neoway

Get connected Get smart

N58 OpenCPU

API Notes

Issue 3.0 Date 2021-02-05

Notice

This document provides guide for users to use N58.

This document is intended for system engineers (SEs), development engineers, and test engineers.

THIS GUIDE PROVIDES INSTRUCTIONS FOR CUSTOMERS TO DESIGN THEIR APPLICATIONS. PLEASE FOLLOW THE RULES AND PARAMETERS IN THIS GUIDE TO DESIGN AND COMMISSION. NEOWAY WILL NOT TAKE ANY RESPONSIBILITY OF BODILY HURT OR ASSET LOSS CAUSED BY IMPROPER OPERATIONS.

THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE DUE TO PRODUCT VERSION UPDATE OR OTHER REASONS.

EVERY EFFORT HAS BEEN MADE IN PREPARATION OF THIS DOCUMENT TO ENSURE ACCURACY OF THE CONTENTS, BUT ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

Neoway provides customers complete technical support. If you have any question, please contact your account manager or email to the following email addresses:

Sales@neoway.com

Support@neoway.com

Website: http://www.neoway.com

Contents

About This Document

Scope

This document is applicable to the N58 series.

Audience

This document is intended for system engineers (SEs), development engineers, and test engineers.

Change History

Issue	Date	Change	Changed By
1.0	2020-03	Initial draft	Wu Guoqing
1.1	2020-08	Optimized API descriptions. Added some APIs.	Wen Jingshun
1.2	2020-08	Added the file operation APIs	Wen Jingshun
3.0	2021-01	• Added the BLE APIs. • Added the MQTT APIs. • Added the APIs used to obtain the socket status, forcibly delete the file folder, set the time synchronization switch, obtain the device boot time, and obtain the hardware version. • Added the websocket APIs • Added the SIM and Network related APIs • Modified the size of storage space	Yang Mengsha Shui Ying Hu Jun Yang Xiaojun Wang Chen Huang Wei Gang Li JinTao

Conventions

Symbol	Indication
	This warning symbol means danger. You are in a situation that could cause fatal device damage or even bodily damage.
	Means reader be careful. In this situation, you might perform an action that could result in module or product damages.
	Means note or tips for readers to use the module

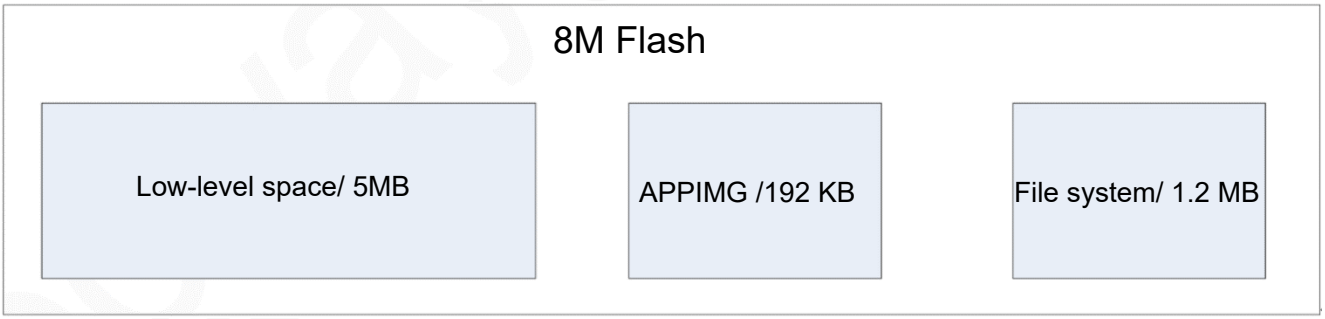
1 About N58 OpenCPU

The N58 OpenCPU module runs FreeRTOS. It mainly provides the following hardware resources.

- ARM Cortex-A5 processor, 500 MHz
- Supports the installation and running of applications that are developed based on libc by C language
- Memory

  RAM: 128 KB (can be extended to 1 MB)
  ROM: 192 KB (extend it as required)
  File system: 1.2 MB
- Supports multiple network modes: GSM/GPRS &&LTE Cat1
- Supports various services including GPS, BT, Wi-Fi Scan, etc.
- Supports USB2.0, ADC, UART, SPI, I2C, and GPIO
- Supports key and LCD backlight

1.1 Flash Space



- Programming language

  Neoway provides API functions of RTOS in C language and header files that define relevant data structures.
- Compilation environment

  Compile an application with the cross compiler provided by Neoway in Windows OS and generate an executable program image.
- Operation method

  Use the RESEARCHDOWNLOAD tool to flash the executable image into N58, and the program automatically starts with the startup of the system.

1.2 Basic Functions

- PPP dialing
- Network related functions
- Voice
- SMS message
- Location
- Wi-Fi scanning
- SIM cards
- Audio
- FOTA upgrade
- Sleep mode
- I/O interfaces including GPIO, ADC, SPI, UART, RTC, I2C, etc.
- Switch of LCD backlight
- Matrix buttons
- Virtual AT functions

1.3 Development Process

Process description

1. Read the APPIMG from the flash into the RAM
2. Run the APPIMG

   Enter **appimg_enter**, the entry function of appimg, to run tasks.

2 APIs

2.1 Power Management

PM APIs can be found in **nwy_pm.h**. They are used to control the module to enter sleep mode or exist from sleep mode, detect the power status of the module, and control the module to shut down quickly or shut down in normal mode.

nwy_pm_state_set

Function	Int nwy_pm_state_set(int mode)
Description	To control the module to enter sleep mode or exist from sleep mode
Parameter	Mode: 0: Wake-up mode 1: Sleep mode
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_power_state

Function	int nwy_power_state(void)
Description	To detect the power status of the module
Parameter	N/A
Return value	Successful: Power status of the module 1: Normal 2: Abnormal Failed: NWY_ERROR

nwy_power_off

Function	int nwy_power_off(int option)
Description	To control the module to shut down quickly or shut down in normal mode.

Parameter	option: Modes of shutting down the module 0: quick shutdown 1: shutdown in normal mode
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_set_pmu_power_level

Format	int nwy_set_pmu_power_level(uint32_t id, uint32_t mv)
Description	To select the voltage value of the submodule
Parameter	id: submodule ID mv: range of the voltage (from 1800 to 3000, unit: mV)
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_subpower_switch

Function	bool nwy_subpower_switch(unsigned int id, bool enabled, bool lp_enabled);
Description	To select the switch of the subpower.
Parameter	id: sub power id enabled: 0-close,1-open at normal run lp_enabled: 0-close,1-open at low power mode
Return value	Failed: 0 Successful: 1

nwy_powerkey_poweroff_ctrl

Function	int nwy_powerkey_poweroff_ctrl(bool enable);
Description	To enable/disable the shutdown control of the **power_n** button.
Parameter	enable: Enable/Disable the shutdown control of power_n button. 0: Disable 1: Enable
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_get_chip_id

Format	int nwy_get_chip_id(uint8_t *uid)
Description	To obtain the module ID.
Parameter	uid: module ID
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_set_auto_poweroff

Function	bool nwy_set_auto_poweroff(uint16_t shut_vol, uint16_t dead_vol, int count, ChargerNoticeCB_t cb)
Description	Automatic shutdown.
Parameter	shut_vol: Shutdown voltage, unit:mV dead_vol: Threshold voltage of module shutdown count: Shutdown voltage alarm times (when the times exceed **count** under the shutdown voltage, the module shuts down automatically). cb: callback function
Return value	Failed: 0 Successful: 1

nwy_set_back_light_level

Format	bool nwy_set_back_light_level(uint32_t back_light, uint32_t level)
Description	To set the backlight brightness.
Parameter	back_light: backlight index (NWY_POWER_RGB_IB0 - NWY_POWER_RGB_IB2) level: Backlight brightness (0 - 63 that is respectively corresponding to 1.68 mA - 54.6mA@3.6V)
Return value	0: failed 1: successful

nwy_bootup_alarm_set

Format	int nwy_bootup_alarm_set(uint32_t sec_in_day)
Description	To set the bootup alarm clock (daily repeat)
Parameter	sec_in_day: alarm clock time, the number of seconds from 0:0:0
Return value	Successful: NWY_SUCCESS

	Failed: NWY_ERROR

nwy_bootup_alarm_del

Format	int nwy_bootup_alarm_ del (void)
Description	To delete the bootup alarm clock (daily repeat)
Parameter	NA
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_shutdown_with_alarm

Format	int nwy_shutdown_with_alarm(void)
Description	Shutdown with the bootup alarm clock, The RTC device of the alarm clock keeps running in the shutdown mode. Ensure that you have set the bootup alarm clock before invoking this API. Otherwise the invoking is invalid.
Parameter	**NA**
Return value	Successful:-No value is returned. Failed: NWY_ERROR

2.2 I/O Interfaces

2.2.1 UART

UART APIs can be found in **nwy_at_uart.h**. They are used to configure the UART settings and transmit data through UART.

nwy_uart_init

Function	int nwy_uart_init(uint32_t name,nwy_uart_mode_t mode);
Description	To initialize UART.
Parameter	name: UART channel
Return value	Successful: hd Failed: -1

nwy_uart_set_baud

Function	bool nwy_uart_set_baud(uint8 t hd,uint32_t baud);
Description	To set the baud rate of UART
Parameter	hd: Return value of UART init. Baud: baud rate
Return value	N/A

nwy_uart_get_baud

Function	bool nwy_uart_get_baud(uint8 t hd,uint32_t *baud);
Description	To query the baud rate of UART
Parameter	hd: Return value of UART init. Baud: baud rate
Return value	Failed: 0 Successful: 1

nwy_uart_set_para

Function	bool nwy_uart_set_para(uint8 t hd,nwy_uart_parity_t parity, nwy_uart_data_bits_t data_size, nwy_uart_stop_bits_t stop_size, bool flowctrl);
Description	To configure UART settings
Parameter	hd: Return value of UART init. parity: parity data_size: data bits stop_size: stop bits flowctrl: flow control
Return value	N/A

nwy_uart_get_para

Function	bool nwy_uart_get_para(uint8 t hd,nwy_uart_parity_t*parity, nwy_uart_data_bits_t *data_size, nwy_uart_stop_bits_t *stop_size, bool *flowctrl);
Description	To query the current settings of UART
Parameter	hd: Return value of UART init. Parity: parity data_size: data bits stop_size: stop bits

	flowctrl: flow control
Return value	Failed: 0 Successful: 1

nwy_uart_send_data

Function	int nwy_uart_send_data(uint8_t hd, uint8 t *data_ptr, uint32_t length);
Description	To send data through UART
Parameter	hd: Return value of UART init. data_ptr: data to be sent length: data length
Return value	Successful: number of bytes sent. Failed: 0

nwy_uart_reg_recv_cb

Function	bool nwy_uart_reg_recv_cb(nwy_uart_recv_callback_t recv_cb);
Description	To receive data through UART
Parameter	recv_cb: callback function of receiving data
Return value	Failed: 0 Successful: 1

nwy_uart_deinit

Function	bool nwy_uart_deinit(uint8_t hd);
Description	To turn off UART.
Parameter	hd: Return value of uart init
Return value	Successful: 1 Failed: 0

2.2.2 I2C

I2C APIs can be found in **nwy_i2c.h**.

nwy_i2c_init

Function	int nwy_i2c_init(const char* i2cDev, unsigned char slaveAddr)
Description	To open an I2C device of the module and set the address of the slave device
Parameter	i2cDev: I2C device name slaveAddr: address of the slave device
Return value	Successful: handles of I2C devices Failed: NWY_ERROR

nwy_i2c_read

Function	int nwy_i2c_read(intfd, unsigned char slaveAddr, unsigned char ofstAddr,unsigned char* ptrBuff, unsigned short length)
Description	To read the data in a specified length from a specified I2C slave device
Parameter	fd: handle of the I2C device slaveAddr: address of the slave device ofstAddr: register address ptrBuff: pointer that points to the buffer that is used to hold data length: Length of data to be read
Return value	Successful: **0** (no data) or **>0** (read data) Failed: NWY_ERROR

nwy_i2c_write

Function	int nwy_i2c_write(intfd, unsigned char slaveAddr, unsigned char ofstAddr,unsigned char* ptrData, unsigned short length)
Description	To write the data in a specified length to a specified I2C slave device
Parameter	fd: handle of the I2C device slaveAddr: address of the slave device ofstAddr: register address ptrData: pointer that points to the buffer to be written to length: length of the data to be written
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_i2c_raw_get_byte

Function	int nwy_i2c_raw_get_byte(int fd, uint8_t *data, int start_flag, int stop_flag)
Description	To read the data of I2C device by byte.
Parameter	fd: handle of the I2C device *data: data read start_flag: start flag (only the first byte is set to 1) stop_flag: stop flag (only the last byte is set to 1)
Return value	Successful: NWY_SUCESS Failed: <0

nwy_i2c_raw_put_byte

Function	int nwy_i2c_raw_put_byte(int fd, uint8_t data, int start_flag, int stop_flag)
Description	To write the data of I2C device by byte.
Parameter	fd: handle of the I2C device data: data written start_flag: start flag (only the first byte is set to 1) stop_flag: stop flag (only the last byte is set to 1)
Return value	Successful: NWY_SUCESS Failed: a value <0

nwy_i2c_deinit

Function	int nwy_i2c_deinit(int fd);
Description	To close the I2C device.
Parameter	fd: handle of the I2C device.
Return value	Successful: NWY_SUCESS Failed: a value <0

2.2.3 SPI

SPI APIs can be found in **nwy_spi.h**. They are used to initialize the module and transmit data through the SPI bus.

nwy_spi_init

Function	Int nwy_spi_init(char *spibus, uint8_t mode, uint32_t speed, uint8_t bits)
Description	To enable and configure the SPI bus
Parameter	spibus: name of the SPI bus, "SPI1" and "SPI2" are supported. mode: bus mode, ranging from mode0 to mode3 speed: bus speed rate bits: bits
Return value	Successful: handle of the SPI bus Failed: SPI_EC_ERROR

nwy_spi_transfer

Function	int nwy_spi_transfer(inthd, uint8_t cs, uint8_t *tx, uint8_t *rx, uint32_t size)
Description	To transmit data through the SPI bus
Parameter	hd: handle of the SPI bus cs: chip selections, cs0 and cs1 are supported. tx: transmit data rx: receive data size: data size
Return value	Successful: SPI_EC_SUCESS Failed: SPI_EC_ERROR

nwy_spi_deinit

Function	int nwy_spi_deinit(inthd)
Description	To disable the SPI bus
Parameter	hd: handle of the SPI bus
Return value	Successful: SPI_EC_SUCESS Failed: SPI_EC_ERROR

2.2.4 GPIO

GPIO APIs can be found in **nwy_gpio_open.h**. They are used to set the GPIO to input, output or interrupt mode. GPIO 0,2, and 3 are the default GPIO port. You can also configure the GPIO port as required. For the custom GPIO configuration, contact Neoway FAE.

nwy_gpio_get_direction

Function	int nwy_gpio_get_direction(uint32 gpio_id)
Description	To obtain the GPIO direction
Parameter	gpio_id: GPIO ID
Return value	gpio direction

nwy_gpio_get_value

Function	int nwy_gpio_get_value(uint32 gpio_id)
Description	To obtain the level state of the GPIO pin.
Parameter	gpio_id: GPIO ID
Return value	GPIO pin

nwy_gpio_set_direction

Function	int nwy_gpio_set_direction(uint32 gpio_id, wy_dir_mode_t dirct)
Description	To configure the GPIO direction
Parameter	gpio_id: GPIO ID dirct: GPIO direction
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_gpio_set_value

Function	int nwy_gpio_set_value(uint32 gpio_id, nwy_value_t value)
Description	To configure the level state of the GPIO pin.
Parameter	gpio_id: GPIO ID value: nwy_value_t
Return value	Successful: NWY_SUCCESS

	Failed: NWY_ERROR

nwy_open_gpio_irq_config

Function	int nwy_open_gpio_irq_config(uint32_t gpio_id, uint8_t irq_mode, nwy_irq_callbackcb);
Description	To set the interrupt mode of a GPIO
Parameter	gpio_id: GPIO ID irq_mode: GPIO interrupt mode cb: nwy_irq_callback
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_gpio_pullup_or_pulldown

Function	int nwy_gpio_pullup_or_pulldown(uint32_t gpio_id, int pull);
Description	To set pull-up/pull-down of GPIO.
Parameter	gpio_id:GPIO ID pull: 0 - pull down 1 - pull up
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_gpio_open_irq_enable

Function	int nwy_gpio_open_irq_enable(uint32_t gpio_id);
Description	To enable GPIO interrupt.
Parameter	gpio_id: GPIO ID
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_gpio_open_irq_disable

Function	int nwy_gpio_open_irq_disable(uint32_t gpio_id);
Description	To disable GPIO interrupt.
Parameter	gpio_id: GPIO ID
Return value	Successful: NWY_SUCCESS

	Failed: NWY_ERROR

nwy_close_gpio

Function	int nwy_close_gpio(uint32_t gpio_id);
Description	To disable GPIO.
Parameter	gpio_id: GPIO ID
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

2.2.5 ADC

ADC API can be found in nwy_adc.h. It is used to collect voltage values through the ADC port.

nwy_adc_get

Function	int nwy_adc_get(nwy_adc_t channel,nwy_adc_aux_scale_t scale);
Description	To get ADC value.
Parameter	channel: ADC channel scale: voltage range
Return value	Voltage value obtained. (unit:mV)

2.3 Peripherals

2.3.1 Audio

Audio APIs can be found in **nwy_audio_api.h**.

nwy_audio_pa_control

Function	int nwy_audio_pa_control(nwy_pa_type pa_type, int gpio_num, nwy_pa_mode mode)
Description	To select the type of the external audio amplifier.
Parameter	pa_type: audio amplifier type gpio_num: the GPIO that is connected to the amplifier switch.

	mode: gain selection (valid for the specific power amplifier)
Return value	Successful: NWY_SUCESS Failed: others

nwy_audio_player_open

Function	int nwy_audio_player_open(player_event_handler handler)
Description	To initialize the audio player
Parameter	handler: callback function that indicates the audio has finished playing.
Return value	Successful: NWY_SUCESS Failed: other values

nwy_audio_player_stop

Function	int nwy_audio_player_stop(void)
Description	To stop playing audio
Parameter	N/A.
Return value	Successful: NWY_SUCESS Failed: other values

nwy_audio_player_close

Function	void nwy_audio_player_close(void)
Description	To close the audio player.
Parameter	N/A.
Return value	void

nwy_audio_player_play

Function	int nwy_audio_player_play (uint8_t *pdata, uint32_t len)
Description	To play buffer data
Parameter	pdata: PCM data len: data length
Return value	Successful: NWY_SUCESS Failed: other values

nwy_audio_set_handset_vol

Function	void nwy_audio_set_handset_vol(constint step)
Description	To set the volume level for the speaker
Parameter	Step: volume level, ranging from 0 to 100
Return value	void

nwy_audio_get_handset_vol

Function	unsigned int nwy_audio_get_handset_vol(void)
Description	To get the speaker volume level.
Parameter	Void
Return value	0-100

nwy_audio_set_mic_vol

Function	void nwy_audio_set_mic_vol(const int step)
Description	To set the mic volume.
Parameter	step: volume level, ranging from 0 to 100.
Return value	void

nwy_audio_get_mic_vol

Function	unsigned int nwy_audio_get_mic_vol(void)
Description	To query the mic volume level.
Parameter	void
Return value	0-100

nwy_audio_recorder_open

Function	int nwy_audio_recorder_open(record_event_handler handler)
Description	To initialize the recording.
Parameter	record_event_handler handler: callback function of the recording.
Return value	Successful: NWY_SUCESS Failed: others

nwy_audio_recorder_start

Function	int nwy_audio_recorder_start(void)
Description	To start recording.
Parameter	N/A.
Return value	Successful: NWY_SUCESS Failed: others

nwy_audio_recorder_stop

Function	int nwy_audio_recorder_stop(void)
Description	To stop recording.
Parameter	N/A.
Return value	Successful: NWY_SUCESS Failed: others

nwy_audio_recorder_close

Function	void nwy_audio_recorder_close(void)
Description	To close recording.
Parameter	void
Return value	void

nwy_tts_playbuf

Function	int nwy_tts_playbuf(char * data, int size, nwy_tts_encode_t type, tts_cb cb, void* cb_param)
Description	To play TTS.
Parameter	data: TTS data pointer. size: TTS data length. type: TTS encoding type. cb: callback of the play complete status. cb_param: callback interface parameter.
Return value	Successful: NWY_SUCESS Failed: others

nwy_tts_stop_play

Function	void nwy_tts_stop_play(void)
Description	To stop TTS playing.
Parameter	void
Return value	void

nwy_audio_tone_play

Function	void nwy_audio_tone_play(const char * tone, unsigned time, int vol)
Description	To play tone audio.
Parameter	Tone:dtmf Time: duration time (unit:ms) vol: volume, ranging from 0 to 15.
Return value	void

2.3.2 LED

LED APIs can be found in **nwy_led.h**.

nwy_led_back_light_open

Function	int nwy_led_back_light_open(int mode);
Description	To enable LED backlight
Parameter	Mode: LED channel to turn on backlight.
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_led_back_light_close

Function	int nwy_led_back_light_close(int mode);
Description	To turn off LED backlight
Parameter	mode: LED channel to turn off backlight.
Return value	Successful: NWY_SUCCESS Failed: NWY_ERROR

2.3.3 Keypad

Keypad API can be found in **nwy_keypad.h**.

reg_nwy_key_cb

Function	void reg_nwy_key_cb(nwy_key_event_cb_tcb);
Description	To register a callback function of the keypad event
Parameter	cb:typedef void (*nwy_key_event_cb_t)(nwy_key_t key, nwy_keyState_tevt);
Return value	N/A.

nwy_keypad_debounce_time

Format	void nwy_keypad_debounce_time(uint32_t debunc,uint32_t itv)
Description	To set the keypad debounce time.
Parameter	debunc: debounce time (ms) Itv: debounce time (ms)
Return value	N/A.

2.3.4 LCD

LCD APIs can be found in **nwy_lcd_bus.h**.

nwy_lcd_bus_init

Function	void nwy_lcd_bus_init(nwy_lcd_bus_config_t *lcd_bus_config);
Description	To initialize the LCD driver.
Parameter	nwy_lcd_bus_config_t *lcd_bus_config: configuration information of GOUDA LCD interface.
Return value	N/A.

nwy_lcd_bus_write_cmd

Function	bool nwy_lcd_bus_write_cmd(unsignedchar cmd);
Description	Write a command to the LCD

Parameter	cmd:addr Command to write
Return value	Successful: true Failed: false

nwy_lcd_bus_write_data

Function	bool nwy_lcd_bus_write_data(unsigned char data);
Description	To write a data to the LCD.
Parameter	data: which data you want to write
Return value	Successful: true Failed: false

nwy_lcd_bus_write_datas

Function	unsigned int nwy_lcd_bus_write_datas(void *data, unsigned int size);
Description	To write data to the LCD.
Parameter	*data: datas you want to write size:data size
Return value	Data size

nwy_lcd_bus_deinit

Function	void nwy_lcd_bus_deinit(void);
Description	To disable the LCD driver.
Parameter	void
Return value	void

2.3.5 SD

SD APIs can be found in **nwy_file.h**.

nwy_sdk_sdcard_mount

Function	bool nwy_sdk_sdcard_mount(void)
Description	To mount the SD card.

Parameter	void
Return value	Successful: true Failed: false

nwy_sdk_sdcard_unmount

Function	void nwy_sdk_sdcard_unmount(void)
Description	To unmount the SD card.
Parameter	void
Return value	void

nwy_read_sdcart_status

Function	int nwy_read_sdcart_status(void)
Description	To read the SD card status.
Parameter	void
Return value	1: mount status 0: unmount status

2.3.6 Spi flash

The definitions of these API functions can be found in **nwy_spi.h**.

nwy_vfs_block_device_create

Format	nwy_block_device_t *nwy_vfs_block_device_create(nwy_spi_flash_t *dev)
Description	To create the logical block device.
Parameter	dev: SPI flash information. Currently only the block size and the number of blocks are used, other parameters are omitted.
Return value	The logical block device.

nwy_vfs_mount

Format	int nwy_vfs_mount(const char *base_path, nwy_block_device_t *nwy_bdev)
Description	To mount the block device to the file system.

Parameter	base_path: mount point of this block device in the file system. nwy_bdev: logical block device
Return value	-1: failed 0: successful

nwy_read_sdcart_status

Format	int nwy_vfs_mkfs(nwy_block_device_t *nwy_bdev)
Description	To format the block device.
Parameter	nwy_bdev: logical block device
Return value	-1: failed 0: successful

2.4 Services

2.4.1 Data

Data APIs can be found in **nwy_data.h**. They are used to establish or close a dial-up connection, configure the dial-up settings and obtain the dial-up status and information related to dialing.

nwy_data_get_srv_handle

Function	int nwy_data_get_srv_handle(nwy_data_cb_funccb_func)
Description	To obtain the resource handle of a dial-up connection and configure the callback function The handle is used to operate the connection. The callback function is used to report the status change of the connection.
Parameter	cb_func: registered callback function, used to report status change of the connection
Return value	Successful: obtained resource handle, a positive integer number Failed: error codes

nwy_data_release_srv_handle

Function	void nwy_data_release_srv_handle(inthndl)

Description	To release the resource handle of a specified dial-up connection and to unregister the callback function. The dial-up connection will be closed if it is valid before unregistering the callback function.
Parameter	hndl: resource handle of a dial-up connection. It is obtained by calling nwy_data_get_srv_handle().
Return value	N/A.

nwy_data_start_call

Function	int nwy_data_start_call(inthndl, nwy_data_start_call_v02_t *param)
Description	To establish a specified dial-up connection This API function is asynchronous. The dialup result is obtained by calling the callback function to which **nwy_data_get_srv_handle()** registers.
Parameter	hndl: a specified resource handle, obtained by calling nwy_data_get_srv_handle(). para: a dialup parameter
Return value	Successful: 0 Failed: a value <0

nwy_data_stop_call

Function	int nwy_data_stop_call(inthndl)
Description	To close a specified dial-up connection This API function is asynchronous. The result is obtained by calling the callback function to which nwy_data_get_srv_handle() registers.
Parameter	hnd: specified resource handle, obtained by calling nwy_data_get_srv_handle().
Return value	Successful: 0 Failed: a value <0

nwy_data_get_ip_addr

Function	int nwy_data_get_ip_addr(inthndl, nwy_data_addr_t_info * info_ptr, int *len)
Description	To obtain the IP address of a dial-up connection This function is called only after a dialup connection is established successfully.
Parameter	hndl: a specified resource handle, obtained by calling nwy_data_get_srv_handle(). info_ptr: structure or structure array that is used to return the address information. Ensure that the structure array is large enough when the dialup connection obtains multiple addresses. For details, see the structure definitions in

	nwy_data_addr_t_info. len: quantity of obtained addresses
Return value	Successful: 0 Failed: a value <0

nwy_data_set_profile

Function	int nwy_data_set_profile(intprofile_idx, nwy_data_profile_type_tprofile_type, nwy_data_profile_info_t *profile_info);
Description	To specify a profile ID and type and to modify the parameters of the profile.
Parameter	profile_idx: ID of the profile to be modified. profile_type: type of profile to be modified. profile_info: modified parameters of the profile, including PDP type, APN, authentication method, user name, and password. For details, see nwy_data_profile_info_t.
Return value	Successful: 0 Failed: a value <0

nwy_data_get_profile

Function	int nwy_data_get_profile(intprofile_idx, nwy_data_profile_type_tprofile_type, nwy_data_profile_info_t *profile_info);
Description	To obtain the current parameters of the specified profile with a specified ID and type.
Parameter	profile_idx: profile ID to be obtained profile_type: profile type to be obtained profile_info: returned profile information
Return value	Successful: 0 Failed: a value <0

nwy_ip4addr_ntoa

Function	char* nwy_ip4addr_ntoa(const nwy_ip4_addr_t *addr)
Description	To convert the format of the IPv4 address.
Parameter	addr: IPv4 address
Return value	IPv4 address

nwy_ip6addr_ntoa

Function	char *nwy_ip6addr_ntoa(const nwy_ip6_addr_t *addr)
Description	To convert the format of the IPv6 address.
Parameter	addr: IPv6 address
Return value	IPv6 address

2.4.2 Voice

Voice APIs can be found in **nwy_voice.h**.

nwy_voice_call_start

Function	int nwy_voice_call_start(uint8_t sim_id,char *phone_num)
Description	To make a call
Parameter	sim_id: only SIM 1 is supported currently. phone_num: destination phone number
Return value	Successful: NWY_RES_OK Failed: nwy_error_t

nwy_voice_call_end

Function	int nwy_voice_call_end(uint8_t sim_id)
Description	To hang up a call
Parameter	sim_id: only SIM 1 is supported currently.
Return value	Successful: NWY_RES_OK Failed: nwy_error_t

nwy_voice_call_autoanswver

Function	int nwy_voice_call_autoanswver()
Description	To answer a call
Parameter	None
Return value	Successful: NWY_RES_OK Failed: nwy_error_t

nwy_voice_call_hold

Function	int nwy_voice_call_hold(uint8_t sim_id)
Description	To hold an incoming call
Parameter	sim_id: only SIM 1 is supported currently.
Return value	Successful: NWY_RES_OK Failed: nwy_error_t

nwy_voice_call_unhold

Function	int nwy_voice_call_unhold(uint8_t sim_id)
Description	To answer a call from a hold-on state
Parameter	sim_id: only SIM 1 is supported currently.
Return value	Successful: NWY_RES_OK Failed: other values

nwy_voice_setvolte

Function	int nwy_voice_setvolte(uint8_t sim_id, uint8_t setvolte)
Description	To configure VoLTE
Parameter	sim_id: only SIM 1 is supported currently. setvolte: used to enable IMS
Return value	Successful: NWY_RES_OK Failed: other values

nwy_get_voice_callerid

Function	void nwy_get_voice_callerid(char* nwy_call_rsp)
Description	To get the called phone information.
Parameter	nwy_call_state: called phone information.
Return value	void

nwy_get_voice_state

Function	void nwy_get_voice_state(char* nwy_call_state)
Description	To get the called status.
Parameter	nwy_call_rsp: called status
Return value	void

2.4.3 SIM

SIM APIs can be found in **sim.h**. They are used to obtain SIM card information and configure card status.

nwy_sim_get_card_status

Function	nwy_sim_status nwy_sim_get_card_status()
Description	To obtain the SIM card status
Parameter	N/A.
Return value	The SIM card is ready: **NWY_SIM_STATUS_READY**The SIM card is not inserted: **NWY_SIM_STATUS_NOT_INSERT** The SIM card is locked by PIN: **NWY_SIM_STATUS_PIN1** The SIM card is locked by PUK: **NWY_SIM_STATUS_PUK1** The SIM card is busy: **NWY_SIM_STATUS_BUSY**

nwy_sim_get_iccid

Function	nwy_result_type nwy_sim_get_iccid(nwy_sim_result_type *iccid_buf)
Description	To obtain ICCID of the SIM card
Parameter	iccid_buff: ICCID of the memory card
Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_get_imsi

Function	nwy_result_type nwy_sim_get_imsi(nwy_sim_result_type *imsi_buf)
Description	To obtain IMSI of the SIM card
Parameter	imsi_buff: IMSI of the memory card

Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_enable_pin

Function	nwy_result_type nwy_sim_enable_pin(nwy_sim_result_type *sim_lock)
Description	To enable PIN of the SIM card
Parameter	sim_lock: PIN code of the SIM card.
Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_disable_pin

Function	nwy_result_type nwy_sim_disable_pin(nwy_sim_result_type *sim_unlock)
Description	To disable PIN of the SIM card
Parameter	sim_unlock: PIN code of the SIM card.
Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_get_pin_mode

Function	nwy_result_type nwy_sim_get_pin_mode(nwy_sim_result_type *pin_mode)
Description	To obtain the PIN status of the SIM card
Parameter	pin_mode: PIN status of the SIM card. For details, see the definitions in **sim.h**.
Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_get_imei

Function	nwy_result_type nwy_result_type nwy_sim_get_imei(nwy_sim_result_type *imei)
Description	To obtain the IMEI number of the device.
Parameter	Device's IMEI number
Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_set_simid

Function	nwy_result_type nwy_sim_set_simid(uint8 nSwitchSimID);
Description	To switch SIM cards.
Parameter	nSwitchSimIDs: SIM card ID, ranging from 0 to 1.
Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_get_simid

Function	uint8 nwy_sim_get_simid()
Description	To get the current SIM card ID.
Parameter	void
Return value	SIM card ID

nwy_sim_get_msisdn

Format	int nwy_sim_get_msisdn(nwy_sim_id_t sim_id, char* msisdn_buf, size_t buf_len)
Description	To obtain MSISDN of the SIM card.
Parameter	sim_id: ID of the SIM card slot, for details, see nwy_sim.h. msisdn_buf: MSISDN value buf_len: invalid currently
Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_csim

Format	int nwy_sim_csim(nwy_sim_id_t sim_id, char *indata, int indata_len, char *outdata, int outdata_len)
Description	To send the APDU command to the SIM card.
Parameter	nwy_sim_id_t sim_id: SIMID the is operated. char *indata: APDU command that is inputted. int indata_len: length of the APDU command that is inputted. char *outdata: buffer that holds the result of the command. int outdata_len: length of the buffer that holds the result of the command. Confirm the SIMID of the current operation before executing the command. If it does not match the actual situation, you need to call the nwy_sim_set_simid interface to

	switch. After the switch is successful, the module needs to restart.
Return value	-1: fail to execute the command. A positive number: indicates the command length that is outputted.

nwy_sim_verify_pin

Format	int nwy_sim_verify_pin(nwy_sim_id_t sim_id, const char* pin)
Description	To verify the PIN code of the SIM card.
Parameter	sim_id: ID of the SIM card slot, for details, see nwy_sim.h. pin: PIN of the SIM card.
Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_unblock

Format	int nwy_sim_unblock(nwy_sim_id_t sim_id, const char* puk, const char* new_pin)
Description	To input the PUK code. After the SIM card is locked by a PIN, you can invoke this API to enter the PUK to unlock the SIM card.
Parameter	sim_id: ID of the SIM card slot, for details, see nwy_sim.h. puk: PUK of the SIM card. pin: PIN of the SIM card.
Return value	Successful: NWY_RES_OK Failed: NWY_RES_ERROR

nwy_sim_get_lacid

Format	int nwy_sim_get_lacid(int *lac, int *cid)
Description	To obtain the LAC and CID value.
Parameter	Lac: location area code, cell code cid: cell ID.
Return value	Successful: NWY_SUCESS Failed: NWY_GEN_E_UNKNOWN

2.4.4 USSD

The definitions of these API function can be found in nwy_ussd.h, used to realize the function of sending USSD, and so on.

nwy_ussd_send_msg

Format	int nwy_ussd_send_msg(uint8 simid, char* ussd_string, uint8 dcs, nwy_ussd_info_t* ussd_info)
Description	To sen USSD and obtain the Return value.
Parameter	sim_id: ID of the SIM card slot. ussd_string: Character string of the USSD command. dcs: coding method, 0 by default. ussd_info: Return value of USSD.
Return value	Successful: NWY_USSD_SUCCESS Failed: NWY_USSD_ERROR

2.4.5 SMS

SMS APIs can be found in **nwy_sms.h**. They are used to receive and send SMS messages as well as set their storage location.

nwy_init_sms_option

Function	nwy_result_t nwy_init_sms_option()
Description	To initialize the parameters of the SMS service
Parameter	N/A.
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

nwy_sms_set_storage

Function	nwy_result_t nwy_sms_set_storage(nwy_sms_storage_type_e sms_storage)
Description	To set the storage location of SMS messages
Parameter	nwy_sms_storage_type_e sms_storage: for details, see **nwy_sms.h**
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

nwy_sms_get_storage

Function	nwy_sms_storage_type_e nwy_sms_storage_type_e nwy_sms_get_storage()
Description	To obtain the storage location of SMS messages
Parameter	N/A.
Return value	nwy_sms_storage_type_e type of the storage location of the SMS messages, for details, see **nwy_sms.h**.

nwy_set_report_option

Function	nwy_result_t nwy_result_t nwy_set_report_option(uint8_t mode, uint8_t mt, uint8_t bm, uint8_t ds, uint8_t bfr)
Description	To set the report mode of SMS messages
Parameter	uint8_t modeuint8_t mtuint8_t bm uint8_t ds uint8_t bfr For details, see the definition of AT+CNMI in the AT Commands Manual.
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

nwy_sms_set_sca

Function	nwy_result_t nwy_sms_set_sca(char *sca, unsigned tosca)
Description	To set the number for a SMS center
Parameter	char *sca unsigned tosca For details, see the definition of AT+CSCA in AT Commands Manual
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

nwy_sms_get_sca

Function	nwy_result_t nwy_sms_get_sca(nwy_sms_result_type *sca)
Description	To obtain the SMSC number
Parameter	nwy_sms_result_type *sca For details, see **nwy_sms.h**.
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

nwy_sms_send_message

Function	nwy_result_t nwy_sms_send_message(nwy_sms_info_type_t *p_sms_data)
Description	To send an SMS message
Parameter	nwy_sms_info_type_t *p_sms_data For details, see **nwy_sms.h**
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

nwy_sms_recv_message

Function	void nwy_sms_recv_message(nwy_sms_recv_info_type_t *sms_data)
Description	To receive an SMS message
Parameter	nwy_sms_recv_info_type_t *sms_data For details, see **nwy_sms.h**
Return value	N/A

nwy_sms_delete_message

Function	nwy_result_t nwy_sms_delete_message(uint16_t nindex, nwy_sms_storage_type_e nStorage)
Description	To delete an SMS message
Parameter	uint16_t nindex nwy_sms_storage_type_e nStorage For details, see **nwy_sms.h**.
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

nwy_sms_read_message

Function	nwy_result_t nwy_sms_read_message(unsigned nindex, nwy_sms_recv_info_type_t *sms_data);
Description	To read an SMS message
Parameter	unsigned nindex nwy_sms_recv_info_type_t *sms_data For details, see **nwy_sms.h**
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

nwy_sms_send_pdu_message

Function	nwy_result_t nwy_sms_send_pdu_message(char *data, int data_len);
Description	To send a PDU SMS message.
Parameter	*data: PDU data data_len: PDU data length, see nwy_sms.h.
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

nwy_sms_delete_message_by_type

Format	nwy_result_t nwy_sms_delete_message_by_type(nwy_sms_msg_dflag_e delflag, nwy_sms_storage_type_e nStorage)
Description	To delete one type of SMS message.
Parameter	nwy_sms_msg_dflag_e delflag nwy_sms_storage_type_e nStorage, for detail, see nwy_sms.h.
Return value	Successful: NWY_SMS_SUCCESS Failed: NWY_SMS_ERROR

2.4.6 Location

Location APIs can be found in **nwy_loc.h**. They are used to enable the positioning function and configure the positioning parameters.

nwy_loc_start_navigation

Function	int nwy_loc_start_navigation()
Description	To enable the positioning function
Parameter	N/A.
Return value	Successful: true Failed: false

nwy_loc_stop_navigation

Function	int nwy_loc_stop_navigation()
Description	To disable the positioning function

Parameter	N/A.
Return value	Successful: true Failed: false

nwy_loc_set_position_mode

Function	int nwy_loc_set_position_mode(nwy_loc_position_mode_tpos_mode)
Description	To set the position mode
Parameter	mode: the position mode, for details, see the definitions in **nwy_loc.h**.
Return value	Successful: true Failed: false

nwy_loc_set_startup_mode

Function	int nwy_loc_set_startup_mode(nwy_loc_startup_mode startup)
Description	To set the startup mode
Parameter	startup: startup mode. For details, see **nwy_loc.h**.
Return value	Successful: true Failed: false

nwy_loc_nmea_format_mode

Function	int nwy_loc_nmea_format_mode(uint16 cmd,int16 sel_parameter)
Description	To set the report frequency and fields of NMEA data
Parameter	cmd: select the report frequency or the report field, corresponding to the value of **sel_parameter**. For details, see the datasheet of the external chip, for example, *CASIC Multi-mode Satellite Navigation Receiver Protocol Specifications* of Zhongke Microelectronics.
Return value	Successful: true Failed: false

nwy_loc_get_nmea_data

Function	void nwy_loc_get_nmea_data(char*data)
Description	To obtain NMEA data actively
Parameter	data: NMEA data to be read

Return value	N/A.

nwy_loc_set_server

Function	int nwy_loc_set_server(char *str_url,intport,char *user,char *password)
Description	To configure the AGNSS server
Parameter	str_url: domain name of the AGNSS server port: port number user: user name password: password
Return value	Successful: true Failed: false

nwy_loc_cipgsmloc_open

Function	bool nwy_loc_cipgsmloc_open (bool value,nwy_loc_cipgsmloc_callback cb)
Description	To open LBS positioning.

Parameter	value: open or close LBS positioning.
	Callback function definition
	typedef struct
	{
	double lat; //latitude
	double lng; //longitude
	double alt; //degree of accuracy
	}nwy_cipgsmloc_info_t;
	
	typedef struct {
	char result; //0 == nwy_cipgsmloc_info_t 1 == errmsg
	union
	{
	nwy_cipgsmloc_info_t data;
	char errmsg[48];
	}info;
	}nwy_log_cipgsmloc_result_t;
	typedef void (*nwy_loc_cipgsmloc_callback)(nwy_log_cipgsmloc_result_t *text);
	nwy_loc_cipgsmloc_callback cb // the callback function that informs the latitude and longitude of upper layer.

Return value	Successful: true Failed: false

nwy_loc_agps_open

Function	bool nwy_loc_agps_open(bool value)
Description	To enable the A-GPS function
Parameter	value: enabling or disabling the A-GPS function
Return value	Successful: true Failed: false

nwy_lbs_get_info

Function	int nwy_lbs_get_info(char *strimei, nwy_lbs_plmn_info *pPlmn, int *strRssi)
Description	To obtain the LBS information.
Parameter	strimei: IMEI number pPlmn: plmn information strRssi: signal value
Return value	Successful: true Failed: false

2.4.7 Wi-Fi Scanning

Wi-Fi scanning APIs can be found in **nwy_wifi.h**. It is used to scan Wi-Fi hotspots

nwy_wifi_scan

Function	Int nwy_wifi_scan(nwy_wifi_scan_list_t *scan_list)
Description	To scan and list Wi-Fi hotspots
Parameter	scan_list: Wi-Fi hotspot list
Return value	Fail to start the device: NWY_WIFI_OPEN_FAILED No Wi-Fi hotspot is found: NWY_WIFI_SCAN_FAILED 0: a Wi-Fi hotspot is scanned successfully

2.4.8 Network

Network APIs can be found in **nwy_network.h**. They are used to query and set networks registration and network modes. These functions can be used to register callback function for network status monitoring.

nwy_nw_get_register_info

Function	int nwy_nw_get_register_info(nwy_nw_regs_info_type_t *p_regs_info)
Description	To obtain the current network registration information
Parameter	p_regs_info: network registration information. It is an output parameter. For details, see **nwy_network.h**.
Return value	Successful: 0 Failed: error codes

nwy_nw_get_network_mode

Function	int nwy_nw_get_network_mode(nwy_nw_mode_type_t *p_mode)
Description	To obtain the current network mode
Parameter	p_mode: network mode. It is an output parameter. For details, see the definitions in **nwy_network.h**.
Return value	Successful: 0 Failed: error codes

nwy_nw_set_network_mode

Function	int nwy_nw_set_network_mode(nwy_nw_mode_type_t mode)
Description	To set the current network mode
Parameter	mode: network mode. For details, see **nwy_network.h**.
Return value	Successful: 0 Failed: error codes

nwy_nw_get_operator_name

Function	int nwy_nw_get_operator_N/Ame (nwy_nw_operator_name_t *opt_name)
Description	To obtain the operator of the network that the module registers to, including Long EONS (Enhanced Operator name String), Short EONS, MCC and MNC. For details, see the definition of nwy_nw_operator_name_t.
Parameter	opt_name: operator name that is returned. It is an output parameter.
Return value	Successful: 0 Failed: error codes

nwy_nw_get_signal_csq

Function	int nwy_nw_get_signal_csq(int *csq_val)
Description	To obtain the CSQ of the current network
Parameter	csq_val: CSQ of the current network. It is an output parameter. Its value is the same with the value of CSQ that is queried by the AT command.
Return value	Successful: 0 Failed: error codes

nwy_nw_register_callback_fnuc

Function	int nwy_nw_register_callback_fnuc(nwy_nw_cb_funccb)
Description	To register a network callback function that is customized
Parameter	cb: network callback function
Return value	Successful: 0 Failed: error codes

nwy_nw_unregister_callback_fnuc

Function	int nwy_nw_unregister_callback_fnuc()
Description	To unregister a network callback function that is customized
Parameter	N/A.
Return value	Successful: 0 Failed: error codes

nwy_cb_func

Function	void nwy_cb_func(nwy_nw_regs_ind_type_tind_type, void *ind_struct)
Description	To set the type of a custom callback function
Parameter	ind_type: message type. For details, see **nwy_network.h**. ind_struct: message structure pointer
Return value	Successful: 0 Failed: error codes

nwy_nw_get_forbidden_plmn

Function	int nwy_nw_get_forbidden_plmn(nwy_nw_fplmn_list_t *fplmn_list)
Description	To obtain the list of Forbidden PLMN
Parameter	fplmn_list: obtained FPLMN list.
Return value	Successful: 0 Failed: error codes

nwy_nw_manual_network_scan

Function	int nwy_nw_manual_network_scan(nwy_nw_net_scan_cb_funcscan_cb)
Description	To scan networks manually It is an asynchronous function. Its scanning result is returned by calling the callback function.
Parameter	scan_cb: callback function. After networks are scanned, this function returns the scanning result.
Return value	Successful: 0 Failed: error codes

nwy_nw_manual_network_select

Function	int nwy_nw_manual_network_select(nwy_nw_net_select_param_t *net_select)
Description	To register to a network manually Register a network in the list that is obtained by calling **nwy_nw_manual_network_scan** .
Parameter	net_select: parameters of the network to be register to, including network PLMN and wireless access technologies.
Return value	Successful: 0 Failed: error codes

nwy_nw_band_lock

Function	int nwy_nw_band_lock(uint32_t act, const char *set_band)
Description	To lock the module to one or multiple frequency bands
Parameter	act: 2: GSM 4: LTE set_band: frequency band to be locked, hexadecimal value. For example,

	nwy_nw_band_lock (4,"c000000004"), indicating that the module is locked to LTE B3, B39, and B40 . (You can query the list of frequency bands supported through AT+BANDLOCK or AT+NBANDLOCK. A maximum of five frequency bands can be set simultaneously)
Return value	Successful: 0 Failed: error codes

nwy_nw_freq_lock

Function	int nwy_nw_freq_lock(uint16_t *nfreq, int n)
Description	To lock single or multiple frequency channels A maximum of 9 ARFCNs can be locked simultaneously
Parameter	nfreq: frequency channel to be locked. n refers to the quantity of the frequency channels.
Return value	Successful: 0 Failed: error codes

nwy_nw_get_IMS_state

Function	int nwy_nw_get_IMS_state(uint8_t* on_off)
Description	To obtain the current IMS status
Parameter	on_off: status of IMS 1: enabled 0: disabled
Return value	Successful: 0 Failed: error codes

nwy_nw_set_IMS_state

Function	int nwy_nw_set_IMS_state(uint8_t on_off)
Description	To enable or disable the IMS function
Parameter	on_off: status of IMS to be set 1: enable 0: disable
Return value	Successful: 0 Failed: error codes Before setting IMS, lock the network mode to **LTE ONLY**. Otherwise, errors will be returned.

nwy_nw_get_signal_rssi

Format	int nwy_nw_get_signal_rssi(uint8_t *rssi)
Description	To obtain the actual RSSI value of the current network
Parameter	rssi: RSSI value of the current network
Return value	Successful: 0 Failed: an error code

nwy_nw_get_netmsg

Format	int nwy_nw_get_ netmsg (nwy_serving_cell_info *pNetmsg)
Description	To obtain the information of the current serving cell.
Parameter	pNetmsg: Current network service cell information structure pointer
Return value	Successful: 0 Failed: an error code

nwy_nw_get_cfgdftpdn_info

Format	int nwy_nw_get_cfgdftpdn_info(nwy_nw_cfgdftpdn_t* cfgdftpdn_info)
Description	To obtain the current default bearer PDN information.
Parameter	cfgdftpdn_info: current default bearer PDN information obtained.
Return value	Successful: 0 Failed: an error code

nwy_nw_set_cfgdftpdn_info

Format	int nwy_nw_set_cfgdftpdn_info(nwy_nw_cfgdftpdn_t* cfgdftpdn_info)
Description	To set the current default bearer PDN information.
Parameter	cfgdftpdn_info: Current default bearer PDN information that is set.
Return value	Successful: 0 Failed: an error code

nwy_nw_get_default_pdn_apn

Format	char* nwy_nw_get_default_pdn_apn()
Description	To obtain the default bearer APN.
Parameter	N/A.
Return value	Successful: first address of the APN information string Failed: an error code

nwy_nw_get_neighborLocatorInfo

Format	int nwy_nw_get_neighborLocatorInfo(nwy_locator_report_cb report_cb)
Description	To obtain the neighbor cell information.
Parameter	report_cb: callback function, after the scanning finishes, the scanned cell information is reported.
Return value	Successful: 0 Failed: an error code

2.4.9 FOTA

FOTA APIs can be found in **nwy_fota.h**. They are used to perform firmware upgrades when the device is connected to a network.

nwy_fota_update

Function	unsigned int nwy_fota_update(const void *data, unsigned int size)
Description	To write the FOTA upgrade package into the flash of the module
Parameter	data: data pointer of the upgrade package size: size of the upgrade package
Return value	Return the size of the upgrade package that is written into the flash of the module.

nwy_version_update

Function	int nwy_version_update(boolbRst)
Description	To trigger a FOTA upgrade
Parameter	bRst: flag of immediate restarting true: after the upgrade package is verified, the upgrade is triggered and the module restarts immediately to upgrade.

	false: after the upgrade package is verified, the upgrade is triggered and the module is upgraded when restarting.
Return value	Successful: 0 Failed: error codes

nwy_get_update_result

Function	int nwy_get_update_result(void)
Description	To query whether the firmware is upgraded successfully
Parameter	N/A.
Return value	Successful: 0 Failed: error codes

2.4.10 Virtual AT

AT APIs can be found in **nwy_vir_at.h**. They are used to transmit and receive AT commands.

nwy_sdk_at_parameter_init

Function	void nwy_sdk_at_parameter_init()
Description	To initialize the virtual AT function
Parameter	N/A
Return value	void

nwy_sdk_at_cmd_send

Function	int nwy_sdk_at_cmd_send(nwy_at_info *pInfo, char *resp, int timeout)
Description	To send AT commands
Parameter	pInfo: pointer that points to the information to be sent through the AT command resp: Return value timeout: timeout period ranges from 5 to 30s.
Return value	Successful: 0 Failed: error codes

nwy_sdk_at_unsolicited_cb_reg

Function	int nwy_sdk_at_unsolicited_cb_reg(char *at_prefix, void *p_func);
Description	To register an unsolicited report function
Parameter	at_prefix: character string reported in an unsolicited manner p_func: function used to process the unsolicited report
Return value	Successful: 0 Failed: error codes

2.4.11 Socket

Socket APIs can be found in **nwy_socket.h**.

nwy_socket_open

Function	int nwy_socket_open (int domain, int type, int protocol);
Description	To create and open a socket
Parameter	Domain: protocol family 0: AF_UNSPEC 1: AF_INET AF_INET6 = 0 Type: socket types 1: SOCK_STREAM 2: SOCK_DGRAM 3: SOCK_RAW Protocol: protocol types 6: IPPROTO_TCP 17: IPPROTO_UDP
Return value	Socket descriptors

nwy_socket_send

Function	int nwy_socket_send(int s, const void *data, size_t size, int flags);
Description	To send data through a socket
Parameter	s: socket descriptor data: data to be sent size: content size flags: 0 generally

Return value	Successful: the quantity of received bytes Failed: error code (a value lower than 0)

nwy_socket_recv

Function	int nwy_socket_recv(int s, void *mem, size_t len, int flags);
Description	To receive data through a socket
Parameter	s: socket descriptor mem: the buffer that is used to store the received data len: data length flags: set to 0 generally
Return value	Successful: the quantity of received bytes Failed: error codes (a value lower than 0)

nwy_socket_sendto

Function	int nwy_socket_sendto(int s, const void *data, size_t size, int flags,const struct sockaddr *to, socklen_t tolen)
Description	To send data to the destination address
Parameter	s: socket descriptor data: the buffer that is used to store the received data size: data length flags: 0 generally to: destination address and port number tolen: address length
Return value	Successful: the quantity of received bytes Failed: error codes (a value lower than 0)

nwy_socket_recvfrom

Function	int nwy_socket_recvfrom(int s, void *mem, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);
Description	To receive data from the source address
Parameter	s: socket descriptor mem: the buffer that is used to store the received data len: data length flags: set to 0 generally from: the source address and the source port number fromlen: address length

Return value	Successful: the quantity of received bytes Failed: error codes (a value lower than 0)

nwy_socket_setsockopt

Function	int nwy_socket_setsockopt(int s, int level, int optname, const void * optval, socklen_t optlen);
Description	To set the value of socket options
Parameter	s: socket descriptor level: a protocol level that the option pertains to optname: name of the option to be accessed optval: points to the buffer that contains new options optlen: option length
Return value	Successful: 0 Failed: others

nwy_socket_getsockopt

Function	int nwy_socket_getsockopt(int s, int level, int optname, void * optval, socklen_t * optlen);
Description	To return the value of socket options
Parameter	s: socket descriptor level: a protocol level that the option pertains to optname: name of the option to be accessed optval: points to the buffer that returns new options optlen: maximum length of the option value
Return value	Successful: 0 Failed: others

nwy_gethostbyname

Function	char* nwy_gethostbyname (const char *name);
Description	To obtain the host address according to the domain name
Parameter	name: domain name
Return value	Failed: NULL

nwy_ gethostbyname1

Function	char* nwy_gethostbyname1 (const char *name, int *isipv6);
Description	To obtain the host address according to the domain name
Parameter	name: domain name isipv6: used to determine whether the returned address adopts IPv6
Return value	Failed: NULL

nwy_socket_close

Function	int nwy_socket_close(int socket);
Description	To close the socket
Parameter	socket: socket descriptor
Return value	Successful: 0 Failed: others

nwy_socket_connect

Function	int nwy_socket_connect(int socket, const struct sockaddr *name, socklen_t namelen);
Description	To create a socket link
Parameter	socket: socket descriptor name: server socket namelen: name length of the server socket
Return value	Successful: 0 Failed: others

nwy_socket_bind_lport

Function	int nwy_socket_bind_lport(int socket, uint16_t lport)
Description	To bind to a socket
Parameter	socket: socket descriptor lport: port number
Return value	Successful: 0 Failed: others

nwy_socket_bind

Function	int nwy_socket_bind(int socket, struct sockaddr *addr, socklen_t *addrlen)
Description	To bind to a socket link
Parameter	socket: socket descriptor addr: socket address addrlen: address length
Return value	Successful: 0 Failed: others

nwy_socket_listen

Function	int nwy_socket_listen(int socket, int backlog)
Description	To set a socket to listen
Parameter	socket: socket descriptor backlog: maximum quantity of clients that the socket can listen to
Return value	Successful: 0 Failed: others

nwy_socket_accept

Function	int nwy_socket_accept (int socket, struct sockaddr *addr, socklen_t *addrlen)
Description	To accept a connection request from a socket.
Parameter	socket: socket descriptor addr: socket information addrlen: address length
Return value	Successful: 0 Failed: others

nwy_socket_select

Function	int nwy_socket_select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout)
Description	To select a socket to listen
Parameter	maxfdp1: range of all the file descriptors, that is, the maximum value of the file descriptor plus 1. readset: to monitor the read status of file descriptors Value greater than 0 indicates there are readable files

	writeset: to monitor the write status of file descriptors, value greater than 0 indicates there are writeable files. exceptset: to monitor file abnormalities timeout: timeout period of the select function. • NULL: the select function blocks • 0 seconds 0 ms: the select function is set to a pure non-blocking function. Regardless of whether the file descriptor changes, a value is returned immediately to continue performing its function. If the file does not change, 0 is returned. If the file changes, a positive value is returned. • A value greater than 0: timeout period If an event occurs within the period, a value greater than 0 is returned. A value must be returned when the timeout period is over.
Return value	• The select function encounters abnormalities: a value lower than 0 is returned. • Some files can be read or written: a value greater than 0 is returned. • The function times out and no files can be read or written or the files are wrong: 0 is returned.

nwy_socket_shutdown

Function	int nwy_socket_shutdown(int s, int how)
Description	To close a socket.
Parameter	s: socket descriptor how: SHUT_RD SHUT_WR SHUT_RDWR
Return value	Successful: 0 Failed: -1

nwy_socket_get_ack

Format	int nwy_socket_get_ack(int socket);
Description	To obtain the total length of data that is successfully received by the peer device.
Parameter	socket: socket descriptor
Return value	the total length of data that is successfully received by the peer device.

nwy_socket_get_sent

Format	int nwy_socket_get_sent(int socket);
Description	To obtain the total length of data sent by the socket
Parameter	socket: socket descriptor

Return value	the total length of data sent by the socket

nwy_socket_bind_netif

Format	int nwy_socket_bind_netif(int sockid, int simid, int cid);
Description	To bind the socket to a specific data link.
Parameter	socket: socket descriptor simid: SIM card ID cid: profile ID
Return value	Failed: -1 Successful: 0

2.4.12 FTP

FTP APIs can be found **nwy_ftp.h**.

nwy_ftp_login

Function	int nwy_ftp_login(nwy_ftp_login_t *ftp_param, resultcb cb)
Description	To log in to the FTP service.
Parameter	nwy_ftp_login_t *ftp_param, resultcb: see **nwy_ftp.h**.
Return value	Successful: 0 Failed: -1

nwy_ftp_get

Function	int nwy_ftp_get(const char* filename, uint8_t type, int offset, int len)
Description	To download data form the FTP server.
Parameter	const char* filename, uint8_t type, int offset, int len, see **nwy_ftp.h**.
Return value	Successful: 0 Failed: -1

nwy_ftp_put

Function	int nwy_ftp_put(const char* filename, uint8_t type ,uint8_t mode, const char *data, int len)

Description	To upload data form the FTP server.
Parameter	const char* filename, uint8_t type , uint8_t mode, const char *data, int len: see **nwy_ftp.h**.
Return value	Successful: 0 Failed: -1

nwy_ftp_filesiz

Function	int nwy_ftp_filesize(const char* filename, uint16 tout)
Description	To query the file size of the FTP server.
Parameter	const char* filename, uint16 tout: see **nwy_ftp.h**.
Return value	Successful: 0 Failed: -1

nwy_ftp_logout

Function	int nwy_ftp_logout(uint16 tout)
Description	To log out form the FTP server.
Parameter	uint16 tout: time-out period
Return value	Successful: 0 Failed: -1

nwy_multiplex_ftp_login

Format	int nwy_ multiplex_ftp_login(nwy_ftp_login_t *ftp_param, resultcb cb)
Description	To log in to the FTP server.
Parameter	uint16_t channel , nwy_ multiplex_ftp_login_t *ftp_param, resultcb cb, for detail, see nwy_ftp.h.
Return value	Successful: 0 Failed: -1

nwy_multiplex_ftp_get

Format	int nwy_ multiplex_ftp_get(uint16_t channel, const char* filename, uint8_t type, int offset, int len)
Description	To download data from the FTP server.

Parameter	uint16_t channel , const char* filename, uint8_t type, int offset, int len, for details, see nwy_ftp.h.
Return value	Successful: 0 Failed: -1

nwy_multiplex_ftp_put

Format	int nwy_ multiplex_ftp_put(uint16_t channel, const char* filename, uint8_t type ,uint8_t mode, const char *data, int len)
Description	To upload data to the FTP server.
Parameter	uint16_t channel , const char* filename, uint8_t type , uint8_t mode, const char *data, int len, for details, see nwy_ftp.h.
Return value	Successful: 0 Failed: -1

nwy_multiplex_ftp_filesiz

Format	int nwy_ multiplex_ftp_filesize(uint16_t channel, const char* filename, uint16 tout)
Description	To query the size of the file in the FTP server,
Parameter	uint16_t channel, const char* filename, uint16 tout, for details, see nwy_ftp.h.
Return value	Successful: 0 Failed: -1

nwy_multiplex_ftp_logout

Format	int nwy_ multiplex_ftp_logout(uint16_t channel,uint16 tout)
Description	To log out from the FTP server.
Parameter	uint16_t channel: channel number uint16 tout: timeout period
Return value	Successful: 0 Failed: -1

2.4.13 HTTP/HTTPS

HTTP/HTTPS APIs can be found in **nwy_http.h**.

nwy_http_setup

Function	int nwy_http_setup(uint16_t channel, const char *url, int port, httpresultcb cb)
Description	To establish an HTTP connection.
Parameter	channel: ranging from 0 to 7. url: destination URL port: destination port cb: callback function of event notification, see **nwy_http.h**.
Return value	Successful: 0 Failed: -1

nwy_http_setup_ex

Format	int nwy_http_setup_ex(uint16_t channel, const char *url, int port, void *context,httpresultcb_ex cb)
Description	To set up an HTTP connection (support adding custom parameters).
Parameter	uint16_t channel: channel number ranges from 0 to 7. const char *url: destination path int port: destination port number void *context: custom parameter of upper layer httpresultcb_ex cb: callback function of the event notification, for details, see nwy_http.h. typedef void (*httpresultcb_ex)(nwy_http_result_t *event, void *context); //notify events
Return value	Successful: 0 Failed: -1

nwy_https_setup

Function	int nwy_https_setup(uint16_t channel, const char *url, int port, httpresultcb cb, nwy_app_ssl_conf_t *ssl_cfg)
Description	To establish an HTTPS connection.
Parameter	channel: ranging from 0 to 7. url: destination URL

	port: destination port cb: callback function of event notification, see **nwy_http.h**. *ssl_cfg for the information of SSL configuration, see **nwy_http.h**.
Return value	Successful: 0 Failed: -1

nwy_http_get

Function	int nwy_http_get(uint8_t keepalive, int offset, int size, boolean is_https);
Description	To initiate a GET request.
Parameter	keepalive: determines whether the connection is a long connection offset: specifies the start position of download. size: specifies the download size. boolean is_https: True: HTTPS False: HTTP see **nwy_http.h**.
Return value	Successful: 0 Failed: -1

nwy_http_head

Function	int nwy_http_head(uint8_t keepalive, boolean is_https)
Description	To initiate a HEAD request.
Parameter	keepalive: determines whether the connection is a long connection boolean is_https: True: HTTPS False: HTTP see **nwy_http.h**.
Return value	Successful: 0 Failed: -1

nwy_http_post

Function	int nwy_http_post(uint8_t keepalive, uint8_t type, const char* data, int len, Boolean is_https)
Description	To initiate a POST request.
Parameter	keepalive: determines whether the connection is a long connection.

	type: data packet type data: post data len: data length boolean is_https: True: HTTPS False: HTTP see **nwy_http.h**.
Return value	Successful: 0 Failed: -1

nwy_http_close

Function	int nwy_http_close(boolean is_https)
Description	To close a connection unsolicitedly.
Parameter	boolean is_https: True: HTTPS False: HTTP see **nwy_http.h**.
Return value	Successful: 0 Failed: -1

nwy_multiplex_http_setup

Format	int nwy_ multiplex_http_setup(uint16_t channel, const char *url, int port, httpresultcb cb)
Description	To set up an HTTP connection.
Parameter	uint16_t channel: channel number ranging from 1 to 7. const char *url: destination path int port: destination port number httpresultcb cb: callback function of the event notification, for details, see nwy_http.h.
Return value	Successful: 0 Failed: -1

nwy_multiplex_https_setup

Format	int nwy_ multiplex_https_setup(uint16_t channel, const char *url, int port, httpresultcb cb, nwy_app_ssl_conf_t *ssl_cfg)

Description	To set up an HTTPS connection.
Parameter	uint16_t channel: channel number ranging from 1 to 7. const char *url: destination path int port: destination port number httpresultcb cb: callback function of the event notification, for details, see nwy_http.h. nwy_app_ssl_conf_t *ssl_cfg SSL: for the specific definition of configuration information, see nwy_http.h.
Return value	Successful: 0 Failed: -1

nwy_ multiplex_http_get

Format	int nwy_ multiplex_http_get(uint16_t channel, uint8_t keepalive, int offset, int size, boolean is_https);
Description	To initiate a GET request.
Parameter	uint16_t channel: channel number ranges from 1 to 7. uint8_t keepalive: whether it is a long connection. int offset: specifies the starting position of the download. int size: specifies the length of the data downloaded. boolean is_https: True: HTTPS False: HTTP For details, see nwy_http.h.
Return value	Successful: 0 Failed: -1

nwy_ multiplex_http_head

Format	int nwy_ multiplex_http_head(uint16_t channel, uint8_t keepalive, boolean is_https)
Description	To initiate a HEAD request.
Parameter	uint16_t channel: channel number ranges from 1 to 7. uint8_t keepalive: whether it is a long connection. boolean is_https: True: HTTPS False: HTTP For details, see nwy_http.h.
Return value	Successful: 0 Failed: -1

nwy_ multiplex_http_post

Format	int nwy_ multiplex_http_post(uint16_t channel, uint8_t keepalive, uint8_t type, const char* data, int len, Boolean is_https)
Description	To initiate a POST request.
Parameter	uint16_t channel: channel number ranges from 1 to 7. uint8_t keepalive: whether it is a long connection. uint8_t type: message type const char* data: post data. int len: data length boolean is_https: True: HTTPS False: HTTP For details, see nwy_http.h.
Return value	Successful: 0 Failed: -1

nwy_ multiplex_http_close

Format	int nwy_multiplex_http_close(uint16_t channel, boolean is_https)
Description	To proactively close the connection.
Parameter	uint16_t channel: channel number ranges from 1 to 7. boolean is_https: True: HTTPS False: HTTP
Return value	Successful: 0 Failed: -1

nwy_cert_add

Function	int nwy_cert_add(const char *file_name, const char *data, int length)
Description	To add the SSL certificate.
Parameter	file_name: name of the certificate. The certificate will be stored in **/nwy/**. data: certificate content length: length of the certificate content For details, see **nwy_http.h**.
Return value	Successful: 0 Failed: -1

nwy_cert_check

Function	int nwy_cert_check(const char *file_name)
Description	To validate the SSL certificate.
Parameter	file_name: name of the certificate.
Return value	Successful: 0 Failed: -1

nwy_cert_del

Function	int nwy_cert_del(const char *file_name)
Description	To delete the SSL certificate.
Parameter	file_name: name of the certificate.
Return value	Successful: 0 Failed: -1

2.4.14 BLE

The definitions of these API functions can be found in nwy_ble.h. They are used to perform the BLE-related functions.

nwy_ble_enable

Format	int nwy_ble_enable();
Description	To enable the Bluetooth.
Parameter	N/A.
Return value	Successful: 1 Failed: 0

nwy_ble_set_adv

Format	int nwy_ble_set_adv(bool enable);
Description	To enable/disable the Bluetooth broadcast.
Parameter	enable: 1: enable 0: disable

Return value	Successful: 1 Failed: 0

nwy_ble_send_data

Format	int nwy_ble_send_data(uint16 datalen, char *data);
Description	Send data over BLE.
Parameter	datalen: data length data: data to be sent.
Return value	Successful: 1 Failed: 0

nwy_read_ble_status

Format	int nwy_read_ble_status();
Description	To read the current BLE status.
Parameter	N/A.
Return value	enable: 1 disable: 0

nwy_ble_disable

Format	int nwy_ble_disable();
Description	To disable the Bluetooth.
Parameter	N/A.
Return value	Successful: 1

nwy_ble_receive_data

Format	Char * nwy_ble_receive_data(int sel);
Description	To receive data.
Parameter	sel: select the data returned
Return value	sel=0, return recv_length, length of the data received. sel=1, return nwy_ble_8910_tp_data_value, the data received. sel= one of other values, return 0 after the data receiving space is cleared.

nwy_ble_get_version

Format	int nwy_ble_get_version(char *version);
Description	To obtain the BLE version information.
Parameter	version: version information that is obtained.
Return value	version information

nwy_ble_set_device_name

Format	int nwy_ble_set_device_name(char * local_name);
Description	To set the BLE device name.
Parameter	local_name: device name
Return value	Successful: 1 Failed: 0

nwy_ble_get_device_name

Format	int nwy_ble_get_device_name(uint16 *device_name);
Description	To obtain the BLE device name.
Parameter	name: device name obtained
Return value	N/A.

nwy_ble_update_conn

Format	int nwy_ble_update_conn(uint16 handle, uint16 intervalMin, uint16 intervalMax, uint16 slaveLatency, uint16 timeoutMulti);
Description	To update the BLE connection parameters.
Parameter	Handle: connection handle IntervalMin: the minimum interval, unit: 1.25 ms, ranges from 7.5 ms to 4s. IntervalMax: the maximum interval, unit: 1.25 ms, ranges from 7.5 ms to 4s, IntervalMin<= intervalMax. slaveLatency: latency timeoutMulti: timeout period of the connection, unit: 10ms, ranges from 100 ms to 32s
Return value	N/A.

nwy_ble_register_callback

Format	int nwy_ble_register_callback(void (*ble_reg_cb)
Description	To register the data callback function.
Parameter	Ble_reg_cb: callback function
Return value	Successful: 1

nwy_set_manufacture

Format	int nwy_set_manufacture (UINT8 Manufacture_Data[8])
Description	To customize the data parameter of manufacture.
Parameter	Manufacture_Data[8], the data parameter
Return value	Successful: 1

nwy_ble_beacon

Format	int nwy_ble_beacon (UINT8 uuid[16],UINT8 major[2],UINT8 minor[2])
Description	Beacon funtcion
Parameter	UINT8 uuid[16],UINT8 major[2],UINT8 minor[2], the Beacon parameter
Return value	Successful: 1 Failed: 0

nwy_ble_set_service

Format	int nwy_ble_set_service (unint32_t srv_uuid[])
Description	To set the UUID service.
Parameter	srv_uuid[], the UUID service that is set.
Return value	Successful: 1 Failed: 0

nwy_ble_set_character

Format	int nwy_ble_set_character (UINT8 char_index,uint32_t char_uuid[],uint8_t prop)
Description	To set the character information.
Parameter	char_index: character index char_uuid[]: character UUID

	prop: character property
Return value	Successful: 1 Failed: 0

2.4.15 Standard MQTT

The definitions of these API functions can be found in MQTTClient.h and nwy_mqtt.h. They are used to perform the Standard MQTT related functions.

MQTTIsConnected

Format	int MQTTIsConnected(MQTTClient* client)
Description	To obtain the MQTT connection status.
Parameter	client: MQTT instance
Return value	Successful: 0 Failed: other values

NetworkInit

Format	void NetworkInit(Network* n)
Description	To initialize the network
Parameter	n: network connection parameters
Return value	N/A.

MQTTClientInit

Format	void MQTTClientInit(MQTTClient* c, Network* network, unsigned int command_timeout_ms,unsigned char* sendbuf, size_t sendbuf_size, unsigned char* readbuf, size_t readbuf_size)
Description	To initialize the MQTT instance.
Parameter	c: MQTT instance network: network connection parameter command_timeout_ms: timeout period sendbuf: sending buff sendbuf_size: size of the sending buffer readbuf: reading buffer readbuf_size: size of the reading buffer

Return value	N/A.

NetworkConnect

Format	int NetworkConnect(Network* n, char* addr, int port)
Description	To establish a network connection.
Parameter	n: network connection parameters addr: URL of the server. port: port number of the server.
Return value	Successful: 0 Failed: other values

MQTTClientInit_defaultMessage

Format	void MQTTClientInit_defaultMessage(MQTTClient* client, messageHandler messageHandler
Description	To register the callback function.
Parameter	client: MQTT instance messageHandler: message processing function
Return value	N/A.

nwy_MQTTConnect

Format	int nwy_MQTTConnect(MQTTClient* c, MQTTPacket_connectData* options)
Description	To set up an MQTT connection.
Parameter	c: MQTT instance cptions: connection parameter, including client identifier, user name, password, security time, and so on.
Return value	Successful: 0 Failed: other values

nwy_MQTTPublish

Format	int nwy_MQTTPublish(MQTTClient* c, const char* topicName, MQTTMessage* message)
Description	To publish an MQTT message.

Parameter	c: MQTT instance topicName: topic name message: message body, including the message, message length, retained mark, and so on.
Return value	Successful: 0 Failed: other values

MQTTSubscribe

Format	int MQTTSubscribe(MQTTClient* c, const char* topicFilter, enum QoS qos, messageHandler messageHandler)
Description	MQTT subscribes the topic. To register the message reporting processing function.
Parameter	c: MQTT instance topicFilter: topic name qos: quality of sevice. messageHandler: message processing function
Return value	Successful: 0 Failed: other values

MQTTUnsubscribe

Format	int MQTTUnsubscribe(MQTTClient* c, const char* topicFilter)
Description	To cancel a subscription.
Parameter	c: MQTT instance topicFilter: topic name
Return value	Successful: 0 Failed: other values

MQTTDisconnect

Format	int MQTTDisconnect(MQTTClient* c)
Description	To disconnect the MQTT connection and release the resource.
Parameter	c: MQTT instance
Return value	Successful: 0 Failed: other values

NetworkDisconnect

Format	void NetworkDisconnect(Network* n)
Description	To close the network connection.
Parameter	n: network connection parameters
Return value	N/A.

MQTTYield

Format	int MQTTYield(MQTTClient* c, int timeout_ms)
Description	To receive network messages and distribute the messages to the user's callback function.
Parameter	c: MQTT instance timeout_ms: timeout period of the receive message.
Return value	0

2.4.16 Alibaba MQTT

The definitions of these API functions can be found inmqtt_api.h. They are used to perform the Alibaba MQTT related functions.

IOT_SetupConnInfo

Format	int IOT_SetupConnInfo(const char *product_key, const char *device_name, const char *device_secret, void **info_ptr)
Description	To set the IoT connection parameters.
Parameter	product_key: secret key Device_name: device name device_secret: info_ptr: info structure
Return value	N/A.

IOT_MQTT_CheckStateNormal

Format	int IOT_MQTT_CheckStateNormal(void *handle)

Description	To obtain the IoT connection status.
Parameter	handle: IoT instance
Return value	1 indicates the connection is established.

IOT_MQTT_Construct

Format	void *IOT_MQTT_Construct(MQTTInitParams *pParams)
Description	To create the instance and connect to the server.
Parameter	n: network connection parameters addr: URL of the server. port: port number of the server.
Return value	Successful: 0 Failed: other values

IOT_MQTT_Subscribe

Format	int IOT_MQTT_Subscribe(void *handle, const char *topic_filter, iotx_mqtt_qos_t qos, iotx_mqtt_event_handle_func_fpt topic_handle_func, void *pcontext)
Description	To subscribe a topic.
Parameter	handle: IoT instance topic_filter: topic name qos: quality of sevice. topic_handle_func: report handling function pcontext: user context, it will be sent back through the callback function.
Return value	Successful: 0 Failed: a value < 0

IOT_MQTT_Publish

Format	int IOT_MQTT_Publish(void *handle, const char *topic_name, iotx_mqtt_topic_info_pt topic_msg)
Description	To publish a message.
Parameter	handle: IoT instance topic_filter: topic name topic_msg: message body

Return value	Successful: 0 When the message is QoS1, the Return value is the MQTT message ID of the reported message. Failed: other values

IOT_MQTT_Unsubscribe

Format	int IOT_MQTT_Unsubscribe(void *handle, const char *topic_filter);
Description	To cancel a subscription.
Parameter	handle: IoT instance topic_filter: topic name
Return value	Successful: 0 Failed: a value < 0

IOT_MQTT_Destroy

Format	int IOT_MQTT_Destroy(void **pClient)
Description	To disconnect the connection, destroy the instance, and release the resource.
Parameter	pClient: IoT instance
Return value	Successful: 0 Failed: other values

IOT_MQTT_Yield

Format	int IOT_MQTT_Yield(void *handle, int timeout_ms);
Description	To receive network messages and distribute the messages to the user's callback function.
Parameter	handle: IoT instance timeout_ms: timeout period of the receive message.
Return value	0

2.4.17 Websocket

The definitions of these API functions can be nwy_rws_proc.h. They are used to perform the websocket-related functions.

nwy_ws_open

Format	void *nwy_ws_open(const char *url, const nwy_rws_opt_t *opt, nwy_rws_data_fun_cb data_cb, nwy_rws_close_fun_cb close_cb, void *arg);
Description	To open the websocket.
Parameter	url: the URL that is enabled. opt: option when opening websocket typedef struct _nwy_rws_opt { int ping_interval; /* the ping interval,if input,used the value(the unit is second),other used the default value, */ int ping_timout; /* ping out time ,now not support.*/ int should_handshake;/* if need handshake,the value is 1,ohter 0 */ char *origin; /* if input,used the origin,other used the host */ char *key; /* if input,used the key ,other used default value */ }nwy_rws_opt_t; data_cb: callback deta function typedef void (*nwy_rws_data_fun_cb)(void *t, const void *data, unsigned int len, void *arg); close_cb: the callback function that is used to close the websocket. typedef void (*nwy_rws_close_fun_cb)(void *t,int reason, void *arg);
Return value	Successful: websocket handle Failed: NULL

nwy_ws_send_binary

Format	int nwy_ws_send_binary(void *ws_client, const void *data, unsigned int size);
Description	To send the binary data..
Parameter	ws_client: websocket handle data: data sent. size: size of the data sent
Return value	Successful: 0 Failed: other values

nwy_rws_send_text

Format	int nwy_rws_send_text(void *ws_client, const char *data);
Description	To send the text data
Parameter	ws_client: websocket handle data: data sent.
Return value	Successful: 0 Failed: other values

nwy_rws_close

Format	int nwy_rws_close(void * ws_client);
Description	To close the websocket.
Parameter	ws_client: websocket handle
Return value	Successful: 0 Failed: other values

2.4.18 Packet manage

The definitions of these API functions can be nwy_ip_packet.h. They are used to perform the data package related functions.

nwy_get_recv_ip_packet_reg

Format	void nwy_get_recv_ip_packet_reg(nwy_get_ip_packet cb);
Description	To register to receive the IP packet callback function.
Parameter	cb: receive packet processing function
Return value	NA

nwy_get_send_ip_packet_reg

Format	void nwy_get_recv_ip_packet_reg(nwy_get_ip_packet cb);
Description	To register to send the IP packet callback function.
Parameter	cb: send packet processing function
Return value	NA

2.5 System

2.5.1 Device Management

DM APIs can be found in **nwy_dm.h**.

nwy_dm_get_dev_model

Function	int nwy_dm_get_dev_model(char *model_buf, int buf_len);
Description	To get the device information.
Parameter	model_buf: device information buf_len: lengt
Return value	Successful: NWY_SUCESS Failed: others

nwy_dm_get_inner_version

Function	int nwy_dm_get_inner_version(char *version_buf, int buf_len)
Description	To get the version number.
Parameter	version_buf: version number buf_len: lengt
Return value	Successful: NWY_SUCESS Failed: others

nwy_dm_get_open_sdk_version

Function	int nwy_dm_get_open_sdk_version(char *version_buf, int buf_len)
Description	To get the version number of the open SDK.
Parameter	version_buf: version number buf_len: lengt
Return value	Successful: NWY_SUCESS Failed: others

nwy_dm_get_device_version

Format	void nwy_dm_get_device_version(char* dev_ver, char* lin_ver);
Description	To query the device version number and baseline version number.
Parameter	dev_ver: device version number lin_ver: baseline version number
Return value	NA

nwy_dm_get_rftemperature

Format	int nwy_dm_get_rftemperature(float *outvalue);
Description	To query the internal temperature of the chip.
Parameter	outvalue: the internal temperature of the chip
Return value	Successful: 0 Failed: -1

nwy_dm_get_hw_version

Format	void nwy_dm_get_hw_version(char *hw_ver, int buf_len)
Description	To obatin the hardware version number.
Parameter	hw_ver: the hardware version buffer that is stored. buf_len: buffer length
Return value	NA

2.5.2 Message Queue

Message queue APIs can be found in **nwy_osi_api.h**.

nwy_create_msg_Que

Function	nwy_osiMessageQueue_t *nwy_create_msg_Que(uint32 msg_count, uint32 msg_size)
Description	To initialize the message queue.
Parameter	msg_count: count of the message queues msg_size: size of the message content.
Return value	Failed: NULL Successful: Others

nwy_delete_msg_que

Function	void nwy_delete_msg_que(nwy_osiMessageQueue_t *mq)
Description	To delete the message queue.
Parameter	mq: message queue
Return value	N/A.

nwy_put_msg_que

Function	bool nwy_put_msg_que(nwy_osiMessageQueue_t *mq,const void *msg, uint32 timeout)
Description	To send a message
Parameter	mq: message queue msg: message content Timeout: time-out period (unit: ms) 0 indicates always waiting; value will be returned immediately after 0xffffffff message queue is full.
Return value	Successful: true Failed: false

nwy_get_msg_que

Function	bool nwy_get_msg_que(nwy_osiMessageQueue_t *mq,const void *msg, uint32 timeout)
Description	To accept the message.
Parameter	mq: message queue msg: message content Timeout: time-out period (unit: ms) 0 indicates always waiting; value is returned immediately after 0xffffffff message queue is full.
Return value	Successful: true Failed: false

nwy_get_queue_pendingevent_cnt

Format	uint32_t nwy_get_queue_pendingevent_cnt(nwy_osiMessageQueue_t *mq);
Description	To obtain the number of messages stored in the queue.
Parameter	mq: message queue
Return value	the number of messages stored in the message queue

nwy_get_queue_spaceevent_cnt

Format	uint32_t nwy_get_queue_spaceevent_cnt(nwy_osiMessageQueue_t *mq);
Description	To obtain the number of idle messages in the queue
Parameter	mq: message queue
Return value	the number of idle messages in the message queue

2.5.3 File Operation

File operation APIs can be found in **nwy_file.h**.

nwy_sdk_fopen

Function	int nwy_sdk_fopen(const char *path, nwy_file_action_e flags);
Description	To open a file.
Parameter	path: file path flags: see **nwy_file_action_e**
Return value	Successful: return file descriptor Failed: a value <0

nwy_sdk_fclose

Function	int nwy_sdk_fclose(int fd);
Description	To close a file.
Parameter	fd: file descriptor
Return value	Successful: NWY_SUCESS Failed: others

nwy_sdk_fread

Function	uint32 nwy_sdk_fread(int fd, void *dst, uint32 size);
Description	To read a file.
Parameter	fd: file descriptor dst: read data buffer size: number of bytes
Return value	Successful: number of read bytes Failed: others

nwy_sdk_fwrite

Function	uint32 nwy_sdk_fwrite(int fd, const void *data, uint32 size);
Description	To write a file.
Parameter	fd: file descriptor data: data to be written size: number of bytes
Return value	Successful: written data in bytes Failed: others

nwy_sdk_fseek

Function	int nwy_sdk_fseek(int fd, int offset, nwy_fseek_offset_e mode);
Description	To set the file pointer.
Parameter	fd: file descriptor Offset: offset mode: see **nwy_fseek_offset_e**
Return value	Successful: offset Failed: others

nwy_sdk_fsize

Function	long nwy_sdk_fsize(const char *path);
Description	To get the file size.
Parameter	path: file path
Return value	Successful: the file size in bytes. Failed: others

nwy_sdk_fsize_fd

Function	int nwy_sdk_fsize_fd(int fd);
Description	To get the file size.
Parameter	fd: file descriptor
Return value	Successful: file size in bytes Failed: -1

nwy_sdk_fexist

Format	bool nwy_sdk_fexist(const char *path);
Description	To determine whether the file exists
Parameter	path: file path
Return value	True: existence False: no existence

nwy_sdk_get_stat_path

Format	int nwy_sdk_get_stat_path(const char *path, struct stat *st);
Description	To obtain the file related information according to the file name.
Parameter	path: file name st: file information
Return value	Successful: 0 Failed: -1

nwy_sdk_get_stat_fd

Format	int nwy_sdk_get_stat_fd(int fd, struct stat *st);
Description	To obtain the file related information according to the file descriptor.
Parameter	fd: file descriptor st: file information
Return value	Successful: 0 Failed: -1

nwy_sdk_fsync

Format	int nwy_sdk_fsync(int fd);
Description	To synchronize data in buffer to the file.
Parameter	fd: file descriptor
Return value	Successful: 0 Failed: -1

nwy_sdk_ftrunc_fd

Format	int nwy_sdk_ftrunc_fd(int fd, long len)
Description	To specify the file to be modified to a specific length according to the file descriptor. If the size of the source file is larger than the specified size, the excess part will be deleted.
Parameter	fd: file descriptor len: length of the specified file.
Return value	Successful: 0 Failed: -1

nwy_sdk_ftrunc_path

Format	int nwy_sdk_ftrunc_path(const char *path, long len);
Description	To specify the file to be modified to a specific length according to the file name. If the size of the source file is larger than the specified size, the excess part will be deleted.
Parameter	fd: file descriptor len: length of the specified file.
Return value	Successful: 0 Failed: -1

nwy_sdk_file_unlink

Function	int nwy_sdk_file_unlink(const char* path);
Description	To delete a file
Parameter	path: file path
Return value	Successful: NWY_SUCESS Failed: others

nwy_sdk_frename

Format	int nwy_sdk_frename(const char *oldpath, const char *newpath);
Description	To rename the file.
Parameter	oldpath: the old file name newpath: the new file name
Return value	Successful: 0 Failed: -1

nwy_sdk_vfs_opendir

Format	nwy_dir *nwy_sdk_vfs_opendir(const char *name);
Description	To open a file folder.
Parameter	name: name of the file folder
Return value	Successful: nwy_dir pointer Failed: NULL

nwy_sdk_vfs_readdir

Format	nwy_dirent *nwy_sdk_vfs_readdir(nwy_dir *pdir);
Description	To read a file folder.
Parameter	pdir: file folder descriptor
Return value	Successful: nwy_dirent pointer Failed: NULL

nwy_sdk_vfs_telldir

Format	long nwy_sdk_vfs_telldir(nwy_dir *pdir);
Description	The current reading position of the pdir directory stream. The Return value represents the offset from the beginning of the file. The Return value returns the next reading position.
Parameter	pdir: directory flow
Return value	Successful: offset from the beginning Failed: -1

nwy_sdk_vfs_seekdir

Format	void nwy_sdk_vfs_seekdir(nwy_dir *pdir, long loc);
Description	To set the reading position of the directory stream.
Parameter	pdir: directory flow loc: the offset from the beginning of the directory file
Return value	NA

nwy_sdk_vfs_rewinddir

Format	void nwy_sdk_vfs_rewinddir(nwy_dir *pdir);
Description	To read the location offset to the position where the directory stream started.
Parameter	pdir: directory flow
Return value	NA

nwy_sdk_vfs_closedir

Format	int nwy_sdk_vfs_closedir(nwy_dir *pdir)
Description	To close the folder directory flow.
Parameter	pdir: directory flow
Return value	Successful: 0 Failed: -1

nwy_sdk_vfs_mkdir

Function	int nwy_sdk_vfs_mkdir(const char *name);
Description	To create a directory.
Parameter	path: path of the directory
Return value	Successful: NWY_SUCESS Failed: others

nwy_sdk_vfs_rmdir

Function	int nwy_sdk_vfs_rmdir(const char *name);
Description	To delete the directory.
Parameter	name: path of the directory

Return value	Successful: NWY_SUCESS Failed: others

nwy_sdk_vfs_ls

Function	int nwy_sdk_vfs_ls(void);
Description	To obtain the remaining space size of the user.
Parameter	void
Return value	The remaining space size in bytes.

nwy_sdk_vfs_free_size

Format	int nwy_sdk_vfs_free_size(const char *path);
Description	To obtain the remaining space of the file system.
Parameter	Path: base_path: mount point of this block device in the file system.
Return value	The file system mount point

nwy_sdk_sfile_init

Function	int nwy_sdk_sfile_init(const char *path);
Description	To initialize a secure file operation (power-cut protection).
Parameter	path: file name
Return value	Successful: 0 Failed: -1

nwy_sdk_sfile_read

Function	long nwy_sdk_sfile_read(const char *path, void *dst, long size);
Description	To initiate a secure file read operation.
Parameter	path: file name dst: buffer used to store the read data size: size of the storage buffer
Return value	Successful: size of the data read Failed: -1

nwy_sdk_sfile_write

Function	long nwy_sdk_sfile_write(const char *path, void *data, long size);
Description	To initiate a secure file writing operation.
Parameter	path: file name dst: buffer to be written with data size: buffer size
Return value	Successful: size of the data written Failed: -1

nwy_sdk_sfile_size

Function	long nwy_sdk_sfile_size(const char *path);
Description	To get the size of a secure file.
Parameter	path: file name
Return value	Successful: size of the secure file Failed: -1

nwy_sdk_fread_path

Function	long nwy_sdk_fread_path(const char *path, void *dst, long size);
Description	To read the file according the file name.
Parameter	pPath: file name dst: buffer to be written with data size: buffer size
Return value	Successful: size of the data read Failed: -1

nwy_sdk_fwrite_path

Function	long nwy_sdk_fwrite_path(const char *path, void *data, long size);
Description	To write data according to the file name.
Parameter	path: file name dst: buffer to be written with data size: buffer size
Return value	Successful: size of the data written Failed: -1

nwy_sdk_vfs_rmdir_recursive

Format	int nwy_sdk_vfs_rmdir_recursive(const char *name)
Description	To forcibly delete a folder.
Parameter	name: name of the file folder
Return value	Successful: 0 Failed: -1

2.5.4 Thread

Thread APIs can be found in **nwy_osi_api.h**.

nwy_create_thread

Function	nwy_osiThread_t *nwy_create_thread(const char *N/Ame, nwy_osiCallback_t func, void *argument,int32 priority, uint32 stack_size, uint32 event_count);
Description	To create a thread.
Parameter	name thread name func thread entry function argument thread entry function argument priority thread priority stack_size thread stack size in byte event_count thread event queue depth (count of events can be hold)
Return value	Successful: return the thread pointer Failed: NULL

nwy_create_thread_withstack

Function	nwy_osiThread_t *nwy_create_thread_withstack(const char *name, nwy_osiCallback_t func, void *argument,uint32 priority, void *stack, uint32 stack_size,uint32 event_count);
Description	To create a thread.
Parameter	name thread name func thread entry function argument thread entry function argument priority thread priority stack starting address of the stack stack_size thread stack size in byte

	event_count thread event queue depth (count of events can be hold)
Return value	Successful: return the thread pointer Failed: NULL

nwy_get_current_thread

Function	nwy_osiThread_t *nwy_get_current_thread();
Description	To get all tasks of the current function.
Parameter	N/A
Return value	Successful: return the thread pointer Failed: NULL

nwy_set_thread_priority

Function	bool nwy_set_thread_priority(nwy_osiThread_t *thread, uint32 priority);
Description	To set the priority.
Parameter	thread: thread priority: priority
Return value	Successful: NWY_SUCESS Failed: others

nwy_get_thread_priority

Function	uint32 nwy_get_thread_priority(nwy_osiThread_t *thread);
Description	To get the thread priority.
Parameter	thread: thread
Return value	Successful: thread priority.

nwy_suspend_thread

Function	void nwy_suspend_thread(nwy_osiThread_t *thread);
Description	To suspend a thread.
Parameter	thread: thread
Return value	N/A

nwy_resume_thread

Function	void nwy_resume_thread(nwy_osiThread_t *thread);
Description	To resume a thread.
Parameter	thread: thread
Return value	N/A

nwy_send_thead_event

Function	bool nwy_send_thead_event(nwy_osiThread_t *thread, nwy_osiEvent_t *event, uint32 timeout);
Description	To send an event to the thread.
Parameter	thread: the thread that accepts events event: event Timeout: time-out period, 0 indicates that the function returns immediately after sending events.
Return value	Successful: true Failed: false

nwy_wait_thead_event

Function	bool nwy_wait_thead_event(nwy_osiThread_t *thread, nwy_osiEvent_t *event, uint32 timeout)
Description	The thread used to accept events.
Parameter	thread: current thread event: event timeout: time-out period, 0 indicates wait forever
Return value	Successful: true Failed: false

nwy_exit_thread

Function	void nwy_exit_thread();
Description	To exist from a thread.
Parameter	Void
Return value	Void

nwy_get_thread_pendingevent_cnt

Format	uint32_t nwy_get_thread_pendingevent_cnt(nwy_osiThread_t *thread);
Description	To obtain the number of events stored in the message queue.
Parameter	thread: thread pointer
Return value	The number of events stored in the current thread.

nwy_get_thread_spaceevent_cnt

Format	uint32_t nwy_get_thread_spaceevent_cnt(nwy_osiThread_t *thread);
Description	To obtain the number of free events in the thread queue.
Parameter	thread: thread pointer
Return value	the number of free events in the thread queue

nwy_sleep

Format	void nwy_sleep(uint32 ms);
Description	Sleep delay
Parameter	sleep: delay time, unit: ms
Return value	NA

nwy_usleep

Format	void nwy_usleep(uint32 ms);
Description	Sleep delay
Parameter	sleep: delay time, unit: μs
Return value	NA

2.5.5 mutex

mutex related APIs can be found in **nwy_osi_api.h**.

nwy_create_mutex

Function	nwy_osiMutex_t *nwy_create_mutex();
Description	To initialize the mutually exclusive lock.

Parameter	Void
Return value	Successful: return the mutually exclusive lock Failed: NULL

nwy_lock_mutex

Function	void nwy_lock_mutex(nwy_osiMutex_t *mutex, uint32 timeout);
Description	To add a lock.
Parameter	mutex: mutually exclusive lock timeout: time-out period, unit: ms (0: no timeout)
Return value	Void

nwy_unlock_mutex

Function	bool nwy_unlock_mutex(nwy_osiMutex_t *mutex);
Description	To unlock.
Parameter	mutex: mutually exclusive lock
Return value	Successful: true Failed: false

nwy_delete_mutex

Function	void nwy_delete_mutex(nwy_osiMutex_t *mutex);
Description	To delete a mutually exclusive lock
Parameter	mutex: mutually exclusive lock
Return value	Void

2.5.6 pipe

Pipe APIs can be found in **nwy_osi_api.h**.

nwy_osiPipe_create

Function	nwy_osiPipe_t *nwy_osiPipe_create(unsigned size)
Description	To initialize a pipe.

Parameter	size: buffer size of the pipe.
Return value	Successful: return the created pipe. Failed: NULL

nwy_osiPipe_delete

Function	void nwy_osiPipe_delete(nwy_osiPipe_t *pipe)
Description	To delete the pipe
Parameter	Pipe: pipe that is created.
Return value	void

nwy_osiPipe_read

Function	int nwy_osiPipe_read(nwy_osiPipe_t *pipe, void *buf, unsigned size)
Description	To read data form the pipe.
Parameter	Pipe: pipe that is created. dst: buffer to be read data size: size of the data buffer.
Return value	Successful: number of bytes of data read

osiPipeWrite

Function	int osiPipeWrite(osiPipe_t *pipe, const void *buf, unsigned size);
Description	To write data into the pipe.
Parameter	Pipe: pipe that is created. dst: buffer to be written data size: size of the data buffer.
Return value	number of bytes of data written.

2.5.7 Semaphore

Semaphore APIs can be found in **nwy_osi_api.h**.

nwy_semaphore_create

Function	nwy_osiSemaphore_t *nwy_semaphore_create(uint32 max_count, uint32 init_count);
Description	To initialize the semaphore.
Parameter	max_count:maximum count of the semaphore init_count:initial count of the semaphore
Return value	Successful: return the semaphore Failed: NULL

nwy_semaphore_acquire

Function	bool nwy_semaphore_acquire(nwy_osiSemaphore_t *sem, uint32 timeout);
Description	To acquire the semaphore.
Parameter	sem: semaphore timeout: time-out period, unit:ms (0 indicates no timeout)
Return value	Successful: true Failed: false

nwy_semahpore_release

Function	void nwy_semahpore_release(nwy_osiSemaphore_t *sem);
Description	To release the semaphore.
Parameter	sem: semaphore
Return value	Void

nwy_semahpore_delete

Function	void nwy_semahpore_delete(nwy_osiSemaphore_t *sem);
Description	To delete the semaphore.
Parameter	sem: semaphore
Return value	Void

2.5.8 Time

Time APIs can be found in **nwy_api.h**.

nwy_set_time

Function	void nwy_set_time(nwy_time_t *julian_time, char timezone)
Description	To set the time interface.
Parameter	julian_time: input time timezone: time zone
Return value	N/A.

nwy_get_time

Function	int nwy_get_time(nwy_time_t *julian_time, char *timezone)
Description	To get the time interface.
Parameter	julian_time: time outputted timezone: time zone outputted
Return value	Failed: 0 Successful: 1

nwy_updatetime_ntp

Format	int nwy_updatetime_ntp(char* url, unsigned long timeout, char* tz, unsigned char dst, nwy_update_time_cb cb_unc)
Description	To synchronize the network time.
Parameter	url: requested URL timeout: timeout period ranges from 1 to 30s. tz: selections of time zone, format “E/W digital”, E: Eastern time zone (0-13), W: Western time zone (0-12) dst: daylight saving time switch 0: disable 1: enable cb_unc: result callback function
Return value	0: failed 1: successful

nwy_get_time_zone_switch

Format	int nwy_get_time_zone_switch(nwy_time_zone_switch *status)
Description	To obtain the clock synchronization switch status.
Parameter	status: the obtained clock synchronization switch status NWY_TIME_ZONE_DISABLE = 0, NWY_TIME_ZONE_ENABLE = 1
Return value	Successful: NWY_SUCESS Failed: other values

nwy_set_time_zone_switch

Format	int nwy_set_time_zone_switch(nwy_time_zone_switch status)
Description	To set the clock synchronization switch status.
Parameter	status: set the clock synchronization status NWY_TIME_ZONE_DISABLE = 0, NWY_TIME_ZONE_ENABLE = 1
Return value	Successful: NWY_SUCESS Failed: other values

nwy_get_up_time_us

Format	int nwy_get_up_time_us(void)
Description	To obtain the device boot time.
Parameter	NA
Return value	the current time of the device, unit: μs

2.5.9 Timer

Timer APIs can be found in **nwy_osi_api.h**.

nwy_timer_init

Function	nwy_osTimer_t *nwy_timer_init(nwy_osiThread_t *thread, nwy_osiCallback cb, void *ctx)
Description	To initialize the timer.
Parameter	thread: thread used to process timer messages. cb: callback function of the timer

	*ctx: context of the callback function
Return value	Failed: NULL

nwy_timer_deinit

Function	void nwy_timer_deinit(nwy_osiTimer_t *timer)
Description	To delete a timer
Parameter	timer: timer
Return value	N/A.

nwy_start_timer

Function	bool nwy_start_timer(nwy_osiTimer_*timer, uint32 ms)
Description	One-off timer
Parameter	timer: timer ms: time-out period (unit:ms)
Return value	Successful: true Failed: false

nwy_start_timer_periodic

Function	bool nwy_start_timer_periodic(nwy_osiTimer_*timer, uint32 ms)
Description	Periodical timer
Parameter	timer: timer ms: time-out period (unit:ms)
Return value	Successful: true Failed: false

nwy_stop_timer

Function	void nwy_stop_timer(nwy_osiTimer_t *timer)
Description	To stop the timer.
Parameter	timer: timer
Return value	Successful: true Failed: false