_BOND_CANCEL  *BondCancel);
```

7.5.8 Unbond or Disconnect

```
INT16  rsi_bt_unbond(RSI_BT_CMD_UNBOND  *Unbond);
```

7.5.9 Set Pin Type

```
INT16 rsi_bt_set_pin_type(RSI_BT_CMD_SET_PIN_TYPE *SetPinType);
```

7.5.10 Get Pin Type

```
INT16 rsi_bt_query_pin_type(void);
```

7.5.11 User Confirmation

```
INT16  rsi_bt_user_confirmation(RSI_BT_CMD_USER_CONFIRMATION  *UserConf);
```

7.5.12 Passkey Request Reply

```
INT16 rsi_bt_passkey_request_reply(RSI_BT_CMD_PASSKEY_REPLY *PasKeyReply);
```

7.5.13 Pincode Reply

```
INT16 rsi_bt_pincode_reply(RSI_BT_CMD_PINCODE_REPLY *PincodeReply);
```

7.5.14 Get Local Device Role

```
INT16 rsi_bt_query_role(RSI_BT_CMD_QUERY_ROLE *GetRole);
```

7.5.15 Set Local Device Role

```
INT16 rsi_bt_set_role(RSI_BT_CMD_SET_ROLE *SetRole);
```

7.5.16 Get Service List

```
INT16 rsi_bt_query_services(RSI_BT_CMD_QUERY_SERVICES *GetServ);
```

7.5.17 Search Service

```
INT16 rsi_bt_search_service(RSI_BT_CMD_SEARCH_SERVICE *SearchServ);
```

7.5.18 Linkkey Reply

```
INT16 rsi_bt_linkkey_reply(RSI_BT_CMD_LINKKEY_REPLY *LinkKeyReply);
```

7.5.19 Enable SSP mode

```
INT16 rsi_bt_enable_ssp_mode(RSI_BT_CMD_SET_SSP_MODE *Setsspmode);
```

7.5.20 Accept SSP confirm

```
INT16 rsi_bt_accept_ssp_confirm(RSI_BT_CMD_ACCEPT_SSP_CONFIRM *acceptssp);
```

7.5.21 Reject SSP confirm

```
INT16 rsi_bt_reject_ssp_confirm(RSI_BT_CMD_REJECT_SSP_CONFIRM *rejectssp);
```

7.5.22 Start sniff mode

```
INT16 rsi_bt_start_sniff_mode (RSI_BT_CMD_SNIFF_MODE *SniffMode);
```

7.5.23 Exit sniff mode

```
INT16 rsi_bt_exit_sniff_mode (RSI_BT_CMD_SNIFF_EXIT *SniffExit);
```

7.5.24 Sniff subrating mode Currently, Sniff Subrating mode is not supported.

```
INT16 rsi_bt_sniff_subrating_mode (RSI_BT_CMD_SNIFF_SUBRATING *SniffSubrating);
```

7.6 BT SPP Prototypes

7.6.1 SPP connect

```
INT16  rsi_bt_spp_connect(RSI_BT_CMD_SPP_CONNECT  *SPPConn);
```

7.6.2 SPP Disconnect

```
INT16  rsi_bt_spp_disconnect(RSI_BT_CMD_SPP_DISCONNECT  *SPPDisConn);
```

7.6.3 SPP Transfer

```
INT16 rsi_bt_spp_transfer(RSI_BT_CMD_SPP_TRANSFER *SPPTransfer);
```

7.7 BT A2DP Prototypes

7.7.1 A2DP Init

```
INT32 rsi_bt_a2dp_init();
```

7.7.2 A2DP Burst mode

```
INT32 rsi_bt_a2dp_burst_mode (uint8_t *remote_dev_addr, uint8_t enable);
```

7.7.3 A2DP Connect

```
INT32 rsi_bt_a2dp_connect(uint8_t *remote_dev_addr);
```

7.7.4 Send PCM or MP3 Data

```
int32_t rsi_bt_a2dp_send_pcm_mp3_data(uint8_t *remote_dev_addr, uint8_t *pcm_mp3_data, uint16_t
pcm_mp3_data_len, uint8_t audio_type);
```

7.7.5 Send SBC Data

```
int32_t rsi_bt_a2dp_send_sbc_aac_data(uint8_t *remote_dev_addr, uint8_t *sbc_aac_data, uint16_t
sbc_aac_data_len, uint8_t audio_type);
```

7.7.6 A2DP Disconnect

```
INT32 rsi_bt_a2dp_disconnect(uint8_t *remote_dev_addr);
```

7.8 BT AVRCP Prototypes

7.8.1 AVRCP Init

```
int32_t rsi_bt_avrcp_init(void);
```

7.8.2 AVRCP Connect

```
int32_t rsi_bt_avrcp_conn(uint8_t *remote_dev_addr);
```

7.8.3 AVRCP Disconnect

```
int32_t rsi_bt_avrcp_disconn(uint8_t *remote_dev_addr);
```

7.8.4 AVRCP Play

```
int32_t rsi_bt_avrcp_play(uint8_t *remote_dev_addr);
```

7.8.5 AVRCP Pause

```
int32_t rsi_bt_avrcp_pause(uint8_t *remote_dev_addr);
```

7.8.6 AVRCP Notify

```
int32_t rsi_bt_avrcp_notify (uint8_t *remote_dev_addr, uint8_t event_id, notify_val_t *p_notify_val);
```

7.8.7 AVRCP Notify response

```
int32_t rsi_bt_avrcp_reg_notify_resp (uint8_t *remote_dev_addr, uint8_t status);
```

7.8.8 AVRCP Play status response

```
int32_t rsi_bt_avrcp_play_status_resp (uint8_t *remote_dev_addr, uint8_t play_status, uint32_t song_len,
uint32_t song_pos);
```

7.8.9 AVRCP Get capcabilites response

```
int32_t rsi_bt_avrcp_cap_resp (uint8_t *remote_dev_addr, uint8_t cap_type, uint8_t nbr_caps, uint32_t
*p_caps);
```

7.8.10 AVRCP Get attribute list response

```
int32_t rsi_bt_avrcp_att_list_resp (uint8_t *remote_dev_addr, uint8_t nbr_atts, uint8_t *p_atts);
```

7.8.11 AVRCP Get attribute value list response

```
int32_t rsi_bt_avrcp_att_val_list_resp (uint8_t *remote_dev_addr, uint8_t nbr_vals, uint8_t *p_vals);
```

7.8.12 AVRCP Element attribute response

```
int32_t rsi_bt_avrcp_ele_att_resp (uint8_t *remote_dev_addr, uint8_t nbr_atts, att_text_t *p_att_list);
```

7.9 PER Commands Prototypes

7.9.1 PER Transmit

```
INT16 rsi_bt_per_transmit(RSI_BT_CMD_PER_TRANSMIT *uPerTransmit);
```

7.9.2 PER Receive

```
INT16 rsi_bt_per_receive(RSI_BT_CMD_PER_RECEIVE *uPerReceive);
```

7.9.3 PER CW Mode

```
INT16  rsi_bt_per_cw_mode(RSI_BT_CMD_PER_CW_MODE  *uPerCw);
```

7.9.4 PER Stats

```
INT16 rsi_per_stats(rsi_uPerStats *uPerStats);
```

7.9.5 Afh map

```
INT16  rsi_bt_per_afhmap(RSI_BT_CMD_AFH_MAP  *uafhmap);
```

8 Embedded BT Classic Appendix A :Sample flows

8.1 Configure BT device in Master mode

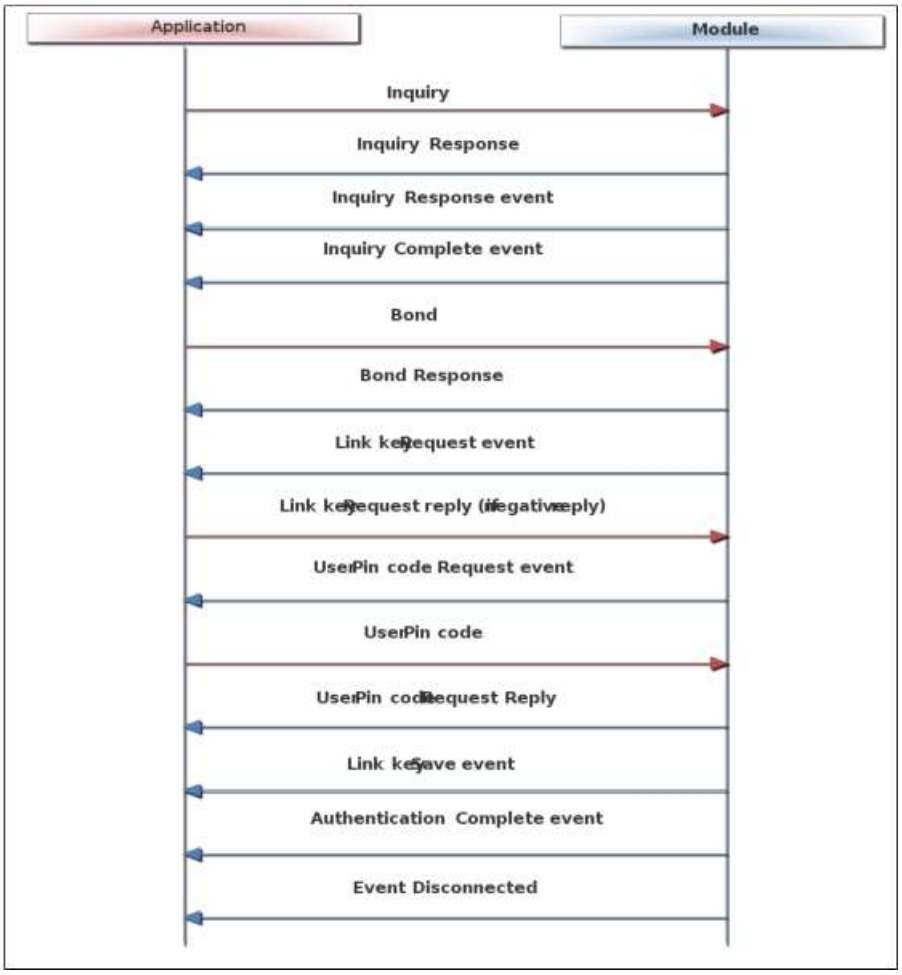


Figure 42: Sample flow in BT master Mode while Link key reply is negative

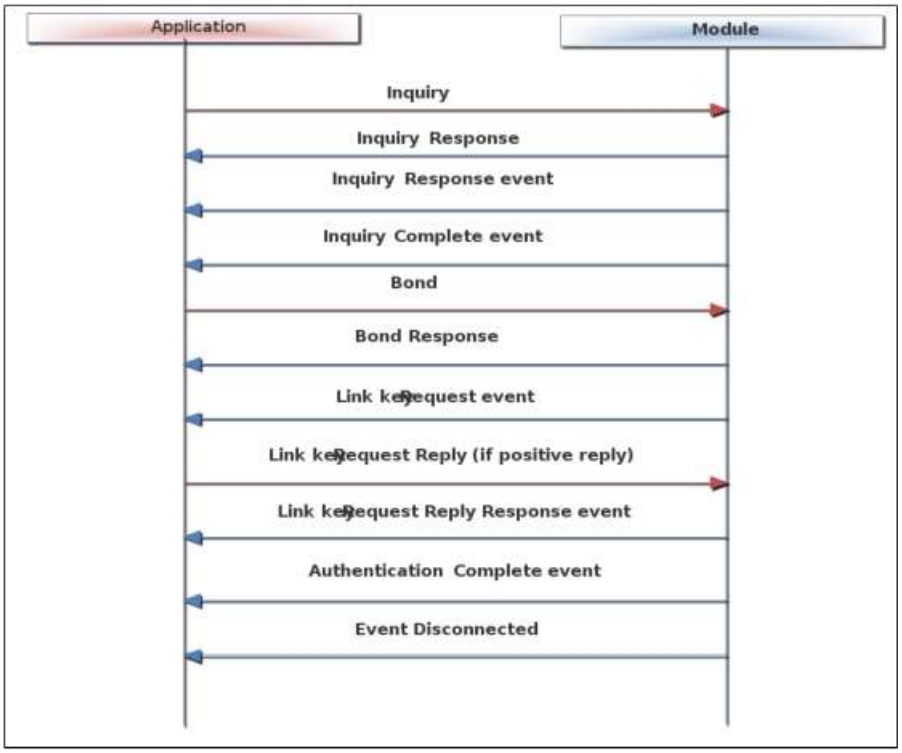


Figure 43: Sample flow in BT master Mode while Link key reply is positive

8.2 Configure BT device in Slave mode

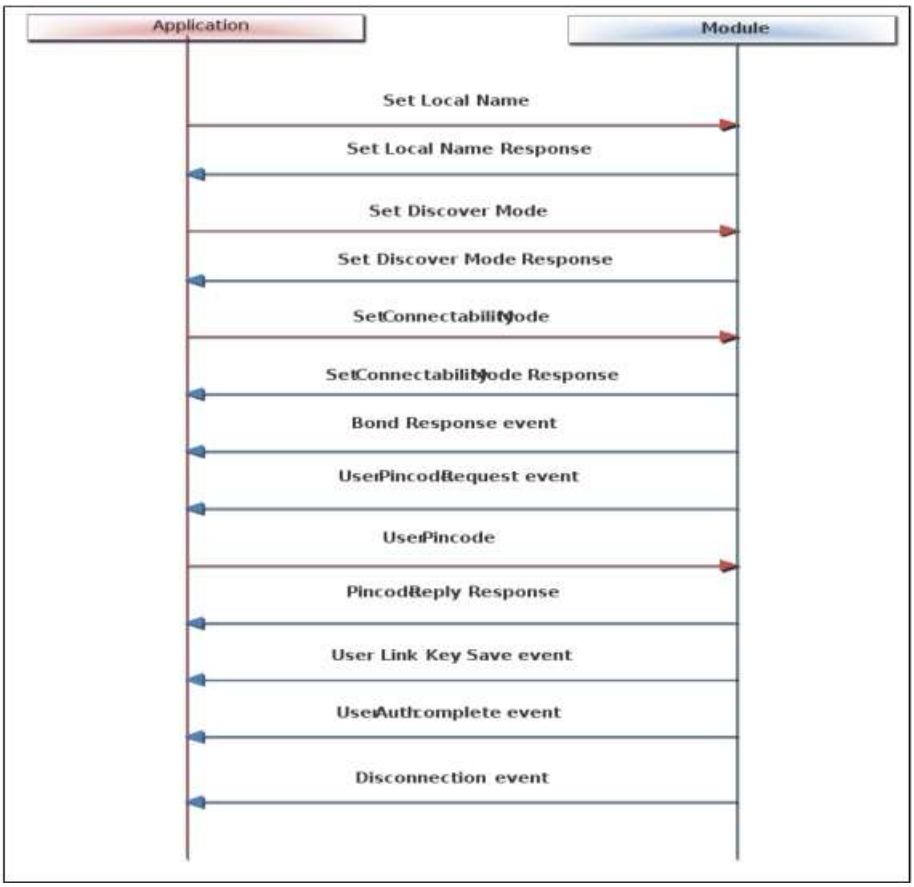


Figure 44: Sample flow in BT Slave Mode

8.3 Configure BT device in Master Mode and do SPP Tx

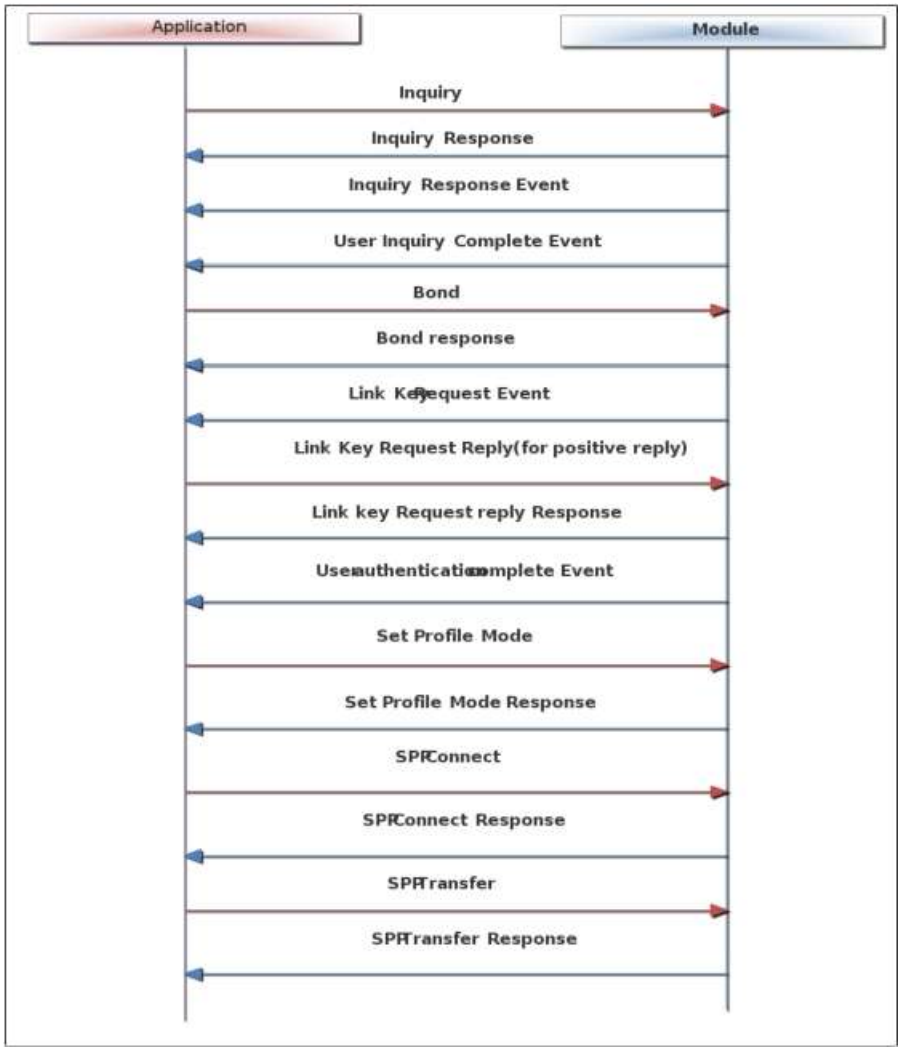


Figure 45: Sample flow in BT Master Mode and do SPP Tx

8.4 Configure BT device in Slave Mode and do SPP Tx

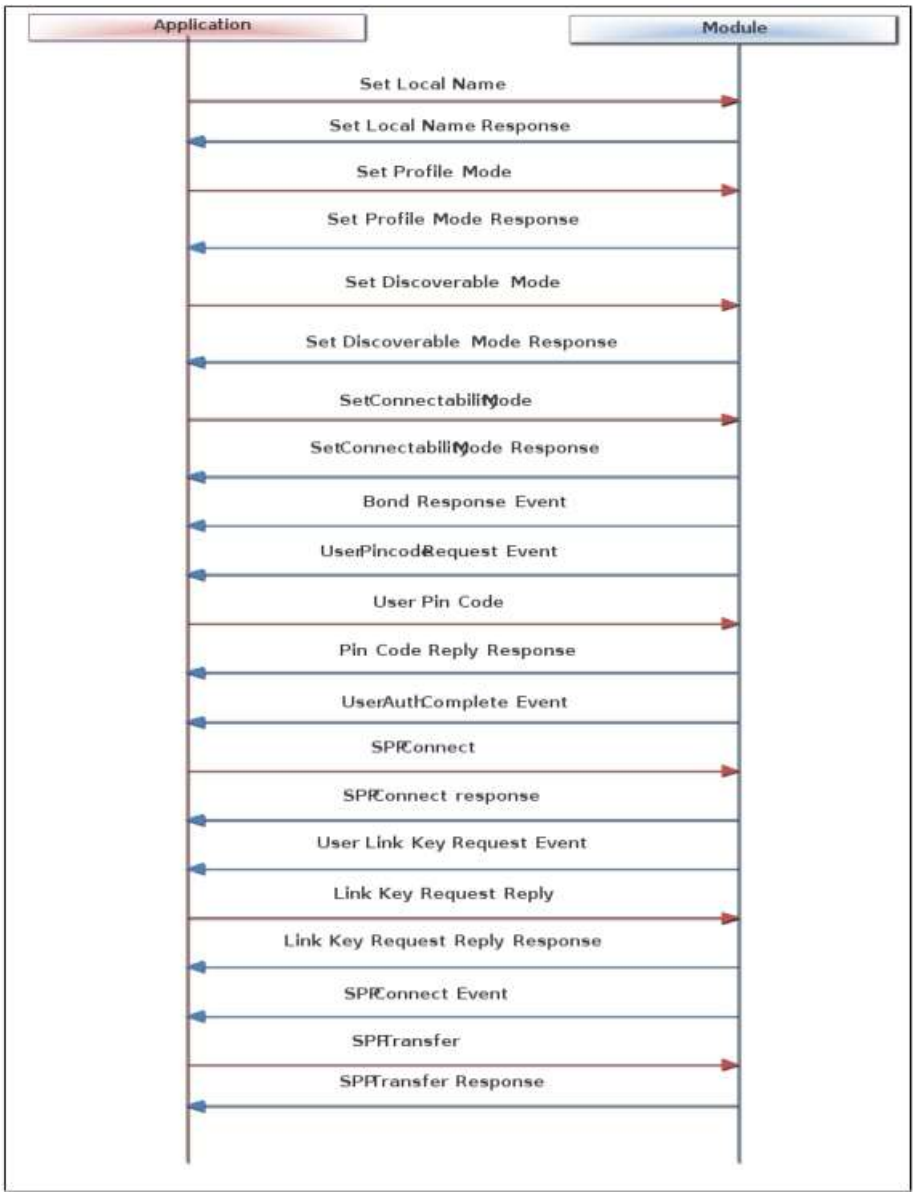
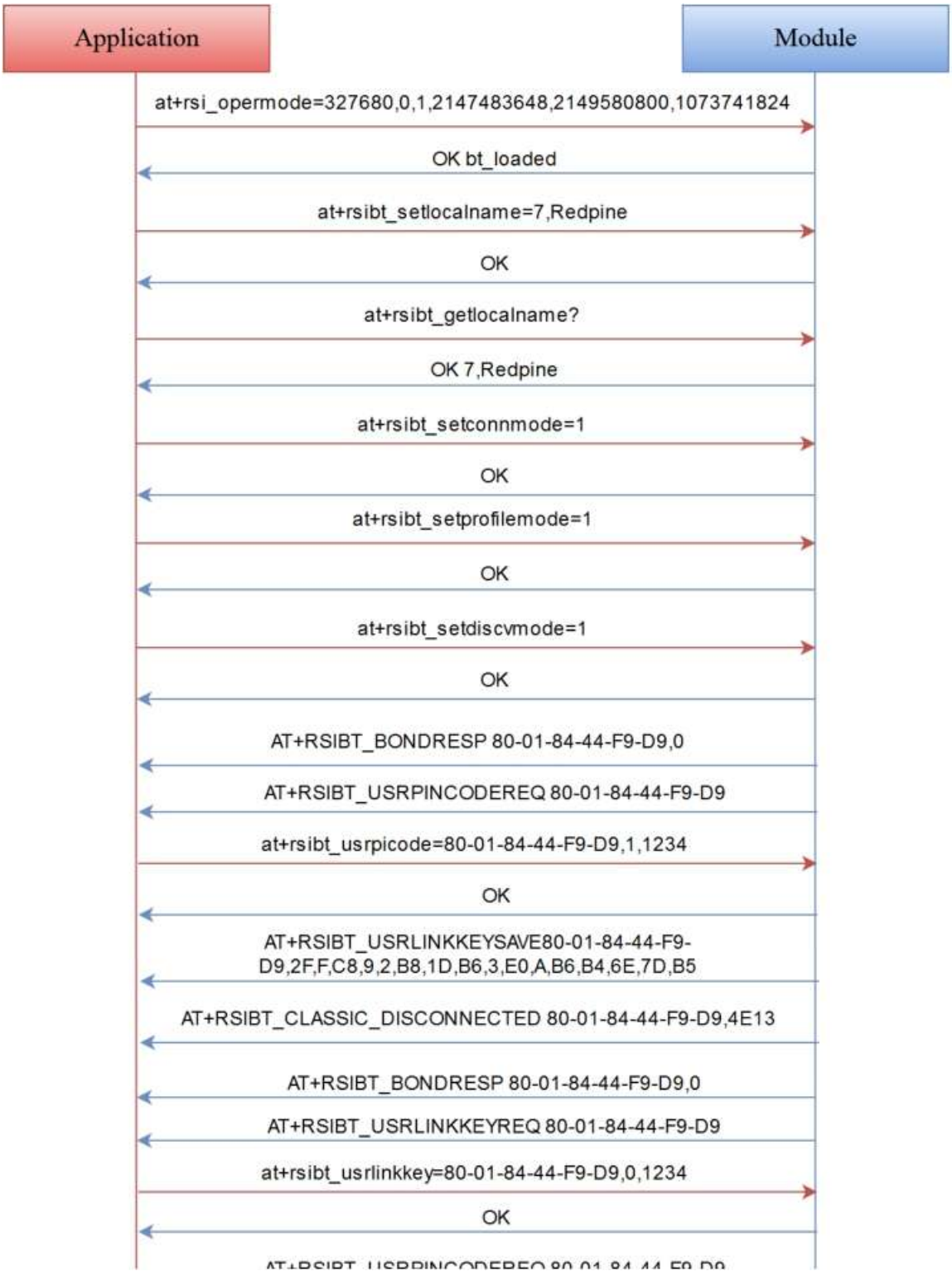


Figure 46: Sample flow in BT Slave Mode and do SPP Tx

8.5 AT command sequence to perform SPP data transfer in BT Master mode

Figure 47: AT command flow in BT Master mode

8.6 AT command sequence to perform SPP data transfer in BT Slave mode



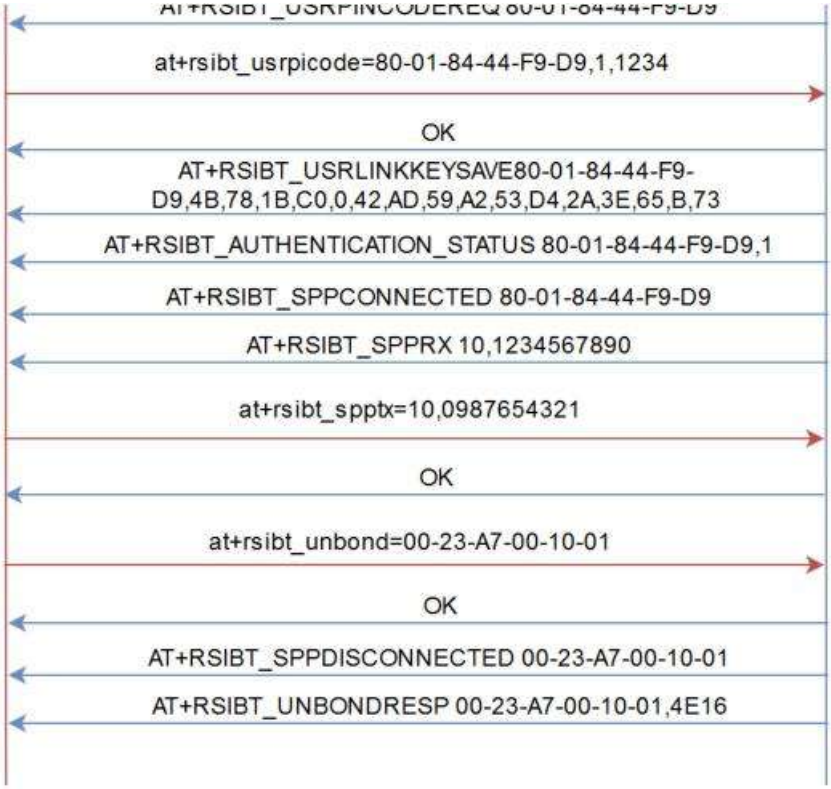


Figure 48: AT command flow in BT Slave mode