

NRC7292 Evaluation Kit

User Guide

(SDK API Manual)

Ultra-low power & Long-range Wi-Fi

Ver1.6
July 23, 2020

NEWRACOM, Inc.

NRC7292 Evaluation Kit User Guide (SDK API Manual)

Ultra-low power & Long-range Wi-Fi

© 2020 NEWRACOM, Inc.

All right reserved. No part of this document may be reproduced in any form without written permission from NEWRACOM.

NEWRACOM reserves the right to change in its products or product specification to improve function or design at any time without notice.

Office

NEWRACOM, Inc.

25361 Commercentre Drive, Lake Forest, CA 92630 USA

<http://www.NEWRACOM.com>

Contents

1	API Reference.....	4
1.1	Wi-Fi	4
1.2	Timer	14
1.3	UART.....	15
1.4	GPIO	19
1.5	I2C	21
1.6	ADC.....	23
1.7	PWM.....	24
1.8	SPI.....	26
1.9	HTTP Client.....	28
1.10	FOTA.....	31
1.11	SerialFlash	33
1.12	Power save	34
2	Middleware API Reference	35
2.1	FreeRTOS.....	35
2.2	lwIP	35
2.3	MbedTLS.....	35
2.4	cJSON.....	35
2.5	MQTT.....	36
2.6	CoAP	36
2.7	Mini-XML.....	36
3	Revision history.....	37

1 API Reference

1.1 Wi-Fi

1.1.1 void nrc_wifi_register_event_handler (event_callback_fn fn)

Register a Wi-Fi event handler callback function pointer. The callback function will be called each time a Wi-Fi event happens and should not be time-consuming.

Parameters:

<i>fn</i>	Callback function pointer
-----------	---------------------------

Returns:

N/A

1.1.2 bool nrc_wifi_set_dhcp (bool dhcp, char *ip_addr)

Set the DHCP option and manually configure the static IP if DHCP mode is disabled.

Parameters:

<i>dhcp</i>	true: Enable DHCP, false: Disable DHCP and use a static IP.
<i>ip_addr</i>	Static IP address (must be provided if DHCP is set to false)

Returns:

true: success

false: fail

1.1.3 bool nrc_wifi_get_dhcp (void)

Set the DHCP mode.

Parameters:

N/A

Returns:

true: enabled

false: disabled

1.1.4 **int nrc_wifi_get_nd (void)**

Request a network index associated with the Wi-Fi connection.

Parameters:

N/A

Returns:

On success, a nonnegative network index is returned. Otherwise, one of the negative-valued error codes (WIFI_GET_ND or WIFI_SET_FAIL) is returned.

1.1.5 **int nrc_wifi_get_network_index (void)**

Get the current network index.

Parameters:

N/A

Returns:

The current network index.

1.1.6 **int nrc_wifi_set_security (int index, int mode, char *password)**

Set the security parameters for Wi-Fi connection.

Parameters:

<i>index</i>	Network index
<i>mode</i>	Security mode: OPEN, WPA, WPA2 (recommended)
<i>password</i>	Password

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.7 **int nrc_wifi_set_ssid (int index, char * ssid)**

(For STA only) Set the SSID of the AP to establish connection with.

Parameters:

<i>index</i>	Network index
<i>ssid</i>	SSID (max 20 bytes)

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.8 **int nrc_wifi_scan (void)**

Start scanning and block until the scanning procedure is complete.

Parameters:

N/A

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.9 **int nrc_wifi_scan_results (SCAN_RESULTS *results)**

Get scan results. (should be called after calling nrc_wifi_scan())

Parameters:

<i>Results</i>	scan list (refer to SCAN_RESULTS structure)
----------------	---

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.10 int nrc_wifi_connect (int index)

Attempt a new connection with the AP and block until the procedure is complete.

Parameters:

<i>index</i>	network index
--------------	---------------

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.11 int nrc_wifi_connect_async (int index)

Asynchronously attempt a new connection with the AP without blocking.

Parameters:

<i>index</i>	network index
--------------	---------------

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.12 int nrc_wifi_disconnect (int index)

Asynchronously disconnect from the AP without blocking.

Parameters:

<i>index</i>	network index
--------------	---------------

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.13 int nrc_wifi_disconnect_sync (int index)

Disconnect from the AP and block until the disconnect procedure is complete.

Parameters:

<i>index</i>	network index
--------------	---------------

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.14 int nrc_wifi_set_country (char *country_code)

Set the country code.

Parameters:

<i>country_code</i>	Country code ("US", "KR", "JP", "CN", "EU", "TW") Default: "KR" (refer to Appendix)
---------------------	--

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.15 int nrc_wifi_get_ip (void)

Request a dynamic IP via DHCP or set a static IP. For DHCP, the function blocks until the allocation procedure is complete.

Parameters:

N/A

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.16 char* nrc_wifi_get_ip_address (void)

Get the current IP address.

Parameters:

N/A

Returns:

On success, the IP address is returned. On error, NULL is returned.

1.1.17 int nrc_wifi_softap_set_ip (char* ip_addr)

Set the Soft AP IP address.

Parameters:

<i>ip_addr</i>	IP address
----------------	------------

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SOFTAP_FAIL is returned.

1.1.18 int nrc_wifi_softap_set_conf (int index, char *ssid, int channel, int sec_mode, char *password)

Set the Soft AP configuration parameters.

Parameters:

<i>index</i>	Network index
<i>ssid</i>	SSID
<i>channel</i>	S1G(Sub 1G) channel index (refer to Appendix)
<i>sec_mode</i>	Security mode
<i>password</i>	Password (only valid if <i>sec_mode</i> is not open)

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.19 int nrc_wifi_softap_start (int index)

Start the Soft AP operation and block until the start procedure is complete.

Parameters:

<i>index</i>	Network index
--------------	---------------

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.20 int nrc_wifi_softap_start_dhcp_server (void)

Start a DHCP server and block until the start procedure is complete. (Only valid while operating as a Soft AP)

Parameters:

N/A

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.21 WLAN_STATE_ID nrc_wifi_get_state (void)

Get the current Wi-Fi connection state.

Parameters:

N/A

Returns:

WLAN_STATE_INIT
WLAN_STATE_READY
WLAN_STATE_TRY_CONNECT
WLAN_STATE_CONNECTED
WLAN_STATE_TRY_GET_IP
WLAN_STATE_GET_IP
WLAN_STATE_TRY_DISCONNECT
WLAN_STATE_DISCONNECTED
WLAN_STATE_SOFTAP_CONF
WLAN_STATE_SOFTAP_START
WLAN_STATE_DHCP_START

1.1.22 void nrc_wifi_set_state (WLAN_STATE_ID state)

Configure the Wi-Fi connection state.

Parameters:

<i>state</i>	Wi-Fi connection state
--------------	------------------------

Returns:

N/A

1.1.23 int nrc_wifi_set_tx_power (int tx_power)

Configure the TX power level.

Parameters:

<i>tx_power</i>	TX power level (1 ~ 30) in dBm
-----------------	--------------------------------

Returns:

On success, the status code WIFI_SUCCESS is returned. On error, the status code WIFI_SET_FAIL is returned.

1.1.24 int nrc_wifi_get_tx_power (void)

Get the current TX power level.

Parameters:

N/A

Returns:

TX power level (1 ~ 30) in dBm

1.1.25 int8_t nrc_wifi_get_rssi (void)

Get the current RSSI level.

Parameters:

N/A

Returns:

RSSI in dBm

1.1.26 int nrc_wifi_get_snr (void)

Get the current SNR level.

Parameters:

N/A

Returns:

SNR in dB

1.1.27 char* nrc_wifi_get_mac_address (void)

Get the MAC address.

Parameters:

N/A

Returns:

The stored MAC Address.

1.1.28 void nrc_wifi_set_rate_control (bool enable)

Set the MCS rate control option.

Parameters:

<i>enable</i>	true: enable, false: disable
---------------	------------------------------

Returns:

N/A

1.1.29 bool nrc_wifi_get_rate_control (void)

Get the MCS rate control option.

Parameters:

N/A

Returns:

true: enabled, false: disabled

1.1.30 void nrc_wifi_set_mcs (uint8_t mcs)

Set the MCS index.

Parameters:

<i>mcs</i>	MCS index (0~7 or 10)
------------	-----------------------

Returns:

N/A

1.1.31 **bool nrc_wifi_get_mcs (void)**

Get the current MCS index

Parameters:

N/A

Returns:

The current MCS index (0~7 or 10).

1.2 Timer

1.2.1 void nrc_timers_init (void)

Initialize timers

Parameters:

N/A

Returns:

N/A

1.2.2 timer_id nrc_timer_create (uint64_t time, bool repeat, timer_callback handler)

Create and start a timer.

Parameters:

<i>time</i>	Time duration value in microsecond
<i>repeat</i>	true (repeat) or false (single)
<i>handler</i>	Callback handler function when the timer expired

Returns:

The timer ID. (Can be used to stop the timer using nrc_timer_stop())

1.2.3 void nrc_timer_stop (timer_id id)

Stop the timer associated with the timer ID.

Parameters:

<i>id</i>	Timer ID
-----------	----------

Returns:

N/A

1.3 UART

1.3.1 **bool nrc_uart_set_channel (int ch)**

Set the UART channel index.

Parameters:

<i>ch</i>	UART channel index (0~3)
-----------	--------------------------

Returns:

On success, true is returned. Otherwise, false is returned.

1.3.2 **void nrc_uart_register_intr_handler (int ch, intr_handler_fn cb)**

Register an interrupt handler for the specified UART channel. The callback function should not be time-consuming.

Parameters:

<i>ch</i>	UART channel index (0~3)
<i>cb</i>	Callback function of interrupt handler

Returns:

N/A

1.3.3 **void nrc_uart_set_config (NRC_UART_CONFIG * conf)**

Set the configuration for the current UART channel.

Parameters:

<i>conf</i>	UART configuration structure (included in api_uart.h)
-------------	---

Returns:

N/A

1.3.4 void nrc_uart_set_interrupt (int ch, bool tx_en, bool rx_en)

Enable or disable interrupt for the specified UART channel.

Parameters:

<i>ch</i>	UART channel index (0~3)
<i>tx_en</i>	true: enable TX interrupt, false: disable TX interrupt
<i>rx_en</i>	true: enable RX interrupt, false: disable RX interrupt

Returns:

N/A

1.3.5 void nrc_uart_intr_clr (int ch, bool tx_int, bool rx_int, bool to_int)

Clear interrupt with a specific channel.

Parameters:

<i>ch</i>	UART channel index (0~3)
<i>tx_en</i>	true: clear TX interrupt, false: do nothing
<i>rx_en</i>	true: clear RX interrupt, false: do nothing
<i>to_int</i>	true: clear RX timeout interrupt, false: do nothing

Returns:

N/A

1.3.6 bool nrc_uart_get (int ch, char * c)

Get a byte from the specified UART channel.

Parameters:

<i>ch</i>	UART channel index (0~3)
<i>c</i>	Pointer to which the received byte will be written

Returns:

true(1) or false(0)

1.3.7 int nrc_uart_get_intr_type (int ch)

Get the interrupt type for the specified UART channel.

Parameters:

<i>ch</i>	UART channel index (0~3)
-----------	--------------------------

Returns:

Interrupt type

UART_INT_NONE

UART_INT_ERROR

UART_INT_TIMEOUT

UART_INT_RX_DONE

UART_INT_TX_EMPTY

1.3.8 bool nrc_uart_put (int ch, char c)

Send a byte through the specified UART channel.

Parameters:

<i>ch</i>	UART channel index (0~3)
<i>c</i>	Byte to send.

Returns:

true(1) or false(0)

1.3.9 void nrc_uart_console_enable(void)

Enable the UART debug console on UART channel 3.

Parameters:

N/A

Returns:

N/A

1.3.10 void nrc_uart_printf(const char *f, ...)

Print on the debug console.

Parameters:

<i>f</i>	Format
...	Optional arguments

Returns:

N/A

1.3.11 void nrc_uart_vprintf(const char *f, va_list ap)

Print on the debug console.

Parameters:

<i>f</i>	Format
<i>ap</i>	Arguments

Returns:

N/A

※ Defined structure

```
typedef struct {
    int ch;                                /* UART Channel Index */
    NRC_UART_DATA_BIT db;                  /* Data bit */
    int br;                                /* Baud rate */
    NRC_UART_STOP_BIT stop_bit;            /* Stop bit */
    NRC_UART_PARITY_BIT parity_bit;        /* Parity bit */
    NRC_UART_HW_FLOW_CTRL hw_flow_ctrl;    /* HW flow control */
    NRC_UART_FIFO fifo;                    /* FIFO */
} NRC_UART_CONFIG;
```

1.4 GPIO

1.4.1 void nrc_gpio_config (NRC_GPIO_CONFIG * conf)

Set the GPIO configuration (direction, alternative function, mode).

Parameters:

<i>conf</i>	GPIO configuration (refer to api_gpio.h for more information)
-------------	---

Returns:

N/A

1.4.2 void nrc_gpio_register_intr_handler (NRC_GPIO_PIN pin, intr_handler_fn cb)

Register an interrupt handler callback function with the specified GPIO pin.

Parameters:

<i>pin</i>	GPIO pin index (refer to the datasheet for more information)
<i>cb</i>	Interrupt handler callback function

Returns:

N/A

1.4.3 int nrc_gpio_inputb (int pin)

Read from the specified GPIO pin.

Parameters:

<i>pin</i>	GPIO pin index (refer to the datasheet for more information)
------------	--

Returns:

0(LOW) or 1(HIGH)

1.4.4 void nrc_gpio_outputb (int pin, int level)

Write to the specified GPIO pin.

Parameters:

<i>pin</i>	GPIO pin index (refer to the datasheet for more information)
<i>level</i>	GPIO level (0: LOW or 1: HIGH)

Returns:

N/A

※ Defined structure

```
typedef struct {  
    NRC_GPIO_PIN gpio_pin;      /* Pin number */  
    NRC_GPIO_DIR gpio_dir;      /* Direction */  
    NRC_GPIO_ALT gpio_alt;      /* Alternative function */  
    NRC_GPIO_MODE gpio_mode;    /* Mode */  
} NRC_GPIO_CONFIG;
```

1.5 I2C

1.5.1 void nrc_i2c_init (uint32_t clk)

Initialize the I2C controller.

Parameters:

<i>clk</i>	I2C controller clock speed in Hz. (Max: 400,000)
------------	--

Returns:

N/A

1.5.2 void nrc_i2c_enable (bool enable)

Enable or disable the I2C controller.

Parameters:

<i>enable</i>	true(enable) or false(disable)
---------------	--------------------------------

Returns:

N/A

1.5.3 void nrc_i2c_ch_reset (void)

Reset the I2C controller.

Parameters:

N/A

Returns:

N/A

1.5.4 void nrc_i2c_start (void)

Start the I2C operation.

Parameters:

N/A

Returns:

N/A

1.5.5 **bool nrc_i2c_writebyte (uint8_t data)**

Write data to the I2C controller.

Parameters:

<i>data</i>	Data to write
-------------	---------------

Returns:

true or false

1.5.6 **bool nrc_i2c_readbyte (uint8_t * data, bool ack)**

Read data from the I2C controller.

Parameters:

<i>data</i>	Pointer to store the read byte.
<i>ack</i>	true(ACK) or false(NACK)

Returns:

true

1.5.7 **bool nrc_i2c_waitack (void)**

Block until receiving ACK or NACK from the I2C controller.

Parameters:

N/A

Returns:

true(ACK) or false(NACK)

1.5.8 **void nrc_i2c_stop (void)**

Stop the I2C operation.

Parameters:

N/A

Returns:

N/A

1.6 ADC

1.6.1 void nrc_nadc_init (void)

Initialize the ADC controller.

Parameters:

N/A

Returns:

N/A

1.6.2 void nrc_nadc_fini (void)

Finalize the ADC controller.

Parameters:

N/A

Returns:

N/A

1.6.3 uint16_t nrc_nadc_get_data (uint32_t id)

Read data from the specified channel

Parameters:

<i>id</i>	Channel ID (1~3)
-----------	------------------

Returns:

Data value (0x000 ~ 0x1FF)

1.7 PWM

1.7.1 void nrc_pwm_init (uint8_t ch, uint8_t gpio_num, uint8_t use_high_clk)

Initialize a PWM controller and assign it to the specified GPIO pin.

Parameters:

<i>ch</i>	PWM channel index (0~3)
<i>gpio_num</i>	GPIO pin allocation index (8~11)
<i>use_high_clk</i>	If 0, then a 1-bit pulse duration is about 20.8us. Otherwise, the duration is about 10.4us

Returns:

N/A

1.7.2 void nrc_pwm_config (uint8_t ch, uint32_t pattern1, uint32_t pattern2, uint32_t pattern3, uint32_t pattern4)

Configure the duty cycle for the specified PWM channel. One duty cycle consists of 128-bit pulse patterns. The pattern begins at the MSB of pattern1 and ends at the LSB of pattern4.

Parameters:

<i>Ch</i>	PWM channel index (0~3)
<i>pattern1</i>	Pattern bits 0~31
<i>pattern2</i>	Pattern bits 32~63
<i>pattern3</i>	Pattern bits 64~95
<i>pattern4</i>	Pattern bits 96~127

Returns:

N/A

1.7.3 void nrc_pwm_enable (uint32_t ch, bool enable)

Enable the specified PWM channel.

Parameters:

<i>ch</i>	PWM channel index (0~3)
<i>enable</i>	true(enable) or false(disable)

Returns:

N/A

1.8 SPI

1.8.1 void nrc_spi_init (enum spi_mode mode, enum spi_frame_bits bits, uint32_t clock)

Initialize the SPI controller.

Parameters:

<i>mode</i>	SPI mode ([CPOL, CPHA] = 0: [L, L], 1: [L, H], 2: [H, L], 3: [H, H])
<i>bits</i>	SPI frame bits
<i>clock</i>	SPI clock frequency

Returns:

N/A

1.8.2 void nrc_spi_enable (bool enable)

Enable or disable SPI.

Parameters:

<i>enable</i>	true(enable) or false(disable)
---------------	--------------------------------

Returns:

N/A

1.8.3 void nrc_spi_writebyte (uint8_t reg, uint8_t data)

Write a byte value to the specified register address.

Parameters:

<i>Reg</i>	Register address
<i>Data</i>	Byte value

Returns:

N/A

1.8.4 `uint8_t nrc_spi_readbyte (uint8_t reg)`

Read from the specified register address.

Parameters:

<i>reg</i>	Register address
------------	------------------

Returns:

The byte value read from the specified register address.

1.8.5 `uint32_t nrc_spi_xfer (uint8_t *wbuffer, uint8_t *rbuffer, uint32_t size)`

Transfer data between the master and the slave.

Parameters:

<i>wbuffer</i>	Write buffer pointer
<i>rbuffer</i>	Read buffer pointer
<i>size</i>	Number of bytes to transfer

Returns:

The actual number of transferred bytes.

1.9 HTTP Client

1.9.1 `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)`

Send a GET request to the specified URL.

Parameters:

<i>handle</i>	Connection handle
<i>url</i>	Destination URL
<i>custom_header</i>	Custom HTTP request header. The headers (" <code><method></code> <code><uri></code> HTTP/1.1") and "Host: <code><host-name></code> " will be always sent by default. The string <code>custom_header</code> will be appended to the default header.
<i>data</i>	Pointer to the 'httpc_data_t' data handling structure.
<i>certs</i>	Pointer to the 'ssl_certs_t' for certificates structure.

Returns:

Error code (`httpc_ret_e`). See `api_httpc.h` for more information.

1.9.2 `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)`

Send a POST request to the specified URL.

Parameters:

<i>handle</i>	Connection handle
<i>url</i>	Destination URL
<i>custom_header</i>	Custom HTTP request header. The headers (" <code><method></code> <code><uri></code> HTTP/1.1") and "Host: <code><host-name></code> " will be always sent by default. The string <code>custom_header</code> will be appended to the default header.
<i>data</i>	Pointer to the 'httpc_data_t' data handling structure.
<i>certs</i>	Pointer to the 'ssl_certs_t' for certificates structure.

Returns:

Error code (`httpc_ret_e`). See `api_httpc.h` for more information.

1.9.3 `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)`

Send a PUT request to the specified URL.

Parameters:

<i>handle</i>	Connection handle
<i>url</i>	Destination URL
<i>custom_header</i>	Custom HTTP request header. The headers (" <code><method><uri> HTTP/1.1</code> ") and " <code>Host: <host-name></code> " will be always sent by default. The string <code>custom_header</code> will be appended to the default header.
<i>data</i>	Pointer to the ' <code>httpc_data_t</code> ' data handling structure.
<i>certs</i>	Pointer to the ' <code>ssl_certs_t</code> ' for certificates structure.

Returns:

Error code (`httpc_ret_e`). See `api_httpc.h` for more information.

1.9.4 `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)`

Send a DELETE request to the specified URL.

Parameters:

<i>handle</i>	Connection handle
<i>url</i>	Destination URL
<i>custom_header</i>	Custom HTTP request header. The headers (" <code><method><uri> HTTP/1.1</code> ") and " <code>Host: <host-name></code> " will be always sent by default. The string <code>custom_header</code> will be appended to the default header.
<i>data</i>	Pointer to the ' <code>httpc_data_t</code> ' data handling structure.
<i>certs</i>	Pointer to the ' <code>ssl_certs_t</code> ' for certificates structure.

Returns:

Error code (`httpc_ret_e`). See `api_httpc.h` for more information.

1.9.5 `httpc_ret_e nrc_httpc_rcv_response(con_handle_t *handle, httpc_data_t *data)`

Retrieves the response data when there're remains after executing the request functions

Parameters:

<i>handle</i>	Connection handle
<i>data</i>	Pointer to the 'httpc_data_t' data handling structure.

Returns:

Error code (`httpc_ret_e`). See `api_httpc.h` for more information.

1.9.6 `void nrc_httpc_close(con_handle_t *handle)`

Close a HTTP connection.

Parameters:

<i>handle</i>	Connection handle
---------------	-------------------

Returns:

N/A

※ Defined structure

```
typedef struct {
    const char *server_cert;    /* Server certification. */
    const char *client_cert;    /* Client certification. */
    const char *client_pk;      /* Client private key. */
    int server_cert_length;     /* Server certification length, server_cert buffer size. */
    int client_cert_length;     /* Client certification length, client_cert buffer size. */
    int client_pk_length;       /* Client private key length, client_pk buffer size. */
} ssl_certs_t;
```

```
typedef struct {
    char *data_out;             /* Pointer of the output buffer for data sending. */
    uint32_t data_out_length;   /* Output buffer length. */
    char *data_in;              /* Pointer of the input buffer for data receiving. */
    uint32_t data_in_length;    /* Input buffer length. */
    int rcvd_size;              /* Actual received data size. */
} httpc_data_t;
```

1.10FOTA

1.10.1 void nrc_fota_write (uint32_t dst, uint8_t *src, uint32_t len)

Write data to the serial flash.

Parameters:

<i>Dst</i>	Destination address in the serial flash
<i>Src</i>	Pointer to the start of the data
<i>Len</i>	Length of the data

Returns:

N/A

1.10.2 void nrc_fota_erase (uint32_t dst, uint32_t len)

Erase the serial flash memory block at the specified address.

Parameters:

<i>dst</i>	Destination address
<i>len</i>	Length of bytes to erase

Returns:

N/A

1.10.3 void nrc_fota_set_info (uint32_t len, uint32_t crc)

Set the firmware metainformation (length and CRC value).

Parameters:

<i>len</i>	Length of the firmware
<i>crc</i>	CRC value of the firmware binary

Returns:

N/A

1.10.4 void nrc_fota_update_done (fota_info_t* fw_info)

Requests the system to update the newly downloaded firmware and reset. The function must be called after the new firmware has been successfully downloaded and stored on the system.

Parameters:

<i>fw_info</i>	Firmware metainformation (length and CRC value)
----------------	---

Returns:

N/A

1.10.5 uint32_t nrc_fota_cal_crc (uint8_t* data, uint32_t len)

Compute and retrieve the CRC value of the data

Parameters:

<i>data</i>	Data
<i>len</i>	Length of the data

Returns:

The CRC value of the data.

※ Defined structure

```
typedef struct {  
    uint32_t fw_length;    /* Length of a firmware binary file */  
    uint32_t crc;          /* CRC of a firmware binary file */  
} fota_info_t;
```


1.11 SerialFlash

1.11.1 bool nrc_sf_erase_user_config(uint8_t user_area)

Erase 4KB user config area

Parameters:

user_area	User config area
-----------	------------------

Returns:

true: success

false: fail

1.11.2 bool nrc_sf_read_user_config(uint8_t user_area, uint8_t *data, size_t size)

Read user config data from the user_config address of the serial flash

Parameters:

user_area	User config area
<i>data</i>	User config data
<i>size</i>	Length of the data

Returns:

true: success

false: fail

1.11.3 bool nrc_sf_write_user_config(uint8_t user_area, uint8_t *data, size_t size)

Write user config data to the user_config address of the serial flash

Parameters:

user_area	User config area
<i>data</i>	User config data
<i>size</i>	Length of the data

Returns:

true: success

false: fail

1.12 Power save

1.12.1 `bool nrc_ps_set_sleep(uint8_t sleep_mode, uint16_t interval)`

Set the POWER SAVE MODE and POWER SAVE PROTOCOL.

The function supports two sleep modes.

※ Currently, Modem sleep and Non-Tim are not supported. These modes will be implemented.

Parameters:

<i>sleep_mode</i>	Mode (0 :Modem Sleep 1 : Deep Sleep)
<i>interval</i>	0: Tim / non-zero : Non-Tim (ms)

Returns:

true: enabled

false: disabled

1.12.2 `bool nrc_ps_set_gpio_wakeup_pin(int pin_number)`

Configure a WAKEUP-GPIO-PIN when system state is uCode or deepsleep.

Call this function before deepsleep if user want to config the WAKEUP-GPIO-PIN.

Parameters:

<i>pin_number</i>	Select wakeup GPIO Pin number (0 ~ 31)
-------------------	--

Returns:

true: enabled

false: disabled

1.12.3 `bool nrc_ps_set_wakeup_source(uint8_t wakeup_source)`

Configure WAKEUP SOURCES when system state is deepsleep.

Call this function before deepsleep if user want to config the wakeup sources.

Parameters:

<i>wakeup_source</i>	WAKEUP_SOURCE_RTC WAKEUP_SOURCE_GPIO
----------------------	--

Returns:

true: enabled

false: disabled

2 Middleware API Reference

2.1 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>

2.2 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>

2.3 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>

2.4 cJSON

cJSON is an ultralightweight JSON parser in ANSI C.

- GitHub Page:
 - <https://github.com/DaveGamble/cJSON>

2.5 MQTT

MQTT (MQ Telemetry Transport) is an open OASIS and ISO standard (ISO/IEC PRF 20922) lightweight, publish-subscribe network protocol that transports messages between devices. Any network protocol that provides ordered, lossless, bi-directional connections can support MQTT.

- Official Website:
 - <https://mqtt.org/>
- Online Documentation:
 - <http://mqtt.org/documentation>
- GitHub Page:
 - <https://github.com/eclipse/paho.mqtt.embedded-c>

2.6 CoAP

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol (RFC 7252) for use with constrained nodes and constrained networks in the Internet of Things. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

- Official website:
 - <https://coap.technology/>
- Specification (RFC 7252):
 - <https://tools.ietf.org/html/rfc7252>
- C-Implementation of CoAP:
 - <https://libcoap.net/>

2.7 Mini-XML

Mini-XML is a tiny XML library that can be used to read and write XML and XML-like data files without requiring large non-standard libraries.

- Official website:
 - <https://www.msweet.org/mxml/>
- Online Documentation:
 - <https://www.msweet.org/mxml/mxml.html>
- GitHub Page:
 - <https://github.com/michaelsweet/mxml>

3 Revision history

Revision No	Date	Comments
Ver 1.0	11/01/2018	Initial version for customer release created
Ver 1.1	03/25/2019	APIs for Wi-Fi, Timer, ADC, and SPI updated
Ver 1.2	04/05/2019	APIs for Wi-Fi, Connection, Timer, I2C, ADC, SPI, PWM updated
Ver 1.3	07/02/2019	APIs for Wi-Fi, Connection updated
Ver 1.4	11/07/2019	Add HTTP Client API and FOTA API
Ver 1.5	07/05/2020	Add Serial Flash API
Ver 1.6	07/23/2020	Add Power save