RS9113 WiSeConnect™

Simple API Porting Guide

Version 0.4

December 2016

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

Disclaimer:

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

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

Table Of Contents

Table of Figures

No table of figures entries found.

Table of Tables

No table of figures entries found.

1 Overview

RS9113-WiSeConnect™ Release package contains Wireless library/API's to facilitate user application development. RS9113 WiSeConnect™ module supports UART/SPI/USB/USB-CDC as host interfaces. This document contains description about RS9113-WiSeConnect™ API's need to port on host platform to use WiSeConnect™ Wireless libraries.

Hardware abstraction layer contains the platform specific functions which are supposed to be ported. OS abstraction layer contains the OS specific functions which are supposed to be ported if OS is required.

Note: These APIS are applicable to all the WiSeConnect variants like **WiSeConnect Plus, WiSeMCU** and **WYZBEE**. The term WiSeConnect refers to its appropriate variant.

2 Hardware Abstraction Layer

This Section contains description about HAL API's expected to be ported on host platform to use WiSeConnectTM Wireless Library.

This document focuses on the SPI and UART interfaces. RS9113 WiSeConnect acts as a SPI slave for control and data transfer. The Appendix A shows how to get simple example to study porting the SPI slave interface of RS9113 WiSeConnect with Spansion board and UART interface on Windows. For more information on RS9113 WiSeConnect and Spansion MB9BF568NBGL, see the RS9113 WiSeConnect and Spansion MB9BF568NBGL datasheets respectively.

HAL related files are available in the following path

RS9113.NBZ.WC.GEN.OSI.x.x.x\host\sapis\hal

2.1 SPI interface handling API

This section contains API's used by Wireless library to perform SPI transfer to/from module.

Driver uses `rsi_spi_transfer` function to send and receive data on SPI. In this function, platform specific SPI transfer function is supposed to be called.

2.1.1 rsi_spi_transfer

Source File: rsi_hal_mcu_spi.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int16_t rsi_spi_transfer(
            uint8_t *tx_buff,
            uint8_t *rx_buff,
            uint16_t transfer_length,
            uint8_t mode)
```

Description

This API is used by Wireless Library to perform SPI transfer from/to module.

Parameters

Parameter	Description
tx_buff	Pointer to buffer contains data to send to module. This buffer can be null, if receive only operation.
rx_buff	Pointer to buffer to hold data receive

Parameter	Description
	from module. This buffer can be null, if transfer only operation.
transfer_length	length of the transfer
mode	Mode of transfer 0 – 8 bit mode

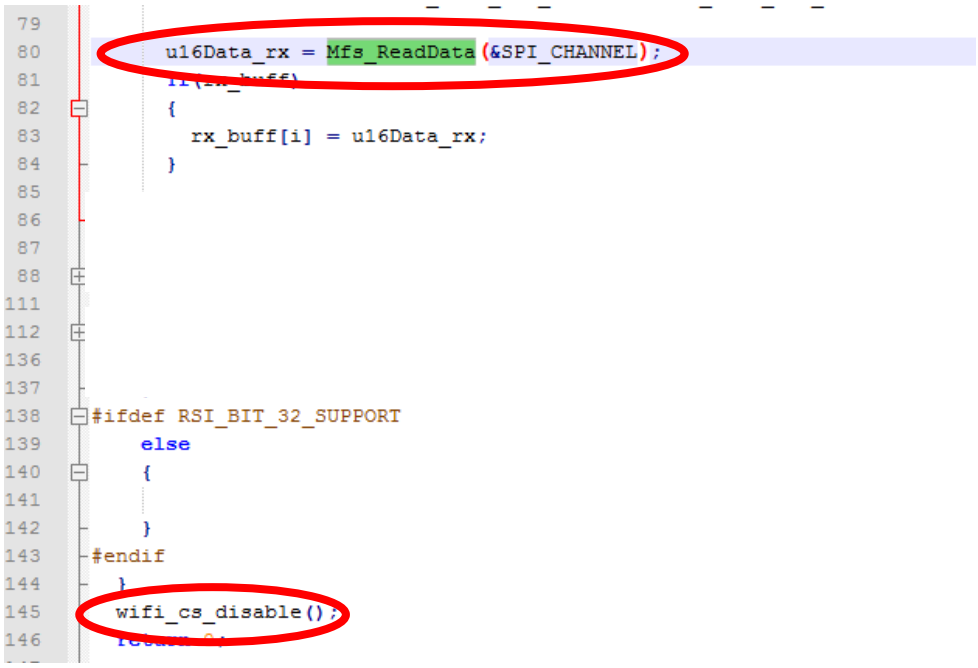
Return Values

0 – on Success

Example

The following sample code snapshot shows how to call platform specific SPI data transfer functions.

```
47  int16_t rsi_spi_transfer(uint8_t *tx_buff,uint8_t *rx_buff, uint16_t transfer_length,uint8_t mode)
48  {
49
50    volatile uint8_t      u8Reg;
51    volatile uint16_t   u16Data_tx, u16Data_rx;
52    volatile uint16_t   i;
53
54    wifi_cs_enable ();
55    for (i = 0; i < transfer_length; i++)
56    {
57  #ifdef RSI_BIT_32_SUPPORT
58      if (mode == RSI_MODE_8BIT)
59  #endif
60      {
61        u16Data_tx = 0;
62        u16Data_rx = 0;
63
64
65        do
66        {
67          u8Reg = Mfs_GetStatus(&SPI_CHANNEL, MFS_CSIO_SSR_TDRE);
68        } while ((u8Reg & MFS_CSIO_SSR_TDRE) != MFS_CSIO_SSR_TDRE);
69        if(tx_buff)
70        {
71          u16Data_tx = (uint16_t)tx_buff[i];
72        }
73        Mfs_WriteData(&SPI_CHANNEL, u16Data_tx);
74
75        do
76        {
77          u8Reg = Mfs_GetStatus(&SPI_CHANNEL, MFS_CSIO_SSR_RDRF);
78        } while ((u8Reg & MFS_CSIO_SSR_RDRF) != MFS_CSIO_SSR_RDRF);
```



2.2 UART interface handling API

This section contains API's used by Wireless library to perform UART interface with module. Following are the list of UART macros to be set for interface with module.

Parameter	Description
RSI_UART_DEVICE	Set UART device port
BAUDRATE	UART Baud rate to be set
RSI_PRE_DESC_LEN	Put Pre descriptor length
UART_HW_FLOW_CONTROL	Enable UART hardware flow control 0 - disable, 1- Enable
RSI_FRAME_DESC_LEN	Give Frame descriptor length
RSI_SKIP_CARD_READY	Skip card ready if in UART mode
RSI_USB_CDC_DEVICE	UART device or USB-CDC device 0-UART,1-USB-CDC

2.2.1 2.6.1 rsi_uart_send

Source File: rsi_hal_mcu_uart.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

int16_t rsi_uart_send(uint8_t *ptrBuf, uint16_t bufLen)

Description

This API is used by Wireless Library to perform UART send from/to module

Parameters

Parameter	Description
ptrBuf	Pointer to the buffer with the data to be sent/received.
bufLen	Number of bytes to send

Return Values

0 – on Success

Example

```
257  int16_t rsi_uart_send(uint8_t *ptrBuf, uint16_t bufLen)
258  {
259    int16_t retval = 0;
260
261  #ifdef RSI_ENABLE_DEBUG_PRINT
262    uint16_t ii = 0;
263
264    printf("\n **TX PACKET SENT** \n");
265    for(ii = 0; ii < bufLen; ii++)
266    {
267      if(ii && ((ii % 16 ) == 0) )
268      {
269        printf("\n");
270      }
271      printf(" 0x%.2x ",ptrBuf[ii]);
272    }
273    printf("\n ");
274  #endif
275  #ifdef WINDOWS
276    DWORD written;
277    retval = WriteFile(rsi_linux_app_cb.ttyfd, ptrBuf, bufLen, &written, NULL);
278  #else
279    //! write function call to write on the interface
280    retval = write(rsi_linux_app_cb.ttyfd, ptrBuf, bufLen);
281    if (retval == bufLen)
282    {
283      retval = 0;
284    }
285  #endif
286    return retval;
287  }
```

2.6.2 rsi_uart_recv

Source File: rsi_hal_mcu_uart.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

int16_t rsi_uart_recv(uint8_t *ptrBuf, uint16_t bufLen)

Description

This API is used by Wireless Library to perform UART receive from/to module

Parameters

Parameter	Description

ptrBuf	Pointer to the buffer with the data to be sent/received.
bufLen	Number of bytes to receive

Return Values

0 – on Success

Example

```
int16_t rsi_uart_recv(uint8_t *ptrBuf, uint16_t bufLen)
{

rsi_uart_byte_read();
}
```

2.6.3 rsi_uart_byte_read

Source File: rsi_hal_mcu_uart.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

uint8_t rsi_uart_byte_read(void)

Description

This API is used to read the byte data from module through UART interface.

Parameters

none

Return Values

Read character

Example

```
219  uint8_t rsi_uart_byte_read()
220  {
221    uint8_t ch = 0;
222  #ifdef WINDOWS
223    DWORD read, Err;
224    COMSTAT CST;
225  #endif
226  #ifdef WINDOWS
227   while(1)
228      {
229        ClearCommError(rsi_linux_app_cb.ttyfd, (LPDWORD)&Err, &CST);
230        if(CST.cbInQue != 0)break;
231        Sleep(10);
232      }
233      ReadFile(rsi_linux_app_cb.ttyfd, &ch, 1, &read, NULL);
234  #else
235    //! read each character
236    read(rsi_linux_app_cb.ttyfd, &ch, 1);
237  #endif
238    //! return the read character
239    return ch;
240  }
```

2.6.4 rsi_uart_init

Source File: rsi_hal_mcu_uart.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

int32_t rsi_uart_init(void)

Description

This API is used by Wireless Library to perform initialize UART interface with module

Parameters

none

Return Values

0 – on Success

!=0 – on Failure

2.6.5 rsi_uart_deinit

Source File: rsi_hal_mcu_uart.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

int32_t rsi_uart_deinit(void)

Description

This API is used by Wireless Library to perform Deinitialize UART interface with module

Parameters

none

Return Values

0 – on Success

!=0 – on Failure

2.3 Interrupt handling API

This section contain descriptions about API related to interrupts need to be ported on host platform.

2.3.1 rsi_hal_intr_config

Source File: rsi_hal_mcu_interrupt.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_intr_config(
 void (*rsi_interrupt_handler)())
```

Description

This API is used by Wireless Library to configure to receive packet pending interrupt from module.

Parameters

Parameter	Description
rsi_interrupt_handler	Pointer to a function that should be called

Parameter	Description
	in interrupt handler. Wireless library will perform specific handling in this function.

Return Values

None

Example:

In the *rsi_hal_mcu_interrupt.c* file you can set the active high level triggered for a pin which is going to be used for interrupts.

```
void rsi_hal_intr_config(void (* rsi_interrupt_handler)())
{
  stc_exint_config_t stcExtIntConfig;

  PDL_ZERO_STRUCT(stcExtIntConfig);

  stcExtIntConfig.abEnable[RSI_HAL_MODULE_INTERRUPT_PIN] = TRUE;   // INT15
  stcExtIntConfig.aenLevel[RSI_HAL_MODULE_INTERRUPT_PIN] = ExIntHighLevel;  //ExIntHighLevel;
  stcExtIntConfig.apfnExintCallback[RSI_HAL_MODULE_INTERRUPT_PIN] = rsi_interrupt_handler;

 Exint_Init(&stcExtIntConfig);



)
```

2.3.2 rsi_hal_intr_mask

Source File: rsi_hal_mcu_interrupt.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_intr_mask(void)
```

Description

This API is used by Wireless Library to mask/disable receive packet pending interrupt from module.

Parameters

None

Return Values

None

Example

```
void rsi_hal_intr_mask(void)
{
  Exint_DisableChannel(RSI_HAL_MODULE_INTERRUPT_PIN);

}
```

2.3.3 rsi_hal_intr_unmask

Source File: rsi_hal_mcu_interrupt.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_intr_unmask(void)
```

Description

This API is used by Wireless Library to unmask receive packet pending interrupt from module.

Parameters

None

Return Values

None

Example

```
93  void rsi_hal_intr_unmask(void)
94  {
95    Exint_EnableChannel(RSI_HAL_MODULE_INTERRUPT_PIN);
96
97  }
```

2.3.4 rsi_hal_intr_clear

Source File: rsi_hal_mcu_interrupt.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_intr_clear(void)
```

Description

This API is used by Wireless Library to clear receive packet pending interrupt from module, after receiving pending packet from module.

Parameters

None

Return Values

None

Example

```
110  void rsi_hal_intr_clear(void)
111  {
112    FM4_EXTI->EICL &= (0xFFFFFFFFu ^ (1u << RSI_HAL_MODULE_INTERRUPT_PIN));
113
114  }
```

2.3.5 rsi_hal_intr_pin_status

Source File: rsi_hal_mcu_interrupt.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint8_t rsi_hal_intr_pin_status(void)
```

Description

This API is used by Wireless Library to check the status of interrupt pin, to check whether packet pending from module or not.

Parameters

None

Return Values

None

Example

```
128  uint8_t rsi_hal_intr_pin_status(void)
129  {
130    //! Return interrupt status
131    //! FM4_INTREQ->IRQ018MON_f.EXTINT;
132   return FM4_GPIO->PDIR3_f.P30;
```

2.4 GPIO Port handling API

This section contains descriptions about API's related to GPIO's need to be ported on host platform, which are used by Wireless Library.

These GPIOs are used to reset the module, power save mode and for other handshakes.

These are provided in *rsi_hal_mcu_ioports.c* file in HAL folder.

2.4.1 rsi_hal_config_gpio

Source File: rsi_hal_mcu_ioports.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_config_gpio(
                uint8_t gpio_number,
                uint8_t mode,
                uint8_t value)
```

Description

This API is used by Wireless Library to configure GPIO's on host platform which are connected to module. Following are List of GPIO's used by Wireless Library.

GPIO Numbers	Description
RSI_HAL_RESET_PIN	GPIO to reset WiSeConnect Module
RSI_HAL_MODULE_INTERRUPT_PIN	GPIO to receive packet pending interrupt
RSI_HAL_WAKEUP_INDICATION_PIN	GPIO to receive module wakeup from power save indication
RSI_HAL_SLEEP_CONFIRM_PIN	GPIO to give sleep confirmation to module to go to sleep in power save
RSI_HAL_INTERFACE_READY_PIN	GPIO to receive SPI interface busy and ready indication from module

Table 1 : GPIO PIN Mapping

Note: User can change the GPIO macro definition according to Host GPIO port numbers.

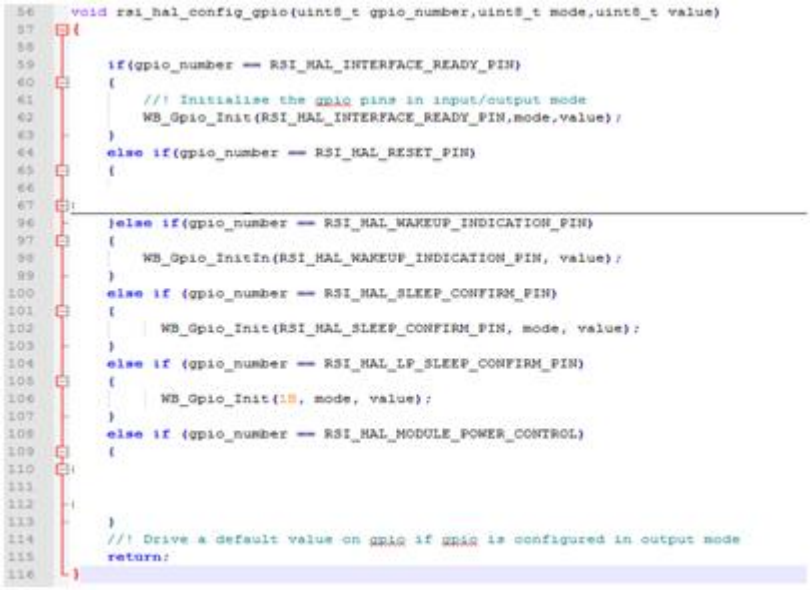
Parameters

Parameter	Description
gpio_number	Unique GPIO number (constant) used to differentiate GPIO's used by Wireless Library. HAL layer can map these GPIO number to actual GPIO number/ports used in host platform. Refer **Table 1 : GPIO PIN Mapping** for GPIO number
mode	Bit map used to configure GPIO. BIT(0) : 0 – Configure GPIO in input mode 1 – Configure GPIO in output mode BIT(1-7) : Reserved
value	Default value to drive on GPIO, if GPIO configured in output mode. 0- Low 1- High

Return Values

None

Example



```
void rsi_hal_config_gpio(uint8_t gpio_number,uint8_t mode,uint8_t value)
{

    if(gpio_number == RSI_HAL_INTERFACE_READY_PIN)
    {
        //! Initialise the gpio pins in input/output mode
        WB_Gpio_Init(RSI_HAL_INTERFACE_READY_PIN,mode,value);
    }
    else if(gpio_number == RSI_HAL_RESET_PIN)
    {

    }else if(gpio_number == RSI_HAL_WAKEUP_INDICATION_PIN)
    {
        WB_Gpio_InitIn(RSI_HAL_WAKEUP_INDICATION_PIN, value);
    }
    else if (gpio_number == RSI_HAL_SLEEP_CONFIRM_PIN)
    {
        WB_Gpio_Init(RSI_HAL_SLEEP_CONFIRM_PIN, mode, value);
    }
    else if (gpio_number == RSI_HAL_LP_SLEEP_CONFIRM_PIN)
    {
        WB_Gpio_Init(18, mode, value);
    }
    else if (gpio_number == RSI_HAL_MODULE_POWER_CONTROL)
    {


    }
    //! Drive a default value on gpio if gpio is configured in output mode
    return;
}
```

2.4.2 rsi_hal_set_gpio

Source File: rsi_hal_mcu_ioports.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_set_gpio(uint8_t gpio_number)
```

Description

This API is used by Wireless Library to driver value (1) on specified GPIO configured in output mode.

Parameters

Parameter	Description
gpio_number	Unique GPIO number (constant) used to differentiate GPIO's used by Wireless Library. HAL layer can map these GPIO number to actual GPIO number/ports used in host platform. Refer **Table 1 : GPIO PIN Mapping** for GPIO number

Parameter	Description

Return Values

None

Example

```
void rsi_hal_set_gpio(uint8_t gpio_number)
{

    if(gpio_number == RSI_HAL_INTERFACE_READY_PIN)
    {
        //! drives a high value on GPIO
        WB_Gpio_Put(RSI_HAL_INTERFACE_READY_PIN,1);
    }
    else if(gpio_number == RSI_HAL_RESET_PIN)
    {


        return;
    }
    else if(gpio_number == RSI_HAL_SLEEP_CONFIRM_PIN)
    {
        WB_Gpio_Put(RSI_HAL_SLEEP_CONFIRM_PIN,1);
    }
    else if(gpio_number == RSI_HAL_LP_SLEEP_CONFIRM_PIN)
    {
        WB_Gpio_Put(18,1);
    }
    else if(gpio_number == RSI_HAL_MODULE_POWER_CONTROL)
    {



    }
}
```

2.4.3 rsi_hal_get_gpio

Source File: rsi_hal_mcu_ioports.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint8_t rsi_hal_get_gpio(uint8_t gpio_number)
```

Description

This API is used by Wireless Library to get the value driven on specified GPIO configured in input mode.

Parameters

Parameter	Description
gpio_number	Unique GPIO number (constant) used to differentiate GPIO's used by Wireless Library. HAL layer can map these GPIO number to actual GPIO number/ports

Parameter	Description
	used in host platform. Refer **Table 1 : GPIO PIN Mapping** for GPIO number

Return Values

0 – Low

1 – High

Example

```
191  uint8_t rsi_hal_get_gpio(uint8_t gpio_number)
192  {
193      volatile uint8_t gpio_value = 0;
194
195
196      if(gpio_number == RSI_HAL_INTERFACE_READY_PIN)
197      {
198          //! Get the gpio value
199          gpio_value =  WB_Gpio_Get(38);
200      }
201      else if(gpio_number == RSI_HAL_WAKEUP_INDICATION_PIN)
202      {
203          //! Get the gpio value
204          gpio_value =  WB_Gpio_Get(RSI_HAL_WAKEUP_INDICATION_PIN);
205      }
206
207      return gpio_value;
208  }
```

2.4.4 rsi_hal_clear_gpio

Source File: rsi_hal_mcu_ioports.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_clear_gpio(uint8_t gpio_number)
```

Description

This API is used by Wireless Library to driver value (0) on specified GPIO configured in output mode.

Parameters

Parameter	Description
gpio_number	Unique GPIO number (constant) used to differentiate GPIO's used by Wireless Library. HAL layer can map these GPIO number to actual GPIO number/ports used in host platform. Refer **Table 1 : GPIO PIN Mapping** for GPIO number

Return Values

None

Example

```
222  void rsi_hal_clear_gpio(uint8_t gpio_number)
223  {
224
225      if(gpio_number == RSI_HAL_RESET_PIN)
226      {
227  +
245      ,
246      else if(gpio_number == RSI_HAL_SLEEP_CONFIRM_PIN)
247      {
248
249          WB_Gpio_Put(RSI_HAL_SLEEP_CONFIRM_PIN,0);
250
251      }
252      else if(gpio_number == RSI_HAL_LP_SLEEP_CONFIRM_PIN)
253      {
254
255          WB_Gpio_Put(1B,0);
256
257      }
258      else if(gpio_number == RSI_HAL_MODULE_POWER_CONTROL)
259      {
260  +
261
262
263      }
264      //! drives a low value on GPIO
265      return;
266  }
```

2.5 Random number generation API

This section contains API used by wireless Library to read random number.

2.5.1 rsi_get_random_number

Source File: rsi_hal_mcu_random.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint32_t rsi_get_random_number(void)
```

Description

This API is used by Wireless Library to get the random number.

Parameters

None

Return Values

32 bit random number

2.6 Timer handling API

This section contains API's used by wireless Library to perform timer handling

2.6.1 rsi_timer_start

Source File: rsi_hal_mcu_timer.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int32_t rsi_timer_start(
          uint8_t timer_node,
          uint8_t mode,
          uint8_t type,
          uint32_t duration,
          void (*rsi_timer_expiry_handler)(void))
```

Description

This API is used by Wireless Library to start the timer.

Parameters

Parameter	Description
timer_node	Unique timer number
mode	Millisecond timer/ Microsecond timer
type	Single shot timer/ period timer
duration	Time out value
rsi_timer_expiry_handler	Handler should called on timeout

Return Values

0 – on Success

!=0 – on Failure

2.6.2 rsi_timer_stop

Source File: rsi_hal_mcu_timer.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int32_t rsi_timer_stop(
            uint8_t timer_node)
```

Description

This API is used by Wireless Library to stop the timer.

Parameters

Parameter	Description
timer_node	Unique timer node number

Return Values

0 – on Success

!=0 – on Failure

2.6.3 rsi_timer_read

Source File: rsi_hal_mcu_timer.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint32_t rsi_timer_read(
           uint8_t timer_node)
```

Description

This API is used by Wireless Library to read the timer count

Parameters

Parameter	Description
timer_node	Unique timer node number

Return Values

Read timer value

2.6.4 rsi_delay_ms

Source File: rsi_hal_mcu_timer.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_delay_ms(uint32_t delay_ms)
```

Description

This API is used by Wireless Library to have delay in milli seconds

Parameters

Parameter	Description
delay_ms	delay in milli seconds

Return Values

none

2.6.5 rsi_delay_us

Source File: rsi_hal_mcu_timer.c

Path: RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_delay_us(uint32_t delay_us)
```

Description

This API is used by Wireless Library to have delay in micro seconds

Parameters

Parameter	Description
delay_ms	delay in micro seconds

Return Values

none

3 OS Interface Layer

WiSeConnect™ wireless library supports both OS and Non OS platforms. This section describes the OS wrapper's required to port to the platform specific OS with Wireless Library.

For Example: Driver uses rsi_mutex_create function to create the semaphore. In this function, OS specific mutex create functions are supposed to be called.

Note: Enable `RSI_WITH_OS macro` in SAPIS if OS is used

Below are the functions can be found in rsi_os_wrapper.c at following path
RS9113.NBZ.WC.GEN.OSI.1.x.x\host\sapis\os\free_rtos

3.1 rsi_critical_section_enrty

Prototype

```
rsi_reg_flags_t rsi_critical_section_entry()
```

Description

This API is used by Wireless Library to protect the critical section by disabling interrupts. This API implementation should contain code to disable interrupts and return the interrupt status before disabling interrupt (used to the restore interrupt status in exit critical section).

Parameters

None

Return Values

`rsi_reg_flags_t`

Platform specific data structure to hold the interrupt status before entering critical section.

3.2 rsi_critical_section_exit

Prototype

```
void  rsi_critical_section_exit(rsi_reg_flags_t xflags)
```

Description

This API is used by Wireless Library to exit critical section by restoring interrupt status which was stored while entering critical section. This API implementation should contain code to restore interrupt status based on `xflags`.

Parameters

Parameter	Description
`xflags`	hold interrupt status to restore on exit critical section

Return Values

None

3.3 rsi_mutex_create

Prototype

```
rsi_error_t rsi_mutex_create(rsi_mutex_handle_t *mutex)
```

Description

This API is used by Wireless Library to create and initialize mutex instance.

Parameters

Parameter	Description
`mutex`	Instance of `rsi_mutex_handle_t` type, where definition of `rsi_mutex_handle_t` structure should map to platform specific OS mutex structure.

Return Values

`=0` - On success

`!=0` – On Failure

3.4 rsi_mutex_lock

Prototype

```
rsi_error_t rsi_mutex_lock(
     volatile rsi_mutex_handle_t *mutex,
     uint32_t timeout_ms)
```

Description

This API is used by Wireless Library to acquire lock on mutex.

Parameters

Parameter	Description
mutex	Instance of rsi_mutex_handle_t type, where definition of rsi_mutex_handle_t structure should map to platform specific OS mutex structure.
timeout_ms	Maximum time in milliseconds to wait to acquire mutex lock. If timeout_ms is 0 then wait till mutex lock is acquired.

Return Values

=0 - On success

!=0 – On Failure

3.5 rsi_mutex_unlock

Prototype

```
rsi_error_t rsi_mutex_unlock(
           volatile rsi_mutex_handle_t *mutex)
```

Description

This API is used by Wireless Library to release lock on mutex.

Parameters

Parameter	Description
mutex	Instance of rsi_mutex_handle_t type, where definition of rsi_mutex_handle_t structure should map to platform specific OS mutex structure.

Return Values

=0 - On success

!=0 – On Failure

3.6 rsi_mutex_destory

Prototype

```
rsi_error_t rsi_mutex_destory(rsi_mutex_handle_t *mutex)
```

Description

This API is used by Wireless Library to destroy mutex instance.

Parameters

Parameter	Description
mutex	Instance of rsi_mutex_handle_t type, where definition of rsi_mutex_handle_t structure should map to platform specific OS mutex structure.

Return Values

=0 - On success

!=0 – On Failure

3.7 rsi_semaphore_create

Prototype

```
rsi_error_t rsi_semaphore_create(
            rsi_semaphore_handle_t *semaphore,
            uint32_t count)
```

Description

This API is used by Wireless Library to create and initialize semaphore instance.

Parameters

Parameter	Description
semaphore	Instance of rsi_semaphore_handle_t type, where definition of rsi_semaphore_handle_t structure should map to platform specific OS semaphore structure.
count	resource count

Return Values

=0 - On success

!=0 – On Failure

3.8 rsi_semaphore_destroy

Prototype

```
rsi_error_t rsi_semaphore_destroy(
            rsi_semaphore_handle_t *semaphore)
```

Description

This API is used by Wireless Library to destroy semaphore instance.

Parameters

Parameter	Description
semaphore	Instance of rsi_semaphore_handle_t type, where definition of rsi_semaphore_handle_t structure

Parameter	Description
	should map to platform specific OS semaphore structure.

Return Values

=0 - On success

!=0 – On Failure

3.9 rsi_semaphore_wait

Prototype

```
rsi_error_t rsi_semaphore_wait(
            rsi_semaphore_handle_t *semaphore,
            uint32_t timeout_ms )
```

Description

This API is used by Wireless Library to acquire or wait for semaphore.

Parameters

Parameter	Description
semaphore	Instance of rsi_semaphore_handle_t type, where definition of rsi_semaphore_handle_t structure should map to platform specific semaphore structure.
timeout_ms	Maximum time to wait to acquire semaphore. If timeout_ms is 0 then wait till acquire semaphore.

Return Values

=0 - On success

!=0 – On Failure

3.10 rsi_semaphore_post

Prototype

```
rsi_error_t rsi_semaphore_post(
            rsi_semaphore_handle_t *semaphore)
```

Description

This API is used by Wireless Library to release semaphore, which was acquired.

Parameters

Parameter	Description
semaphore	Instance of rsi_semaphore_handle_t type, where definition of rsi_semaphore_handle_t structure should map to platform specific OS semaphore structure.

Return Values

=0 - On success

!=0 – On Failure

3.11 rsi_semaphore_reset

Prototype

```
rsi_error_t rsi_semaphore_reset(
            rsi_semaphore_handle_t *semaphore)
```

Description

This API is used by Wireless Library to the semaphore to initial state.

Parameters

Parameter	Description
semaphore	instance of rsi_semaphore_handle_t

Parameter	Description
	type, where definition of `rsi_semaphore_handle_t` structure should map to platform specific OS semaphore structure.

Return Values

=0 - On success

!=0 – On Failure

3.12 rsi_task_create

Prototype

```
rsi_error_t rsi_task_create(
           rsi_task_function_t *task_function,
           uint8_t *stack_buffer,
           uint32_t stack_size,
           void *parameters,
           uint32_t task_priority,
           rsi_task_handle_t  *task_handle)
```

Description

This API is used by Wireless Library to create platform specific OS task/thread.

Parameters

Parameter	Description
`task_function`	Pointer to function to be executed by created thread. Prototype of the function `void (*task_function) (void *paramters)`
`stack_buffer`	Pointer to buffer to hold task stack
`stack_size`	size of the stack buffer
`parameter`	Pointer parameters to be passed to `task_function`
`task_priority`	task priority (0 – highest priority task)
`task_handle`	Instance of `rsi_task_handle_t` type

Parameter	Description
	(task control block), where definition of `rsi_task_handle_t` structure should map to platform specific OS task control block structure.

Return Values

=0 - On success

!=0 – On Failure

3.13 rsi_task_destroy

Prototype

```
void rsi_task_destroy(rsi_task_handle_t *task_handle)
```

Description

This API is used by Wireless Library to destroy task/thread, which was already created using `rsi_task_create` API.

Parameters

Parameter	Description
`task_handle`	Instance of `rsi_task_handle_t` type (task control block), where definition of `rsi_task_handle_t` structure should map to platform specific task OS control block structure.

Return Values

=0 - On success

!=0 – On Failure

4 Creation of a project for porting

This section contains the detailed description of creating a new project for porting the APIs on any host platform. This project uses binary format to communicate with the module.

4.1 Interfaces supported

Module supports SPI, UART, USB and USB-CDC interfaces. The same APIs can be used. This document explains about the SPI and UART but for USB and USB-CDC there will be similar steps.

For communicating with a host application CPU, **RSI_SPI_INTERFACE** macro should be defined for SPI interface. Find **RSI_SPI_INTERFACE** macro in the driver source codes to see more information on the SPI uses. This macro can be defined in the compiler symbols.

Similarly to enable UART interface define **RSI_UART_INTERFACE**.

4.2 Supported modes

WLAN Only

BLE Only

BT Only

ZB Only

WLAN Station + BLE

WLAN Station (TCP/IP bypass) + BT

WLAN Station + ZB End device

4.3 Directory structure

This package contains the following folders:

Build: This folder contains Makefile to compile all APIs for x 86 platforms

Docs: This folder contains API guide document which describes usage of API's

Driver : This folder contains core driver APIs.

Examples : This folder contains different example applications along with app notes for each example. User can use these applications and can develop required application accordingly

Hal: This folder contains HAL APIs which user need to ported to host platform

Include: This folder contains all header files

Nwk : This folder contains networking related APIs

Wlan : This folder contains wlan related APIs

Zigbee : This folder contains ZigBee related APIs

bt_ble : This folder contains BT Classic and BLE related APIs

common: This folder contains common files for BLE ,BT ,WLAN and ZigBee

4.4 Steps to create a project

For creating a simple project follow the below steps:

4.4.1 WLAN only project

1. Copy/ Add all the files present in the following sapis folder
   - driver/
   - hal/
   - include/
   - nwk/
   - wlan/
   - common/
2. Port the HAL API's present in the HAL folder files based on the platform. Please refer section *2 Hardware Abstraction Layer*
3. copy **`tcp_client`** folder present in the following path
   `sapis/examples/wlan/`
4. Configure the parameters in **`rsi_tcp_client.c`** file as explained in

rsi_tcp_client.pdf application note which is present in the same application folder **sapis/examples/wlan/tcp_client/**

5. Build and run the application

4.4.2 BT only project

1. Copy/ Add all the files present in the following sapis folder

   driver/

   hal/

   include/

   nwk/

   bt_ble/

   common/

2. Port the HAL API's present in the HAL folder files based on the platform. Please refer section *2 Hardware Abstraction Layer*

3. copy **bt_ssp_test_app** folder present in the following path **sapis/examples/bt/**

4. Configure the parameters in **rsi_bt_ssp_test_app.c** file as explained in **rsi_bt_ssp_test.pdf** application note which is present in the same application folder **sapis/examples/bt/bt_spp_test_app/**

5. Define **RSI_BT_ENABLE** and **RSI_BLE_ENABLE** macros.

6. Build and run the application

4.4.3 BLE only project

1. Copy/ Add all the files present in the following sapis folder

   driver/

   hal/

   include/

   nwk/

   bt_ble/

   common/

2. Port the HAL API's present in the HAL folder files based on the platform. Please refer section *2 Hardware Abstraction Layer*

3. copy **simple_chat** folder present in the following path **sapis/examples/ble/**

4. Configure the parameters in **rsi_ble_simple_chat.c** file as explained in **rsi_ble_sample_chat.pdf** application note which is present in the same application folder **sapis/examples/ble/sample_chat/**

5. Define **RSI_BT_ENABLE** and **RSI_BLE_ENABLE** macros.

6. Build and run the application

4.4.4 ZigBee only project

1. Copy/ Add all the files present in the following sapis folder

   driver/

   hal/

   include/

   nwk/

   zigbee/ rsi_zb_apis.c

   common/

2. Port the HAL API's present in the HAL folder files based on the platform. Please refer section *2 Hardware Abstraction Layer*

3. copy **`switch`** folder present in the following path **`sapis/examples/zigbee/`**

4. Configure the parameters in **`rsi_zb_config.h`** file as explained in **`rsi_zigbee_switch.pdf`** application note which is present in the same application folder **`sapis/examples/zigbee/switch/`**

5. Define **`RSI_ZB_ENABLE`** macro.

6. Remove **`rsi_bt_ble.c`** file from driver folder **`sapis/driver/`**

7. Build and run the application

4.4.5 Wlan+BT project

1. Copy/ Add all the files present in the following sapis folder

   driver/

   hal/

   include/

   nwk/

   bt_ble/

   common/

   wlan/

2. Port the HAL API's present in the HAL folder files based on the platform. Please refer section *2 Hardware Abstraction Layer*3. copy **`wlan_bt_bridge`** folder present in the following path **`sapis/examples/wlan_bt/`**

4. Configure the parameters in **`rsi_bt_app.c`** and **`rsi_wlan_app.c`** file as explained in **`rsi_wlan_station_bt_bridge.pdf`** application note which is present in the same application folder **`sapis/examples/wlan_bt/wlan_bt_bridge/`**

5. Define **`RSI_BT_ENABLE`** and **`RSI_BLE_ENABLE`** macros.

6. Build and run the application

4.4.6 Wlan+BLE project

1. Copy/ Add all the files present in the following sapis folder

   driver/

   hal/

   include/

   nwk/

   bt_ble/

   common/

   wlan/

2. Port the HAL API's present in the HAL folder files based on the platform. Please refer section *2 Hardware Abstraction Layer*

3. copy **wlan_ap_ble_bridge** folder present in the following path **sapis/examples/wlan_ble/**

4. Configure the parameters in **rsi_ble_app.c** and **rsi_wlan_ap_app.c** file as explained in **rsi_wlan_ap_ble_bridge.pdf** application note which is present in the same application folder **sapis/examples/wlan_ble/wlan_ap_ble_bridge/**

5. Define **RSI_BT_ENABLE** and **RSI_BLE_ENABLE** macros.

6. Build and run the application

4.4.7 Wlan+ZigBee project

1. Copy/ Add all the files present in the following sapis folder

   driver/

   hal/

   include/

   nwk/

   zigbee/ rsi_zb_apis.c

   common/

   wlan/

2. Port the HAL API's present in the HAL folder files based on the platform. Please refer section *2 Hardware Abstraction Layer*

3. copy **wlan_zigbee_switch** folder present in the following path **sapis/examples/wlan_zigbee/**

4. Configure the parameters in **rsi_wlan_app.c** and **rsi_zb_app.c** file as explained in **rsi_wlan_zigbee_switch.pdf** application note which is present in the same application folder **sapis/examples/wlan_zigbee/wlan_zigbee_switch/**

5. Define **`RSI_ZB_ENABLE`** macro.
6. Remove **`rsi_bt_ble.c`** file from driver folder **`sapis/driver/`**
7. Build and run the application

5 Appendix A

SPI

Following folder contains reference project for Spansion micro controller which does not use OS:

sapis/ platforms/spansion_MB9BF568NBGL/no_os/

Reference project for Spansion micro controller which uses FreeRTOS is available at following path:

sapis/platforms/spansion_MB9BF568NBGL/freeRTOS/

UART

A reference project is available for UART at following path.

RS9113.NBZ.WC.GEN.OSI.x.x.x\host\sapis\platforms\windows_uart

User can port similarly on the microcontroller platform.