NEW RADIO COMMUNICATION FOR NEW ERA

NRC7394 Evaluation Kit User Guide

(Standalone SDK API)

Ultra-low power & Long-range Wi-Fi

Ver 1.3.1
Dec. 12, 2024

NEWRACOM, Inc.

NRC7394 Evaluation Kit User Guide (Standalone SDK API)
Ultra-low power & Long-range Wi-Fi

Office

Newracom, Inc.
505 Technology Drive, Irvine, CA 92618 USA
http://www.newracom.com

Contents

1 Overview 16
2 General 17
2.1.1 Error Type 17
3 Wi-Fi 18
3.1 Data Type 18
3.1.1 API Status Return Value 18
3.1.2 Device Mode 18
3.1.3 Wi-Fi State 19
3.1.4 Country Code 19
3.1.5 Security Mode 20
3.1.6 Bandwidth 20
3.1.7 IP Mode 20
3.1.8 Address status 21
3.1.9 Scan mode 21
3.1.10 SCAN_RESULT 21
3.1.11 SCAN_RESULTS 22
3.1.12 AP_INFO 22
3.1.13 STA State 22
3.1.14 STA_INFO 23
3.1.15 STA_LIST 23
3.1.16 Tx Power Type 23
3.1.17 Guard Interval(GI) Type 23
3.1.18 Version Type 24
3.1.19 Ignore Broadcast SSID 24
3.1.20 SAE PWE 24
3.1.21 EAP 25
3.1.22 Network mode 25
3.1.23 TID 25
3.2 Function Call 26
3.2.1 nrc_wifi_get_device_mode 26
3.2.2 nrc_wifi_get_mac_address 26
3.2.3 nrc_wifi_get_tx_power 27
3.2.4 nrc_wifi_set_tx_power 27
3.2.5 nrc_wifi_get_rssi 28

3.2.6 nrc_wifi_get_average_rssi 28
3.2.7 nrc_wifi_get_snr 28
3.2.8 nrc_wifi_get_rate_control 29
3.2.9 nrc_wifi_set_rate_control 29
3.2.10 nrc_wifi_get_mcs_info 30
3.2.11 nrc_wifi_get_mcs 30
3.2.12 nrc_wifi_set_mcs 31
3.2.13 nrc_wifi_get_cca_threshold 31
3.2.14 nrc_wifi_set_cca_threshold 31
3.2.15 nrc_wifi_set_tx_time 32
3.2.16 nrc_wifi_enable_duty_cycle 32
3.2.17 nrc_wifi_disable_duty_cycle 33
3.2.18 nrc_wifi_tx_avaliable_duty_cycle 34
3.2.19 nrc_wifi_get_state 34
3.2.20 nrc_wifi_add_network 34
3.2.21 nrc_wifi_remove_network 35
3.2.22 nrc_wifi_country_from_string 35
3.2.23 nrc_wifi_country_to_string 35
3.2.24 nrc_wifi_get_country 36
3.2.25 nrc_wifi_set_country 36
3.2.26 nrc_wifi_get_channel_bandwidth 37
3.2.27 nrc_wifi_get_channel_freq 37
3.2.28 nrc_wifi_set_channel_freq 38
3.2.29 nrc_wifi_set_channel_freq_bw 38
3.2.30 nrc_wifi_set_ssid 38
3.2.31 nrc_wifi_get_bssid 39
3.2.32 nrc_wifi_set_bssid 39
3.2.33 nrc_wifi_softap_get_ignore_broadcast_ssid 40
3.2.34 nrc_wifi_softap_set_ignore_broadcast_ssid 40
3.2.35 nrc_wifi_set_security 41
3.2.36 nrc_wifi_set_eap_security 41
3.2.37 nrc_wifi_set_pmk 42
3.2.38 nrc_wifi_set_sae_pwe 43
3.2.39 nrc_wifi_get_sae_pwe 43
3.2.40 nrc_wifi_get_scan_freq 44
3.2.41 nrc_wifi_set_scan_freq 44
3.2.42 nrc_wifi_get_scan_freq_nons1g 45
3.2.43 nrc_wifi_set_scan_freq_nons1g 45

3.2.44 nrc_wifi_get_aid 46
3.2.45 nrc_wifi_scan 47
3.2.46 nrc_wifi_scan_ssid 47
3.2.47 nrc_wifi_scan_timeout 48
3.2.48 nrc_wifi_scan_results 48
3.2.49 nrc_wifi_abort_scan 49
3.2.50 nrc_wifi_connect 49
3.2.51 nrc_wifi_disconnect 49
3.2.52 nrc_wifi_wps_pbc 50
3.2.53 nrc_wifi_wps_cancel 50
3.2.54 nrc_wifi_softap_set_conf 51
3.2.55 nrc_wifi_softap_set_bss_max_idle 52
3.2.56 nrc_wifi_softap_get_max_num_sta 52
3.2.57 nrc_wifi_softap_set_max_num_sta 53
3.2.58 nrc_wifi_softap_set_ip 53
3.2.59 nrc_wifi_softap_start 54
3.2.60 nrc_wifi_softap_start_timeout 54
3.2.61 nrc_wifi_softap_stop 55
3.2.62 nrc_wifi_softap_disassociate 55
3.2.63 nrc_wifi_softap_deauthenticate 56
3.2.64 nrc_wifi_softap_start_dhcp_server 56
3.2.65 nrc_wifi_softap_stop_dhcp_server 57
3.2.66 nrc_wifi_softap_get_sta_list 57
3.2.67 nrc_wifi_softap_get_sta_by_addr 57
3.2.68 nrc_wifi_softap_get_sta_num 58
3.2.69 nrc_wifi_softap_get_beacon_interval 58
3.2.70 nrc_wifi_softap_set_beacon_interval 59
3.2.71 nrc_wifi_register_event_handler 59
3.2.72 nrc_wifi_unregister_event_handler 60
3.2.73 nrc_addr_get_state 60
3.2.74 nrc_wifi_get_ip_mode 60
3.2.75 nrc_wifi_set_ip_mode 61
3.2.76 nrc_wifi_get_ip_address 61
3.2.77 nrc_wifi_set_ip_address 62
3.2.78 nrc_wifi_stop_dhcp_client 62
3.2.79 nrc_wifi_set_dns 63
3.2.80 nrc_wifi_add_etharp 63
3.2.81 nrc_wifi_send_addba 64

3.2.82 nrc_wifi_send_delba 64
3.2.83 nrc_wifi_set_tx_aggr_auto 65
3.2.84 nrc_wifi_set_passive_scan 65
3.2.85 nrc_wifi_set_simple_bgscan 66
3.2.86 nrc_wifi_get_ap_info 66
3.2.87 nrc_wifi_set_rf_power 67
3.2.88 nrc_wifi_set_use_4address 67
3.2.89 nrc_wifi_get_use_4address 67
3.2.90 nrc_get_hw_version 68
3.2.91 nrc_wifi_get_gi 69
3.2.92 nrc_wifi_set_gi 69
3.2.93 nrc_wifi_set_beacon_loss_detection 69
3.2.94 nrc_wifi_get_listen_interval 70
3.2.95 nrc_wifi_set_listen_interval 70
3.2.96 nrc_wifi_get_mic_scan 71
3.2.97 nrc_wifi_set_mic_scan 71
3.2.98 nrc_wifi_set_enable_auth_control 72
3.2.99 nrc_wifi_get_enable_auth_control 72
3.2.100 nrc_wifi_set_auth_control_param 73
3.2.101 nrc_wifi_get_auth_control_param 73
3.2.102 nrc_wifi_set_auth_control_scale 74
3.2.103 nrc_wifi_get_auth_control_scale 74
3.2.104 nrc_wifi_get_auth_current_ti 74
3.3 Callback Functions & Events 75
4 System 76
4.1 Function Call 76
4.1.1 nrc_get_rtc 76
4.1.2 nrc_reset_rtc 76
4.1.3 nrc_sw_reset 76
4.1.4 nrc_get_user_factory 77
4.1.5 nrc_get_user_factory_info 77
4.1.6 nrc_led_trx_init 78
4.1.7 nrc_led_trx_deinit 78
4.1.8 nrc_wdt_enable 78
4.1.9 nrc_wdt_disable 79
4.1.10 nrc_set_app_version 79
4.1.11 nrc_get_app_version 79
4.1.12 nrc_set_app_name 80

4.1.13 nrc_get_app_name 80
4.1.14 nrc_get_sdk_version 80
4.1.15 nrc_set_flash_device_info 81
4.1.16 nrc_get_flash_device_info 81
4.1.17 nrc_get_user_data_area_address 81
4.1.18 nrc_get_user_data_area_size 82
4.1.19 nrc_erase_user_data_area 82
4.1.20 nrc_write_user_data 82
4.1.21 nrc_read_user_data 83
4.1.22 nrc_get_xtal_status 83
5 DMA 84
5.1 Data Type 84
5.1.1 DMA Errors 84
5.1.2 DMA Burst Size 84
5.1.3 DMA Width 84
5.1.4 DMA AHBM 85
5.1.5 DMA Peripheral ID 85
5.1.6 DMA Peripheral 85
5.1.7 DMA Descriptor 86
5.2 Function Call 86
5.2.1 nrc_dma_enable 86
5.2.2 nrc_dma_disable 86
5.2.3 nrc_dma_is_enabled 87
5.2.4 nrc_dma_get_channel 87
5.2.5 nrc_dma_valid_channel 87
5.2.6 nrc_dma_peri_init 88
5.2.7 nrc_dma_config_m2m 88
5.2.8 nrc_dma_config_m2p 89
5.2.9 nrc_dma_config_p2m 90
5.2.10 nrc_dma_config_p2p 90
5.2.11 nrc_dma_start 91
5.2.12 nrc_dma_stop 91
5.2.13 nrc_dma_busy 92
5.2.14 nrc_dma_src_addr 92
5.2.15 nrc_dma_dest_addr 92
5.2.16 nrc_dma_desc_print 93
5.2.17 nrc_dma_desc_init 93
5.2.18 nrc_dma_desc_link 94

5.2.19 nrc_dma_desc_set_addr 94
5.2.20 nrc_dma_desc_set_addr_inc 94
5.2.21 nrc_dma_desc_set_size 95
5.2.22 nrc_dma_desc_set_width 95
5.2.23 nrc_dma_desc_set_bsize 96
5.2.24 nrc_dma_desc_set_inttc 96
5.2.25 nrc_dma_desc_set_ahb_master 97
5.2.26 nrc_dma_desc_set_protection 97
5.3 Callback Functions & Events 98
6 UART 99
6.1 Data Type 99
6.1.1 Channel 99
6.1.2 UART Data Bit 99
6.1.3 UART Stop Bit 99
6.1.4 UART Parity Bit 100
6.1.5 UART Hardware Flow Control 100
6.1.6 UARTFIFO 100
6.1.7 UART Configuration 100
6.1.8 UART Interrupt Type 101
6.2 Function Call 101
6.2.1 nrc_uart_set_config 101
6.2.2 nrc_hw_set_channel 101
6.2.3 nrc_uart_get_interrupt_type 102
6.2.4 nrc_uart_set_interrupt 102
6.2.5 nrc_uart_clear_interrupt 102
6.2.6 nrc_uart_put 103
6.2.7 nrc_uart_get 103
6.2.8 nrc_uart_register_interrupt_handler 104
6.2.9 nrc_uart_console_enable 104
6.3 Callback Functions & Events 105
7 UART DMA 106
7.1 Data Type 106
7.1.1 UART Device Configuration 106
7.1.2 UART DMA buffer 106
7.1.3 UART RX Parameter 106
7.1.4 UART and DMA configuration 106
7.2 Function Call 107
7.2.1 nrc_uart_dma_open 107

7.2.2 nrc_uart_dma_close 107
7.2.3 nrc_uart_dma_change 107
7.2.4 nrc_uart_dma_read 108
7.2.5 nrc_uart_write 108
7.3 Callback Functions & Events 108
8 GPIO 110
8.1 Data Type 110
8.1.1 GPIO Pin 110
8.1.2 GPIO Direction 110
8.1.3 GPIO Mode 110
8.1.4 GPIO Level 110
8.1.5 GPIO Alternative Function 111
8.1.6 GPIO Configurations 111
8.1.7 GPIO Interrupt Trigger Mode 111
8.1.8 GPIO Interrupt Trigger Level 111
8.2 Function Call 112
8.2.1 nrc_gpio_config 112
8.2.2 nrc_gpio_output 112
8.2.3 nrc_gpio_outputb 112
8.2.4 nrc_gpio_input 113
8.2.5 nrc_gpio_inputb 113
8.2.6 nrc_gpio_trigger_config 114
8.2.7 nrc_gpio_register_interrupt_handler 114
8.3 Callback Functions & Events 115
9 I2C 116
9.1 Data Type 116
9.1.1 I2C_CONTROLLER_ID 116
9.1.2 I2C_WIDTH 116
9.1.3 I2C_CLOCK_SOURCE 116
9.1.4 i2c_device_t 117
9.2 Function Call 117
9.2.1 nrc_i2c_init 117
9.2.2 nrc_i2c_enable 117
9.2.3 nrc_i2c_reset 118
9.2.4 nrc_i2c_start 118
9.2.5 nrc_i2c_stop 119
9.2.6 nrc_i2c_writebyte 119
9.2.7 nrc_i2c_readbyte 119

10 ADC 121
10.1 Data Type 121
10.1.1 ADC Channel 121
10.2 Function Call 121
10.2.1 nrc_adc_init 121
10.2.2 nrc_adc_deinit 121
10.2.3 nrc_adc_get_data 122
11 PWM 123
11.1 Data Type 123
11.1.1 PWM Channel 123
11.2 Function Call 124
11.2.1 nrc_pwm_hw_init 124
11.2.2 nrc_pwm_set_config 124
11.2.3 nrc_pwm_set_enable 125
12 SPI 126
12.1 Data Type 126
12.1.1 SPI Mode 126
12.1.2 SPI Frame Bits 126
12.1.3 SPI Controller ID 127
12.1.4 spi_device_t 127
12.2 Function Call 128
12.2.1 nrc_spi_master_init 128
12.2.2 nrc_spi_init_cs 128
12.2.3 nrc_spi_enable 128
12.2.4 nrc_spi_start_xfer 129
12.2.5 nrc_spi_stop_xfer 129
12.2.6 nrc_spi_xfer 129
12.2.7 nrc_spi_writebyte_value 130
12.2.8 nrc_spi_readbyte_value 131
12.2.9 nrc_spi_write_values 131
12.2.10 nrc_spi_read_values 132
12.2.11 nrc_spi_slave_init 132
12.2.12 nrc_spi_slave_read 132
12.2.13 nrc_spi_slave_write 133
13 SPI DMA 134
13.1 Function Call 134
13.1.1 spi_dma_init 134

13.1.2 spi_dma_write 134
13.1.3 spi_dma_read 134
14 HTTP Client 136
14.1 Data Type 136
14.1.1 HTTP Client Return Types 136
14.1.2 HTTP Client Connection Handle 136
14.1.3 SSL Certificate Structure 137
14.1.4 HTTP Client Data Type 137
14.2 Function Call 137
14.2.1 nrc_httpc_get 137
14.2.2 nrc_httpc_post 138
14.2.3 nrc_httpc_put 139
14.2.4 nrc_httpc_delete 139
14.2.5 nrc_httpc_delete 140
14.2.6 nrc_httpc_recv_response 141
14.2.7 nrc_httpc_close 141
15 FOTA 142
15.1 Data Type 142
15.1.1 FOTA Information 142
15.1.2 Broadcast FOTA mode 142
15.2 Function Call 143
15.2.1 nrc_fota_is_support 143
15.2.2 nrc_fota_write 143
15.2.3 nrc_fota_erase 143
15.2.4 nrc_fota_set_info 144
15.2.5 nrc_fota_update_done 144
15.2.6 nrc_fota_update_done_bootloader 144
15.2.7 nrc_fota_cal_crc 145
15.2.8 nrc_bcast_fota_init 145
15.2.9 nrc_bcast_fota_set_mode 146
15.2.10 nrc_bcast_fota_enable 146
16 Power save 147
16.1 Data Type 147
16.1.1 Power Save Wakeup Source 147
16.1.2 Power Save Wakeup Reason 147
16.2 Function Call 147
16.2.1 nrc_ps_deep_sleep 147
16.2.2 nrc_ps_sleep_alone 148

16.2.3 nrc_ps_sleep_forever 148
16.2.4 nrc_ps_wifi_tim_deep_sleep 149
16.2.5 nrc_ps_clear_sleep_mode 149
16.2.6 nrc_ps_set_gpio_wakeup_pin 150
16.2.7 nrc_ps_set_gpio_wakeup_pin2 150
16.2.8 nrc_ps_set_wakeup_source 151
16.2.9 nrc_ps_wakeup_reason 151
16.2.10 nrc_ps_wakeup_gpio_ext 151
16.2.11 nrc_ps_set_gpio_direction 152
16.2.12 nrc_ps_set_gpio_out 152
16.2.13 nrc_ps_set_gpio_pullup 152
16.2.14 nrc_ps_add_schedule 153
16.2.15 nrc_ps_add_gpio_callback 153
16.2.16 nrc_ps_start_schedule 154
16.2.17 nrc_ps_resume_deep_sleep 154
17 PBC (Push Button) 155
17.1 Data Type 155
17.1.1 pbc_ops 155
17.2 Function Call 155
17.2.1 wps_pbc_fail_cb 155
17.2.2 wps_pbc_timeout_cb 156
17.2.3 wps_pbc_success_cb 156
17.2.4 wps_pbc_button_pressed_event 157
17.2.5 init_wps_pbc 157
18 Middleware API Reference 158
18.1 FreeRTOS 158
18.2 WPA_supplicant 158
18.3 lwIP 158
18.1 MbedTLS 159
18.2 NVS library 159
19 Abbreviations 160
20 Revision history 161

List of Tables

Table 2.1 Error Type 17
Table 3.1 tWIFI_STATUS 18
Table 3.2 tWIFI_DEVICE_MODE 18
Table 3.3 tWIFI_STATE_ID 19
Table 3.4 tWIFI_COUNTRY_CODE 19
Table 3.5 tWIFI_SECURITY 20
Table 3.6 tWIFI_BANDWIDTH 20
Table 3.7 tWIFI_IP_MODE 20
Table 3.8 tNET_ADDR_STATUS 21
Table 3.9 tWIFI_SCAN_MODE 21
Table 3.10 SCAN_RESULT 21
Table 3.11 Security Flags 22
Table 3.12 SCAN_RESULTS 22
Table 3.13 AP_INFO 22
Table 3.14 tWIFI_STA_STATE 22
Table 3.15 STA_INFO 23
Table 3.16 STA_LIST 23
Table 3.17 Tx Power Type 23
Table 3.18 Guard Interval(GI) Type 24
Table 3.19 Version Type 24
Table 3.20 tWIFI_IGNORE_BROACAST_SSID 24
Table 3.21 SAE_PWE Type 24
Table 3.22 EAP Type 25
Table 3.23 Network mode Type 25
Table 3.24 EAP Type 25
Table 3.25 tWIFI_EVENT_ID 75
Table 5.1 DMA_ERROR 84
Table 5.2 DMA_BSIZE 84
Table 5.3 DMA_WIDTH 84
Table 5.4 DMA_AHBM 85
Table 5.5 DMA_PERI_ID 85
Table 5.6 dma_peri_t 85
Table 5.7 dma_desc_t 86
Table 6.1 NRC_UART_CHANNEL 99
Table 6.2 NRC_UART_DATA_BIT 99
Table 6.3 NRC_UART_STOP_BIT 99
Table 6.4 NRC_UART_PARITY_BIT 100
Table 6.5 NRC_UART_HW_FLOW_CTRL 100
Table 6.6 NRC_UART_FIFO 100
Table 6.7 NRC_UART_CONFIG 100

Table 6.8 NRC_UART_INT_TYPE 101
Table 7.1 uart_dma_t 106
Table 7.2 uart_dma_buf_t 106
Table 7.3 uart_dma_rx_params_t 106
Table 7.4 uart_dma_info_t 107
Table 8.1 NRC_GPIO_PIN 110
Table 8.2 NRC_GPIO_DIR 110
Table 8.3 NRC_GPIO_MODE 110
Table 8.4 NRC_GPIO_LEVEL 111
Table 8.5 NRC_GPIO_ALT 111
Table 8.6 NRC_GPIO_CONFIG 111
Table 8.7 nrc_gpio_trigger_t 111
Table 8.8 nrc_gpio_trigger_t 111
Table 9.1 I2C_CONTROLLER_ID 116
Table 9.2 I2C_WIDTH 116
Table 9.3 I2C_CLOCK_SOURCE 116
Table 9.4 i2c_device_t 117
Table 10.1 ADC_CH 121
Table 11.1 PWM_CH 123
Table 12.1 SPI_MODE 126
Table 12.2 SPI_FRAME_BITS 126
Table 12.3 SPI_CONTROLLER_ID 127
Table 12.4 spi_device_t 127
Table 14.1 httpc_ret_e 136
Table 14.2 con_handle_t 136
Table 14.3 ssl_certs_t 137
Table 14.4 httpc_data_t 137
Table 15.1 FOTA_INFO 142
Table 15.2 Broadcast FOTA mode 142
Table 16.1 POWER_SAVE_WAKEUP_SOURCE 147
Table 16.2 POWER_SAVE_WAKEUP_REASON 147
Table 17.1 pbc_ops 155
Table 19.1 Abbreviations and acronyms 160

List of Figures

Figure 1.1 NRC7394 SDK Architecture 16

1 Overview

This document introduces the Application Programming Interface (API) forstandalone NRC7394 Software Development Kit (SDK). These APIs are used for Wi-Fi operations and events and other peripherals on the NRC7394 Evaluation Boards (EVB).

The user application is implemented using SDK API, 3rd party libraries and system hardware abstract layer (HAL) APIs. The lwIP is used for TCP/IP related codes. The mbedtls is related to encryption and decryption. The FreeRTOS is a real-time operating system kernel for embedded devices. It provides methods for multiple threads or tasks, mutexes, semaphores and software timers. Wi-Fi API is implemented based on wpa_supplicant. It provides the general Wi-Fi operations such as scan, connect, set Wi-Fi configurations and get system status information such as RSSI, SNR.

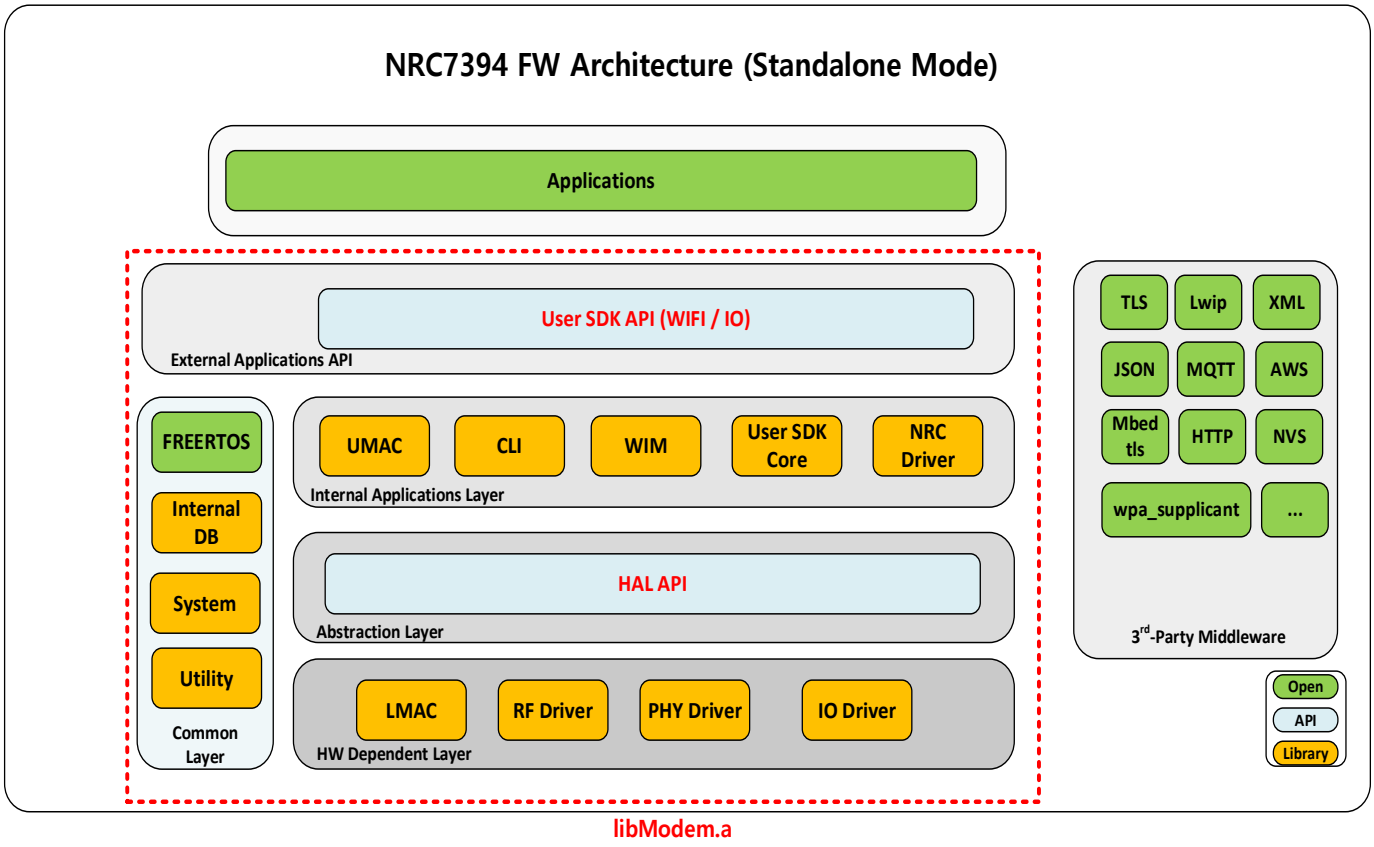

Figure 1.1 NRC7394 SDK Architecture

2 General

The general data types are defined at the “NRC7394/API/Inc/nrc_types.h”.

2.1.1 Error Type

nrc_err_t is an operation function return type. These types are defined at the “lib/sdk/inc/nrc_types.h”.

Table 2.1 Error Type

Name	Description
NRC_SUCCESS	Operation was successful
NRC_FAIL	Operation failed

3 Wi-Fi

The Wi-Fi API provides functions to:
• Scan & connect to AP
• Configuration the Wi-Fi settings
• Set and get the IP address

3.1 Data Type

These types are defined at the “sdk/nrc_types.h”.

3.1.1 API Status Return Value

tWIFI_STATUS is returned by API functions to indicate whether a function call succeeded or failed.

Table 3.1 tWIFI_STATUS

Name	Description
WIFI_SUCCESS	Operation successful
WIFI_NOMEM	No memory
WIFI_INVALID	Invalid parameter
WIFI_INVALID_STATE	Invalid Wi-Fi state
WIFI_TIMEOUT	Operation timeout
WIFI_TIMEOUT_DHCP	Get IP address is timeout
WIFI_FAIL	Operation failed
WIFI_FAIL_INIT	Wi-Fi initial is failed
WIFI_FAIL_CONNECT	Wi-Fi connection is failed
WIFI_FAIL_DHCP	Get DHCP client is failed
WIFI_FAIL_SET_IP	Set IP address is failed
WIFI_FAIL_SOFTAP	SoftAP start is failed
WIFI_FAIL_SOFTAP_NOSTA	No station is connected to softAP.

3.1.2 Device Mode

tWIFI_DEVICE_MODE is the bandwidth.

Table 3.2 tWIFI_DEVICE_MODE

Name	Description
WIFI_MODE_STATION	Station
WIFI_MODE_AP	Access Point

3.1.3 Wi-Fi State

tWIFI_STATE_ID is the Wi-Fi state.

Table 3.3 tWIFI_STATE_ID

Name	Description
WIFI_STATE_UNKNOWN	Not initialized or unknown state
WIFI_STATE_INIT	Initial
WIFI_STATE_CONFIGURED	Wi-Fi configuration is done
WIFI_STATE_TRY_CONNECT	Try to connect
WIFI_STATE_CONNECTED	Connected
WIFI_STATE_TRY_DISCONNECT	Try to disconnect
WIFI_STATE_DISCONNECTED	Disconnected
WIFI_STATE_SOFTAP_CONFIGURED	Set the SoftAP configuration
WIFI_STATE_SOFTAP_TRY_START	Try to start SoftAP
WIFI_STATE_SOFTAP_START	SoftAP is started
WIFI_STATE_DHCPS_START	DHCP server is started

3.1.4 Country Code

tWIFI_COUNTRY_CODE is the country code.

Table 3.4 tWIFI_COUNTRY_CODE

Name	Description
WIFI_CC_UNKNOWN	Unknown value
WIFI_CC_JP	Japan
WIFI_CC_US	United States of America
WIFI_CC_EU	Europe
WIFI_CC_CN	China
WIFI_CC_NZ	New Zealand
WIFI_CC_AU	Australia
WIFI_CC_K1	Korea USN1(921.5~922.5) Mhz – LBT (Non Standard)
WIFI_CC_K2	Korea USN5(925.5~929) Mhz – MIC detection
WIFI_CC_T8	Singapore (800Mhz)
WIFI_CC_T9	Singapore (900Mhz)
WIFI_CC_S8	Taiwan (866.6~868Mhz)
WIFI_CC_S9	Taiwan (917.5~923Mhz)

3.1.5 Security Mode

tWIFI_SECURITY is the security mode. The NRC7394 supports the OPEN, WPA2, WPA3-SAE and WPA3-OWE security protocols.

Table 3.5 tWIFI_SECURITY

Name	Description
WIFI_SEC_OPEN	Open
WIFI_SEC_WPA2	WPA2
WIFI_SEC_WPA3_OWE	WPA3 OWE
WIFI_SEC_WPA3_SAE	WPA3 SAE

※ If you intend to use WPA3 in the STA (Station), it will be necessary to modify the AP (Access Point) configurations to support WPA3 as well. Please refer the Appendix A. in 'UG-7394-004-Standalone SDK.pdf'

(1) In order to enable WPA3-OWE (Opportunistic Wireless Encryption), please make the following modification in the 'wpa_auth.c' file:
Change the value of 'eapol_key_timeout_subseq' to 2000.
(2) NRC7394 does not support PWE (Password-Only Wakeup Enabled).
To utilize WPA3-SAE (Simultaneous Authentication of Equals) without PWE, remove the 'sae_pwe=1' line from the host configuration file, such as 'hostapd.conf'.

3.1.6 Bandwidth

tWIFI_BANDWIDTH is the bandwidth.

Table 3.6 tWIFI_BANDWIDTH

Name	Description
WIFI_1M	1 MHz bandwidth
WIFI_2M	2 MHz bandwidth
WIFI_4M	4 MHz bandwidth

3.1.7 IP Mode

tWIFI_IP_MODE is the IP mode.

Table 3.7 tWIFI_IP_MODE

Name	Description
WIFI_STATIC_IP	Static IP
WIFI_DYNAMIC_IP	Dynamic IP, which uses the DHCP client

3.1.8 Address status

tNET_ADDR_STATUS is the IP address status.

Table 3.8 tNET_ADDR_STATUS

Name	Description
NET_ADDR_NOT_SET	IP address is not set
NET_ADDR_DHCP_STARTED	DHCP client is started
NET_ADDR_SET	IP address is set

3.1.9 Scan mode

tWIFI_SCAN_MODEis the scan type.

Table 3.9 tWIFI_SCAN_MODE

Name	Description
WIFI_SCAN_MODE_ACTIVE	Actively probes for networks by sending requests
WIFI_SCAN_MODE_PASSIVE	Listens for network signals without sending probe requests

3.1.10 SCAN_RESULT

This is a union of data types for SCAN_RESULT.

Table 3.10 SCAN_RESULT

Type	Element	Description
char*	items[5]	This is union values. Each array entry points members. Items[0] : BSSID items[1] : Frequency items[2] : Signal level items[3] : Flags items[4] : SSID
char*	bssid	BSSID, which is fixed-length, colon-separated hexadecimal ASCII string. (Ex. “84:25:3f:01:5e:50”)
char*	freq	Frequency. The frequency is equivalent Wi-Fi channel (2.4/5G frequency) (Ex. “5205”). See the “S1G Channel“
char*	sig_level	Numeric ASCII string of RSSI. (Ex. “-25”). The unit is dBm
char*	flags	ASCII string of the security model for the network.
char*	ssid	ASCII string of SSID.
tWIFI_SECURITY	security	Security. See the “ Security Mode”
tWIFI_BANDWIDTH	bandwidth	Bandwidth. See the “Bandwidth”

Table 3.11 Security Flags

Name	Description
WPA2-EAP	Wi-Fi Protected Access 2 – Extensible Authentication Protocol
WPA2-PSK	Wi-Fi Protected Access 2 – Pre-Shared Key
WPA3-SAE	Wi-Fi Protected Access 3 – Simultaneous Authentication of Equals
WPA3-OWE	Wi-Fi Protected Access 3 – Opportunistic Wireless Encryption

3.1.11 SCAN_RESULTS

This is a structure for function nrc_wifi_scan_results().

Table 3.12 SCAN_RESULTS

Type	Element	Description
int	n_result	number of scanned BSSID
SCAN_RESULT	result[MAX_SCAN_RESULTS]	scan results

※'MAX_SCAN_RESULTS' is a maximum scan results and 30.

3.1.12 AP_INFO

AP information

Table 3.13 AP_INFO

Type	Element	Description
uint8_t	bssid[6]	BSSID
uint8_t	ssid[32]	ASCII string of SSID.
uint8_t	ssid_len	ssid length
uint8_t	cc[2]	ASCII string of the country code
uint16_t	ch	Channel index
uint16_t	freq	Frequency. The frequency is equivalent Wi-Fi channel (2.4/5G frequency) (Ex. "5205"). See the "S1G Channel"
tWIFI_BANDWIDTH	bw	Bandwidth. See the "Bandwidth"
tWIFI_SECURITY	security	Security. See the " Security Mode"

3.1.13 STA State

tWIFI_STA_STATE is the STA state which is connected to AP.

Table 3.14tWIFI_STA_STATE

Name	Description
WIFI_STA_INVALID	STA is not existed in AP information
WIFI_STA_AUTH	STA is authenticated
WIFI_STA_ASSOC	STA is associated

3.1.14 STA_INFO

Station's information which is connected to AP.

Table 3.15 STA_INFO

Type	Element	Description
tWIFI_STA_STATE	state	The state of station. See the "STA state"
int8_t	rssi	Received Signal Strength Indicator value (dBm)
uint8_t	snr	Signal-to-noise ratio
uint8_t	tx_mcs	MCS(Modulation and Coding Scheme) index used for transmission.
uint8_t	rx_mcs	MCS(Modulation and Coding Scheme) index used for transmission.
uint16_t	aid	Association ID
uint8_t	addr[6]	MAC address

3.1.15 STA_LIST

Station lists which are connected to AP

Table 3.16 STA_LIST

Type	Element	Description
uint16_t	total_num	Total number of stations
STA_INFO	sta[MAX_STA_CONN_NUM]	The array of station information

3.1.16 Tx Power Type

The Tx power type can be configured for the Wi-Fi radio.

Table 3.17 Tx Power Type

Name	Description
WIFI_TXPOWER_AUTO	Automatically adjust its Tx power based on the current network conditions. It use the board data.
WIFI_TXPOWER_LIMIT	Automatically adjust its Tx power based on the current network conditions and Max Tx power is limited. It use the board data.
WIFI_TXPOWER_FIXED	The device will use a fixed Tx power level

3.1.17 Guard Interval(GI) Type

The guard interval(GI) type can be configured for the Wi-Fi radio.

Table 3.18 Guard Interval(GI) Type

Name	Description
WIFI_GI_UNKNOWN	Unknown value
WIFI_GI_LONG	Use the long guard interval(GI)
WIFI_GI_SHORT	Use the short guard interval(GI)

3.1.18 Version Type

VERSION_T which includes major, minor and patch.

Table 3.19 Version Type

Type	Element	Description
uint8_t	major	major version
uint8_t	minor	minor version
uint8_t	patch	patch version

3.1.19 Ignore Broadcast SSID

tWIFI_IGNORE_BROACAST_SSID is the ignore broadcast types.

Table 3.20 tWIFI_IGNORE_BROACAST_SSID

Name	Description
WIFI_IGNORE_BROADCAST_SSID_FULL	Probe requests for broadcast SSID are not ignored. It sends the SSID in beacons.
WIFI_IGNORE_BROADCAST_SSID_EMPTY	An empty (length=0) SSID is sent in beacons, and probe requests for broadcast SSID are ignored.
WIFI_IGNORE_BROADCAST_SSID_CLEAR	The SSID is cleared (ASCII 0), but its original length is retained. This may be necessary for clients that do not support an empty SSID. Probe requests for broadcast SSID are ignored.

3.1.20 SAE PWE

tWIFI_SAE_PWE is the sae_pwe types. SAE PWE stands for Simultaneous Authentication of Equals (SAE) Password Element (PWE).

Table 3.21 SAE_PWE Type

Name	Description
WIFI_SAE_PWE_HAP	Support hunting-and-pecking loop only
WIFI_SAE_PWE_H2E	Support hash-to-element only

WIFI_SAE_PWE_BOTH	Support both hunting-and-pecking loop and hash-to-element enabled

3.1.21 EAP

tWIFI_EAP is the EAP(Extensible Authentication Protocol) types.

Table 3.22 EAP Type

Name	Description
WIFI_EAP_NONE	None
WIFI_EAP_TLS	Use TLS
WIFI_EAP_TTLS	Use TTLS-MSCHAPv2
WIFI_EAP_PEAP	Use PEAPv0-MSCHAPv2

3.1.22 Network mode

tWIFI_NETWORK_MODE is the network mode types.

Table 3.23 Network mode Type

Name	Description
WIFI_NETWORK_MODE_BRIDGE	Bridged mode
WIFI_NETWORK_MODE_NAT	NAT mode

3.1.23 TID

tWIFI_TID is the Wi-Fi TID(Traffic Identifier) types.

Table 3.24 EAP Type

Name	Description
WIFI_TID_BE	Best effort traffic
WIFI_TID_BK	Background traffic
WIFI_TID_VI	Video traffic
WIFI_TID_VO	Voice traffic

3.2 Function Call

These APIs are defined at the “sdk/api/api_wifi.h”.

3.2.1 nrc_wifi_get_device_mode

This function retrieves the device mode of the specified network index.

Prototype :

tWIFI_STATUS nrc_wifi_get_device_mode(int vif_id, tWIFI_DEVICE_MODE *mode)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

mode

Type: tWIFI_DEVICE_MODE *

Purpose: Device mode(STA or AP). See “Device Mode”.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.2 nrc_wifi_get_mac_address

This function retrieves the MAC address of the specified network index.

Prototype :

tWIFI_STATUS nrc_wifi_get_mac_address(int vif_id, char *addr)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

addr

Type: char*

Purpose: A pointer to get MAC address which is colon-separated hexadecimal ASCII string. (Ex. “84:25:32:11:5e:50”).

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.3 nrc_wifi_get_tx_power

This function retrieves the transmit (TX) power in decibel-milliwatts (dBm), which is valid once the connection is established.

Prototype :

tWIFI_STATUS nrc_wifi_get_tx_power(int vif_id, uint8_t *txpower)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

txpower

Type: int*

Purpose: A pointer to store the TX power value in dBm.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.4 nrc_wifi_set_tx_power

This function sets the transmit (TX) power and its type. It should be invoked following the configuration of the country code.

Prototype :

tWIFI_STATUS nrc_wifi_set_tx_power(uint8_t txpower, uint8_t type)

Input Parameters :

txpower

Type: int

Purpose: TX Power (in dBm) (1~30)

type

Type: uint8_t

Purpose: Auto(0): The device will automatically adjust its Tx power based on the current network conditions and signal strength.
Limit(1): The device will use a specified maximum Tx power limit.
Fixed(2): The device will use a fixed Tx power level.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

※ The AUTO (0) and LIMIT (1) options operate auto TX gain adjustment using board data file.

3.2.5 nrc_wifi_get_rssi

This function retrieves the received signal strength indicator (RSSI) value for STA.

Prototype :

tWIFI_STATUS nrc_wifi_get_rssi (int vif_id, int8_t *rssi)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

rssi

Type: int8_t*

Purpose: A pointer to store the RSSI value in decibels (dB).

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.6 nrc_wifi_get_average_rssi

This function retrieves the average received signal strength indicator (RSSI) value with 4 packets for STA.

Prototype :

tWIFI_STATUS nrc_wifi_get_average_rssi (int vif_id, int8_t *rssi)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

rssi

Type: int8_t*

Purpose: A pointer to store the RSSI value in decibels (dB).

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.7 nrc_wifi_get_snr

This function retrieves the signal-to-noise ratio (SNR) value for STA.

Prototype :

tWIFI_STATUS nrc_wifi_get_snr(int vif_id, uint8_t *snr)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
snr
Type: uint8_t*
Purpose: A pointer to store the SNR value in decibels (dB).

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.8 nrc_wifi_get_rate_control

This function retrieves the status of the MCS (Modulation and Coding Scheme) rate control option.

Prototype :

bool nrc_wifi_get_rate_control(int vif_id)

Input Parameters :

vif_id
Type: int
Purpose: Network index.

Returns :

Status : 1(enable) or 0(disable)

3.2.9 nrc_wifi_set_rate_control

This function sets the MCS (Modulation and Coding Scheme) rate control option.

Prototype :

tWIFI_STATUS nrc_wifi_set_rate_control(int vif_id, bool enable)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
enable
Type: bool
Purpose: Specifies whether to enable or disable the rate control.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.10 nrc_wifi_get_mcs_info

This function retrieve the Modulation and Coding Scheme(MCS) information both TX and RX.

Prototype :

tWIFI_STATUS nrc_wifi_get_mcs_info(int vif_id, uint8_t *tx_mcs, uint8_t *rx_mcs)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

listen_interval

Type: uint8_t*

tx_mcs

Pointer to store the MCS for transmission(TX).

Purpose: Rate control(RC) OFF : manually configured MCS for transmission (TX)

Rate control(RC) ON : the latest MCS set by Rate Control(RC) for transmission and updated by tx data frames

interval_ms

Type: uint8_t*

rx_mcs

Purpose: Pointer to store the MCS for reception(RX).

the latest MCS for reception(RX) and updated by rx data frames

Returns :

WIFI_SUCCESS, if the operation was successful.

An error code of type tWIFI_STATUS for any other errors.

3.2.11 nrc_wifi_get_mcs

This function gets the Modulation Coding Scheme (MCS) value.

Prototype :

tWIFI_STATUS nrc_wifi_get_mcs (int vif_id, uint8_t *mcs)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

mcs

Type: uint8_t

Purpose: A pointer to store the MCS (0 ~ 7, 10)

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.12 nrc_wifi_set_mcs

This function sets the Modulation Coding Scheme (MCS) value. It is applied when the rate control is disabled.

Prototype :

tWIFI_STATUS nrc_wifi_set_mcs(uint8_t mcs)

Input Parameters :

mcs

Type: uint8_t

Purpose: The Modulation Coding Scheme value (0 ~ 7, 10)

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.13 nrc_wifi_get_cca_threshold

This function gets the Clear Channel Assessment (CCA) threshold.

Prototype :

tWIFI_STATUS nrc_wifi_set_cca_threshold(int vif_id, int* cca_threshold)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

cca_threshold

Type: int*

Purpose: CCA threshold in dBm (decibel-milliwatts).

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.14 nrc_wifi_set_cca_threshold

This function sets the Clear Channel Assessment (CCA) threshold for a specific network.

Prototype :

tWIFI_STATUS nrc_wifi_set_cca_threshold(int vif_id, int cca_threshold)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

cca_threshold

Type: int

Purpose: CCA threshold in dBm (decibel-milliwatts) (-100 to -35).

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.15 nrc_wifi_set_tx_time

This function configures the carrier sense time and pause time for packet transmission. It performs channel sensing before transmitting packets, waiting for the carrier sense time. If the channel is busy, it backs off; if it's idle, it transmits packets for the specified resume time (which may be shorter). After transmission, a pause time is observed before the module can sense the channel again for subsequent transmissions.

Prototype :

tWIFI_STATUS nrc_wifi_set_tx_time(uint16_t cs_time, uint32_t pause_time)

Input Parameters :

cs_time

Type: uint16_t

Purpose: Carrier sensing time for "Listen before Talk(LBT)" in microseconds (0 to 12480).

pause_time

Type: uint32_t

Purpose: Pause time between transmissions in microseconds.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.16 nrc_wifi_enable_duty_cycle

This function enables the duty cycle feature, which allows for controlling the transmission duration within a specified window.

Prototype :

tWIFI_STATUS nrc_wifi_enable_duty_cycle(uint32_t window, uint32_t duration, uint32_t margin)

Input Parameters :

window

Type: uint32_t

Purpose: Specifies the duty cycle window in microseconds

duration

Type: uint32_t

Purpose: Specifies the allowed transmission duration within the duty cycle window in microseconds.

margin

Type: uint32_t

Purpose: Specifies the duty margin in microseconds.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

Essentially, the final duty cycle is determined by the formula: (duration - margin) / window. In other words, the effect is the same for the following two cases:

- nrc_wifi_enable_duty_cycle(window, duration, margin)
- nrc_wifi_enable_duty_cycle(window, duration - margin, 0)

All three units are in microseconds. Normally, you'd just set the margin to be 0 so that the duty cycle is simply (duration / window) over the specified time window.

The seemingly redundant margin parameter exists to provide an option for users to indirectly adjust the level of guarantee in terms of strictly meeting the regulatory duty cycle limit using a single variable parameter. For example, if we simply set the window and duration parameter values so that duration / margin exactly equals the duty cycle limit, the actual physical duty cycle of the transmitter may occasionally exceed this limit by a miniscule amount (e.g. even if the duty cycle limit and the configured duty cycle limit are both 2.8%, the actual measured physical duty cycle sequence might be something like 2.805%, 2.799%, 2.803%, 2.798% ...). By applying a positive margin value, we can mitigate the occurrence of such excessive duty cycle usage.

3.2.17 nrc_wifi_disable_duty_cycle

This function disables the duty cycle feature, allowing unrestricted transmission without any limitations.

Prototype :

tWIFI_STATUS nrc_wifi_disable_duty_cycle(void)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.18 nrc_wifi_tx_avaliable_duty_cycle

This function checks whether the transmission is currently available within the duty cycle window.

Prototype :

bool nrc_wifi_tx_avaliable_duty_cycle(void)

Returns :

True (1) / False (0)

3.2.19 nrc_wifi_get_state

This function retrieves the current Wi-Fi connection state for a specific network index.

Prototype :

tWIFI_STATE_ID nrc_wifi_get_state(int vif_id)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

Returns :

Current Wi-Fi state, if the operation was successful.

WIFI_STATE_UNKNOWN, if error. See "Wifi_STATE".

3.2.20 nrc_wifi_add_network

This function adds a network index associated with the Wi-Fi connection.

Prototype :

tWIFI_STATUS nrc_wifi_add_network(int *vif_id)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

※ After calling this function, the assigned network index will be stored in the vif_id variable. You can use this network index for further Wi-Fi configuration or operations.

3.2.21 nrc_wifi_remove_network

This function removes a network index associated with the Wi-Fi connection.

Prototype :

tWIFI_STATUS nrc_wifi_remove_network(int vif_id)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

※ By specifying the network index (vif_id) to this function, you can remove the associated network from the Wi-Fi connection. After removing the network, it will no longer be available for Wi-Fi operations.

3.2.22 nrc_wifi_country_from_string

This function retrieves the country code index based on the input string representation of the country code.

Prototype :

tWIFI_COUNTRY_CODE nrc_wifi_country_from_string(const char *str_cc)

Input Parameters :

str_cc

Type: const char*

Purpose: A pointer to a null-terminated string that represents the country code. Valid country code strings. See “Country Code”.

Returns :

tWIFI_COUNTRY_CODE. See “Country Code”.

3.2.23 nrc_wifi_country_to_string

This function retrieves a string representation of the country code based on the provided country code index.

Prototype :

const char *nrc_wifi_country_to_string(int vif_id, tWIFI_COUNTRY_CODE cc)

Input Parameters :

vif_id

Type: int
Purpose: Network index.

cc
Type: tWIFI_COUNTRY_CODE
Purpose: The country code index (tWIFI_COUNTRY_CODE). See “Country Code”

sReturns :

If successful, NULL terminated country code.
NULL if cc provided is not supported.

3.2.24 nrc_wifi_get_country

This function retrieves the current country code used for Wi-Fi operation. The country code represents the regulatory domain.

Prototype :

tWIFI_STATUS nrc_wifi_get_country(tWIFI_COUNTRY_CODE *cc)

Input Parameters :

cc
Type: char*
Purpose: A pointer to a variable of type tWIFI_COUNTRY_CODE where the country code will be populated. See "Country Code" for the available country code options. See “Country Code”.

Returns :

WIFI_SUCCESS, if the operation was successful.
An error code of type tWIFI_STATUS for any other errors.

3.2.25 nrc_wifi_set_country

This function sets the country code for the specified network index, allowing the Wi-Fi operation to comply with the regulations of the specified regulatory domain.

Prototype :

tWIFI_STATUS nrc_wifi_set_set_country (int vif_id, tWIFI_COUNTRY_CODE cc)

Input Parameters :

vif_id
Type: Int
Purpose: Network index.

cc
Type: tWIFI_COUNTRY_CODE
Purpose: The country code to set. See “Country Code”.

Returns :

WIFI_SUCCESS, if the operation was successful.
An error code of type tWIFI_STATUS for any other errors.

3.2.26 nrc_wifi_get_channel_bandwidth

This function retrieves the channel bandwidth for the specified network index.

Prototype :

tWIFI_STATUS nrc_wifi_get_channel_bandwidth(int vif_id, uint8_t *bandwidth)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

bandwidth
- Type: uint8_t *
- Purpose: A pointer to a variable of type uint8_t to store the channel bandwidth. The possible values are 0 (1M BW), 1 (2M BW), or 2 (4M BW).

Returns :

WIFI_SUCCESS, if the operation was successful.
An error code of type tWIFI_STATUS for any other errors.

3.2.27 nrc_wifi_get_channel_freq

This function retrieves the frequency for Sub-1GHz channels for the specified network index.

Prototype :

tWIFI_STATUS nrc_wifi_get_channel_freq(int vif_id, uint16_t *s1g_freq)

Input Parameters :

vif_id
- Type: Int
- Purpose: Network index.

s1g_freq
- Type: uint16_t *
- Purpose: A pointer to a variable of type uint16_t to store the S1G channel frequency in MHz/10.

Returns :

WIFI_SUCCESS, if the operation was successful.
An error code of type tWIFI_STATUS for any other errors.

3.2.28 nrc_wifi_set_channel_freq

This function sets the frequency for Sub-1GHz channels for the specified network index.

Prototype :

tWIFI_STATUS nrc_wifi_set_channel_freq(int vif_id, uint16_t s1g_freq)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

s1g_freq

Type: uint16_t

Purpose: The desired S1G channel frequency in MHz/10.

Returns :

WIFI_SUCCESS, if the operation was successful.

An error code of type tWIFI_STATUS for any other errors.

3.2.29 nrc_wifi_set_channel_freq_bw

The function allows to set the S1G channel frequency and bandwidth for a specific network interface.

Prototype :

tWIFI_STATUS nrc_wifi_set_channel_freq_bw(int vif_id, uint16_t s1g_freq, uint8_t bw)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

s1g_freq

Type: uint16_t

Purpose: The desired S1G channel frequency in MHz/10.

bw

Type: uint8

Purpose: The bandwidth (1, 2, or 4 MHz).

Returns :

WIFI_SUCCESS, if the operation was successful.

An error code of type tWIFI_STATUS for any other errors.

3.2.30 nrc_wifi_set_ssid

Set the SSID of the access point (AP) to connect to in STA mode.

Prototype :

tWIFI_STATUS nrc_wifi_set_ssid(int vif_id, char * ssid)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

ssid

Type: char*

Purpose: A pointer to the SSID string (ASCII). The maximum length of the name is 32 bytes.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.31 nrc_wifi_get_bssid

This function is used to get the BSSID (Basic Service Set Identifier) of the connected access point (AP).

Prototype :

tWIFI_STATUS nrc_wifi_get_bssid(int vif_id, char *bssid)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

bssid

Type: char*

Purpose: A pointer to get bssid which is colon-separated hexadecimal ASCII string. (Ex. “84:25:3f:01:5e:50”). The maximum length of the name is 17 bytes.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.32 nrc_wifi_set_bssid

This function is used to set the BSSID (Basic Service Set Identifier) of the access point (AP) to connect to. This function is applicable for station (STA) mode only.

Prototype :

tWIFI_STATUS nrc_wifi_set_bssid(int vif_id, char * bssid)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

bssid

Type: char*

Purpose: A pointer to set bssid which is colon-separated hexadecimal ASCII string. (Ex. “84:25:3f:01:5e:50”). The maximum length of the name is 17 bytes

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

※ By using this function, you can set the BSSID of the specific AP you want to connect to. The BSSID is a unique identifier assigned to each AP in a Wi-Fi network. By setting the BSSID, you can specify the AP you wish to connect to when multiple APs are available with the same SSID.

3.2.33 nrc_wifi_softap_get_ignore_broadcast_ssid

This function gets the ingnore broadcast ssid types.

Prototype :

tWIFI_STATUS nrc_wifi_softap_get_ignore_broadcast_ssid(int vif_id, int *ignore_broadcast_ssid)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

ignore_broadcast_ssid

Type Int*

Purpose: A pointer to store ignore_broadcast_ssid
0: Broadcast SSID as usual (default)
1: Send an empty (length=0) SSID in beacons and ignore probe requests for broadcast SSID.
2: Clear SSID (ASCII 0), but keep the original length. (this may be required with some clients that do not support empty SSID) and ignore probe requests for broadcast SSID

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.34 nrc_wifi_softap_set_ignore_broadcast_ssid

This function sets the ingnore broadcast ssid types.

Prototype :

tWIFI_STATUS nrc_wifi_softap_set_ignore_broadcast_ssid(int vif_id, int ignore_broadcast_ssid)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

ignore_broadcast_ssid

Type Int

Purpose: ignore_broadcast_ssid

0: Broadcast SSID as usual (default)

1: Send an empty (length=0) SSID in beacons and ignore probe requests for broadcast SSID.

2: Clear SSID (ASCII 0), but keep the original length. (this may be required with some clients that do not support empty SSID) and ignore probe requests for broadcast SSID

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.35 nrc_wifi_set_security

This function is used to set the security parameters for a Wi-Fi connection.

Prototype :

void nrc_wifi_set_security (int vif_id, int mode, char *password)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

mode

Type: int

Purpose: Security mode. Refer to "Security Mode" for available options.

password

Type: char*

Purpose: A pointer to set password. (Ex. “123ABDC”). (upto 30 Bytes)

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.36 nrc_wifi_set_eap_security

This function is used to set EAP security parameters for Wi-Fi connection.

Prototype :

tWIFI_STATUS nrc_wifi_set_eap_security (int vif_id, int mode, int eap, char *identity, char *password, const char *ca_cert, const char *client_cert, const char *private_key, char *private_key_password)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

mode
- Type: int
- Purpose: Security mode. Refer to "Security Mode" for available options.

eap
- Type: int
- Purpose: EAP type. Refer to “EAP” for available options

identity
- Type: char *
- Purpose: identity

password
- Type: char *
- Purpose: password

ca_cert
- Type: const char *
- Purpose: CA certificate(only TLS type)

client_cert
- Type: const char *
- Purpose: client certificate(only TLS type)

private_key
- Type: const char *
- Purpose: private key(only TLS type)

private_key_password
- Type: const char *
- Purpose: private key password(only TLS type)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.37 nrc_wifi_set_pmk

This function is used to set the PMK (Pairwise Master Key) parameters for a Wi-Fi connection.

Prototype :

tWIFI_STATUS nrc_wifi_set_pmk(int vif_id, char *pmk)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

pmk

Type: char*

Purpose: A pointer to set Pairwise Master Key(PMK).

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

※ The PMK (Pairwise Master Key) is a pre-shared key used for authentication in WPA/WPA2 security modes. By setting the PMK, you specify the secret key for secure communication during Wi-Fi connections. Once successfully set, the PMK is used in the authentication process when establishing a Wi-Fi connection.

3.2.38 nrc_wifi_set_sae_pwe

This function is used to set SAE mechanism for PWE derivation.

Prototype :

tWIFI_STATUS nrc_wifi_set_sae_pwe (int vif_id, int sae_pwe)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

sae_pwe:

Type: int

Purpose: SAE mechanism for PWE derivation. Refer to "SAE PWE" for available options.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.39 nrc_wifi_get_sae_pwe

This function is used to get SAE mechanism for PWE derivation.

Prototype :

tWIFI_STATUS nrc_wifi_get_sae_pwe (int vif_id, int *sae_pwe)

Input Parameters :

vif_id

Type: int
Purpose: Network index.

sae_pwe:
Type: int*
Purpose: SAE mechanism for PWE derivation. Refer to "SAE PWE" for available options.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.40 nrc_wifi_get_scan_freq

This function is used to retrieve the scan channel list for scanning access points (APs).

Prototype :

tWIFI_STATUS nrc_wifi_get_scan_freq(int vif_id, uint16_t *freq_list, uint8_t *num_freq)

input Parameters :

vif_id
Type: Int
Purpose: Network index.

freq_list
Type: uint16_t*
Purpose: A pointer to the frequency list. The frequency should be assigned equivalent Wi-Fi channel(2.4 / 5G frequency) (Ex. "5205 5200). See the "S1G Channel"

num_freq
Type: uint8_t*
Purpose: A pointer to save the number of frequencies.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.41 nrc_wifi_set_scan_freq

This function is used to set the scan channel list for scanning access points (APs).

Prototype :

tWIFI_STATUS nrc_wifi_set_scan_freq(int vif_id, uint16_t *freq_list, uint8_t num_freq)

Input Parameters :

vif_id
Type: Int
Purpose: Network index.

freq_list
- Type: uint16_t*
- Purpose: A pointer to the frequency list. The frequency should be assigned equivalent Wi-Fi channel(2.4 / 5G frequency) (Ex. “5205 5200). See the “S1G Channel“

num_freq
- Type: uint8_t
- Purpose: number of frequencies.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.42 nrc_wifi_get_scan_freq_nons1g

This function is used to retrieve the scan channel list for scanning access points (APs). It specifically focuses on setting frequencies for non-1g channels.

Prototype :

tWIFI_STATUS nrc_wifi_get_scan_freq_nons1g(int vif_id, uint16_t *freq_list, uint8_t *num_freq)

input Parameters :

vif_id
- Type: Int
- Purpose: Network index.

freq_list
- Type: uint16_t*
- Purpose: A pointer to the frequency list. The frequency should be assigned equivalent Wi-Fi channel(2.4 / 5G frequency) (Ex. “5205 5200). See the “S1G Channel“

num_freq
- Type: uint8_t*
- Purpose: A pointer to save the number of frequencies.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.43 nrc_wifi_set_scan_freq_nons1g

This function serves the purpose of configuring the scan channel list for scanning access points (APs). It specifically focuses on setting frequencies for non-1g channels.

Prototype :

tWIFI_STATUS nrc_wifi_set_scan_freq_nons1g(int vif_id, uint16_t *freq_list, uint8_t num_freq)

Input Parameters :

vif_id

Type: Int

Purpose: Network index.

freq_list

Type: uint16_t*

Purpose: A pointer to the frequency list. The frequency should be assigned equivalent Wi-Fi channel(2.4 / 5G frequency) (Ex. “5205 5200). See the “S1G Channel“

num_freq

Type: uint8_t

Purpose: number of frequencies.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.44 nrc_wifi_get_aid

This function is used to get the Association ID (AID) allocated by the access point (AP) for a specific network interface.

Prototype :

tWIFI_STATUS nrc_wifi_get_aid(int vif_id, int *aid)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

aid

Type: int*

Purpose: A pointer to get association ID, which is signed binary number.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

※ The Association ID (AID) is a unique identifier assigned by the AP to each associated station (STA) during the Wi-Fi connection establishment. This ID is used to differentiate and identify individual STAs within the network.

3.2.45 nrc_wifi_scan

This function is used to initiate a scan for available access points (APs) in the Wi-Fi network. This function allows the device to discover and collect information about available APs, such as their SSID, BSSID, signal strength, and security settings.

Prototype :

int nrc_wifi_scan (int vif_id)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

※ After calling nrc_wifi_scan, the scanning process is initiated, and the device starts scanning the Wi-Fi channels for APs. The scan results can be obtained using nrc_wifi_get_scan_result().

3.2.46 nrc_wifi_scan_ssid

This function initiates a scan for available access points (APs) in the Wi-Fi network and reserves a scan result slot for the specified SSID. It allows the device to gather information about the available APs, ensuring that at least one scan result is dedicated to the provided SSID.

Prototype :

int nrc_wifi_scan_timeout (int vif_id, uint32_t timeout, char *ssid)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

ssid

Type: char*

Purpose: SSID to scan for. If NULL, scan for all SSID's.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

nrc_wifi_softap_get_max_num_sta

3.2.47 nrc_wifi_scan_timeout

This function is used to initiate a scan for available access points (APs) in the Wi-Fi network with a specified timeout duration.

Prototype :

int nrc_wifi_scan_timeout (int vif_id, uint32_t timeout, char *ssid)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

timeout

Type: uint32_t

Purpose: Blocking time in milliseconds. If set to zero, the caller will be blocked until the scan is completed.

ssid

Type: char*

Purpose: SSID to scan for. If NULL, scan for all SSID's.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.48 nrc_wifi_scan_results

This function is used to retrieve the scan results obtained from a previous Wi-Fi scan operation.

Prototype :

tWIFI_STATUS nrc_wifi_scan_results(int vif_id, SCAN_RESULTS *results)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

results

Type: SCAN_RESULTS*

Purpose: A pointer to the SCAN_RESULTS structure to store the scan listsscan lists. See "SCAN_RESULTS".

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.49 nrc_wifi_abort_scan

This function is used to stop the ongoing scan procedure.

Prototype :

int nrc_wifi_abort_scan (int vif_id)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.50 nrc_wifi_connect

This function is used to connect to an access point (AP) with the specified network index. Before calling this function, make sure to set the necessary AP information such as SSID and security parameters using the appropriate functions mentioned earlier.

Prototype :

tWIFI_STATUS nrc_wifi_connect_timeout (int vif_id, uint32_t timeout)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

timeout

Type: uint32_t

Purpose: Blocking time in milliseconds. If set to zero, the caller will be blocked until the connection is established.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.51 nrc_wifi_disconnect

The nrc_wifi_disconnect_timeout function is used to disconnect from the access point (AP).

Prototype :

tWIFI_STATUS nrc_wifi_disconnect_timeout (int vif_id, uint32_t timeout)

Input Parameters :

vif_id

Type: int
Purpose: Network index.

timeout
Type: uint32_t
Purpose: Blocking time in milliseconds.
If zero, the caller will be blocked until the disconnection is completed.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.52 nrc_wifi_wps_pbc

This function is used to initiate the WPS (Wi-Fi Protected Setup) Push Button Configuration method. This method allows for easy and secure Wi-Fi setup by pressing a physical or virtual push button on both the device and the access point.

Prototype :

tWIFI_STATUS nrc_wifi_wps_pbc(int vif_id)

Input Parameters :

vif_id
Type: int
Purpose: Network index.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.53 nrc_wifi_wps_cancel

This function is used to cancle the WPS (Wi-Fi Protected Setup) Push Button Configuration method.

Prototype :

tWIFI_STATUS nrc_wifi_wps_cancel (int vif_id)

Input Parameters :

vif_id
Type: int
Purpose: Network index.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.54 nrc_wifi_softap_set_conf

The nrc_wifi_softap_set_conf function is used to set the configuration for SoftAP (Software Access Point).

Prototype :

tWIFI_STATUS nrc_wifi_softap_set_conf (int vif_id, char *ssid, uint16_t s1g_freq, uint8_t bw, tWIFI_SECURITY sec_mode, char *password)

Input Parameters :

vif_id

Type: int
Purpose: network index

ssid

Type: char *
Purpose: SSID (Service Set Identifier) of the SoftAP.

s1g_freq

Type: uint16_t
Purpose: Sub-1GHz channel frequency for the SoftAP.

bw

Type: uint8_t
Purpose: specify the bandwidth for a wireless connection (0(BW is selected Automatically), 1(WIFI_1M), 2(WIFI_2M), 4(WIFI_4M))

sec_mode

Type: tWIFI_SECURITY
Purpose: Security mode for the SoftAP (tWIFI_SECURITY)

password

Type: char *
Purpose: Password for the SoftAP, used for authentication and encryption.

sae_pwe

Type: int
Purpose: SAE mechanism for PWE derivation. Refer to "SAE PWE" for available options.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.55 nrc_wifi_softap_set_bss_max_idle

This function is used to set the BSS (Basic Service Set) MAX IDLE period and retry count for the SoftAP. This function is typically used when you want to add the BSS Max Idle Information Element (IE) to the SoftAP. This feature is useful for managing the association and disassociation of STAs based on their idle time. If a STA remains idle for a duration longer than the specified BSS Max Idle period, the SoftAP can automatically disassociate the STA. The retry count specifies the number of attempts the SoftAP should make to receive keep-alive packets from the idle STA before considering it disconnected.

Prototype :

tWIFI_STATUS nrc_wifi_softap_set_bss_max_idle(int vif_id, int period, int retry_cnt)

Input Parameters :

vif_id

Type: int
Purpose: Network index

period

Type: int
Purpose: BSS Max Idle period. It specifies the maximum duration (in milliseconds) that a STA (Station) can be idle before being disassociated from the SoftAP. The valid range is from 0 to 2,147,483,647 milliseconds.

retry_cnt

Type: int
Purpose: Retry count for receiving keep-alive packets from the STA. It specifies the number of retries that the SoftAP should attempt to receive a keep-alive packet from an idle STA before considering it as disconnected. The valid range is from 1 to 100.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.56 nrc_wifi_softap_get_max_num_sta

This function get the maximum number of stations that can be connected to the SoftAP (Software Access Point).

Prototype :

tWIFI_STATUS nrc_wifi_softap_get_max_num_sta(int vif_id, uint8_t *max_sta_num)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

ssid
- Type: uint8_t *
- Purpose: max_sta_num

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.57 nrc_wifi_softap_set_max_num_sta

This function set the maximum number of stations that can be connected to the SoftAP (Software Access Point). The default value is 10.

Prototype :

tWIFI_STATUS nrc_wifi_softap_set_max_num_sta(int vif_id, uint8_t max_sta_num)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

ssid
- Type: uint8_t *
- Purpose: max_sta_num

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.58 nrc_wifi_softap_set_ip

This function is used to set the IP address for the SoftAP.

Prototype :

tWIFI_STATUS nrc_wifi_softap_set_ip(int vif_id, char *ipaddr, char *netmask, char *gateway)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

mode

Type: tWIFI_IP_MODE
Purpose: WIFI_STATIC_IP or WIFI_DYNAMIC_IP

ipaddr
Type: char *
Purpose: A pointer to a string representing the IP address to be set. The IP address should be in the IPv4 format (e.g., "192.168.1.10").

netmask
Type: char *
Purpose: netmask for static IP configuration

gateway
Type: char *
Purpose: gateway for static IP configuration

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.59 nrc_wifi_softap_start

This function is used to synchronously start the SoftAP. Blocks until SoftAP startup completes.

Prototype :

tWIFI_STATUS nrc_wifi_softap_start(int vif_id)

Input Parameters :

vif_id
Type: int
Purpose: network index

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.60 nrc_wifi_softap_start_timeout

Start SoftAP asynchronously with timeout.

Prototype :

tWIFI_STATUS nrc_wifi_softap_start_timeout(int vif_id, uint32_t timeout)

Input Parameters :

vif_id
Type: int
Purpose: network index
ip_addr
Type: uint32_t
Purpose: Blocking time in milliseconds. If set to zero, the caller will be blocked until the SoftAP is started.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.61 nrc_wifi_softap_stop

This function is used to stop the SoftAP. When called, this function will stop the SoftAP and release any allocated resources.

Prototype :

tWIFI_STATUS nrc_wifi_softap_stop(int vif_id)

Input Parameters :

vif_id
Type: int
Purpose: network index

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.62 nrc_wifi_softap_disassociate

This function is used to disassociate stations from the SoftAP. It can disassociate all stations or a station specified by its MAC address.

Prototype :

tWIFI_STATUS nrc_wifi_disassociate(int vif_id, char* mac_addr)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
mac_addr
Type: char*

Purpose: A pointer to set the MAC address. It can be set to the broadcast address (ff:ff:ff:ff:ff:ff) to disassociate all stations, or a specific station's MAC address as a colon-separated hexadecimal ASCII string.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.63 nrc_wifi_softap_deauthenticate

This function is used to deauthenticate stations from the SoftAP. It can deauthenticate all stations or a station specified by its MAC address.

Prototype :

tWIFI_STATUS nrc_wifi_softap_deauthenticate (int vif_id, char* mac_addr)

Input Parameters :

vif_id
Type: int
Purpose: Network index.

mac_addr
Type char*
Purpose: A pointer to set the MAC address. It can be set to the broadcast address (ff:ff:ff:ff:ff:ff) to deauthenticate all stations, or a specific station's MAC address as a colon-separated hexadecimal ASCII string.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.64 nrc_wifi_softap_start_dhcp_server

This function is used to start the DHCP server for the SoftAP.

Prototype :

tWIFI_STATUS nrc_wifi_softap_start_dhcp_server(int vif_id)

Input Parameters :

vif_id
Type: int
Purpose: Network index.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.65 nrc_wifi_softap_stop_dhcp_server

This function is used to stop the DHCP server for the SoftAP.

Prototype :

tWIFI_STATUS nrc_wifi_softap_stop_dhcp_server(int vif_id)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.66 nrc_wifi_softap_get_sta_list

This function is used to retrieve information about the connected STAs (Stations) in the SoftAP mode.

Prototype :

tWIFI_STATUS nrc_wifi_softap_get_sta_list(int vif_id, STA_LIST *info)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

info

Type STA_LIST *

Purpose: A pointer to get STA's information. See "STA_LIST" and "STA_INFO" structures for more details on the information provided.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.67 nrc_wifi_softap_get_sta_by_addr

This function is used to retrieve information about a specific STA (Station) in the SoftAP mode using its MAC address.

Prototype :

tWIFI_STATUS nrc_wifi_softap_get_sta_by_addr(int vif_id, uint8_t *addr, STA_INFO *sta)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
addr
Type uint8_t *
Purpose: A pointer to the MAC address of the STA

Type STA_INFO*
Purpose: A pointer to retrieve the STA's information. It should be of type STA_INFO*. See the "STA_INFO"

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.68 nrc_wifi_softap_get_sta_num

This function is used to retrieve the number of STAs (Stations) currently associated with the SoftAP (Access Point).

Prototype :

tWIFI_STATUS nrc_wifi_softap_get_sta_num(int vif_id)

Input Parameters :

vif_id
Type: int
Purpose: Network index.

Returns :

Number of STAs associated with the SoftAP.

3.2.69 nrc_wifi_softap_get_beacon_interval

This function gets the beacon interval

Prototype :

tWIFI_STATUS nrc_wifi_softap_get_beacon_interval(int vif_id, uint16_t *beacon_interval)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
beacon_interval
Type uint16_t*

Purpose: A pointer to store beacon interval(TU). (1TU=1024us) (range range 15..65535)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.70 nrc_wifi_softap_set_beacon_interval

This function sets the beacon interval

Prototype :

tWIFI_STATUS nrc_wifi_softap_get_beacon_interval(int vif_id, uint16_t beacon_interval)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
beacon_interval
Type uint16_t
Purpose: beacon interval(TU). (1TU=1024us) (range range 15..65535)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.71 nrc_wifi_register_event_handler

This function is used to register a Wi-Fi event handler callback function. The callback function will be called when a Wi-Fi event happens. See the "Callback Functions & Events"

Prototype :

tWIFI_STATUS nrc_wifi_register_event_handler(int vif_id, event_callback_fn fn)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
fn
Type: event_callback_fn
Purpose: event handler for Wi-Fi connection.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.72 nrc_wifi_unregister_event_handler

This function removes a Wi-Fi event handler callback function added by nrc_wifi_register_event_handler.

Prototype :

tWIFI_STATUS nrc_wifi_unregister_event_handler(int vif_id, event_callback_fn fn)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

fn

Type: event_callback_fn

Purpose: event handler for Wi-Fi connection.

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.73 nrc_addr_get_state

This function is used to get the IP address setting state for a specific network interface.

Prototype :

tNET_ADDR_STATUS nrc_addr_get_state (int vif_id)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

Returns :

IP address setting state of type tNET_ADDR_STATUS.

3.2.74 nrc_wifi_get_ip_mode

This function is used to get the IP mode for a specific network interface.

Prototype :

tWIFI_STATUS nrc_wifi_get_ip_mode(int vif_id, tWIFI_IP_MODE* mode)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

mode

Type: tWIFI_IP_MODE*
Purpose: A Pointer to a tWIFI_IP_MODE variable.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.75 nrc_wifi_set_ip_mode

This function is used to set the IP mode and IP address for a specific network interface.

Prototype :

tWIFI_STATUS nrc_wifi_set_ip_mode(int vif_id, tWIFI_IP_MODE mode, char* ip_addr)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
mode
Type: tWIFI_IP_MODE
Purpose: IP mode, either WIFI_IP_MODE_STATIC or WIFI_IP_MODE_DYNAMIC.
ip_addr
Type char*
Purpose: A pointer to set static IP which is ASCII string. (Ex. “192.168.200.23”)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.76 nrc_wifi_get_ip_address

This function is used to get the current IP address of a specific network interface.

Prototype :

tWIFI_STATUS nrc_wifi_get_ip_address(int vif_id, char **ip_addr)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
ip_addr
Type: char**
Purpose: A double pointer to get the address of IP address.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.77 nrc_wifi_set_ip_address

This function is used to set the IP address configuration for a specific network interface. It allows you to either request a dynamic IP address via DHCP or set a static IP address.

Prototype :

tWIFI_STATUS nrc_wifi_set_ip_address(int vif_id, tWIFI_IP_MODE mode, uint32_t timeout, char* ipaddr, char *netmask, char *gateway)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

mode
- Type: tWIFI_IP_MODE
- Purpose: WIFI_STATIC_IP or WIFI_DYNAMIC_IP

timeout
- Type: uint32_t
- Purpose: Wait timeout if WIFI_DYNAMIC_IP is selected. It is ignored for WIFI_STATIC_IP.

ipaddr
- Type: char *
- Purpose: IP address for static IP configuration

netmask
- Type: char *
- Purpose: netmask for static IP configuration

gateway
- Type: char *
- Purpose: gateway for static IP configuration

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.78 nrc_wifi_stop_dhcp_client

This function is used to stop the DHCP client for a specific network interface. This function is typically called to terminate the DHCP client and release the obtained IP address lease.

Prototype :

tWIFI_STATUS nrc_wifi_stop_dhcp_client(int vif_id)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.79 nrc_wifi_set_dns

This function is used to set the DNS (Domain Name System) server addresses.

Prototype :

tWIFI_STATUS nrc_wifi_set_dns(char*pri_dns, char *sec_dns)

Input Parameters :

pri_dns
- Type: char*
- Purpose: Primary DNS server

sec_dns
- Type char*
- Purpose: Secondary DNS server

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.80 nrc_wifi_add_etharp

This function is used to add an entry to the Ethernet ARP (Address Resolution Protocol) table.

Prototype :

tWIFI_STATUS nrc_wifi_add_etharp(int vif_id, const char* addr, char *mac_addr)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

addr
- Type: const char*
- Purpose: The IP address you want to add to the ARP table

mac_addr
- Type char*
- Purpose: The MAC address corresponding to the IP address

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.81 nrc_wifi_send_addba

Send ADDBA action frame

Prototype :

tWIFI_STATUS nrc_wifi_send_addba(int vif_id, tWIFI_TID tid, char *mac_addr)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

tid
- Type: tWIFI_TID
- Purpose: traffic identifier (WIFI_TID_BE, WIFI_TID_BK, WIFI_TID_VI, WIFI_TID_VO)

mac_addr
- Type char*
- Purpose: The MAC address

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.82 nrc_wifi_send_delba

Send DELBA action frame

Prototype :

tWIFI_STATUS nrc_wifi_send_delba(int vif_id, tWIFI_TID tid, char *mac_addr)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

tid
- Type: tWIFI_TID
- Purpose: traffic identifier (WIFI_TID_BE, WIFI_TID_BK, WIFI_TID_VI, WIFI_TID_VO)

mac_addr
- Type char*
- Purpose: The MAC address

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.83 nrc_wifi_set_tx_aggr_auto

Enable automatic tx aggreation

Prototype :

tWIFI_STATUS nrc_wifi_set_tx_aggr_auto(int vif_id, tWIFI_TID tid, uint8_t max_agg_num)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

tid
- Type: tWIFI_TID
- Purpose: traffic identifier (WIFI_TID_BE, WIFI_TID_BK, WIFI_TID_VI, WIFI_TID_VO)

max_agg_num
- Type uint8_t
- Purpose: Max aggregation number

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.84 nrc_wifi_set_passive_scan

This function is used to enable or disable passive scanning in the Wi-Fi module. Passive scanning is a type of Wi-Fi scanning where the Wi-Fi module listens for beacon frames transmitted by access points without actively transmitting probe requests. It allows the module to collect information about nearby access points without actively participating in the scanning process.

* A passive scan generally takes more time, since the client must listen and wait for a beacon versus actively probing to find an AP.

* For passive scan operation, AP should be disabled the short beacon in EVK start.py
short_bcn_enable = 0 # 0 (disable) or 1 (enable)

Prototype :

tWIFI_STATUS nrc_wifi_set_passive_scan(bool passive_scan_on)

Input Parameters :

vif_id
- Type: bool
- Purpose: passive_scan_on (1:enable, 0:disable)

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS) for any other errors.

3.2.85 nrc_wifi_set_simple_bgscan

This function configures a basic background scan for roaming purposes. For example, with short_interval set to 30 seconds, long_interval to 300 seconds, and signal_threshold at -40 dBm: if the signal strength falls below -40 dBm, background scanning occurs every 30 seconds; if the signal is stronger, scanning occurs every 300 seconds.

Prototype :

tWIFI_STATUS nrc_wifi_set_simple_bgscan(int vif_id, uint16_t short_interval, int signal_threshold, uint16_t long_interval)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

short_interval
- Type: uint16_t
- Purpose: short scan interval (sec)

signal_threshold
- Type: int
- Purpose: short/long interval choice signal threshold (db) (ex : -45)

long_interval
- Type: uint16_t
- Purpose: long scan interval (sec)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.86 nrc_wifi_get_ap_info

This function is used to retrieve information about staions information.

Prototype :

tWIFI_STATUS nrc_wifi_get_ap_info(int vif_id, AP_INFO *info)

Input Parameters :

vif_id
- Type: int
- Purpose: Network index.

info
- Type STA_LIST *

Purpose: A pointer to the AP_INFO structure where the AP's information will be stored. See "AP_INFO"

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.87 nrc_wifi_set_rf_power

This function is used to turn on or off the RF (Radio Frequency) power.

Prototype :

tWIFI_STATUS nrc_wifi_set_rf_power(bool power_on)

Input Parameters :

power_on
Type: bool
Purpose: turn on/off rf power.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.88 nrc_wifi_set_use_4address

This function is used to set whether to use four-address support. Four-address support is used in Wi-Fi networks to enable communication between two clients connected to the same AP (Access Point) using Layer 2 bridging.

Prototype :

tWIFI_STATUS nrc_wifi_set_use_4address(bool value)

Input Parameters :

value
Type: bool
Purpose: Enable / disable 4-address support.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS) for any other errors.

3.2.89 nrc_wifi_get_use_4address

This function is used to get the current setting of whether four-address support is enabled or disabled. Four-address support is used in Wi-Fi networks to enable communication between two clients connected to the same AP (Access Point) using Layer 2 bridging.

Prototype :

bool nrc_wifi_get_use_4address(void)

Input Parameters :

void

Returns :

True, the 4-address is enabled

False, the 4-address is disabled

3.2.90 nrc_get_hw_version

This function retrieves the hardware version of the Wi-Fi module, which is stored in the flash memory. It allows you to access and retrieve the specific hardware version information of the Wi-Fi module directly from the flash memory.

Prototype :

uint16_t nrc_get_hw_version(void)

Input Parameters :

void

Returns :

hw_version

3.2.91 nrc_wifi_get_gi

This function gets the Guard Interval(GI) type.

Prototype :

tWIFI_STATUS nrc_wifi_get_gi(tWIFI_GI* mcs)

Input Parameters :

mcs

Type: tWIFI_GI*

Purpose: A pointer to store the guard interval (0:Long GI, 1:Short GI)

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.92 nrc_wifi_set_gi

This function sets the Guard Interval (GI) type for a wireless connection. It should be called before association. The default is a long guard interval.

Prototype :

tWIFI_STATUS nrc_wifi_set_gi(tWIFI_GI mcs)

Input Parameters :

mcs

Type: tWIFI_GI

Purpose: The guard interval type (0:Long GI, 1:Short GI)

Returns :

WIFI_SUCCESS, if the operation was successful.

Error code (tWIFI_STATUS): In case of any other errors.

3.2.93 nrc_wifi_set_beacon_loss_detection

This function configures the operation mode for beacon loss detection, applicable specifically to a Station (STA). By default, the beacon loss detection is enabled, with a beacon loss threshold set at 30.
(ex) 30 * BI(100) * TU(1024us) = about 3 sec

Prototype :

tWIFI_STATUS nrc_wifi_set_beacon_loss_detection(int vif_id, bool enable,
uint8_t beacon_loss_thresh)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

enable

Type: bool
Purpose: Specifies whether to enable (1) or disable (0) beacon loss detection.

beacon_loss_thresh
Type: uint8_t
Purpose: disconnection threshold about beacon loss

Returns :

WIFI_SUCCESS, if the operation was successful.
An error code of type tWIFI_STATUS for any other errors.

3.2.94 nrc_wifi_get_listen_interval

This function gets the listen interval for a Station (STA), which determines the duration the station remains in a sleep state before it wakes up to receive buffered data from the connected Access Point (AP). It's important to note that the listen interval should be kept shorter than the BSS (Basic Service Set) maximum idle time.

Prototype :

```
tWIFI_STATUS nrc_wifi_get_listen_interval(int vif_id, uint16_t *listen_interval,
                                          uint32_t *interval_ms)
```

Input Parameters :

vif_id
Type: int
Purpose: Network index.

listen_interval
Type: uint16_t
Purpose: listen interval
Listen Interval Time (us) = listen_interval * beacon_interval * 1TU (1024 us)

interval_ms
Type: uint32_t
Purpose: listen interval time (ms). It should be set after association.

Returns :

WIFI_SUCCESS, if the operation was successful.
An error code of type tWIFI_STATUS for any other errors.

3.2.95 nrc_wifi_set_listen_interval

This function sets the configuration of the listen interval for a Station (STA), determining the period the station stays in a sleep state before awakening to receive buffered data from the connected Access Point (AP). It's important to ensure that the set listen interval remains shorter than the BSS (Basic Service Set) maximum idle time.

Prototype :

```
tWIFI_STATUS nrc_wifi_set_listen_interval(int vif_id, uint16_t listen_interval)
```

Input Parameters :

vif_id
Type: int
Purpose: Network index.
listen_interval
Type: uint16_t
Purpose: listen interval
Listen Interval Time (us) = listen_interval * beacon_interval * 1TU (1024 us)

Returns :

WIFI_SUCCESS, if the operation was successful.
An error code of type tWIFI_STATUS for any other errors.

3.2.96 nrc_wifi_get_mic_scan

This function retrieve MIC scan settings and channel detection count (applicable only to K2(KR MIC)).

Prototype :

tWIFI_STATUS nrc_wifi_get_mic_scan(bool *enable, bool *channel_move, uint32_t *cnt_detected)

Input Parameters :

enable
Type: bool*
Purpose: Pointer to a variable to store the MIC scan enable setting
channel_move
Type: bool*
Purpose: Pointer to a variable to store the channel move setting when the current channel is invalid
cnt_detected
Type: uint32_t*
Purpose: Pointer to a variable to store the channel detection count

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.97 nrc_wifi_set_mic_scan

This function configure MIC scan settings and channel detection count (applicable only to K2(KR MIC))

Prototype :

tWIFI_STATUS nrc_wifi_set_mic_scan(bool enable, bool channel_move)

Input Parameters :

enable
Type: bool

Purpose: MIC scan disable / enable (0|1)

channel_move

Type: bool

Purpose: channel move setting when the current channel is invalid. This is used for access points (AP) mode. The default is false.

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.98 nrc_wifi_set_enable_auth_control

This function enable Enables or disables DAC (Distributed Authentication Control) as per the 802.11ah specification (Section 11.3.9.3).

Prototype :

tWIFI_STATUS nrc_wifi_set_enable_auth_control(int vif_id, bool enable)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

enable

Type: bool

Purpose: Authentication control disable / enable (0|1)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.99 nrc_wifi_get_enable_auth_control

This function get authentication control status.

Prototype :

tWIFI_STATUS nrc_wifi_get_enable_auth_control(uint8_t *enable)

Input Parameters :

vif_id

Type: int

Purpose: Network index.

enable

Type: uint8_t*

Purpose: Pointer to the DAC status (0 = disabled, 1 = enabled)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.100 nrc_wifi_set_auth_control_param

This function configures the DAC (Distributed Authentication Control) parameters

Prototype :

tWIFI_STATUS nrc_wifi_set_auth_control_param(uint8_t slot, uint8_t ti_min, uint8_t ti_max)

Input Parameters :

slot
- Type: uint8_t
- Purpose: authentication slot (in units of TUs)

ti_min
- Type: uint8_t
- Purpose: minimum transmission interval (in units of beacon intervals)

ti_max
- Type: uint8_t
- Purpose: maximum transmission interval (in units of beacon intervals)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.101 nrc_wifi_get_auth_control_param

This function get authentication parameters

Prototype :

tWIFI_STATUS nrc_wifi_set_auth_control_param(uint8_t* slot, uint8_t* ti_min, uint8_t* ti_max)

Input Parameters :

slot
- Type: uint8_t*
- Purpose: authentication slot (in units of TUs)

ti_min
- Type: uint8_t*
- Purpose: minimum transmission interval (in units of beacon intervals)

ti_max
- Type: uint8_t*
- Purpose: maximum transmission interval (in units of beacon intervals)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.102 nrc_wifi_set_auth_control_scale

This function sets scale factor

Prototype :

tWIFI_STATUS nrc_wifi_set_auth_control_scale(uint8_t scale_factor)

Input Parameters :

scale_factor

Type:	uint8_t
Purpose:	The scale factor (1: in units of BI for TI_MIN/TI_MAX or 10: in units of 10*BI for TI_MIN/TI_MAX)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.103 nrc_wifi_get_auth_control_scale

This function gets scale factor

Prototype :

tWIFI_STATUS nrc_wifi_get_auth_control_scale(uint8_t* scale_factor)

Input Parameters :

scale_factor

Type:	uint8_t*
Purpose:	The scale factor (1: in units of BI for TI_MIN/TI_MAX or 10: in units of 10*BI for TI_MIN/TI_MAX)

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.2.104 nrc_wifi_get_auth_current_ti

This function get current Transmission Interval(TI) value of DAC

Prototype :

tWIFI_STATUS nrc_wifi_get_auth_current_ti(int *ti)

Input Parameters :

value
Type: uint8_t*
Purpose: TI value

Returns :

WIFI_SUCCESS, if the operation was successful.
Error code (tWIFI_STATUS): In case of any other errors.

3.3 Callback Functions & Events

Prototype :

void (*event_callback_fn)(int vif_id, tWIFI_EVENT_ID event, int data_len, void *data)

Input Parameters :

vif_id
Type: int
Purpose: Network index.
event
Type: tWIFI_EVENT_ID
Purpose: Wi-Fi Event
data_len
Type: int
Purpose: Data length.
data
Type: void *
Purpose: Data address

Table 3.25 tWIFI_EVENT_ID

Name	Data	Description
WIFI_EVT_SCAN	N/A	Scan is started
WIFI_EVT_SCAN_DONE	N/A	Scan is finished
WIFI_EVT_CONNECT_SUCCESS	MAC Address	Connection
WIFI_EVT_DISCONNECT	MAC Address	Disconnection
WIFI_EVT_AP_STARTED	N/A	SoftAP is started
WIFI_EVT_VENDOR_IE	VendorIE data	Vendor IE
WIFI_EVT_AP_STA_CONNECTED	MAC Address	STA is connected
WIFI_EVT_AP_STA_DISCONNECTED	MAC Address	STA is disconnected
WIFI_EVT_ASSOC_REJECT	MAC Address	Association is rejected

4 System

The system API provides functions to:
• Set and get the system configuration values
• Set the debug log level

4.1 Function Call

The header file for system APIs are defined at the “sdk/inc/api_system.h”.

4.1.1 nrc_get_rtc

Retrieve the real time clock value since cold boot

Prototype :

nrc_err_t nrc_get_rtc(uint64_t* rtc_time)

Input Parameters :

rtc_time

Type: uint64_t*

Purpose: A pointer to get RTC time.

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

4.1.2 nrc_reset_rtc

Reset the real time clock to 0

Prototype :

voidnrc_reset_rtc(void)

Input Parameters :

None

Returns :

None

4.1.3 nrc_sw_reset

Reset software

Prototype :

void nrc_sw_reset(void)

Input Parameters :

None

Returns :

None

4.1.4 nrc_get_user_factory

Get user factory data in flash memory

Prototype :

nrc_err_t nrc_get_user_factory(char* data, uint16_t buf_len)

Input Parameters :

data

Type: char*

Purpose: A pointer to store user factory data

buf_len

Type: uint16_t

Purpose: buffer length (should be 512 Bytes)

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

4.1.5 nrc_get_user_factory_info

This function get base address and total size of user factory area.

Prototype :

nrc_err_t nrc_get_user_factory_info(uint32_t *addr, uint32_t *size)

Input Parameters :

data

Type: uint32_t *

Purpose: A pointer to store base address of user factory area

buf_len

Type: uint32_t*

Purpose: A pointer to store total size of user factory area

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

4.1.6 nrc_led_trx_init

Initializes the Tx/Rx LED blinking feature

Prototype :

nrc_err_t nrc_led_trx_init(int tx_gpio, int rx_gpio, int timer_period, bool invert)

Input Parameters :

tx_gpio
- Type: int
- Purpose: The GPIO pin for the Tx LED

rx_gpio
- Type: int
- Purpose: The GPIO pin for the Rx LED

timer_period
- Type: int
- Purpose: The period for checking the status of the LED blinking

invert
- Type: bool
- Purpose: invert the LED blinking signal

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

4.1.7 nrc_led_trx_deinit

Deinitializes the Tx/Rx LED blinking feature

Prototype :

nrc_err_t nrc_led_trx_deinit(void)

Input Parameters :

None

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

4.1.8 nrc_wdt_enable

Enable watchdog monitoring. The default is enabled

Prototype :

nrc_err_t nrc_wdt_enable(void)

Input Parameters :

None

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

4.1.9 nrc_wdt_disable

Disable watchdog monitoring

Prototype :

nrc_err_t nrc_wdt_disable(void)

Input Parameters :

None

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

4.1.10 nrc_set_app_version

Set application version, which is a mandatory to use broadcast FOTA

Prototype :

nrc_err_t nrc_set_app_version(VERSION_T* version)

Input Parameters :

version

Type: VERSION_T* (see Version Type)
Purpose: A pointer of application version

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

4.1.11 nrc_get_app_version

Get application version, which is a mandatory to use broadcast FOTA

Prototype :

VERSION_T* nrc_get_app_version(void)

Input Parameters :

void

Returns :

VERSION_T* (see Version Type)

4.1.12 nrc_set_app_name

Set application name, which is a mandatory to use broadcast FOTA

Prototype :

nrc_err_t nrc_set_app_name(char* appname)

Input Parameters :

appname

Type: char*

Purpose: A pointer of application name (Max 32 bytes)

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

4.1.13 nrc_get_app_name

Get application name, which is a mandatory to use broadcast FOTA

Prototype :

char* nrc_get_app_name(void)

Input Parameters :

void

Returns :

char*

4.1.14 nrc_get_sdk_version

Get SDK version

Prototype :

VERSION_T* nrc_get_sdk_version(void)

Input Parameters :

void

Returns :

VERSION_T* (see Version Type)

4.1.15 nrc_set_flash_device_info

This function is designed to facilitate the storage of device information data in the flash memory, with a dedicated space of 4KB allocated for this purpose.

Prototype :

nrc_err_t nrc_set_flash_device_info(uint8_t* data, uint16_t len)

Input Parameters :

data
- Type: char*
- Purpose: A pointer to store user factory data

buf_len
- Type: uint16_t
- Purpose: buffer length (should be 4KB Bytes)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

4.1.16 nrc_get_flash_device_info

This function serves the purpose of getting data from the device information area within the flash memory. The allocated space for this area is 4KB.

Prototype :

nrc_err_t nrc_get_flash_device_info(uint8_t* data, uint16_t len)

Input Parameters :

data
- Type: char*
- Purpose: A pointer to store user factory data

buf_len
- Type: uint16_t
- Purpose: buffer length (should be 4KB Bytes)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

4.1.17 nrc_get_user_data_area_address

This function gets the address of the user data area in flash memory.

Prototype :

uint32_t nrc_get_user_data_area_address(void)

Input Parameters :

None

Returns :

The size of the user data area address.

4.1.18 nrc_get_user_data_area_size

This function gets the size of the user data area in flash memory.

Prototype :

uint32_t nrc_get_user_data_area_size(void)

Input Parameters :

None

Returns :

The size of the user data area in bytes.

4.1.19 nrc_erase_user_data_area

This function erases the user data area in flash memory.

Prototype :

uint32_t nrc_get_user_data_area_size(void)

Input Parameters :

None

Returns :

WIFI_SUCCESS, if the operation was successful. The size of t

An error code of type tWIFI_STATUS for any other errors.

4.1.20 nrc_write_user_data

This function writes user data to the user data area in flash memory.

Prototype :

nrc_err_t nrc_write_user_data(uint32_t user_data_offset, uint8_t* data, uint32_t size)

Input Parameters :

user_data_offset

Type: uint32_t

Purpose: The offset from the user data area's base address. It should be aligned to a 4-byte boundary.

data

Type: uint8_t*

Purpose: A pointer to the data to be written.

size

Type: uint32_t

Purpose: The size of data to write.

Returns :

WIFI_SUCCESS, if the operation was successful.

An error code of type tWIFI_STATUS for any other errors.

4.1.21 nrc_read_user_data

This function read user data from the user data area in flash memory

Prototype :

nrc_err_t nrc_read_user_data(uint8_t* buffer, uint32_t user_data_offset, uint32_t size)

Input Parameters :

buffer

Type: uInt8_t*

Purpose: A pointer to the buffer where the data will be stored.

user_data_offset

Type: uint32_t

Purpose: The offset from the user data area's base address. It should be aligned to a 4-byte boundary.

size

Type: uint32_t

Purpose: The size of data to read.

Returns :

WIFI_SUCCESS, if the operation was successful.

An error code of type tWIFI_STATUS for any other errors.

4.1.22 nrc_get_xtal_status

This function gets the status of the crystal (xtal)

※ It is designed specifically for NRC7394.

Prototype :

uint8_t nrc_get_xtal_status(void)

Input Parameters :

None

Returns :

The Crystal(xtal) status

: 0(Crystal status not checked), 1(Crystal is working), 2(Crystal is not working)

5 DMA

DMA (Direct memory access) subsystem API's.
Data transfer between system memory and peripheral devices.

5.1 Data Type

These types are defined in "lib/sdk/inc/api_dma.h".

5.1.1 DMA Errors

Enum value that describes the return values.

Table 5.1 DMA_ERROR

Name	Description
NRC_DMA_OK	Operation was successful.
NRC_DMA_EPERM	Operation cannot be performed.
NRC_DMA_EINVAL	Given argument is invalid.
NRC_DMA_EBUSY	Underlying DMA is busy.

5.1.2 DMA Burst Size

Enum value that describes the DMA burst data size.

Table 5.2 DMA_BSIZE

Name	Description
NRC_DMA_BSIZE_1	Burst size 1 bit.
NRC_DMA_BSIZE_4	Burst size 4 bits.
NRC_DMA_BSIZE_8	Burst size 8 bits.
NRC_DMA_BSIZE_16	Burst size 16 bits.
NRC_DMA_BSIZE_32	Burst size 32 bits.
NRC_DMA_BSIZE_64	Burst size 64 bits.
NRC_DMA_BSIZE_128	Burst size 128 bits.
NRC_DMA_BSIZE_256	Burst size 256 bits.

5.1.3 DMA Width

Enum value that describes the DMA width to be used.

Table 5.3 DMA_WIDTH

Name	Description
NRC_DMA_WIDTH_8	DMA to transfer 8 bits at a time.
NRC_DMA_WIDTH_16	DMA to transfer 16 bits at a time.

NRC_DMA_WIDTH_32	DMA to transfer 32 bits at a time.

5.1.4 DMA AHBM

AHBM interface number.

Table 5.4 DMA_AHBM

Name	Description
NRC_DMA_AHB_M1	AHBM interface 1
NRC_DMA_AHB_M2	AHBM interface 2

5.1.5 DMA Peripheral ID

DMA peripheral identifications.

Table 5.5 DMA_PERI_ID

Name	Description
NRC_DMA_PERI_SSP0_RX	SPI channel 0 peripheral for RX.
NRC_DMA_PERI_SSP0_TX	SPI channel 0 peripheral for TX.
NRC_DMA_PERI_SSP1_RX	SPI channel 1 peripheral for RX.
NRC_DMA_PERI_SSP1_TX	SPI channel 1 peripheral for TX.
NRC_DMA_PERI_HSUART0_RX	UART channel 0 peripheral for RX.
NRC_DMA_PERI_HSUART0_TX	UART channel 0 peripheral for TX.
NRC_DMA_PERI_HSUART1_RX	UART channel 1 peripheral for RX.
NRC_DMA_PERI_HSUART1_TX	UART channel 1 peripheral for TX.
NRC_DMA_PERI_HSUART2_RX	UART channel 2 peripheral for RX.
NRC_DMA_PERI_HSUART2_TX	UART channel 2 peripheral for TX.
NRC_DMA_PERI_HSUART3_RX	UART channel 3 peripheral for RX.
NRC_DMA_PERI_HSUART3_TX	UART channel 3 peripheral for TX.
NRC_DMA_PERI_SSP2_RX	SPI channel 2 peripheral for RX.
NRC_DMA_PERI_SSP2_TX	SPI channel 2 peripheral for TX.
NRC_DMA_PERI_SSP3_RX	SPI channel 3 peripheral for RX.
NRC_DMA_PERI_SSP3_TX	SPI channel 3 peripheral for TX.

5.1.6 DMA Peripheral

DMA peripheral data structure passed to DMA controller.

Table 5.6 dma_peri_t

Name	Description
Addr	Peripheral location address
ID	Peripheral Identification

AddrInc	Automatic address increase enable/disable
FlowCtrl	Flow control enable/disable

5.1.7 DMA Descriptor

DMA channel control information passed to DMA controller.

Table 5.7 dma_desc_t

Name	Description
SrcAddr	Source address
DestAddr	Destination address
Next	Next dma_desc_t element
XferSize	Transfer size
SBSize	Source burst size
DBSize	Destination burst size
SWidth	Source transfer width
Dwidth	Destination transfer width
SAHBM	Source AHB master
DAHBM	Destination AHB master
SAInc	Source address increment
DAInc	Destination address increment
Privileged	Protection: Privileged mode
Bufferable	Protection: Buffered access
Cacheable	Protection: Cached access
IntTC	Terminal count interrupt enable

5.2 Function Call

5.2.1 nrc_dma_enable

This function enables DMA subsystem on NRC SoC.

Prototype :

void nrc_dma_disable (void)

Input Parameters :

N/A

Returns :

N/A

5.2.2 nrc_dma_disable

This function disables DMA subsystem on NRC SoC.

Prototype :

void nrc_dma_disable (void)

Input Parameters :

N/A

Returns :

N/A

5.2.3 nrc_dma_is_enabled

Check if DMA subsystem is enabled on NRC Soc.

Prototype :

bool nrc_dma_is_enabled (void)

Input Parameters :

Returns :

true if DMA is enabled.
False if DMA is not enabled.

5.2.4 nrc_dma_get_channel

This function returns the DMA channel that is available to be used on the system.
If the argument highest set to non-zero, then the highest channel number will be returned from 8 channels available on the system. If the highest is set to zero, then low channel number will be returned. i.e., If firmware is already using DMA channel 0 and 7, non-zero highest will result in returning channel number 6 and zero highest will return channel number 1.

Prototype :

int nrc_dma_get_channel (int highest)

Input Parameters :

Highest
Type: Int
Purpose: Desired high or low channel.

Returns :

Available channel number.
-1 if there is no available channel.

5.2.5 nrc_dma_valid_channel

Validate if channel number provided are within 0 to 7.

Prototype :

bool nrc_dma_valid_channel (int channel)

Input Parameters :

channel
Type: Int
Purpose: Channel number to be validated.

Returns :

true if channel number given is valid.
false otherwise.

5.2.6 nrc_dma_peri_init

Initialize dma_peri_t data structure using supplied data.

Prototype :

int nrc_dma_peri_init (dma_peri_t *peri, int id, uint32_t addr, bool addr_inc, bool flow_ctrl)

Input Parameters :

peri
Type: dma_peri_t
Purpose: DMA configuration data structure.
id
Type: int
Purpose: Identification
addr
Type: uint32_t
Purpose: Address to be used for DMA.
addr_inc
Type: bool
Purpose: Whether to automatically increase address.
flow_ctrl
Type: bool
Purpose: If flow control should be used.

Returns :

NRC_DMA_OK, if the operation was successful
NRC_DMA_EINVAL, if error occurs.

5.2.7 nrc_dma_config_m2m

Configure DMA to be used within the system.

Prototype :

int nrc_dma_config_m2m (int channel, dma_isr_t inttc_isr, dma_isr_t interr_isr)

Input Parameters :

channel

Type: int
Purpose: DMA channel to be used.

inttc_isr
Type: dma_isr_t
Purpose: Interrupt handler that will be called upon completion of DMA operation.

interr_isr
Type: dma_isr_t
Purpose: Interrupt handler that will be call if there is any error occurred during DMA operation.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EPERM if DMA is not enabled.
NRC_DMA_EBUSY if DMA is already enabled.
NRC_DMA_EINVAL if error occurs.

5.2.8 nrc_dma_config_m2p

Configure DMA to be used for the data direction from SoC to peripheral.

Prototype :

int nrc_dma_config_m2p (int channel, dma_peri_t *dest_peri, dma_isr_t inttc_isr, dma_isr_t interr_isr)

Input Parameters :

channel
Type: int
Purpose: DMA channel to be used.

src_peri
Type: dma_peri_t
Purpose: source peripheral address.
i.e. SSP0_BASE_ADDR + 8 for data section of SPI channel 0.
(See SoC specification for details.)

inttc_isr
Type: dma_isr_t
Purpose: Interrupt handler that will be called upon completion of DMA operation.

interr_isr
Type: dma_isr_t
Purpose: Interrupt handler that will be call if there is any error occurred during DMA operation.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EPERM if DMA is not enabled.
NRC_DMA_EBUSY if DMA is already enabled.

NRC_DMA_EINVAL if error occurs.

5.2.9 nrc_dma_config_p2m

Configure DMA to be used for the data incoming from peripheral to SoC.

Prototype :

int nrc_dma_config_p2m (int channel, dma_peri_t *src_peri, dma_isr_t inttc_isr, dma_isr_t interr_isr)

Input Parameters :

channel
- Type: int
- Purpose: DMA channel to be used.

src_peri
- Type: dma_peri_t
- Purpose: source peripheral address. i.e. SSP0_BASE_ADDR + 8 for data section of SPI channel 0. (See SoC specification for details.)

inttc_isr
- Type: dma_isr_t
- Purpose: Interrupt handler that will be called upon completion of DMA operation.

interr_isr
- Type: dma_isr_t
- Purpose: Interrupt handler that will be call if there is any error occurred during DMA operation.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EPERM if DMA is not enabled.
NRC_DMA_EBUSY if DMA is already enabled.
NRC_DMA_EINVAL if error occurs.

5.2.10 nrc_dma_config_p2p

Configure DMA to be used between the peripherals.

Prototype :

int nrc_dma_config_p2p (int channel, dma_peri_t *src_peri, dma_isr_t inttc_isr, dma_isr_t interr_isr)

Input Parameters :

channel
- Type: int
- Purpose: DMA channel to be used.

src_peri
- Type: dma_peri_t

source peripheral address.
Purpose: i.e. SSP0_BASE_ADDR + 8 for data section of SPI channel 0.
(See SoC specification for details.)

inttc_isr
Type: dma_isr_t
Purpose: Interrupt handler that will be called upon completion of DMA operation.

interr_isr
Type: dma_isr_t
Purpose: Interrupt handler that will be call if there is any error occurred during DMA operation.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EPERM if DMA is not enabled.
NRC_DMA_EBUSY if DMA is already enabled.
NRC_DMA_EINVAL if error occurs.

5.2.11 nrc_dma_start

Starts the DMA operation for the channel and the data given as dma_desc_t.

Prototype :

int nrc_dma_start (int channel, dma_desc_t *desc)

Input Parameters :

channel
Type: int
Purpose: DMA channel to be used.

desc
Type: dma_desc_t
Purpose: Linked list of DMA descriptors.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EPERM if DMA is not enabled.
NRC_DMA_EBUSY if DMA is already enabled.
NRC_DMA_EINVAL if error occurs.

5.2.12 nrc_dma_stop

Stops the DMA operation.

Prototype :

int nrc_dma_stop (int channel)

Input Parameters :

channel

Type: int

Purpose: DMA channel to be used.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EPERM if DMA is not enabled.
NRC_DMA_EBUSY if DMA is already enabled.
NRC_DMA_EINVAL if error occurs.

5.2.13 nrc_dma_busy

Check whether the given channel is busy or not.

Prototype :

bool nrc_dma_busy (int channel)

Input Parameters :

channel

Type: int

Purpose: DMA channel to be used.

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

5.2.14 nrc_dma_src_addr

Retrieves the source memory address of DMA channel.

Prototype :

uint32_t nrc_dma_src_addr (int channel)

Input Parameters :

channel

Type: int

Purpose: DMA channel to be used.

Returns :

Source address.

5.2.15 nrc_dma_dest_addr

Retrieves the destination memory address of DMA channel.

Prototype :

uint32_t nrc_dma_dest_addr (int channel)

Input Parameters :

channel

Type: int

Purpose: DMA channel to be used.

Returns :

Destination address.

5.2.16 nrc_dma_desc_print

Prints the dma_desc_t centents for given desc.

Prototype :

void nrc_dma_desc_print (dma_desc_t *desc)

Input Parameters :

desc

Type: dma_desc_t

Purpose: dma_desc_t to be printed.

Returns :

N/A

5.2.17 nrc_dma_desc_init

Initialize DMA descriptor using given arguments.

Prototype :

int nrc_dma_desc_init (dma_desc_t *desc, uint32_t src_addr, uint32_t dest_addr, uint16_t size)

Input Parameters :

desc

Type: dma_desc_t

Purpose: dma_desc_t to be initialized.

src_addr

Type: uint32_t

Purpose: Source address to be used.

dest_addr

Type: uint32_t

Purpose: Destination address to be used.

size

Type: uint16_t

Purpose: Size of data to be transferred.

Returns :

NRC_DMA_OK, if successful.

NRC_DMA_EINVAL, otherwise.

5.2.18 nrc_dma_desc_link

Set the next field in dma_desc_t for given desc to complete dma_desc_t linked list.

Prototype :

int nrc_dma_desc_link (dma_desc_t *desc, dma_desc_t *next)

Input Parameters :

desc

Type: dma_desc_t

Purpose: dma_desc_t next field to be updated.

next

Type: dma_desc_t

Purpose: Dma_desc_t to be linked to desc provided.

Returns :

NRC_DMA_OK, if the operation was successful.

NRC_DMA_EINVAL all other errors.

5.2.19 nrc_dma_desc_set_addr

Update source and destination for given desc.

Prototype :

int nrc_dma_desc_set_addr (dma_desc_t *desc, uint32_t src_addr, uint32_t dest_addr)

Input Parameters :

Desc

Type: dma_desc_t

Purpose: dma_desc_t to be updated.

src_addr

Type: uint32_t

Purpose: Source address to update to.

dest_addr

Type: uint32_t*

Purpose: Destination address to update to.

Returns :

NRC_DMA_OK, if the operation was successful.

NRC_DMA_EINVAL, all other errors.

5.2.20 nrc_dma_desc_set_addr_inc

Update whether to automatically increment source and destination address for given dma_desc_t.

Prototype :

int nrc_dma_desc_set_addr_inc (dma_desc_t *desc, bool src_inc, bool dest_inc)

Input Parameters :

desc

Type: dma_desc_t

Purpose: dma_desc_t to be updated.

src_inc

Type: bool

Purpose: Enable/disable automatic source address incrementation.

dest_inc

Type: bool

Purpose: Enable/disable automatic destination address incrementation.

Returns :

NRC_DMA_OK, if the operation was successful.

NRC_DMA_EINVAL, all other errors.

5.2.21 nrc_dma_desc_set_size

Update transfer size for given dma_desc_t.

Prototype :

int nrc_dma_desc_set_size (dma_desc_t *desc, uint16_t size)

Input Parameters :

desc

Type: dma_desc_t

Purpose: dma_desc_t to be updated.

size

Type: uint16_t

Purpose: Transfer size to update to.

Returns :

NRC_DMA_OK, if the operation was successful.

NRC_DMA_EINVAL, all other errors.

5.2.22 nrc_dma_desc_set_width

Update data width in bits for given dma_desc_t.

Prototype :

int nrc_dma_desc_set_width (dma_desc_t *desc, uint8_t src_width, uint8_t dest_width)

Input Parameters :

desc
Type: dma_desc_t
Purpose: dma_desc_t to be updated.
src_width
Type: uint8_t
Purpose: Source data width in bits.
dest_width
Type: uint8_t*
Purpose: Destination data width in bits.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EINVAL, all other errors.

5.2.23 nrc_dma_desc_set_bsize

Update data burst size for given dma_desc_t.

Prototype :

int nrc_dma_desc_set_bsize (dma_desc_t *desc, uint8_t src_bsize, uint8_t dest_bsize)

Input Parameters :

desc
Type: dma_desc_t
Purpose: dma_desc_t to be updated.
src_bsize
Type: uint8_t
Purpose: Source burst data size.
dest_bsize
Type: uint8_t*
Purpose: Destination burst data size.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EINVAL, all other errors.

5.2.24 nrc_dma_desc_set_inttc

Enable/disable interrupt for given dma_desc_t.

Prototype :

int nrc_dma_desc_set_inttc (dma_desc_t *desc, bool inttc)

Input Parameters :

desc

Type: dma_desc_t
Purpose: dma_desc_t to be updated.

inttc
Type: bool
Purpose: True to enable, false to disable interrupt.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EINVAL, all other errors.

5.2.25 nrc_dma_desc_set_ahb_master

Set source and destination AHB interface.

Prototype :

int nrc_dma_desc_set_ahb_master (dma_desc_t *desc, int src_ahbm, int dest_ahbm)

Input Parameters :

Desc
Type: dma_desc_t
Purpose: dma_desc_t to be updated.

src_ahbm
Type: Int
Purpose: Source AHB master interface.

dest_ahbm
Type: Int
Purpose: Destination AHB master interface.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EINVAL, all other errors.

5.2.26 nrc_dma_desc_set_protection

Update protection scheme to be used for given dma_desc_t.

Prototype :

int nrc_dma_desc_set_protection (dma_desc_t *desc, bool privileged, bool bufferable, bool cacheable)

Input Parameters :

desc
Type: dma_desc_t
Purpose: dma_desc_t to be updated.

privileged
Type: bool
Purpose: Set whether privileged protection be enabled or disabled.

bufferable
Type: bool
Purpose: Set whether bufferable protection be enabled or disabled.
cacheable
Type: bool
Purpose: Set whether cacheable protection be enabled or disabled.

Returns :

NRC_DMA_OK, if the operation was successful.
NRC_DMA_EINVAL, all other errors.

5.3 Callback Functions & Events

The interrupt handler function pointer type.

Prototype :

typedef void (*dma_isr_t) (int channel)

Input Parameters :

channel
Type: int
Purpose: DMA channel

6 UART

The UART API provides functions to:
• Set the UART channel, configurations, interrupt handler and interrupt type
• Get and put a character and print strings

6.1 Data Type

These types are defined at the “lib/sdk/inc/api_uart.h”.

6.1.1 Channel

NRC_UART_CHANNEL is an UART channel. NRC_UART_CHANNEL is an UART channel. The UART0 channels are not available for user use in NRC7292 EVK. It is dedicated for console.

Table 6.1 NRC_UART_CHANNEL

Name	Description
NRC_UART_CH0	Channel 0
NRC_UART_CH1	Channel 1

6.1.2 UART Data Bit

NRC_UART_DATA_BIT is a data bit size.

Table 6.2 NRC_UART_DATA_BIT

Name	Description
NRC_UART_DB5	Data bit 5
NRC_UART_DB6	Data bit 6
NRC_UART_DB7	Data bit 7
NRC_UART_DB8	Data bit 8

6.1.3 UART Stop Bit

NRC_UART_STOP_BIT is a data bit size.

Table 6.3 NRC_UART_STOP_BIT

Name	Description
NRC_UART_SB1	Stop bit 1
NRC_UART_SB2	Stop bit 2

6.1.4 UART Parity Bit

NRC_UART_PARITY_BIT is a type of parity.

Table 6.4 NRC_UART_PARITY_BIT

Name	Description
NRC_UART_PB_NONE	None
NRC_UART_PB_ODD	Odd parity bit
NRC_UART_PB_EVEN	Even parity bit

6.1.5 UART Hardware Flow Control

NRC_UART_HW_FLOW_CTRL indicate that a UART hardware flow control is enabled or disabled.

Table 6.5 NRC_UART_HW_FLOW_CTRL

Name	Description
NRC_UART_HFC_DISABLE	Disable
NRC_UART_HFC_ENABLE	Enable

6.1.6 UARTFIFO

NRC_UART_FIFO indicate that a UART FIFO is enabled or disabled.

Table 6.6 NRC_UART_FIFO

Name	Description
NRC_UART_FIFO_DISABLE	Disable FIFO
NRC_UART_FIFO_ENABLE	Enable FIFO

6.1.7 UART Configuration

NRC_UART_CONFIG is a configuration about UART.

Table 6.7 NRC_UART_CONFIG

Name	Description
ch	Channel number
db	Data bit
br	Baudrate
stop_bit	Stop bit
parity_bit	Parity bit
hw_flow_ctrl	Enable or disable hardware flow control
fifo	Enable or disable FIFO

6.1.8 UART Interrupt Type

NRC_UART_INT_TYPE is an interrupt type.

Table 6.8 NRC_UART_INT_TYPE

Name	Description
NRC_UART_INT_TIMEOUT	Timeout
NRC_UART_INT_RX_DONE	Rx is done
NRC_UART_INT_TX_EMPTY	Tx is empty

6.2 Function Call

The header file for system APIs are defined at the “sdk/inc/api_uart.h”.

6.2.1 nrc_uart_set_config

Set the UART configurations.

Prototype :

nrc_err_t nrc_uart_set_config(NRC_UART_CONFIG *conf)

Input Parameters :

conf

Type: NRC_UART_CONFIG*

Purpose: A pointer to set uart configurations. See “UART Configuration”

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

6.2.2 nrc_hw_set_channel

Set the UART channel

Prototype :

nrc_err_t nrc_uart_set_channel(int ch)

Input Parameters :

ch

Type: int

Purpose: UART channel

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

6.2.3 nrc_uart_get_interrupt_type

Get the UART interrupt type.

Prototype :

nrc_err_t nrc_uart_get_interrupt_type(int ch, NRC_UART_INT_TYPE *type)

Input Parameters :

ch
- Type: int
- Purpose: UART channel

type
- Type: NRC_UART_INT_TYPE *
- Purpose: A pointer to set UART interrupt type. See "UART Interrupt Type"

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

6.2.4 nrc_uart_set_interrupt

Set the UART interrupt.

Prototype :

nrc_err_t nrc_uart_set_interrupt(int ch, bool tx_en, bool rx_en)

Input Parameters :

ch
- Type: int
- Purpose: UART channel

tx_en
- Type: bool
- Purpose: Tx enable flag

rx_en
- Type: bool
- Purpose: Rx enable flag

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

6.2.5 nrc_uart_clear_interrupt

Clear the UART interrupt.

Prototype :

nrc_err_t nrc_uart_clear_interrupt(int ch, bool tx_int, bool rx_int , bool timeout_int)

Input Parameters :

ch
- Type: int
- Purpose: UART channel

tx_en
- Type: bool
- Purpose: Tx enable flag

rx_en
- Type: bool
- Purpose: Rx enable flag

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

6.2.6 nrc_uart_put

Put the character data to UART.

Prototype :

nrc_err_t nrc_uart_put(int ch, char data)

Input Parameters :

ch
- Type: int
- Purpose: UART channel

data
- Type: char
- Purpose: data

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

6.2.7 nrc_uart_get

Get the character data from UART.

Prototype :

nrc_err_t nrc_uart_get(int ch, char *data)

Input Parameters :

ch
- Type: int

Purpose: UART channel

data

Type: char*

Purpose: A pointer to get data

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

6.2.8 nrc_uart_register_interrupt_handler

Register user callback function for UART input.

Prototype :

nrc_err_t nrc_uart_register_interrupt_handler(int ch, intr_handler_fn cb)

Input Parameters :

ch

Type: int

Purpose: timer channel

cb

Type: intr_handler_fn

Purpose: callback function

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

6.2.9 nrc_uart_console_enable

Enable/disable uart print and console command.

Prototype :

nrc_err_t nrc_uart_console_enable(bool enabled)

Input Parameters :

Enabled

Type: bool

Purpose: true or false to enable or disable console print and command.

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

6.3 Callback Functions & Events

The interrupt handler function pointer type is defined at the “sdk/inc/nrc_types.h”.

Prototype :

typedef void (*intr_handler_fn)(int vector)

Input Parameters :

vector

Type: int

Purpose: input vector

7 UART DMA

Helper API to utilize DMA while receiving UART data. Transmitted data will not utilize DMA.

7.1 Data Type

These types are defined in “lib/sdk/inc/api_uart_dma.h”.

7.1.1 UART Device Configuration

UART device configuration setting data structure.

Table 7.1 uart_dma_t

Name	Description
channel	UART channel number
baudrate	BAUD rate to use
data_bits	Data bit : DB5(0), DB6(1), DB7(2), DB8(3)
stop_bits	Stop bit : SB1(0), SB2(1)
parity	Parity : None(0), Odd(1), Even(2)
hfc	Hardware flow control : enable(0)/disable(1)

7.1.2 UART DMA buffer

DMA buffer to use for UART operation

Table 7.2 uart_dma_buf_t

Name	Description
addr	Buffer address
size	Size of buffer

7.1.3 UART RX Parameter

UART RX parameter to be used for RX data handling.

Table 7.3 uart_dma_rx_params_t

Name	Description
buf	UART DMA buffer : uart_dma_buf_t
cb	User supplied callback to receive UART RX data through DMA

7.1.4 UART and DMA configuration

UART and DMA configuration data to be used for initialization.

Table 7.4 uart_dma_info_t

Name	Description
uart	UART device configuration : uart_dma_t
rx_fifo	UART Buffer for RX data
tx_fifo	UART buffer for TX data (Not used.)
rx_params	UART RX parameter : uart_dma_rx_params_t

7.2 Function Call

7.2.1 nrc_uart_dma_open

Initialize UART and DMA setting.

Prototype :

int nrc_uart_dma_open (uart_dma_info_t *info)

Input Parameters :

info

Type: uart_dma_info_t

Purpose: UART setting to be used.

Returns :

0, if the operation was successful.

-1, all other errors.

7.2.2 nrc_uart_dma_close

Close UART and clean up data used for UART/DMA operations.

Prototype :

void nrc_uart_dma_close (void)

Input Parameters :

N/A

Returns :

N/A

7.2.3 nrc_uart_dma_change

Change UART setting if necessary.

Prototype :

int nrc_uart_dma_change (uart_dma_t *uart)

Input Parameters :

uart

Type: uart_dma_t

Purpose: New UART setting to be used.

Returns :

0, if the operation was successful.

-1, all other errors.

7.2.4 nrc_uart_dma_read

Read UART data received using DMA subsystem.

Prototype :

int nrc_uart_dma_read (char *buf, int len)

Input Parameters :

buf

Type: char*

Purpose: Buffer to collect received data.

len

Type: int

Purpose: Size of buffer.

Returns :

Size of UART data received.

7.2.5 nrc_uart_write

Write data to UART. DNA isn't used for TX.

Prototype :

int nrc_uart_write (char *buf, int len)

Input Parameters :

buf

Type: char*

Purpose: Data to be transmitted through UART.

len

Type: int

Purpose: Size of data to be transmitted.

Returns :

Size of UART data transmitted.

7.3 Callback Functions & Events

UART receive callback to retrieve received data.

Prototype :

typedef void (*uart_dma_rxcb_t)(char *buf, int len)

Input Parameters :

buf

Type: char

Purpose: Buffer to retrieve UART RX data

len

Type: int

Purpose: Buffer length

8 GPIO

The GPIO API provides functions to:
• Set the GPIO configurations and interrupt handler
• Get GPIO input values and set GPIO output values

8.1 Data Type

These types are defined at the “lib/sdk/inc/api_gpio.h”.

8.1.1 GPIO Pin

NRC_GPIO_PIN is a GPIO pin number.

Table 8.1 NRC_GPIO_PIN

Name	Description
GPIO_00~GPIO30	GPIO pin number

※The supported GPIO depends on chips. Please reference the hardware guide document.

8.1.2 GPIO Direction

NRC_GPIO_DIR is a GPIO direction.

Table 8.2 NRC_GPIO_DIR

Name	Description
GPIO_INPUT	Input direction
GPIO_OUTPUT	Output direction

8.1.3 GPIO Mode

NRC_GPIO_MODE is a GPIO mode.

Table 8.3 NRC_GPIO_MODE

Name	Description
GPIO_PULL_UP	Pull up
GPIO_PULL_DOWN	Pull down
GPIO_FLOATING	Floating

8.1.4 GPIO Level

NRC_GPIO_LEVEL is a GPIO level.

Table 8.4 NRC_GPIO_LEVEL

Name	Description
GPIO_LEVEL_LOW	0
GPIO_LEVEL_HIGH	1

8.1.5 GPIO Alternative Function

NRC_GPIO_ALT is an alternative function.

Table 8.5 NRC_GPIO_ALT

Name	Description
GPIO_FUNC	GPIO function
GPIO_NOMAL_OP	GPIO Normal operation

8.1.6 GPIO Configurations

NRC_GPIO_CONFIG is a GPIO configuration.

Table 8.6 NRC_GPIO_CONFIG

Name	Description
gpio_pin	Pin number
gpio_dir	Direction
gpio_alt	Alternative function
gpio_mode	Mode

8.1.7 GPIO Interrupt Trigger Mode

GPIO interrupt trigger type.

Table 8.7 nrc_gpio_trigger_t

Name	Description
TRIGGER_EDGE	Edge trigger
TRIGGER_LEVEL	Level trigger

8.1.8 GPIO Interrupt Trigger Level

GPIO interrupt trigger level.

Table 8.8 nrc_gpio_trigger_t

Name	Description
TRIGGER_HIGH	High trigger
TRIGGER_LOW	Low trigger

8.2 Function Call

The header file for system APIs are defined at the “sdk/inc/api_gpio.h”.

8.2.1 nrc_gpio_config

Set the GPIO configuration.

Prototype :

nrc_err_t nrc_gpio_config(NRC_GPIO_CONFIG *conf)

Input Parameters :

conf

Type: NRC_GPIO_CONFIG*

Purpose: A pointer to set GPIO configurations. See “GPIO Configurations”

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

8.2.2 nrc_gpio_output

Set the GPIO data (32bits).

Prototype :

nrc_err_t nrc_gpio_output(uint32_t *word)

Input Parameters :

conf

Type: uint32_t *

Purpose: A pointer to set GPIO output value (32bits)

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

8.2.3 nrc_gpio_outputb

Set the GPIO data for a specified pin number.

Prototype :

nrc_err_t nrc_gpio_outputb(int pin, intlevel)

Input Parameters :

pin

Type: int

Purpose: GPIO pin number

level
Type: int
Purpose: output value level

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

8.2.4 nrc_gpio_input

Get the GPIO data (32bits).

Prototype :

nrc_err_t nrc_gpio_input(uint32_t *word)

Input Parameters :

conf
Type: uint32_t *
Purpose: A pointer to get GPIO output value (32bits)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

8.2.5 nrc_gpio_inputb

Get the GPIO data for a specified pin number.

Prototype :

nrc_err_t nrc_gpio_inputb(int pin, int *level)

Input Parameters :

pin
Type: int
Purpose: GPIO pin number
level
Type: int
Purpose: A pointer to get GPIO input value

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

8.2.6 nrc_gpio_trigger_config

Configure GPIO interrupt trigger (LEVEL/EDGE, HIGH/LOW signal)

※NRC729 can't support this API.

Prototype :

nrc_err_t nrc_gpio_trigger_config(int vector, nrc_gpio_trigger_t trigger,
nrc_gpio_trigger_level_t level, bool debounce)

Input Parameters :

vector
- Type: int
- Purpose: interrupt vector (INT_VECTOR0 or INT_VECTOR1)

trigger
- Type: nrc_gpio_trigger_t
- Purpose: TRIGGER_EDGE or TRIGGER_LEVEL

level
- Type: nrc_gpio_trigger_level_t
- Purpose: TRIGGER_HIGH or TRIGGER_LOW

debounce
- Type: bool
- Purpose: true or false to enable/disable debounce logic

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

8.2.7 nrc_gpio_register_interrupt_handler

Register GPIO interrupt handler.

Prototype :

nrc_gpio_register_interrupt_handler(int pin, intr_handler_fn cb)

Input Parameters :

pin
- Type: int
- Purpose: pin number

cb
- Type: intr_handler_fn
- Purpose: callback function

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

8.3 Callback Functions & Events

The interrupt handler function pointer type is defined at the “sdk/inc/nrc_types.h”.

Prototype :

typedef void (*intr_handler_fn)(int vector)

Input Parameters :

vector

Type: int

Purpose: input vector

9 I2C

The I2C API provides functions to:
• Set the I2C configurations
• I2C initialize, enable, reset
• Read and write byte via I2C

9.1 Data Type

These types are defined at the “lib/sdk/inc/api_i2c.h”.

9.1.1 I2C_CONTROLLER_ID

I2C_CONTROLLER_ID is an i2c channel.

Table 9.1 I2C_CONTROLLER_ID

Name	Description
I2C_MASTER_0	I2C channel 0
I2C_MASTER_1	I2C channel 1
I2C_MASTER_2	I2C channel 2
I2C_MASTER_MAX	Max channel number

9.1.2 I2C_WIDTH

I2C_WIDTH is an i2c data width.

Table 9.2 I2C_WIDTH

Name	Description
I2C_WIDTH_8BIT	8 Bits
I2C_WIDTH_16BIT	16 Bits

9.1.3 I2C_CLOCK_SOURCE

I2C_CLOCK_SOURCE is an i2c clock source.

Table 9.3 I2C_CLOCK_SOURCE

Name	Description
I2C_CLOCK_CONTROLLER	Clock Controller.
I2C_CLOCK_PCLK	PCLK

9.1.4 i2c_device_t

i2c_device_tis an i2c configurations.

Table 9.4 i2c_device_t

Name	Description
pin_sda	SDA pin
pin_scl	SCL pin
clock_source	clock source, 0:clock controller, 1:PCLK
controller	ID of i2c controller to use
clock	i2c clock (Hz)
width	i2c data width
address	i2c address

9.2 Function Call

The header file for system APIs are defined at the “sdk/inc/api_i2c.h”.

9.2.1 nrc_i2c_init

Initialize the I2C controller.

Prototype :

nrc_err_t nrc_i2c_init(i2c_device_t* i2c)

Input Parameters :

i2c

Type: i2c_device_t*

Purpose: A pointer to set i2c configurations

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

9.2.2 nrc_i2c_enable

Enable or disable the I2C controller.

※ Please disable I2C only after a transaction is stopped.

Prototype :

nrc_err_t nrc_i2c_enable(i2c_device_t* i2c, bool enable)

Input Parameters :

i2c
Type: i2c_device_t*
Purpose: A pointer to set i2c configurations
enable
Type: bool
Purpose: I2C controller enable or disable

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

9.2.3 nrc_i2c_reset

Reset the I2C controller.

Prototype :

nrc_err_t nrc_i2c_reset(i2c_device_t* i2c)

Input Parameters :

i2c
Type: i2c_device_t*
Purpose: A pointer to set i2c configurations

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

9.2.4 nrc_i2c_start

Start the I2C operation.

Prototype :

nrc_err_t nrc_i2c_start(i2c_device_t* i2c)

Input Parameters :

i2c
Type: i2c_device_t*
Purpose: A pointer to set i2c configurations

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

9.2.5 nrc_i2c_stop

Stop the I2C operation.

Prototype :

nrc_err_t nrc_i2c_stop(i2c_device_t* i2c)

Input Parameters :

i2c

Type: i2c_device_t*

Purpose: A pointer to set i2c configurations

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

9.2.6 nrc_i2c_writebyte

Write data to the I2C controller.

Prototype :

nrc_err_t nrc_i2c_writebyte(i2c_device_t* i2c, uint8_t data)

Input Parameters :

i2c

Type: i2c_device_t*

Purpose: A pointer to set i2c configurations

data

Type: uint8_t

Purpose: data

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

9.2.7 nrc_i2c_readbyte

Read data from the I2C controller.

Prototype :

nrc_err_t nrc_i2c_readbyte(i2c_device_t* i2c, uint8_t *data, bool ack)

Input Parameters :

i2c

Type: i2c_device_t*

Purpose: A pointer to set i2c configurations

data

Type: uint8_t*
Purpose: A pointer to store the read data

ack
Type: bool
Purpose: ACK flag. If there's no further reading registers, then false. Otherwise, true

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

10 ADC

The ADC API provides functions to:
• Initialize / De-initialize the ADC controller
• Read the ADC controller data

10.1 Data Type

These types are defined at the “lib/sdk/inc/api_adc.h”.

10.1.1 ADC Channel

ADC_CH is an ADC channel. The supported channel number depends on chips.

Table 10.1 ADC_CH

Name	Description
ADC0 – ADC1	ADC channel

10.2 Function Call

The header file for system APIs are defined at the “sdk/inc/api_adc.h”.

10.2.1 nrc_adc_init

Initialize the ADC controller.

Prototype :

nrc_err_t nrc_adc_init(void)

Input Parameters :

N/A

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

10.2.2 nrc_adc_deinit

De-initialize the ADC controller.

Prototype :

nrc_err_t nrc_adc_deinit(void)

Input Parameters :

N/A

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

10.2.3 nrc_adc_get_data

Read the data from the ADC controller.

Prototype :

uint16_t nrc_adc_get_data (ADC_CH id)

Input Parameters :

id

Type: ADC channel id
Purpose: Channel ID

Returns :

ADC value

11 PWM

The PWM API provides functions to:
• Initialize the PWM controller
• Set configuration and enable for PWM

11.1 Data Type

These types are defined at the “lib/sdk/inc/api_pwm.h”.

11.1.1 PWM Channel

PWM_CH is an PWM channel.

Table 11.1 PWM_CH

Name	Description
PWM_CH0	PWM channel 0
PWM_CH1	PWM channel 1
PWM_CH2	PWM channel 2
PWM_CH3	PWM channel 3
PWM_CH4	PWM channel 0
PWM_CH5	PWM channel 1
PWM_CH6	PWM channel 2
PWM_CH7	PWM channel 3

※ The supported PWM channels are different in each chip. Please reference the hardware guide document.NRC7292(CH0-CH3),NRC7394(CH0-CH7)

11.2Function Call

The header file for system APIs are defined at the “sdk/inc/api_pwm.h”.

11.2.1 nrc_pwm_hw_init

Initialize the ADC controller.

Prototype :

nrc_err_t nrc_pwm_hw_init(uint8_t ch, uint8_t gpio_num, uint8_t use_high_clk)

Input Parameters :

ch

Type: uint8_t

Purpose: PWM channel ID. See “PWM Channel”

gpio_num

Type: uint8_t

Purpose: GPIO number assigned for PWM

use_high_clk

Type: uint8_t

Purpose: If 0, then the pulse duration for 1-bit in each pattern is about 20.8us. Otherwise, about 10.4us

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

11.2.2 nrc_pwm_set_config

Set configuration parameters of PWM. One duty cycle consists of 4 pulse patterns(total 128-bit).

※ It starts with the MSB of pattern1 and ends with the LSB of pattern4.

Prototype :

nrc_err_t nrc_pwm_set_config(uint8_t ch, uint32_t pattern1, uint32_t pattern2, uint32_t pattern3, uint32_t pattern4)

Input Parameters :

ch

Type: uint8_t

Purpose: PWM channel ID. See “PWM Channel”

pattern1

Type: uint32_t

Purpose: 1st pulse pattern(Pattern bits 0~31)

pattern2

Type: uint32_t

Purpose: 2nd pulse pattern(Pattern bits 32~63)

pattern3

Type: uint32_t

Purpose: 3rd pulse pattern(Pattern bits 64~95)

pattern4

Type: uint32_t

Purpose: 4th pulse pattern(Pattern bits 96~127)

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

11.2.3 nrc_pwm_set_enable

Enable the specified PWM channel.

Prototype :

nrc_err_t nrc_pwm_set_enable(uint32_t ch, bool enable)

Input Parameters :

ch

Type: uint32_t

Purpose: PWM channel ID. See "PWM Channel"

enable

Type: bool

Purpose: Enable / disable

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

12 SPI

The SPI API provides functions to:
• Initialize and enable the SPI controller
• Write and read byte via SPI

12.1 Data Type

These types are defined at the “lib/sdk/inc/api_spi.h”.

12.1.1 SPI Mode

SPI_MODE is a SPI mode, which is related to CPOL and CPHA values.

※ Refer the Serial Peripheral Interface. (https://en.wikipedia.org/wiki/Serial_Peripheral_Interface)

Table 12.1 SPI_MODE

Name	Description
SPI_MODE0	SPI mode 0 (CPOL=0, CPHA=0)
SPI_MODE1	SPI mode 1 (CPOL=0, CPHA=1)
SPI_MODE2	SPI mode 2 (CPOL=1, CPHA=0)
SPI_MODE3	SPI mode 3 (CPOL=1, CPHA=1)

12.1.2 SPI Frame Bits

SPI_FRAME_BITS is a number of frame bits.

Table 12.2 SPI_FRAME_BITS

Name	Description
SPI_BIT4	SPI 4-bit frame
SPI_BIT5	SPI 5-bit frame
SPI_BIT6	SPI 6-bit frame
SPI_BIT7	SPI 7-bit frame
SPI_BIT8	SPI 8-bit frame
SPI_BIT9	SPI 9-bit frame
SPI_BIT10	SPI 10-bit frame
SPI_BIT11	SPI 11-bit frame
SPI_BIT12	SPI 12-bit frame
SPI_BIT13	SPI 13-bit frame
SPI_BIT14	SPI 14-bit frame
SPI_BIT15	SPI 15-bit frame
SPI_BIT16	SPI 16-bit frame

12.1.3 SPI Controller ID

SPI_CONTROLLER_ID is a SPI controller ID.

Table 12.3 SPI_CONTROLLER_ID

Name	Description
SPI_CONTROLLER_SPI0	SPI 0
SPI_CONTROLLER_SPI1	SPI 1

12.1.4 spi_device_t

spi_device_tis a spi configurations.

Table 12.4 spi_device_t

Name	Description
pin_miso	SPI MISO pin
pin_mosi	SPI MOSI pin
pin_cs	SPI Chip Select pin
pin_sclk	SPI SCLK pin
frame_bits	SPI frame bits
clock	SPI clock
mode	SPI mode
controller	ID of SPI controller to use
irq_save_flag	irq save flag
Isr_handler	Event handler

12.2Function Call

The header file for system APIs are defined at the “sdk/inc/api_spi.h”.

12.2.1 nrc_spi_master_init

Initialize the SPI controller with the specified mode and bits

Prototype :

nrc_err_t nrc_spi_master_init(spi_device_t* spi)

Input Parameters :

spi

Type: spi_device_t

Purpose: spi configuration. See “spi_device_t”

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

12.2.2 nrc_spi_init_cs

Assign the chip select pin and set active high

Prototype :

nrc_err_t nrc_spi_init_cs(uint8_t pin_cs)

Input Parameters :

pin_cs

Type: uint8_t

Purpose: Assign GPIO for chip select

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

12.2.3 nrc_spi_enable

Enable / disable the SPI controller.

Prototype :

nrc_err_t nrc_spi_enable(spi_device_t* spi, bool enable)

Input Parameters :

spi

Type: spi_device_t

Purpose: spi configuration. See "spi_device_t"

enable

Type: bool

Purpose: Enable / disable

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

12.2.4 nrc_spi_start_xfer

Enable CS to continuously transfer data.

Prototype :

nrc_err_t nrc_spi_start_xfer(spi_device_t* spi)

Input Parameters :

spi

Type: spi_device_t

Purpose: spi configuration. See "spi_device_t"

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

12.2.5 nrc_spi_stop_xfer

Disable CS to continuously transfer data.

Prototype :

nrc_err_t nrc_spi_stop_xfer(spi_device_t* spi)

Input Parameters :

spi

Type: spi_device_t

Purpose: spi configuration. See "spi_device_t"

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

12.2.6 nrc_spi_xfer

Transfer the data between master and slave. User can call nrc_spi_xfer multiple times to transmit data.

※This function should run inside nrc_spi_start_xfer() and nrc_spi_stop_xfer().

Prototype :

nrc_err_t nrc_spi_xfer(spi_device_t* spi, uint8_t *wbuffer, uint8_t *rbuffer, uint32_t size)

Input Parameters :

spi

Type: spi_device_t

Purpose: spi configuration. See "spi_device_t"

wbuffer

Type: uint8_t*

Purpose: A pointer to write data

rbuffer

Type: uint8_t*

Purpose: A pointer to read data

size

Type: uint32_t

Purpose: Number of bytes to transfer

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

12.2.7 nrc_spi_writebyte_value

Write one-byte data to the specified register address.

Prototype :

nrc_err_t nrc_spi_writebyte_value(spi_device_t* spi, uint8_t addr, uint8_t data);

Input Parameters :

spi

Type: spi_device_t

Purpose: spi configuration. See "spi_device_t"

addr

Type: uint8_t

Purpose: register address to write data

data

Type: uint8_t

Purpose: data to write

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

12.2.8 nrc_spi_readbyte_value

Read one-byte data to the specified register address.

Prototype :

nrc_err_t nrc_spi_readbyte_value(spi_device_t* spi, uint8_t addr, uint8_t data);

Input Parameters :

spi
- Type: spi_device_t
- Purpose: spi configuration. See "spi_device_t"

addr
- Type: uint8_t
- Purpose: register address to read data

data
- Type: uint8_t*
- Purpose: A pointer to read data

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

12.2.9 nrc_spi_write_values

Write bytes data to the specified register address.

Prototype :

nrc_err_t nrc_spi_write_values(spi_device_t* spi, uint8_t addr, uint8_t *data, int size)

Input Parameters :

spi
- Type: spi_device_t
- Purpose: spi configuration. See "spi_device_t"

addr
- Type: uint8_t
- Purpose: register address to write data

data
- Type: uint8_t*
- Purpose: A pointer to write data

size
- Type: int
- Purpose: write data size. The unit is bytes.

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

12.2.10 nrc_spi_read_values

Read bytes data to the specified register address.

Prototype :

nrc_err_t nrc_spi_read_values(spi_device_t* spi, uint8_t addr, uint8_t *data, int size)

Input Parameters :

spi
- Type: spi_device_t
- Purpose: spi configuration. See "spi_device_t"

addr
- Type: uint8_t
- Purpose: register address to read data

data
- Type: uint8_t*
- Purpose: A pointer to read data

size
- Type: int
- Purpose: read data size. The unit is bytes.

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

12.2.11 nrc_spi_slave_init

Initialize SPI slave with the specified mode and bits.

Prototype :

nrc_err_t nrc_spi_slave_init(spi_device_t* spi)

Input Parameters :

spi
- Type: spi_device_t
- Purpose: spi configuration. See "spi_device_t" . Only SPI_MODE3 is supported.

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

12.2.12 nrc_spi_slave_read

Read a byte data sent from master.

Prototype :

nrc_err_t nrc_spi_slave_read(uint8_t *data)

Input Parameters :

data

Type: uint8_t*

Purpose: A pointer to a byte data read.

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

12.2.13 nrc_spi_slave_write

Read bytes data to the specified register address.

Prototype :

nrc_err_t nrc_spi_slave_write(uint8_t data)

Input Parameters :

data

Type: uint8_t

Purpose: A byte data to be transferred to master.

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

13 SPI DMA

Helper API's to transfer data through SPI using DMA.

13.1 Function Call

13.1.1 spi_dma_init

Initialize SPI using DMA.
spi_device_t should be initialized for SPI configuration and supplied when calling spi_dma_init.

Prototype :

int spi_dma_init(spi_device_t *spi_dma)

Input Parameters :

spi_dma

Type: spi_device_t

Purpose: spi configuration. See "spi_device_t"

Returns :

0, if the operation was successful.

-1, all other errors.

13.1.2 spi_dma_write

Write data to SPI using DMA. SPI related data should be prepared by user.

Prototype :

void spi_dma_write(uint8_t *data, uint32_t size)

Input Parameters :

data

Type: uint8_t

Purpose: To be transferred to SPI device using DMA.

size

Type: uint32_t

Purpose: Size of data.

Returns :

N/A

13.1.3 spi_dma_read

Read data to SPI using DMA. SPI related data should be prepared by user.

Prototype :

void spi_dma_read(uint8_t *addr, uint8_t *data, uint32_t size)

Input Parameters :

addr

Type: SPI peripheral address.

Purpose: Device specific address to be read.

data

Type: uint8_t

Purpose: Buffer to transferred data to be saved.

size

Type: uint32_t*

Purpose: Size for how much data to be read.

Returns :

N/A

14 HTTP Client

The HTTP client API provides functions to:
• HTTP request method (GET, PUT, POST, DELETE)
• Retrieves the response data about request function
※ Note: These APIs are now publicly accessible and have been moved to lib/http_client.

14.1 Data Type

These types are defined at the “lib/http_client/include/nrc_http_client.h”.

14.1.1 HTTP Client Return Types

httpc_ret_e is a return type for HTTP client.

Table 14.1 httpc_ret_e

Name	Description
HTTPC_RET_ERROR_INVALID_METHOD	Invalid HTTP method
HTTPC_RET_ERROR_TLS_SEND_FAIL	TLS send fail
HTTPC_RET_ERROR_TLS_CONNECTION	TLS connection fail
HTTPC_RET_ERROR_PK_LOADING_FAIL	Private key loading fail
HTTPC_RET_ERROR_CERT_LOADING_FAIL	Certificate loading fail
HTTPC_RET_ERROR_SEED_FAIL	Seed creation fail
HTTPC_RET_ERROR_BODY_SEND_FAIL	Request body send fail
HTTPC_RET_ERROR_HEADER_SEND_FAIL	Request Header send fail
HTTPC_RET_ERROR_INVALID_HANDLE	Invalid handle
HTTPC_RET_ERROR_ALLOC_FAIL	Memory allocation fail
HTTPC_RET_ERROR_SCHEME_NOT_FOUND	Scheme(http:// or https://) not found
HTTPC_RET_ERROR_SOCKET_FAIL	Socket creation fail
HTTPC_RET_ERROR_RESOLVING_DNS	Cannot resolve the hostname
HTTPC_RET_ERROR_CONNECTION	Connection fail
HTTPC_RET_ERROR_UNKNOWN	Unknown error
HTTPC_RET_CON_CLOSED	Connection closed by remote
HTTPC_RET_OK	Success

14.1.2 HTTP Client Connection Handle

con_handle_t is a connection handle type for HTTP client.

Table 14.2 con_handle_t

Name	Description
con_handle_t	Connection handle

14.1.3 SSL Certificate Structure

ssl_certs_t is a SSL certificate structure type.

Table 14.3 ssl_certs_t

Name	Description
ca_cert	Certificate Authority Server certification
client_cert	Client certification
client_pk	Client private key
ca_cert_length	Certificate Authority Server certification I, server_cert buffer size
client_cert_length	Client certification I, client_cert buffer size
client_pk_length	Client private key I, client_pk buffer size

14.1.4 HTTP Client Data Type

httpc_data_t is a data type for HTTP client.

Table 14.4 httpc_data_t

Name	Description
data_out	Connection handle
data_out_length	Output buffer length
data_in	Pointer of the input buffer for data receiving
data_in_length	Input buffer length
recved_size	Received data size

14.2Function Call

The header file for system APIs are defined at the “lib/http_client/include/nrc_http_client.h”..

14.2.1 nrc_httpc_get

Executes a GET request on a given URL.

Prototype :

httpc_ret_e nrc_httpc_get(con_handle_t *handle, const char *url, const char *custom_header, httpc_data_t *data, ssl_certs_t *certs)

Input Parameters :

handle

Type: con_handle_t*

Purpose: Connection handle”

url

Type: const char *

Purpose: URL for the request

custom_header

Type: const char *
Purpose: Customized request header. The request-line("<method><uri> HTTP/1.1") and "Host: <host-name>" will be sent in default internally. Other headers can be set as null-terminated string format.

data
Type: httpc_data_t *
Purpose: A pointer to the #httpc_data_t to manage the data sending and receiving

certs
Type: ssl_certs_t *
Purpose: A pointer to the #ssl_certs_t for the certificates

Returns :

HTTPC_RET_OK, if the operation was successful.
Negative error value, all other errors.

14.2.2 nrc_httpc_post

Executes a POST request on a given URL.

Prototype :

httpc_ret_e nrc_httpc_post(con_handle_t *handle, const char *url, const char *custom_header, httpc_data_t *data, ssl_certs_t *certs)

Input Parameters :

handle
Type: con_handle_t*
Purpose: Connection handle"

url
Type: const char *
Purpose: URL for the request

custom_header
Type: const char *
Purpose: Customized request header. The request-line("<method><uri> HTTP/1.1") and "Host: <host-name>" will be sent in default internally. Other headers can be set as null-terminated string format.

data
Type: httpc_data_t *
Purpose: A pointer to the #httpc_data_t to manage the data sending and receiving

certs
Type: ssl_certs_t *
Purpose: A pointer to the #ssl_certs_t for the certificates

Returns :

HTTPC_RET_OK, if the operation was successful.
Negative error value, all other errors.

14.2.3 nrc_httpc_put

Executes a PUT request on a given URL.

Prototype :

httpc_ret_e nrc_httpc_put(con_handle_t *handle, const char *url, const char *custom_header, httpc_data_t *data, ssl_certs_t *certs)

Input Parameters :

handle
- Type: con_handle_t*
- Purpose: Connection handle"

url
- Type: const char *
- Purpose: URL for the request

custom_header
- Type: const char *
- Purpose: Customized request header. The request-line("<method><uri> HTTP/1.1") and "Host: <host-name>" will be sent in default internally. Other headers can be set as null-terminated string format.

data
- Type: httpc_data_t *
- Purpose: A pointer to the #httpc_data_t to manage the data sending and receiving

certs
- Type: ssl_certs_t *
- Purpose: A pointer to the #ssl_certs_t for the certificates

Returns :

HTTPC_RET_OK, if the operation was successful.
Negative error value, all other errors.

14.2.4 nrc_httpc_delete

Executes a DELETE request on a given URL.

Prototype :

httpc_ret_e nrc_httpc_delete(con_handle_t *handle, const char *url, const char *custom_header, httpc_data_t *data, ssl_certs_t *certs)

Input Parameters :

handle
- Type: con_handle_t*
- Purpose: Connection handle"

url
- Type: const char *
- Purpose: URL for the request

custom_header
Type: const char *
Purpose: Customized request header. The request-line("<method><uri> HTTP/1.1") and "Host: <host-name>" will be sent in default internally. Other headers can be set as null-terminated string format.
data
Type: httpc_data_t *
Purpose: A pointer to the #httpc_data_t to manage the data sending and receiving
certs
Type: ssl_certs_t *
Purpose: A pointer to the #ssl_certs_t for the certificates

Returns :

HTTPC_RET_OK, if the operation was successful.
Negative error value, all other errors.

14.2.5 nrc_httpc_delete

Executes a DELETE request on a given URL.

Prototype :

httpc_ret_e nrc_httpc_delete(con_handle_t *handle, const char *url, const char *custom_header, httpc_data_t *data, ssl_certs_t *certs)

Input Parameters :

handle
Type: con_handle_t*
Purpose: Connection handle"
url
Type: const char *
Purpose: URL for the request
custom_header
Type: const char *
Purpose: Customized request header. The request-line("<method><uri> HTTP/1.1") and "Host: <host-name>" will be sent in default internally. Other headers can be set as null-terminated string format.
data
Type: httpc_data_t *
Purpose: A pointer to the #httpc_data_t to manage the data sending and receiving
certs
Type: ssl_certs_t *
Purpose: A pointer to the #ssl_certs_t for the certificates

Returns :

HTTPC_RET_OK, if the operation was successful.

Negative error value, all other errors.

14.2.6 nrc_httpc_recv_response

Retrieves the response data when there are remains after executing the request functions.

Prototype :

httpc_ret_e nrc_httpc_recv_response(con_handle_t *handle, httpc_data_t *data);

Input Parameters :

handle

Type: con_handle_t*

Purpose: Connection handle”

data

Type: httpc_data_t *

Purpose: A pointer to the #httpc_data_t to manage the data sending and receiving

Returns :

HTTPC_RET_OK, if the operation was successful.

Negative error value, all other errors.

14.2.7 nrc_httpc_close

Close the network connection to release system resources. The connection is included in each request method function and this function should be called with every HTTP request.

Prototype :

void nrc_httpc_close(con_handle_t *handle)

Input Parameters :

handle

Type: bool

Purpose: Enable / disable

Returns :

N/A

15 FOTA

The FOTA API provides functions to:
- Check the support of FOTA and set FOTA information
- Erase and write FOTA area.
- Firmware and boot loader FOTA update done function.
- CRC32 calculation.

15.1 Data Type

These types are defined at the “lib/sdk/inc/api_fota.h”.

15.1.1 FOTA Information

FOTA_INFO is an information about FOTA firmware.

Table 15.1 FOTA_INFO

Name	Description
fw_length	Firmware length
crc	CRC32 value
ready	ready flag (Not used)

15.1.2 Broadcast FOTA mode

The broadcast FOTA mode can be configured for the broadcast FOTA operation.

Table 15.2 Broadcast FOTA mode

Name	Description
BC_FOTA_MODE_ANY	Run broadcast FOTA without AP connection
BC_FOTA_MODE_CONNECTED	Run broadcast FOTA when AP connected

15.2Function Call

The header file for system APIs are defined at the “sdk/inc/api_fota.h”.

15.2.1 nrc_fota_is_support

Check the flash is able to support FOTA

Prototype :

bool nrc_fota_is_support(void)

Input Parameters :

N/A

Returns :

True, if it supports FOTA.
False, if it does not support FOTA.

15.2.2 nrc_fota_write

Write data from source address to destination address in FOTA memory area.

Prototype :

nrc_err_t nrc_fota_write(uint32_t dst, uint8_t *src, uint32_t len)

Input Parameters :

dst
- Type: uint32_t
- Purpose: offset from fota_memory start address

src
- Type: uint8_t*
- Purpose: source address

len
- Type: uint32_t
- Purpose: source data length

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

15.2.3 nrc_fota_erase

Erase FOTA memory area

Prototype :

nrc_err_t nrc_fota_erase(void)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

15.2.4 nrc_fota_set_info

Set FOTA binary information (binary length and crc)

Prototype :

nrc_err_t nrc_fota_set_info(uint32_t len, uint32_t crc)

Input Parameters :

len
- Type: uint32_t
- Purpose: binary size

crc
- Type: uint32_t
- Purpose: crc value for binary

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

15.2.5 nrc_fota_update_done

Updated firmware and reboot.

Prototype :

nrc_err_t nrc_fota_update_done(FOTA_INFO* fw_info)

Input Parameters :

fw_info
- Type: FOTA_INFO*
- Purpose: FOTA binary information (binary length and crc)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

15.2.6 nrc_fota_update_done_bootloader

Updated boot loader and reboot.

Prototype :

nrc_err_t nrc_fota_update_done_bootloader(FOTA_INFO* fw_info)

Input Parameters :

fw_info

Type: FOTA_INFO*

Purpose: FOTA binary information (binary length and crc)

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

15.2.7 nrc_fota_cal_crc

Calculate crc32 value.

Prototype :

nrc_err_t nrc_fota_cal_crc(uint8_t* data, uint32_t len, uint32_t *crc)

Input Parameters :

data

Type: uint8_t*

Purpose: A pointer for data

len

Type: uint32_t

Purpose: length for CRC

crc

Type: uint32_t

Purpose: A pointer to store the calculated crc value

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

15.2.8 nrc_bcast_fota_init

Initialize broadcast FOTA

Prototype :

void nrc_bcast_fota_init(void)

Input Parameters :

N/A

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

15.2.9 nrc_bcast_fota_set_mode

Set the broadcast FOTA mode

Prototype :

void nrc_bcast_fota_set_mode(uint8_t mode)

Input Parameters :

mode

Type: uint8_t

Purpose: BC_FOTA_MODE_ANY or BC_FOTA_MODE_CONNECTED. These types are defined at the “lib/sdk/inc/api_bcast_fota.h”.

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

15.2.10 nrc_bcast_fota_enable

Set the broadcast FOTA mode enable or disable

Prototype :

void nrc_bcast_fota_enable(bool enable)

Input Parameters :

enable

Type: uint8_t

Purpose: enable/disable broadcast FOTA

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

16 Power save

The power save memory API provides functions to:
- Set power save mode
- Set wakeup pin and source

16.1 Data Type

These types are defined at the “lib/sdk/inc/api_ps.h”.

16.1.1 Power Save Wakeup Source

These are related to wakeup source.

Table 16.1 POWER_SAVE_WAKEUP_SOURCE

Define	Value
WAKEUP_SOURCE_RTC	0x00000001 << 0
WAKEUP_SOURCE_GPIO	0x00000001 << 1

16.1.2 Power Save Wakeup Reason

These are related to wakeup reason. These are defined at the “sdk/inc/api_ps.h”.

Table 16.2 POWER_SAVE_WAKEUP_REASON

Define	Description
NRC_WAKEUP_REASON_COLDBOOT	Normal power on
NRC_WAKEUP_REASON_RTC	RTC timeout
NRC_WAKEUP_REASON_GPIO	Wakeup by GPIO
NRC_WAKEUP_REASON_TIM	Unicast packet in TIM sleep mode
NRC_WAKEUP_REASON_TIM_TIMER	RTC timeout in TIM sleep mode
NRC_WAKEUP_REASON_NOT_SUPPORTED	Not supported

16.2 Function Call

The header file for system APIs are defined at the “sdk/inc/api_ps.h”.

16.2.1 nrc_ps_deep_sleep

Command the device to go to Non-TIM mode deep sleep.

If used after a previous WiFi pairing has been completed, the device will utilize the saved WiFi connection information in retention memory for faster pairing recovery.

※ The sleep_ms parameter may be overridden by the MIN (BSS MAX IDLE, LISTEN INTERVAL)
BSS MAX IDLE : set to AP with the default value being about 30 minutes
LISTEN INTERVAl : set to STA with the default value being 0

So If you want to use Deep sleep with duration you want, you have to check listen interval and bss max idle should be larger than it.

Prototype :

nrc_err_t nrc_ps_deep_sleep(uint64_t sleep_ms)

Input Parameters :

interval

Type: uint64_t

Purpose: The duration for deep sleep. The unit is ms. (>= 1000ms)

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

16.2.2 nrc_ps_sleep_alone

Command the device to go to Non-TIM deep sleep.
Unlike nrc_ps_deep_sleep, it will not save pairing information, potentially leading to longer WiFi reconnection time. Additionally, this API will not override the sleep duration specified by the sleep_ms parameter.

Prototype :

nrc_err_t nrc_ps_sleep_alone(uint64_t sleep_ms)

Input Parameters :

timeout

Type: uint64_t

Purpose: Duration for deep sleep. The unit is ms. (>= 1000ms)

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

16.2.3 nrc_ps_sleep_forever

Command the device to go to Non-TIM deep sleep forever.

Unlike nrc_ps_deep_sleep, it will not save pairing information, potentially leading to longer WiFi reconnection time.

Prototype :

nrc_err_t nrc_ps_sleep_forever()

Input Parameters :

N/A

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.4 nrc_ps_wifi_tim_deep_sleep

The function commands device to WiFi TIM sleep. The WiFi wakes up if Traffic Indication Map signal received or sleep duration expired. If sleep_ms is set to 0, the device will wakeup only for TIM traffic.

Prototype :

nrc_err_t nrc_ps_wifi_tim_deep_sleep(uint32_t idle_timout_ms, uint32_t sleep_ms)

Input Parameters :

idle_timout_ms

Type:	uint32_t
Purpose:	Wait time before entering the modem sleep. The unit is ms. (0 <= time < 10000ms)

sleep_ms

Type:	uint32_t
Purpose:	Duration for deep sleep. The unit is ms. (0(not use) or time >= 1000ms)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.5 nrc_ps_clear_sleep_mode

The function commands to clear sleep mode in retention memory.

Prototype :

nrc_err_t nrc_ps_clear_sleep_mode(void)

Input Parameters :

N/A

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.6 nrc_ps_set_gpio_wakeup_pin

Configure a wakeup-gpio-pin when system state is uCode or deep sleep.

※ This function should be called before deep sleep, if user want to set the wakeup-gpio-pin.

Prototype :

nrc_err_t nrc_ps_set_gpio_wakeup_pin(bool check_debounce, int pin_number, bool active_high)

Input Parameters :

check_debounce

Type: bool

Purpose: check mechanical vibration of a switch

pin_number

Type: int

Purpose: GPIO pin number for wakeup when GPIO is enabled for wakeup source

active_high

Type: bool

Purpose: Wakeup polarity : true – active high, false – active low

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

16.2.7 nrc_ps_set_gpio_wakeup_pin2

Configure a 2nd wakeup-gpio-pin when system state is uCode or deep sleep.

Prototype :

nrc_err_t nrc_ps_set_gpio_wakeup_pin(bool check_debounce, int pin_number, bool active_high)

Input Parameters :

check_debounce

Type: bool

Purpose: check mechanical vibration of a switch

pin_number

Type: int

Purpose: GPIO pin number for wakeup when GPIO is enabled for wakeup source

active_high

Type: bool

Purpose: Wakeup polarity : true – active high, false – active low

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

16.2.8 nrc_ps_set_wakeup_source

Configure wakeup sources when system state is deepsleep.

※ This function should be called before deepsleep, if user want to set the wakeup source.

Prototype :

nrc_err_t nrc_ps_set_wakeup_source(uint8_t wakeup_source)

Input Parameters :

wakeup_source

Type: uint8_t

Purpose: wakeup source. See "Power Save Wakeup Source"

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

16.2.9 nrc_ps_wakeup_reason

Get the wakeup reason.

Prototype :

nrc_err_t nrc_ps_wakeup_reason(uint8_t *reason)

Input Parameters :

reason

Type: uint8_t*

Purpose: A pointer to get wakeup reason. See "Power Save Wakeup Reason"

Returns :

NRC_SUCCESS, if the operation was successful.

NRC_FAIL, all other errors.

16.2.10 nrc_ps_wakeup_gpio_ext

Retrieve GPIO interrupt group if multiple GPIO wakeup sources are used

Prototype :

nrc_err_t nrc_ps_wakeup_gpio_ext (uint8_t *reason)

Input Parameters :

reason

Type: uint8_t*

Purpose: A pointer to get wakeup reason.
NRC_WAKEUP_GPIO_EXT0 (GPIO 8, 10, 12, 14, 16, 18, 20, 22)
NRC_WAKEUP_GPIO_EXT1, (GPIO 9, 11, 13, 15, 17, 19, 21, 23)
NRC_WAKEUP_GPIO_EXT2, (GPIO 0 - 7)
NRC_WAKEUP_GPIO_EXT3, (GPIO 24 - 31)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.11 nrc_ps_set_gpio_direction

Set the gpio direction mask in deep sleep.

Prototype :

voidnrc_ps_set_gpio_direction(uint32_t bitmask)

Input Parameters :

bitmask
Type: uint32_t
Purpose: Set bitmask of GPIO direction, as bits 0-31 (input:0, output:1)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.12 nrc_ps_set_gpio_out

Set the gpio pullup mask in deep sleep.

Prototype :

voidnrc_ps_set_gpio_out(uint32_t bitmask)

Input Parameters :

bitmask
Type: uint32_t
Purpose: Set bitmask of GPIO out value, as bits 0-31 (low:0, high:1)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.13 nrc_ps_set_gpio_pullup

Set the gpio pullup mask in deep sleep.

Prototype :

voidnrc_ps_set_gpio_pullup(uint32_t bitmask)

Input Parameters :

bitmask
Type: uint32_t

Purpose: Set bitmask of GPIO pullup value, as bits 0-31 (pulldown:0, pullup:1)

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.14 nrc_ps_add_schedule

Add schedules to the deep sleep scheduler (NON TIM mode) timeout, whether to enable Wi-Fi, and callback function to execute when the scheduled time is reached. Current implementation can accept up to 4 individual schedules. Each individual schedule should have at least one minute apart in timeout. When adding schedule the callback should be able to finish in the time window.

Prototype :

nrc_err_t nrc_ps_add_schedule(uint32_t timeout, bool net_init, scheduled_callback func)

Input Parameters :

timeout
- Type: uint32_t
- Purpose: Sleep duration in msec for this schedule

net_init
- Type: bool
- Purpose: Whether callback will require Wi-Fi connection

func
- Type: scheduled_callback
- Purpose: Scheduled callback function pointer defined as void (*scheduled_callback)()

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.15 nrc_ps_add_gpio_callback

Add gpio exception callback to handle gpio interrupted wake up. This information will be added into retention memory and processed if gpio interrupt occurs. If net_init is set to true, then Wi-Fi and network will be initialized.

Prototype :

nrc_err_t nrc_ps_add_gpio_callback(bool net_init, scheduled_callback func)

Input Parameters :

net_init
- Type: bool
- Purpose: Whether callback will require Wi-Fi connection

func

Type:	scheduled_callback
Purpose:	Scheduled_callback function pointer defined as void (*scheduled_callback) ()

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.16 nrc_ps_start_schedule

Start the scheduled deep sleep configured using nrc_ps_add_schedule.

Prototype :

nrc_err_t nrc_ps_start_schedule()

Input Parameters :

N/A

Returns :

NRC_SUCCESS, if the operation was successful.
NRC_FAIL, all other errors.

16.2.17 nrc_ps_resume_deep_sleep

Command the device to go to deep sleep for remaining scheduled time. This function is used to sleep after none-scheduled wakeup such as GPIO interrupt.

Prototype :

void nrc_ps_resume_deep_sleep()

Input Parameters :

N/A

Returns :

None

17 PBC (Push Button)

WPS-PBC for simple network configuration

17.1 Data Type

These types are defined at the “sdk/inc/api_pbc.h”.

17.1.1 pbc_ops

pbc_ops are a structure type.

Table 17.1 pbc_ops

Name	Description
GPIO_PushButton	WPS-PBC GPIO for push button
nrc_wifi_wps_pbc_fail	WPS-PBC operation fail
nrc_wifi_wps_pbc_timeout	WPS-PBC operation timeout
nrc_wifi_wps_pbc_success	WPS-PBC operation success
nrc_wifi_wps_pbc_pressed	WPS-PBC operation press

17.2 Function Call

The header file for PBC APIs is defined at the “sdk/inc/api_pbc.h”.

17.2.1 wps_pbc_fail_cb

This callback is called when WPS-PBC operation fail

Prototype :

void wps_pbc_fail_cb(void *priv)

Input Parameters :

priv

Type: void*

Purpose: A pointer for nrc_wpa_if

Returns :

N/A

17.2.2 wps_pbc_timeout_cb

This callback is called when there is no connection attempt for 120 second and timeout occurs.

Prototype :

void wps_pbc_timeout_cb(void *priv)

Input Parameters :

priv

Type: void*

Purpose: A pointer for nrc_wpa_if

Returns :

N/A

17.2.3 wps_pbc_success_cb

This callback is called when WPS-PBC operation success

Prototype :

static void wps_pbc_success_cb(void *priv, int net_id, uint8_t *ssid, uint8_t ssid_len, uint8_t security_mode, char *passphrase)

Input Parameters :

priv

Type: void*

Purpose: A pointer for nrc_wpa_if

net_id

Type: int_t

Purpose: network id

ssid

Type: uint8_t

Purpose: SSID

ssid_len

Type: uint8_t

Purpose: SSID length

security_mode

Type: uint8_t

Purpose: See the " Security Mode"

passphrase

Type: char*

Purpose: WPA ASCII passphrase (ASCII passphrase must be between 8 and 63 characters)

Returns :

N/A

17.2.4 wps_pbc_button_pressed_event

This callback is called when user push the button which is connected with GPIO. This GPIO is registered for interrupt.

Prototype :

void wps_pbc_button_pressed_event(int vector)

Input Parameters :

vector

Type: int

Purpose: GPIO pin number for wakeup when GPIO is enabled for wakeup source

Returns :

17.2.5 init_wps_pbc

Initialize WPS-PBC function

Prototype :

void init_wps_pbc(struct pbc_ops *ops)

Input Parameters :

ops

Type: struct pbc_ops *

Purpose: structure contains GPIO and callbacks

Returns :

N/A

18 Middleware API Reference

18.1 FreeRTOS

FreeRTOS is a market-leading real–time operating system (RTOS) for microcontrollers and small microprocessors.

- Official Website:
  - https://www.freertos.org/RTOS.html
- Online Documentation:
  - https://www.freertos.org/features.html
- Git Repository:
  - https://github.com/FreeRTOS/FreeRTOS

18.2 WPA_supplicant

Wpa_supplicant is a WPA Supplicant for Linux, BSD, Mac OS X, and Windows with support for WPA and WPA2 (IEEE 802.11i / RSN). Supplicant is the IEEE 802.1X/WPA component that is used in the client stations. It implements key negotiation with a WPA authenticator, and it controls the roaming and IEEE 802.11 authentication/association of the wlan driver.

- Official website:
  - https://w1.fi/wpa_supplicant/
- Online Documentation:
  - https://w1.fi/wpa_supplicant/devel/
- GitHub Page:
  - git clone git://w1.fi/srv/git/hostap.git

18.3 lwIP

lwIP (lightweight IP) is a widely used open-source TCP/IP stack designed for embedded systems.

- Official Website:
  - http://savannah.nongnu.org/projects/lwip
- Online Documentation:
  - http://www.nongnu.org/lwip
- Git Repository:
  - https://git.savannah.nongnu.org/git/lwip.git

18.1 MbedTLS

MbedTLS is an implementation of the TLS and SSL protocols and the respective cryptographic algorithms and support code required.

- Official Website:
  - https://tls.mbed.org
- Online API Reference:
  - https://tls.mbed.org/api
- GitHub Page:
  - https://github.com/ARMmbed/mbedtls

18.2 NVS library

NVS library used for storing data values in the flash memory. Data are stored in a non-volatile manner, so it is remaining in the memory after power-out or reboot. This lib is inspired and based on TridentTD_ESP32NVS work.

The NVS stored data in the form of key-value. Keys are ASCII strings, up to 15 characters. Values can have one of the following types:

- integer types: uint8_t, int8_t, uint16_t, int16_t, uint32_t, int32_t, uint64_t, int64_t
- zero-terminated string
- variable length binary data (blob)

Refer to the NVS ESP32 lib original documentation for a details about internal NVS lib organization.

19 Abbreviations

Table 19.1 Abbreviations and acronyms

Name	Description
IP	Internet Protocol
LwIP	Lightweight Internet Protocol
SDK	Software Development Kit
SDK	Software Development Kit
API	Application Programming Interface
EVB	Evaluation Board
AP	Access Point
STA	Station
SSID	Service Set Identifier
BSSID	Basic Service Set Identifier
RSSI	Received Signal Strength Indication
SNR	Signal-to-noise ratio
WPA2	Wi-Fi Protected Access 2
WPA3-SAE	Wi-Fi Protected Access 3 – Simultaneous Authentication of Equals
WPA3-OWE	Wi-Fi Protected Access 3 – Opportunistic Wireless Encryption
EAP	Extensible Authentication Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
AID	Association ID
MAC	Medium Access Control
dBm	Decibel-milliwatts
S1G	Sub 1 GHz
HAL	Hardware Abstract Layer
ADC	Analog-to-Digital Converter
UART	Universal Asynchronous Receiver-Transmitter
PWM	Pulse-Width Modulation
SPI	Serial Peripheral Interface
TPC	Transmission Power Control
GPIO	General-purpose input/output
CPOL	Clock Polarity
CPHA	Clock Phase
TIM	Traffic Indication Map
NVS	Non-Volatile Storage
BI	Beacon Interval

20 Revision history

Revision No	Date	Comments
Ver 1.0	8/2/2023	Initial version
Ver 1.1	8/16/2023	Added 'VERSION_T' Update APIs Added functions: nrc_wifi_get_average_rssi(), nrc_set_app_version(), nrc_get_app_version(), nrc_set_app_name(), nrc_get_app_name(), nrc_get_sdk_version() Updated functions: nrc_wifi_get_rssi(), nrc_wifi_get_snr() : Added vif_id in parameter wps_pbc_fail_cb(), wps_pbc_timeout_cb(), wps_pbc_success_cb() : Added void* parameters
Ver 1.2	11/28/2023	Update type defines : Removed 'ADC_AVRG' type Update description about nrc_ps_deep_sleep() Update APIs Added functions: nrc_set_flash_device_info(), nrc_get_flash_device_info(), nrc_wifi_softap_get_max_num_sta(), nrc_wifi_softap_set_max_num_sta(), nrc_wifi_set_tx_aggr_auto(), nrc_wifi_set_beacon_loss_detection(), nrc_wifi_get_listen_interval(), nrc_wifi_set_listen_interval(),nrc_wifi_softap_get_beacon_interval(), nrc_wifi_softap_set_beacon_interval(), nrc_wifi_get_mcs_info(), nrc_wifi_softap_get_ignore_broadcast_ssid(), nrc_wifi_softap_set_ignore_broadcast_ssid(), nrc_get_user_data_area_size(), nrc_write_user_data(), nrc_read_user_data(),nrc_wifi_get_mic_scan(), nrc_wifi_set_mic_scan() tWIFI_IGNORE_BROADCAST_SSID type Updated functions: nrc_wifi_get_tx_power(): Added a vif_id parameter Removed functions: nrc_adc_avrg_sel(), nrc_wifi_softap_get_hidden_ssid(), nrc_wifi_softap_set_hidden_ssid() ※ hidden_ssid APIs are replaced by ignore_broadcast_ssid APIs
Ver 1.3	11/22/2024	Added functions: nrc_get_xtal_status() Updated the decription about nrc_httpc_close() Added chapters for DMA, UART DMA and SPI DMA Updated nrc_ps_set_gpio_wakeup_pin Updated tWIFI_COUNTRY_CODE table, STA_INFO Update APIs

		Added functions: nrc_wifi_set_eap_security(), nrc_wifi_set_sae_pwe(), nrc_wifi_get_sae_pwe(), nrc_wifi_wps_cancel(), nrc_ps_clear_sleep_mode(), nrc_ps_set_gpio_wakeup_pin2(), nrc_wifi_set_simple_bgscan(), nrc_get_user_factory_info() Updated functions: nrc_wifi_softap_set_conf(),nrc_wifi_set_ip_address(), wps_pbc_fail_cb(), wps_pbc_timeout_cb(), wps_pbc_success_cb(), nrc_adc_get_data(), spi_dma_init() SPI Slave operation support API's added. nrc_spi_slave_init(), nrc_spi_slave_read(), nrc_spi_slave_write() Distributed Authentication control support API's added. nrc_wifi_set_enable_auth_control(), nrc_wifi_get_enable_auth_control(), nrc_wifi_set_auth_control_param(), nrc_wifi_get_auth_control_param(), nrc_wifi_set_auth_control_scale(), nrc_wifi_get_auth_control_scale(), nrc_wifi_get_auth_current_ti()
Ver1.3.1	12/12/2024	Added bandwidth in SCAN_RESULT Update nrc_wifi_enable_duty_cycle() description