

# UIS8850 OpenCPU

## API Notes

Issue 1.2 Date 2024-04-03



**Copyright © Neoway Technology Co., Ltd 2024. All rights reserved.**

No part or whole of this document may be reproduced or transmitted by any individual or other entity in any form or by any means without prior written consent of Neoway Technology Co., Ltd.

**neoway** is the trademark of Neoway Technology Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## Notice

This document is specifically for Cat.1 (8850) modules.

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

THIS DOCUMENT PROVIDES SUPPORT FOR PRODUCT DESIGN BY THE USER. THE USER MUST DESIGN AND DEBUG THE PRODUCT ACCORDING TO THE SPECIFICATIONS AND PARAMETERS IN THIS DOCUMENT. NEOWAY ASSUMES NO RESPONSIBILITY FOR PERSONAL INJURY AND PROPERTY LOSS CAUSED BY USER'S IMPROPER OPERATION.

DUE TO PRODUCT VERSION UPDATES OR OTHER REASONS, THE CONTENT OF THIS DOCUMENT WILL BE UPDATED AS NECESSARY WITHOUT PRIOR NOTICE.

UNLESS OTHERWISE AGREED, ALL STATEMENTS, INFORMATION AND SUGGESTIONS IN THIS DOCUMENT CONSTITUTE NO WARRANTY WHETHER EXPRESS OR IMPLIED.

Neoway Technology Co., Ltd. provides customers with comprehensive technical support. For any inquiry, please contact your account manager directly or send an email to the following email address:

Sales@neoway.com

Support@neoway.com

**Website:** <http://www.neoway.com>

# Contents

1 About UIS8850 OpenCPU .....	5
1.1 Flash Space .....	5
1.2 Basic Functional Module .....	6
1.3 Block Diagram of the Development Process .....	6
2 APIs .....	8
2.1 Power Management .....	8
2.2 I/O Interfaces .....	11
2.2.1 UART .....	11
2.2.2 I2C .....	13
2.2.3 SPI .....	15
2.2.4 GPIO .....	17
2.2.5 ADC .....	19
2.3 Peripheral .....	20
2.3.1 Audio .....	20
2.3.2 Spi flash .....	24
2.4 Service .....	24
2.4.1 Data .....	24
2.4.2 SIM .....	29
2.4.3 SMS .....	34
2.4.4 Location .....	38
2.4.5 Wi-Fi Scan .....	44
2.4.6 Network .....	44
2.4.7 FOTA .....	50
2.4.8 Virtual AT .....	51
2.4.9 Socket .....	52
2.4.10 FTP .....	59
2.4.11 HTTP/HTTPS .....	60
2.4.12 Standard MQTT .....	63
2.4.13 Alibaba MQTT .....	67
2.4.14 Voice .....	69
2.5 System .....	71
2.5.1 Device Management .....	71
2.5.2 Message Queue .....	73
2.5.3 File Operation .....	75
2.5.4 Thread .....	84
2.5.5 mutex .....	90
2.5.6 pipe .....	91
2.5.7 Semaphores .....	92
2.5.8 Time .....	93
2.5.9 Timer .....	95

# About This Document

## Scope

This document is applicable to UIS8850 OpenCPU.




## Audience

This document is intended for [system engineers \(SEs\)](#), [development engineers](#), and [test engineers](#).

## Change History

Issue	Date	Change	Changed By
1.0	2022-08	Initial draft	Li Jintao
1.1	2022-06	Modified the IOT APIs. Deleted the APIs not supported by 8850. Optimized the API description.	Shui Ying Li Jintao
1.2	2023-08	Corrected some error content.	Li Jintao

## Conventions

Symbol	Indication
	Indicates danger or warning. This information must be followed. Otherwise, a catastrophic module or user device failure or bodily injury may occur.
	Indicates caution. This symbol alerts the user to important points about using the module. If these points are not followed, the module or user device may fail.
	Indicates instructions or tips. This symbol provides advices or suggestions that may be useful when using the module.

# 1 About UIS8850 OpenCPU

The N706-CB OpenCPU module runs FreeRTOS (version number 10.4.4). It mainly provides the following hardware resources.

- ARM Cortex-A5 processor, 500 MHz main frequency
- Supports the installation and running of applications that are developed based on libc by C language
- Memory
  - BM version: RAM: 340 KB, APPIMG: 324 KB, file system: 420 KB
  - DG version: RAM: 2 MB, APPIMG: 1856 KB, File system: 1216 KB
- Supports network modes: LTE, Cat.1
- Supports various services including GPS and Wi-Fi Scan (DG version)
- Supports USB2.0, ADC, UART, SPI, I2C, GPIO

## 1.1 Flash Space

Figure 1-1 Flash space of DG version

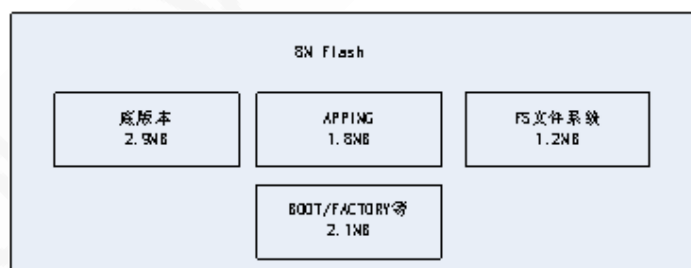
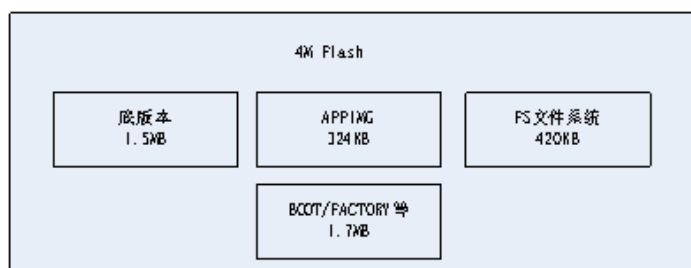


Figure 1-2 Flash space of BM version



- Programming language  
Provides C API functions that run on RTOS and header files which contain the structure definitions.
- Compilation Environment  
It allows developers to build applications on a computer running Windows OS, compile applications using the GCC compiler (IDE), and generate image files (executable files).
- Operating method  
Flash the image file to the flash of the module using a tool and set the file to automatically boot upon power on.

## 1.2 Basic Functional Module

- Data dial-up
- Network registration, network status querying
- SMS related
- Positioning service related (supported by the DG version)
- Wi-Fi scanning related
- SIM card services
- FOTA related
- FOTA
- Sleep and wakeup
- I/O interfaces including GPIO, ADC, SPI, UART, RTC, I2C, etc.
- Virtual AT

## 1.3 Block Diagram of the Development Process



Process description

1. Read the APPIMG from the flash into the RAM

## 2. Run the APPIMG

Enter **apping\_enter**, the entry function of **apping**, to run tasks.

Neoway Confidential

## 2 APIs

### 2.1 Power Management

To enable the system to enter sleep mode or wake up from sleep mode, detect the power supply status, and control fast shutdown or shutdown in a normal process of the module. PM API definitions can be found in **nwy\_pm.h**.

#### nwy\_pm\_psm\_time\_set

Function	int nwy_pm_psm_time_set(unsigned int secs)
Description	To set the time of entering sleep mode the system needs to spend
Parameter	Secs: time
Return Value	Successful: NWY_SUCCESS Failed: NWY_ERROR

#### nwy\_pm\_state\_set

Function	int nwy_pm_state_set(int mode)
Description	To allow the module to enter or exit sleep mode.
Parameter	mode 0: wake-up 1: sleep
Return Value	Successful: NWY_SUCCESS Failed: NWY_ERROR

#### nwy\_power\_state

Function	int nwy_power_state(void)
Description	To detect the power supply status
Parameter	N/A.
Return Value	Successful: 0: abnormal 1: normal



	Failed: NWY_ERROR
--	-------------------

## nwy\_power\_off

Function	int nwy_power_off(int option)
Description	To perform fast shutdown, normal shutdown, or restart operation according <option>
Parameter	option: option for controlling the shutdown of the module 0: fast shutdown 1: shutdown in a normal process 2: restart
Return Value	Successful: NWY_SUCESS Failed: NWY_ERROR

## nwy\_set\_pmu\_power\_level

Function	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 power switch for the submodule
Parameter	id: sub power id enabled: 0: close 1: open at normal run ip_enabled: 0: close 1: open at lowpower mode
Return Value	0: failed Successful: 1

## nwy\_powerkey\_poweroff\_ctrl

Function	int nwy_powerkey_poweroff_ctrl(bool enable);
Description	To enable/disable the shutdown function of the <b>power_n</b> key
Parameter	enable: power_n keys control options for shutdown 0: Disable 1: Enable
Return Value	Successful: NWY_SUCCESS Failed: NWY_ERROR

## nwy\_get\_boot\_causes

Function	unsigned int nwy_get_boot_causes(void)
Description	To obtain the module boot cause
Parameter	NA
Return Value	Successful: startup reason

## nwy\_get\_usb\_status

Function	bool nwy_get_usb_status();
Description	To obtain the USB status
Parameter	NA
Return Value	1- CONNECT 2- DISCONNECT

## nwy\_bootup\_alarm\_set

Function	int nwy_bootup_alarm_set(uint32_t sec_in_day)
Description	To set a scheduled startup 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

Function	int nwy_bootup_alarm_del (void)
----------	---------------------------------

Description	To delete scheduled wake-up alarm (daily recurring alarm).
Parameter	NA
Return Value	Successful: NWY_SUCCESS Failed: NWY_ERROR

## 2.2 I/O Interfaces

### 2.2.1 UART

UART API definitions can be found in **nwy\_uart.h**. To configure UART parameters, and perform UART data reading and writing function.

#### nwy\_uart\_init

Function	int nwy_uart_init(uint32_t name,nwy_uart_mode_t mode);
Description	To initialize the UART
Parameter	name: UART channel mode: 0: AT 1: data
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 UART baud rate
Parameter	Hd: return value of uart init Baud: baud rate
Return Value	Successful: 1 Failed: 0

#### nwy\_uart\_get\_baud

Function	bool nwy_uart_get_baud(uint8_t hd,uint32_t *baud);
Description	To obtain the UART baud rate
Parameter	Hd: return value of uart init Baud: baud rate value

Return Value	Successful: 1 Failed: 0
--------------	----------------------------

## 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 set UART parameters
Parameter	Hd: return value of uart init parity, data_size, stop_size, flowctrl
Return Value	Successful: 1 Failed: 0

## 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 obtain UART configurations
Parameter	Hd: return value of uart init parity, data_size, stop_size, flowctrl
Return Value	Successful: 1 Failed: 0

## nwy\_uart\_send\_data

Function	int nwy_uart_send_data(uint8_t hd, uint8_t *data_ptr, uint32_t length);
Description	To send data via 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\_rcv\_cb

Function	bool nwy_uart_reg_rcv_cb(nwy_uart_rcv_callback_t rcv_cb);
----------	---

Description	To receive data via UART
Parameter	recv_cb: receiving callback function
Return Value	Successful: 1 Failed: 0

### nwy\_uart\_reg\_tx\_cb

Function	bool nwy_uart_reg_tx_cb(uint8_t hd, nwy_uart_send_callback_t tx_cb);
Description	A callback function of completing UART data sending
Parameter	tx_cb: callback function indicating the data is sent
Return Value	Successful: 1 Failed: 0

### nwy\_set\_rx\_frame\_timeout

Function	int nwy_set_rx_frame_timeout(uint8_t hd, int time);
Description	To set the timeout period for UART frames
Parameter	Hd: return value of uart init Time: frame timeout period
Return Value	NA

### nwy\_uart\_deinit

Function	bool nwy_uart_deinit(uint8_t hd);
Description	To close the UART device
Parameter	hd: return value of uart init
Return Value	Successful: 1 Failed: 0

## 2.2.2 I2C

The I2C API definitions 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 and write the slave address
Parameter	i2cDev: I2C device name slaveAddr: slave address
Return Value	Successful: I2C device handle 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 data in specified length from the destination address via the specified I2C slave device
Parameter	fd: I2C device handle slaveAddr: slave address ofstAddr: Register address ptrBuff: points to the buffer that stores data length: Length of the read data
Return Value	Successful: Successful: 0: no data >0: there is 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 data in specified length to the destination address via the specified I2C slave device
Parameter	fd: I2C device handle slaveAddr: slave address ofstAddr: Register address ptrData: points to a buffer that to be written data length: Length of the written data
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 data in bytes from the I2C device
Parameter	fd: I2C device handle *data: read data start_flag: start flag (set the first byte of the data read to 1, other bytes to 0) stop_flag: stop flag (set the last byte of the data read to 1, other bytes to 0)
Return Value	Successful: NWY_SUCESS Failed: values <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 data in bytes to the I2C device
Parameter	fd: I2C device handle length: written data start_flag: start flag (set the first byte of the written read to 1, other bytes to 0) stop_flag: stop flag (set the last byte of the written data to 1, other bytes to 0)
Return Value	Successful: NWY_SUCESS Failed: values <0

### nwy\_i2c\_deinit

Function	int nwy_i2c_deinit(int fd);
Description	To close an I2C device
Parameter	fd: I2C device handle
Return Value	Successful: NWY_SUCESS Failed: values <0

## 2.2.3 SPI

SPI API definitions can be found in nwy\_spi.h. Calling these APIs can perform SPI initialization and data transmission.

### nwy\_spi\_init

Function	int nwy_spi_init(char *spibus, uint8_t mode, uint32_t speed, uint8_t bits)
----------	--

Description	To enable and configure SPI bus
Parameter	spibus: SPI bus name: "SPI1", "SPI2" mode: bus mode: mode0 - mode3 speed: bus speed bits: data bits
Return Value	Successful: a SPI bus handle Failed: <b>SPI_EC_ERROR</b>

### nwy\_spi\_read

Function	uint32_t nwy_spi_read(int hd, void *readaddr, uint32_t len)
Description	To read data via SPI
Parameter	hd: SPI bus handle readaddr: data address len: length of transmitted data
Return Value	Successful: SPI_EC_SUCESS Failed: SPI_EC_ERROR

### nwy\_spi\_cs\_cfg

Function	int nwy_spi_cs_cfg(int hd, uint8_t cs, bool sel)
Description	SPI chip select control
Parameter	hd: SPI bus handle cs: select the CS signal sel: whether to enable the chip select
Return Value	Successful: SPI_EC_SUCESS Failed: SPI_EC_ERROR

### nwy\_spi\_write

Function	uint32_t nwy_spi_write(int hd, void *sendaddr, uint32_t len)
Description	To write data via SPI
Parameter	hd: SPI bus handle sendaddr: data address len: length of transmitted data
Return Value	Successful: SPI_EC_SUCESS Failed: SPI_EC_ERROR



## nwy\_spi\_deinit

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

## 2.2.4 GPIO

GPIO API definitions can be found in **nwy\_gpio\_open.h**. GPIO 0, 2, and 3 are the default GPIO ports. You can configure the GPIO as required. For details, contact Neoway FAEs.

Calling the following APIs can set the GPIO interface to input, output or interrupt mode.

## 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 GPIO pin
Parameter	gpio_id: GPIO ID
Return Value	gpio pin

## nwy\_gpio\_set\_direction

Function	int nwy_gpio_set_direction(uint32 gpio_id, nwy_dir_mode_t direct)
Description	To set the GPIO direction
Parameter	gpio_id: GPIO ID direct: GPIO direction

Return Value	Successful: NWY_SUCCESS Failed: NWY_ERROR
--------------	--

### nwy\_gpio\_set\_value

Function	int nwy_gpio_set_value(uint32_t gpio_id, nwy_value_t value)
Description	To set 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 GPIO interrupt
Parameter	gpio_id: GPIO ID irq_mode: gpioirq 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 the GPIO pull-up/pull-down
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 the GPIO interrupt

Parameter	gpio_id: GPIO ID
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 the 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 the 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 close the GPIO
Parameter	gpio_id: GPIO ID
Return Value	Successful: NWY_SUCCESS Failed: NWY_ERROR

## 2.2.5 ADC

ADC API definitions can be found in **nwy\_adc.h**. To collect voltages

### nwy\_adc\_get

Function	int nwy_adc_get(nwy_adc_t channel, nwy_adc_aux_scale_t scale);
----------	--

Description	To obtain ADC values
Parameter	Channel: ADC channel Scale: voltage range
Return Value	Successful: collected voltage value (unit: mV)

## 2.3 Peripheral

### 2.3.1 Audio

The Audio API definitions 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 external audio amplifier type
Parameter	Pa_type: amplifier type Gpio_num: corresponds to the GPIO connected to the amplifier switch mode: gain selection (effective for a specific amplifier)
Return Value	Successful: NWY_SUCCESS Failed: other values

#### nwy\_audio\_player\_open

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

#### nwy\_audio\_player\_stop

Function	int nwy_audio_player_stop(void)
Description	To stop audio playing
Parameter	N/A.
Return Value	Successful: NWY_SUCCESS

	Failed: other values
--	----------------------

### nwy\_audio\_player\_close

Function	void nwy_audio_player_close(void)
Description	To close the 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 cache data
Parameter	pdata: PCM data len: data length
Return Value	Successful: NWY_SUCCESS Failed: other values

### nwy\_audio\_set\_handset\_vol

Function	void nwy_audio_set_handset_vol(constint step)
Description	To set the loudspeaker volume level
Parameter	Step: volume level (0 - 100)
Return Value	void

### nwy\_audio\_get\_handset\_vol

Function	unsigned int nwy_audio_get_handset_vol(void)
Description	To query the loudspeaker 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 (0 - 100)
Return Value	void

### nwy\_audio\_get\_mic\_vol

Function	unsigned int nwy_audio_get_mic_vol(void)
Description	To query the mic volume
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: recording call back function
Return Value	Successful: NWY_SUCCESS Failed: other values

### nwy\_audio\_recorder\_start

Function	int nwy_audio_recorder_start(void)
Description	To start an audio recording
Parameter	N/A.
Return Value	Successful: NWY_SUCCESS Failed: other values

### nwy\_audio\_recorder\_stop

Function	int nwy_audio_recorder_stop(void)
Description	To stop the audio recording
Parameter	N/A.
Return Value	Successful: NWY_SUCCESS

	Failed: other values
--	----------------------

### nwy\_audio\_recorder\_close

Function	void nwy_audio_recorder_close(void)
Description	To close the video 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 coding type cb: callback function indicates the playing is complete. cb_param: callback parameter interface
Return Value	Successful: NWY_SUCCESS Failed: other values

### 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
Parameter	Tone: DTMF Time: duration (unit: ms) Vol: volume (0 - 15)

Return Value	void
--------------	------

## nwy\_audio\_tone\_detect

Function	void nwy_audio_tone_detect(bool enable, keytonedetectHook_t func)
Description	DTMF detect (phone functionality needs to be supported)
Parameter	enable: enable or disable DTMF detect func: check the call back function
Return Value	void

## nwy\_audio\_file\_record

Function	int nwy_audio_file_record(char *filename, int quality)
Description	To play an audio recording file
Parameter	filename: the name of the audio file to be recorded quality: audio file quality (only valid to AMR)
Return Value	0: successful -1: failed

## nwy\_audio\_file\_play

Function	void nwy_audio_file_play(char *filename)
Description	To play an audio recording file
Parameter	filename: the name of the audio file to be played
Return Value	Void

## 2.3.2 Spi flash

In development

## 2.4 Service

### 2.4.1 Data

Data API definitions can be found in **nwy\_data.h**.



They are mainly used to:

- Set up or close a data dial-up connection.
- Set dial-up parameters.
- Obtain connection status and the related information.

### nwy\_data\_get\_srv\_handle

Function	int nwy_data_get_srv_handle(nwy_data_cb_func cb_func)
Description	<p>To obtain the resource handle of data dial-up connection and set callback function</p> <p>The handle is used to operate the connection.</p> <p>The callback function is used to notify the connection status.</p>
Parameter	cb_func: The registered callback function which is used to notify the connection status.
Return Value	<p>Return the obtained resource handle</p> <p>Successful: 1</p> <p>Failed: a failure code</p>

### nwy\_data\_release\_srv\_handle

Function	void nwy_data_release_srv_handle(int hndl)
Description	<p>To release resources of the specified data dialing deregister the callback function.</p> <p>Note that, before releasing the resources, ensure that the dial-up connection is closed.</p>
Parameter	hndl: Resource handle of the dial-up connection, obtained from <b>nwy_data_get_srv_handle()</b> .
Return Value	N/A.

### nwy\_data\_start\_call

Function	int nwy_data_start_call(int hndl, nwy_data_start_call_v02_t *param)
Description	<p>To start a dial-up connection</p> <p>The result of this command is returned in asynchronous mode. The connection result is obtained through the callback function that is registered</p>

	<b>bynwy_data_get_srv_handle()</b> .
Parameter	hndl: The specified resource handle of the dial-up connection, obtained from <b>nwy_data_get_srv_handle()</b> . para: dial-up parameter
Return Value	Successful: 0 Failed: a value <0

## nwy\_data\_stop\_call

Function	int nwy_data_stop_call(int hndl)
Description	To close the dial-up connection  The result of this command is returned in asynchronous mode. The connection result is obtained through the callback function that is registered by <b>nwy_data_get_srv_handle()</b> .
Parameter	hndl: The specified resource handle of the dial-up connection, obtained from <b>nwy_data_get_srv_handle()</b> .
Return Value	Successful: 0 Failed: a value <0

## nwy\_data\_get\_ip\_addr

Function	int nwy_data_get_ip_addr(int hndl, nwy_data_addr_t_info * info_ptr, int *len)
Description	To obtain the address information of the specified dialing. This API can be invoked only after the dial-up connection is set up successfully.
Parameter	hndl: The specified resource handle of the dial-up connection, obtained from <b>nwy_data_get_srv_handle()</b> . info_ptr: a structure or a set of structures used to return address information If multiple addresses might be returned, reserve a big enough set of structures.  For the address information, see the structure definition in <b>nwy_data_addr_t_info</b> . len: Number of addresses obtained
Return Value	Successful: 0 Failed: a value <0

## nwy\_data\_set\_profile

Function	Int nwy_data_set_profile (int profile_idx, nwy_data_profile_type_t profile_type, nwy_data_profile_info_t * profile_info );
Description	To modify the parameters of the specified profile (with a specified ID and type).
Parameter	profile_idx: Used to specify the ID of the profile to be modified profile_type: Used to specify the type of the profile to be modified. profile_info: Profile parameters, including PDP type, APN, authentication method, user name, and password. See <b>nwy_data_profile_info_t</b> .
Return Value	Successful: 0 Failed: a value <0

### nwy\_data\_get\_profile

Function	int nwy_data_get_profile(int profile_idx, nwy_data_profile_type_t profile_type, nwy_data_profile_info_t * profile_info);
Description	To obtain the 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: The 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 switch the IPv4 address format
Parameter	addr: IPv6 address
Return Value	IPv6 address

### nwy\_ip6addr\_ntoa

Function	char *nwy_ip6addr_ntoa(const nwy_ip6_addr_t *addr)
Description	To switch the IPv6 address format
Parameter	addr: IPv6 address
Return Value	IPv6 address

## nwy\_cmdfunc\_nfrfit

Function	int nwy_cmdfunc_nfrfit(void)
Description	To query the module's comprehensive test result
Parameter	N/A.
Return Value	1: successful 0: failed

## nwy\_cmdfunc\_nfrcal

Function	int nwy_cmdfunc_nfrcal(void)
Description	To query the module's calibration result
Parameter	N/A.
Return Value	1: successful 0: failed

## nwy\_data\_get\_flowcalc\_info

Function	int nwy_data_get_flowcalc_info(nwy_data_flowcalc_info_t* flowcalc_info)
Description	Query the upstream and downstream data traffic of the module
Parameter	flowcalc_info: the upstream and downstream data structures that need to be obtained
Return Value	-2: invalid parameter type 0: the upstream and downstream data is successfully obtained.

## nwy\_set\_dns\_server

Function	int nwy_set_dns_server(int index,char *dns)
Description	To set the DNS server's IP address.
Parameter	Index: sets primary/second DNS dns: sets DNS address character string
Return Value	1: failed 0: successful

## nwy\_get\_dns\_server

Function	int nwy_get_dns_server(char *dnsaddr)
----------	---------------------------------------

Description	To obtain the DNS server's IP address.
Parameter	Dnsaddr: obtains the DNS server address.
Return Value	1: failed 0: successful

## 2.4.2 SIM

SIM API definitions can be found in **nwy\_sim.h**. To operate or query UICC of SIM/USIM/RUIM, including obtaining card status, operating pin, and reading important files in UICC.

### nwy\_sim\_get\_card\_status

Function	nwy_sim_status_t nwy_sim_get_card_status(nwy_sim_id_t sim_id);
Description	To obtain the SIM card status.
Parameter	N/A.
Return Value	Card ready: NWY_SIM_STATUS_READY Card not inserted: NWY_SIM_STATUS_NOT_INSERT PIN locked: NWY_SIM_STATUS_PIN1 PUK locked: NWY_SIM_STATUS_PUK1 Card busy: NWY_SIM_STATUS_BUSY

### nwy\_sim\_get\_iccid

Function	int nwy_sim_get_iccid(nwy_sim_id_t sim_id, char* iccid_buf, size_t buf_len);
Description	To obtain ICCID of the SIM card
Parameter	iccid_buf: ICCID of the memory card
Return Value	Successful: NWY_SUCCESS Failed: the corresponding error code, see <b>nwy_common.h</b> .

### nwy\_sim\_get\_imsi

Function	int nwy_sim_get_imsi(nwy_sim_id_t sim_id, char* imsi_buf, size_t buf_len);
Description	To obtain IMSI of the SIM card
Parameter	sim_id: specifies the SIM card ID. Please set to 0 imsi_buf: buffer used to return IMSI, ensure that the buffer length meets the length requirement.

	buf_len: imsi_buf length
Return Value	Successful: NWY_SUCCESS Failed: the corresponding error code, see <b>nwy_common.h</b> .

### nwy\_sim\_enable\_pin

Function	int nwy_sim_enable_pin(nwy_sim_id_t sim_id, const char* pin)
Description	To enable SIM card PIN
Parameter	sim_id: Specifies an ID number of the card to be operated. pin: character string, the length should comply with 3GPP specifications.
Return Value	Successful: NWY_SUCCESS Failed: the corresponding error code, see <b>nwy_common.h</b>

### nwy\_sim\_disable\_pin

Function	int nwy_sim_disable_pin(nwy_sim_id_t sim_id, const char* pin)
Description	invalid PIN of the SIM card
Parameter	sim_unlock: To input the PINcode of the SIM card
Return Value	Successful: NWY_SUCCESS Failed: the corresponding error code, see <b>nwy_common.h</b>

### nwy\_sim\_get\_pin\_mode

Function	nwy_sim_pin_mode_t nwy_sim_get_pin_mode(nwy_sim_id_t sim_id)
Description	To obtain the SIM card PIN status
Parameter	sim_id: Specifies an ID number of the card to be operated.
Return Value	The current PIN status; for details, see <b>nwy_sim_pin_mode_t</b> .

### nwy\_sim\_get\_simid

Function	uint8 nwy_sim_get_simid()
Description	To obtain the current SIM card
Parameter	void
Return Value	SIM ID: 0 or 1

## nwy\_sim\_get\_msisdn

Function	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 <b>nwy_sim.h</b> . msisdn_buf: MSISDN value. buf_len: invalid currently
Return Value	Successful: NWY_SUCCESS      Failed: a value <0, for details, see <b>nwy_common.h</b>

## nwy\_sim\_verify\_pin

Function	int nwy_sim_verify_pin(nwy_sim_id_t sim_id, const char* pin)
Description	To input PIN 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_SUCCESS Failed: a value <0, for details, see <b>nwy_common.h</b>
Remarks	To input PIN of the SIM card  Note that if a wrong PIN is given three times, the PUK must be inserted in place of the PIN.

## nwy\_sim\_unblock

Function	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, call 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: <b>NWY_SUCESS</b> Failed: a value <0, for details, see <b>nwy_common.h</b> Failed: NWY_RES_ERROR
Remarks	Note that if a wrong PUK is given ten times, the SIM card will be disabled and

never be used to connect to the cellular network again.

## nwy\_sim\_change\_pin

Function	int nwy_sim_change_pin(nwy_sim_id_t sim_id, const char *old_pin, const char *new_pin);
Description	To modify the current PIN
Parameter	sim_id: ID of the SIM card slot, for details, see nwy_sim.h. old_pin: PIN of the SIM card. new_pin: PIN to be set
Return Value	Successful: <b>NWY_SUCESS</b> Failed: a value < 0; for details see <b>nwy_common.h</b> .

The following interfaces are located in **nwy\_poc\_dsds.h**, and the following interfaces can be used to complete the setting of POC dual SIM single standby, including the setting of functional switch, the setting of the default card, the setting of automatic switching, and the interface of switching card.

## nwy\_poc\_dsds\_set\_on\_off

Function	int nwy_poc_dsds_set_on_off(nwy_poc_dsds_onoff_type_e switch_type)
Description	To control the POC switch of the SIM card.
Parameter	switch_type: 0: disable 1: enable
Return Value	Successful: NWY_SUCESS Failed: NWY_GEN_E_UNKNOWN

## nwy\_poc\_dsds\_get\_on\_off

Function	nwy_poc_dsds_onoff_type_e nwy_poc_dsds_get_on_off()
Description	To obtain the POC switch of the SIM card.
Parameter	Void
Return Value	nwy_poc_dsds_onoff_type_e: 0: disable 1: enable



## nwy\_poc\_dsds\_set\_default\_card

Function	int nwy_poc_dsds_set_default_card(nwy_poc_dsds_defaultcard_type_e default_card)
Description	To set the default card used by POC
Parameter	default_card: 0 sim1, 1 sim2
Return Value	Successful: NWY_SUCESS Failed: NWY_GEN_E_UNKNOWN

## nwy\_poc\_dsds\_get\_default\_card

Function	nwy_poc_dsds_defaultcard_type_e nwy_poc_dsds_get_default_card()
Description	To obtain the default card used by POC
Parameter	Void
Return Value	nwy_poc_dsds_defaultcard_type_e: 0sim1, 1sim2

## nwy\_poc\_dsds\_set\_auto\_switch\_card

Function	int nwy_poc_dsds_set_auto_switch_card(nwy_poc_dsds_auto_switch_type_e auto_switch)
Description	To enable the automatic SIM cut-off switch.
Parameter	auto_switch: 0: disable 1: enable
Return Value	Successful: NWY_SUCESS Failed: NWY_GEN_E_UNKNOWN

## nwy\_poc\_dsds\_get\_auto\_switch\_card

Function	nwy_poc_dsds_auto_switch_type_e nwy_poc_dsds_get_auto_switch_card();
Description	To obtain the automatic SIM cut-off switch status.
Parameter	Void
Return Value	nwy_poc_dsds_auto_switch_type_e: 0: disable 1: enable

## nwy\_poc\_dsds\_get\_mastercard\_card

Function	nwy_poc_dsds_mastercard_type_e nwy_poc_dsds_get_mastercard_card()
Description	To obtain the ID of POC currently used.

Parameter	Void
Return Value	nwy_poc_dsds_mastercard_type_e: 0: sim1, 1: sim 2, 3 by default

### nwy\_poc\_dsds\_set\_mastercard\_card

Function	Int nwy_poc_dsds_set_mastercard_card(nwy_poc_dsds_mastercard_type_e master_card);
Description	To set the PIC ID.
Parameter	master_card: 0: SIM 1, 1: SIM 2, 3: default
Return Value	Successful: NWY_SUCESS Failed: NWY_GEN_E_UNKNOWN

## 2.4.3 SMS

SMS API definitions can be found in **nwy\_sms.h**. The APIs are used to send/receive SMS messages and set the storage location.

### nwy\_init\_sms\_option

Function	int nwy_init_sms_option(void)
Description	To initialize the SMS parameters
Parameter	N/A.
Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .

### nwy\_sms\_set\_storage

Function	int nwy_sms_set_storage(nwy_sms_storage_type_e sms_storage)
Description	To set the SMS message location.
Parameter	nwy_sms_storage_type_e sms_storage 1: stored into the module 2: stored into the SIM card
Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .

## nwy\_sms\_get\_storage

Function	nwy_sms_storage_type_e nwy_sms_get_storage(void)
Description	To obtain the SMS message location.
Parameter	N/A.
Return Value	<b>nwy_sms_storage_type_e</b> indicating the message storage location 1: stored into the module 2: stored into the SIM card

## nwy\_set\_report\_option

Function	int nwy_set_report_option(uint8_t mode, uint8_t mt, uint8_t bm, uint8_t ds, uint8_t bfr)
Description	To set the SMS report mode
Parameter	mode: mode of notifying the SMS message after it is received. mt: specifies the rules for managing the received SMS according the message's Data Coding Scheme (DCS). bm: specifies the rules for managing the received Cell Broadcast messages (CBM). ds: specifies the rules for managing the Status Report messages (SMS-STATUS-REPORT). bfr: controls the buffering of URCs: Refers to 3GPP 27.005 The default values for the above parameters are 0, 0, 0, 0, 0.
Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .

## nwy\_sms\_set\_sca

Function	int nwy_sms_set_sca(char *sca, unsigned tosca)
Description	To set the SMSC number.
Parameter	sca: SMSC number tosca: SMSC number type 129: common type 145: international type (contains the character "+") For details, see GSM 04.11
Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .

Remarks	Before setting, confirm with the local operator to ensure that the SMS center number is correct.
---------	--

### nwy\_sms\_get\_sca

Function	int nwy_sms_get_sca(char *sca)
Description	To obtain the SMSC number.
Parameter	sca: the obtained SMSC number.
Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .

### nwy\_sms\_send\_message

Function	int nwy_sms_send_message(nwy_sms_info_type_t *p_sms_data)
Description	To send SMS messages
Parameter	nwy_sms_info_type_t *p_sms_data phone_num: destination number msg_context_len: message length msg_contxet: message content msg_format: coding format 0: GSM7 2: UNICODE For details, see <b>nwy_sms.h</b> .
Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .
Remarks	Returning value only indicates the function execution is successful not data sent successful.

### nwy\_sms\_rcv\_message

Function	void nwy_sms_rcv_message(nwy_sms_rcv_info_type_t *sms_data)
Description	To obtain the received message content
Parameter	nwy_sms_rcv_info_type_t *sms_data oa: sender address oa_size: length of the sender address scts: timestamp dcs: message coding format nIndex: message index number

	nDataLen: message data length pData: message data For details, see <b>nwy_sms.h</b> .
Return Value	N/A.
Remarks	If the SMS report mode is set to "REC UNREAD" (indicates received unread SMS) only an index number will be displayed; you should read the message content according to the index number. If the SMS report mode is set to "REC READ" (indicates received read SMS) the message content will be displayed directly.

### nwy\_sms\_reg\_rcv\_cb

Function	int nwy_sms_reg_rcv_cb(nwy_sms_rcv_cb_t rcv_cb)
Description	A callback function of the URC related to SMS message registration.
Parameter	nwy_sms_rcv_cb_t rcv_cb: callback function For details, see <b>nwy_sms.h</b> .
Return Value	1
Remarks	To register with the callback function of the URC that notifies the receiving of messages

### nwy\_sms\_delete\_message

Function	int nwy_sms_delete_message(uint16_t nindex)
Description	To delete a message
Parameter	nIndex: message index number nStorage: short message storage location 1: stored into the module 2: stored into the SIM card
Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .

### nwy\_sms\_read\_message

Function	int nwy_sms_read_message(unsigned nindex, nwy_sms_rcv_info_type_t *sms_data)
Description	To read a message
Parameter	nIndex: message index number sms_data: message content

Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .
--------------	---

### nwy\_sms\_delete\_message\_by\_type

Function	int nwy_sms_delete_message_by_type(nwy_sms_msg_dflag_e delflag)
Description	To delete a type of messages
Parameter	delflag: specifies a type of messages to be deleted 1: SMSs that have been read 2: SMSs that have been read and sent 3: SMSs that have been read, unsent, and sent 4: all SMSs
Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .

### nwy\_sms\_read\_message\_list

Function	int nwy_sms_read_message_list(nwy_sms_msg_list_t *sms_info)
Description	To obtain index numbers and types of all SMSs
Parameter	nwy_sms_msg_list_t: structure used to obtain SMS Contains the number of SMS messages as well as the index and type of each SMS
Return Value	Successful: 0 Failed: an error code, see <b>nwy_common.h</b> .

## 2.4.4 Location

Used to perform GNSS positioning and set GNSS positioning parameters  
The location API definitions can be found in **nwy\_loc**.

### nwy\_loc\_start\_navigation

Function	nwy_loc_start_navigation(uint8_t nStartMode)
Description	To start GNSS positioning
Parameter	StartMode: 0: START_HOT 1: START_COLD

Return Value	Successful: true Failed: false
--------------	-----------------------------------

## nwy\_loc\_stop\_navigation

Function	int nwy_loc_stop_navigation()
Description	To stop GNSS positioning
Parameter	N/A.
Return Value	Successful: true Failed: false

## nwy\_loc\_set\_position\_mode

Function	int nwy_loc_set_position_mode(uint8_t nSatType)
Description	To set the positioning mode
Parameter	mode: positioning mode <ul style="list-style-type: none"> <li>1: GPS Only</li> <li>2: BDS Only</li> <li>3: GPS_BDS</li> <li>4: GLONASS Only</li> <li>5: GPS_GLONASS</li> <li>8: B1C Only</li> <li>9: GPS_B1C</li> <li>13: GPS_B1C_GLONASS</li> <li>16: GALILEO Only</li> <li>17: GPS_GALILEO</li> <li>18: BDS_GALILEO</li> <li>19: GPS_BDS_GALILEO</li> <li>20: GLONASS_GALILEO</li> <li>21: GPS_GLONASS_GALILEO</li> <li>24: B1C_GALILEO</li> <li>25: GPS_B1C_GALILEO</li> <li>29: GPS_B1C_GLONASS_GALILEO</li> </ul>
Return Value	Successful: true Failed: false
Remarks	There are some baseline versions that do not support all positioning modes mentioned above.

## nwy\_loc\_output\_format

Function	Int nwy_loc_output_format(uint16_t nOutputFormat, uint32_t nTimeInterval)
Description	To set the output format
Parameter	<p>OutputFormat: formats of outputting NMEA statement, comprise the following 8 formats in an output by default.</p> <p>0: GNSS_GGA_TYPE  1: GNSS_GLL_TYPE  2: GNSS_GSA_TYPE  3: GNSS_GSV_TYPE  4: GNSS_RMC_TYPE  5: GNSS_VTG_TYPE  6: GNSS_LOG_TYPE  7: GNSS_TCXO_TYPE  8: GPS_GNSSLOG_TYPE</p> <p>TimeInterval: time interval ranging from 1 to 32768, 1 by default</p>
Return Value	<p>Successful: ture</p> <p>Failed: false</p>
Remarks	N/A.

### nwy\_loc\_nmea\_format\_mode

Function	iint nwy_loc_delete_aiding_data(void)
Description	To delete aiding, ephemeris, and almanac data
Parameter	N/A.
Return Value	<p>Successful: ture</p> <p>Failed: false</p>
Remarks	N/A.

### nwy\_loc\_nmea\_Info\_parse

Function	int nwy_loc_nmea_Info_parse(NWY_GNSS_READ_INFO_CNF_T *locGnsInfo)
Description	To acquire NMEA data
Parameter	<p>locGnsInfo: main position related data analysis</p> <p>float longitude;  float latitude;  float altitude;  NwyGnssTimeStamp utc_time;  uint32_t ttff_time;  uint8_t satellite_num;</p>



	<pre> float speed; float course; uint16_t clockdrift; NWY_GNSS_SCI_TICK_TIME_T m_nSystemTimeStamp; // RTC or any     other system time uint32_t m_nUTCTime;           // second uint8_t m_nUncertaintySemiMajor; uint8_t m_nUncertaintySemiMinor; float m_nBearing; uint16_t m_nDirection;    // UP=0, Down=1 float m_nHorizontalVelocity; // m/s float m_nVerticalVelocity; // m/s uint32_t fix_flag; float m_nHorizontalAccuracy; float m_nPDOP; float m_nHDOP; float m_nVDOP; uint16_t m_nSatelliteIdentifier; uint16_t m_cn0; uint16_t m_nElevation; uint16_t m_nAzimuth; uint8_t m_IsUsed; </pre>
Return Value	<p>Successful: ture Failed: false</p>

## int nwy\_loc\_nmea\_data\_ind

Function	int nwy_loc_nmea_data_ind(nwy_loc_ind_event_func ind_handler)
Description	To obtain primitive NMEA data
Parameter	<pre> nwy_loc_ind_event_func ind_handler typedef struct {     uint16_t length;     uint16_t type;     uint8_t nmea_data[4096]; } NWY_GNSS_OUTPUT_INFO_IND_T; void      (*nwy_loc_ind_event_func)(NWY_GNSS_OUTPUT_INFO_IND_T *ind_msg); nwy_loc_ind_event_func ind_handler callback function used to notify the underlying </pre>
Return Value	<p>Successful: ture Failed: false</p>

Remarks	N/A.
---------	------

## nwy\_loc\_resume

Function	Int nwy_loc_resume(uint8_t enable)
Description	When the priority of the GNSS service is lower than that of the LTE service, this command enables the service to be preoccupied by the LTE service. After the LTE service is completed, the service can automatically resume the GNSS (cold start or hot start) positioning service.
Parameter	The GNSS service resumes the positioning function after being interrupted by the LTE service 0: disable 1: enable
Return Value	Successful: ture Failed: false
Remarks	N/A.

## nwy\_loc\_handle\_priority

Function	int nwy_loc_handle_priority(uint8_t priority)
Description	To set the priority of GNSS service and LTE service to NV, and the setting will take effect after the module is rebooted.
Parameter	priority, the default priority order is 3. 0: GNSS cold start > LTE signaling > GNSS hot start > LTE data 1: GNSS cold start >LTE signaling > LTE data > GNSS hot start 2: GNSS cold start > GNSS hot start >LTE signaling > LTE data 3: LTE signaling > LTE data > GNSS cold start >GNSS hot start
Return Value	Successful: ture Failed: false
Remarks	N/A.

## nwy\_loc\_cipgsmloc\_open

Function	boolnwy_loc_cipgsmloc_open(bool value,nwy_loc_cipgsmloc_callback cb)
Description	To open base station (BS) positioning
Parameter	Value: open or close BS positioning Definitions of the callback function typedef struct {

	<pre>double lat; //latitude double lng; //longitude double alt; //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 latitude and longitude callback function used to notifies the upperlying</pre>
Return Value	<p>Successful: true</p> <p>Failed: false</p>
Remarks	BS positioning is performed using a server provided by Neoway. Ensure that the server operates normally during positioning.

## nwy\_loc\_wifigsmloc\_open

Function	bool nwy_loc_wifigsmloc_open(bool value, nwy_loc_cipgsmloc_callback cb)
Description	To open Wi-Fi positioning
Parameter	<p>Value: enable or disable Wi-Fi positioning</p> <p>Definitions of the callback function</p> <pre>typedef struct {     double lat; //latitude     double lng; //longitude     double alt; //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);</pre>

	nwy_loc_cipgsmloc_callback cb latitude and longitude callback function used to notifies the upperlying
Return Value	Successful: ture Failed: false
Remarks	Before performing Wi-Fi positioning, scan the Wi-Fi hotspots.

## 2.4.5 Wi-Fi Scan

To scan Wi-Fi hotspots

Wi-Fi scan API definitions can be found in **nwy\_wifi.h**.

### nwy\_wifi\_scan

Function	int nwy_wifi_scan(nwy_wifi_scan_list_t *scan_list)
Description	To scan the Wi-Fi list
Parameter	scan_list: Wi-Fi list
Return Value	1: scan the Wi-Fi hotspot successfully

## 2.4.6 Network

Network API definitions can be found in **nwy\_network.h**.

Network APIs are used to set network mode, obtain network registration status and network mode, set network mode, and monitor network status according the registered callback function of the Network API. Note: LTE is the only supported network on 8850, GSM and VoLTE are not supported.

### 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, for details, see the definition in <b>nwy_network.h</b> .
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

### 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: Output parameter, network mode Note that the module developed based on the 8850 platform supports only LTE; other network mode is abnormal.
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

### nwy\_nw\_get\_operator\_name

Function	int nwy_nw_get_operator_name(nwy_nw_operator_name_t *opt_name)
Description	To obtain operator information of the current registered network Long EONS (Enhanced Operator Name String): CHINA MOBILE, Short EONS: CMCC MCC MNC: 460 00 For details, see the definition in the nwy_nw_operator_name_t data structure.
Parameter	opt_name: Operator name
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

### nwy\_nw\_get\_signal\_csq

Function	int nwy_nw_get_signal_csq(unit8_t *csq_val)
Description	To obtain current network signal strength
Parameter	csq_val: Current network signal strength, refer to the AT+CSQ command
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

### nwy\_nw\_register\_callback\_fnuc

Function	int nwy_nw_register_callback_fnuc(nwy_nw_cb_funccb)
Description	To register the callback function of user-defined networks
Parameter	cb: network callback function
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

## nwy\_nw\_unregister\_callback\_fnuc

Function	int nwy_nw_unregister_callback_fnuc()
Description	To de-register the callback function of user-defined networks
Parameter	N/A.
Return Value	Successful: 0 (NWX_SUCESS) Failed: -1 (NWX_ERROR)

## nwy\_cb\_func

Function	void nwy_cb_func(nwy_nw_regs_ind_type_tind_type, void *ind_struct)
Description	To define the callback function type
Parameter	ind_type: indicates message type, see the definition in <b>nwy_network.h</b> . ind_struct: indicates a message structure pointer
Return Value	Successful: 0 (NWX_SUCESS) Failed: -1 (NWX_ERROR)

## nwy\_nw\_get\_forbidden\_plmn

Function	int nwy_nw_get_forbidden_plmn(nwy_nw_fplmn_list_t *fplmn_list)
Description	To obtain Forbidden PLMN lists
Parameter	fplmn_list: obtained FPLMN list
Return Value	Successful: 0 (NWX_SUCESS) Failed: -1 (NWX_ERROR)

## nwy\_nw\_manual\_network\_scan

Function	int nwy_nw_manual_network_scan(nwy_nw_net_scan_cb_funcscan_cb)
Description	To scan networks manually This is an asynchronous function. The scan result is returned through the callback function.
Parameter	scan_cb: callback function. After the scanning completes, this function will be triggered to return the scanning result.
Return Value	Successful: 0 (NWX_SUCESS) Failed: -1 (NWX_ERROR)

## nwy\_nw\_manual\_network\_select

Function	int nwy_nw_manual_network_select(nwy_nw_net_select_param_t *net_select)
Description	To register network manually, that is, to select a network for registration according to the network list obtained by nwy_nw_manual_network_scan.
Parameter	net_select: select the parameter of the registered network including the network's PLMN and the wireless access technology
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

## nwy\_nw\_band\_lock

Function	int nwy_nw_band_lock(uint32_t act, const char *set_band)
Description	To lock a single frequency or multiple frequencies
Parameter	act: only 4 (LTE) is valid (limited by the 8850 platform) Set_band: band(s) to be locked. 5 bands are supported at most (a hexadecimal number, corresponding to a binary bit) Input in hexadecimal format: 4: 0100—— lock band 3 1: 0001—— lock band 1 0X8000000000 — — lock band 39 (notably, the actual value does not include "0x")
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

## nwy\_nw\_freq\_lock

Function	int nwy_nw_freq_lock(uint16_t *nfreq, int n)
Description	To lock single or multiple frequencies, 9 frequencies can be locked simultaneously at most.
Parameter	nfreq: frequency to be locked n: number of frequencies to be locked
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

## nwy\_nw\_get\_signal\_rssi

Function	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 (NWY_SUCESS) Failed: -1 (NWY_ERROR)
--------------	--

### nwy\_nw\_get\_netmsg

Function	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 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

### nwy\_nw\_get\_cfgdftpdn\_info

Function	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 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

### nwy\_nw\_set\_cfgdftpdn\_info

Function	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 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

### nwy\_nw\_get\_default\_pdn\_apn

Function	char* nwy_nw_get_default_pdn_apn()
Description	To obtain the default bearer APN
Parameter	N/A.
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)



## nwy\_nw\_get\_neighborLocatorInfo

Function	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 (NWX_SUCCESS) Failed: -1 (NWX_ERROR)

## nwy\_nw\_get\_radio\_st

Function	int nwy_nw_get_radio_st(int *status)
Description	To obtain the current Radio status
Parameter	Status: 0: flight mode 1: normal mode
Return Value	Successful: 0 (NWX_SUCCESS) Failed: -1 (NWX_ERROR)

## nwy\_nw\_set\_radio\_st

Function	int nwy_nw_set_radio_st(int fun)
Description	To set the current Radio status
Parameter	fun: 0: flight mode 1: normal mode
Return Value	Successful: 0 (NWX_SUCCESS) Failed: -1 (NWX_ERROR)

## nwy\_nw\_get\_lacid

Function	int nwy_nw_get_lacid(char lac[], char cid[]);
Description	To obtain lac and cid
Parameter	Lac: location area code cid: cell id This function can obtain the corresponding parameters, which can be used for preliminary positioning.

## Return Value

Successful: 0 (NWY\_SUCESS)  
Failed: -1 (NWY\_ERROR)

## 2.4.7 FOTA

To enable a network-connected device to perform App firmware upgrades.

### nwy\_fota\_download\_core

Function	int nwy_fota_download_core(ota_package_t *ota_pack)
Description	To write the FOTA upgrade package into Flash
Parameter	ota_pack: OTA package related parameter
Return Value	Successful: 0 (NWY_SUCESS) Failed: -1 (NWY_ERROR)

### nwy\_version\_core\_update

Function	int nwy_version_core_update (boolbRst)
Description	To trigger an FOTA upgrade
Parameter	bRst: flag of immediate restart true: Restart to upgrade immediately After the upgrade package is checked, the upgrade will be triggered automatically and the module will restart after the upgrade is completed. false: Upgrade in the next restart After the upgrade package is checked, the upgrade will be triggered by the manual restart operation and the module will be restarted after the upgrade is completed.
Return Value	Successful: 0 Failed: an error code

### nwy\_get\_update\_result

Function	int nwy_get_update_result(void)
Description	To query the upgrade result
Parameter	N/A.
Return Value	Successful: 0 Failed: an error code

## 2.4.8 Virtual AT

Virtual AT API definitions can be found in **nwy\_vir\_at.h**. To send or 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 *plnfo, char *resp, int timeout)
Description	To send AT commands
Parameter	plnfo: pointer used to send AT content resp: return value timeout: timeout period 5 - 30s
Return Value	Successful: 0 Failed: an error code
Remarks	Executing this function can send up to 1024-byte data.

### nwy\_sdk\_at\_unsolicited\_cb\_reg

Function	int nwy_sdk_at_unsolicited_cb_reg(char *at_prefix, void *p_func);
Description	A URC function related to AT registration
Parameter	at_prefix: a string of the URC p_func: processing function of the URC
Return Value	Successful: 0 Failed: an error code
Remarks	Currently, at most 32 URCs can be registered for notification.

## 2.4.9 Socket

Socket API definitions 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 the socket
Parameter	domain: protocol stacks AF_UNSPEC = 0 AF_INET = 1 AF_INET6 = 0 Type: socket type SOCK_STREAM = 1 SOCK_DGRAM = 2 SOCK_RAW = 3 Protocol: protocol type IPPROTO_TCP=6 IPPROTO_UDP=17
Return Value	a socket descriptor

## nwy\_socket\_send

Function	int nwy_socket_send(int s, const void *data, size_t size, int flags);
Description	To send data over a socket
Parameter	s: socket descriptor data: data to be sent size: data length flags: 0 in general
Return Value	Successful: a value >0 (number of data bytes) Failed: a value <0 (an error code, see <b>nwy_open_socket.h.</b> )

## nwy\_socket\_recv

Function	int nwy_socket_recv(int s, void *mem, size_t len, int flags);
Description	To receive data via a socket
Parameter	s: socket descriptor mem: buffer used to store the received data len: data length flags: 0 in general
Return Value	Successful: a value >0 (number of data bytes) Failed: a value <0 (an error code, see <b>nwy_open_socket.h.</b> )

## 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 a destination address
Parameter	s: socket descriptor data: buffer used to store the received data size: data length flags: 0 in general to: destination IP address and port number tolen: address length
Return Value	Successful: a value >0 (number of data bytes) Failed: a value <0 (an error code, see <b>nwy_open_socket.h.</b> )

## 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 an original address
Parameter	s: socket descriptor mem: buffer used to store the received data len: data length flags: 0 in general from: original IP address and port number fromlen: address length
Return Value	Successful: a value >0 (number of data bytes) Failed: a value <0 (an error code, see <b>nwy_open_socket.h.</b> )

## nwy\_socket\_setsockopt

Function	int nwy_socket_setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);
Description	To set the socket option
Parameter	s: socket descriptor level: the layer of protocol to which this option will be applied optname: option name to be accessed optval: points to the buffer containing new options optlen: option length
Return Value	Successful: 0 Failed: a non-zero value

## nwy\_socket\_getsockopt

Function	int nwy_socket_getsockopt(int s, int level, int optname, void * optval, socklen_t * optlen);
Description	To return the socket option value
Parameter	s: socket descriptor level: the layer of protocol to which this option will be applied optname: option name to be accessed optval: points to the buffer returning option values optlen: maximum length of the option value
Return Value	Successful: 0 Failed: a non-zero value

## nwy\_gethostbyname

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

## nwy\_gethostbyname1

Function	char* nwy_gethostbyname1(const char *name, int *isipv6);
Description	To obtain a host address according to a domain name
Parameter	name: domain name isipv6: specifies whether the return value is an IPv6 address
Return Value	Failed: NULL

## nwy\_socket\_close

Function	int nwy_socket_close(int socket);
Description	To close a socket
Parameter	socket: socket descriptor
Return Value	Successful: 0 Failed: a non-zero value

## nwy\_socket\_connect

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

## nwy\_socket\_bind\_lport

Function	int nwy_socket_bind_lport(int socket, uint16_t lport)
Description	To bind a socket
Parameter	socket: socket descriptor lport: port number
Return Value	Successful: 0 Failed: a non-zero value

## nwy\_socket\_bind

Function	int nwy_socket_bind(int socket, struct sockaddr *addr, socklen_t *addrlen)
Description	To bind a socket connection
Parameter	socket: socket descriptor addr: socket information addrlen: length
Return Value	Successful: 0 Failed: a non-zero value

## nwy\_socket\_listen

Function	int nwy_socket_listen(int socket, int backlog)
Description	To create a socket listener
Parameter	socket: socket descriptor backlog: maximum number of clients that can be listened
Return Value	Successful: 0 Failed: a non-zero value

## nwy\_socket\_accept

Function	int nwy_socket_accept (int socket, struct sockaddr *addr, socklen_t *addrlen)
Description	To receive a socket connection
Parameter	socket: socket descriptor addr: socket information addrlen: length
Return Value	Successful: 0 Failed: a non-zero value

## nwy\_socket\_select

Function	int nwy_socket_select(int maxfdp1, fd_set *readset, fd_set *writerset, fd_set *exceptset, struct timeval *timeout)
Description	To select a socket listener
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; a value greater than 0 indicates there are readable files



	<p>writeset: to monitor the write status of file descriptors; a value greater than 0 indicates there are writable files.</p> <p>exceptset: to monitor file abnormalities</p> <p>timeout: timeout period of the select function</p> <ul style="list-style-type: none"> <li>• NULL: the select function blocks</li> <li>• 0s 0ms: 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.</li> <li>• A value greater than 0: timeout period</li> </ul> <p>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.</p>
Return Value	<ul style="list-style-type: none"> <li>• The select function encounters abnormalities: a value lower than 0 is returned.</li> <li>• Some files can be read or written: a value greater than 0 is returned.</li> <li>• The function times out and no files can be read or written or the files are wrong: 0 is returned.</li> </ul>
Remarks	<p>After a message is received by using the SELECT mechanism, send this message to another thread (e.g. thread A) through an Event, and the message will be processed in the thread. Otherwise, the message will fail to be processed since the SELECT function is blocked, when it is used, which affects the event reception.</p>

## 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 -1: failed

## nwy\_socket\_get\_ack

Function	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

Function	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

Function	int nwy_socket_bind_netif(int sockid, int simid, int cid);
Description	To bind the socket to a specific data connection
Parameter	socket: socket descriptor simid: SIM card ID cid: profile id
Return Value	Failed: -1 Successful: 0
Remarks	This function is applicable for multi-APN processing. Executing this function can distribute different data streams through to different APNs.

### nwy\_socket\_event\_report\_reg

Function	int nwy_socket_event_report_reg(nwy_socket_event_cb cb)
Description	To register the report processing function for the socket event
Parameter	cb: event processing function
Return Value	Failed: -1 Successful: 0

### nwy\_get\_default\_netif\_v4\_addr

Function	int nwy_get_default_netif_v4_addr(ip4_addr_t *addr)
Description	To obtain the IPv4 address of the default network interface
Parameter	addr: the obtained IPv4 address (input parameter)
Return Value	Failed: -1 Successful: 0

## nwy\_socket\_get\_state

Function	int nwy_socket_get_state(int socket)
Description	To obtain the socket status
Parameter	socket: socket ID
Return Value	Refer to the enumerated values of nwy_tcp_state in <b>nwy_socket.h</b> .

## 2.4.10 FTP

The definitions of these API functions can be found in **nwy\_ftp.h**. They are used to perform the FTP-related functions.

### nwy\_ftp\_login

Function	int nwy_ftp_login(nwy_ftp_login_t *ftp_param, resultcb cb)
Description	To log in to the FTP server
Parameter	nwy_ftp_login_t *ftp_param, resultcb cb //refers to <b>nwy_ftp.h</b> .
Return Value	Successful: 0 Failed: -1

### nwy\_ftp\_get

Function	int nwy_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 //refers to <b>nwy_ftp.h</b> .
Return Value	Successful: 0 Failed: -1

### nwy\_ftp\_put

Function	int nwy_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 <b>nwy_ftp.h</b> .
Return Value	Successful: 0 Failed: -1

### nwy\_ftp\_filesize

Function	int nwy_ftp_filesize(uint16_t channel,const char* filename, uint16_t tout);
Description	To query the size of the file in the FTP server
Parameter	uint16_t channel,const char* filename, uint16_t tout //For details, see <b>nwy_ftp.h</b> .
Return Value	Successful: 0 Failed: -1

### nwy\_ftp\_logout

Function	int nwy_ftp_logout(uint16_t channel,uint16_t tout);
Description	To log out from the FTP server
Parameter	uint16_t channel,uint16_t tout nwy_ftp.h
Return Value	Successful: 0 Failed: -1

## 2.4.11 HTTP/HTTPS

The definitions of these API functions can be found in **nwy\_http.h**. They are used to perform the HTTP/HTTPS-related functions

### nwy\_http\_setup

Function	int nwy_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 ranges from 0 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\_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 set up an HTTPS connection
Parameter	uint16_t channel: channel number ranges from 0 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\_http\_get

Function	int nwy_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 0 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 For details, see nwy_http.h.
Return Value	Successful: 0 Failed: -1

## nwy\_http\_head

Function	int nwy_http_head(uint16_t channel, uint8_t keepalive, boolean is_https);
Description	To initiate an HEAD request

Parameter	uint16_t channel: channel number ranges from 0 to 7. uint8_t keepalive: whether it is a long connection. boolean is_https: true https: false For details, see nwy_http.h.
Return Value	Successful: 0 Failed: -1

## nwy\_http\_post

Function	int nwy_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 0 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 For details, see nwy_http.h.
Return Value	Successful: 0 Failed: -1

## nwy\_http\_close

Function	int nwy_http_close(uint16_t channel, boolean is_https);
Description	To initiate a disconnection request
Parameter	uint16_t channel: channel number ranges from 0 to 7. boolean is_https: true https: false
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 an SSL certificate

Parameter	const char *file_name: certificate name, the certificate will be stored in the <b>/nwy/</b> directory. const char *data: certificate content int length: certificate length For details, see <b>nwy_http.h</b> .
Return Value	Successful: 0 Failed: -1

### nwy\_cert\_check

Function	int nwy_cert_check(const char *file_name)
Description	To confirm an SSL certificate
Parameter	const char *file_name: certificate name
Return Value	Successful: 0 Failed: -1

### nwy\_cert\_del

Function	int nwy_cert_del(const char *file_name)
Description	To delete an SSL certificate
Parameter	const char *file_name: certificate name
Return Value	Successful: 0 Failed: -1

## 2.4.12 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.

### MQTTClient\_isConnected

Function	int MQTTClient_isConnected (MQTTClient handle)
Description	To obtain the MQTT connection status.
Parameter	client: MQTT instance
Return Value	Connected: 1 Not connected: a value not equal to 1

## MQTTClient\_init

Function	void MQTTClient _init()
Description	To perform an MQTT initialization process
Parameter	N/A.
Return Value	N/A.
Remarks	You should perform the initialization process before using MQTT; otherwise, the module will crash.

## MQTTClient\_create

Function	int MQTTClient_create(MQTTClient* handle, const char* serverURI, const char* clientId, int persistence_type, void* persistence_context)
Description	To create an MQTT instance
Parameter	handle: MQTT instance serverURI: URL of the server clientId: clientId of the server persistence_type: persistence type, 0 by default persistence_context: NULL by default
Return Value	Successful: 0 Failed: another value
Remarks	MQTT version 5.0 requires setting persistence. The default persistence_type for version 3.1.1 is 0, and the persistence_context is NULL.

## MQTTClient\_setCallbacks

Function	Int MQTTClient_setCallbacks(MQTTClient handle, void* context, MQTTClient_connectionLost* cl, MQTTClient_messageArrived* ma, MQTTClient_deliveryComplete* dc)
Description	To register the callback function.
Parameter	handle: MQTT instance Cl: MQTT lost callback function ma: message processing function dc: callback function of MQTT delivery
Return Value	Report the subscribed topic message in the message handling function.

## MQTTClient\_connect



Function	int MQTTClient_connect(MQTTClient handle, MQTTClient_connectOptions* options)
Description	To set up an MQTT connection
Parameter	handle: MQTT instance options: connection parameter, including client identifier, user name, password, security time, and so on.
Return Value	Successful: 0 Failed: another value
Remarks	After the MQTT connection is established successfully, you should create a thread to circularly monitor the MQTT connection/disconnection status.

## MQTTClient\_publish

Function	int MQTTClient_publish(MQTTClient handle, const char* topicName, int payloadlen, const void* payload, int qos, int retained, MQTTClient_deliveryToken* deliveryToken)
Description	To publish MQTT messages
Parameter	handle: MQTT instance topicName: topic name payloadlen: message length payload: message content Qos: quality of service, ranging from 0 to 2 Retained: reserved flag
Return Value	Successful: 0 Failed: another value

## MQTTClient\_subscribe

Function	int MQTTClient_subscribe(MQTTClient handle, const char* topic, int qos)
Description	MQTT subscribes the topic. To register the message reporting processing function.
Parameter	handle: MQTT instance topic: topic name Qos: quality of service, ranging from 0 to 2
Return Value	Successful: 1 Failed: another value
Remarks	The messages subscribed to the topic are processed in the callback function.

## MQTTClient\_unsubscribe

Function	int MQTTClient_unsubscribe(MQTTClient handle, const char* topic)
Description	To cancel a subscription
Parameter	handle: MQTT instance topic: topic name
Return Value	Successful: 0 Failed: another value

## MQTTClient\_disconnect

Function	int MQTTClient_disconnect(MQTTClient handle, int timeout)
Description	To disconnect the MQTT connection and release the resource
Parameter	handle: MQTT instance timeout: timeout period
Return Value	Successful: 0 Failed: another value
Remarks	N/A.

## MQTTClient\_distroy

Function	Void MQTTClient_distroy(MQTTClient handle)
Description	To release MQTT resources.
Parameter	handle: MQTT instance
Return Value	N/A.
Remarks	N/A.

## MQTTClient\_yield

Function	int MQTTYield()
Description	To receive network messages and distribute the messages to the user's callback function
Parameter	c: MQTT instance
Return Value	0
Remarks	After an MQTT connection is set up, you need to create a thread to obtain network messages.

## MQTTClient\_freeMessage

Function	void MQTTClient_freeMessage(MQTTClient_message** msg)
Description	To release the received message Body
Parameter	Msg: the received message
Return Value	0
Remarks	After the MQTT receives and process the msg, the received msg should be released actively.

## MQTTClient\_free

Function	void MQTTClient_free(void* ptr)
Description	To release some memory of the MQTT library.
Parameter	ptr: memory pointer
Return Value	0
Remarks	After the MQTT receives and process the msg, the topic name should be released actively.

## 2.4.13 Alibaba MQTT

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

## IOT\_SetupConnInfo

Function	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: device secret key info_ptr: structure of the info connect parameter
Return Value	N/A.

## IOT\_MQTT\_CheckStateNormal

Function	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

Function	void *IOT_MQTT_Construct(MQTTInitParams *pParams)
Description	To create an instance and connect to the server
Parameter	pParams: MQTT connection parameters, including server URL, port, three elements, IoT time reporting function, etc.
Return Value	Successful: 0 Failed: another value

## IOT\_MQTT\_Subscribe

Function	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 to a topic
Parameter	Handle: IoT instance topic_filter: topic name qos: quality of service. 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

Function	int IOT_MQTT_Publish(void *handle, const char *topic_name, iotx_mqtt_topic_info_pt topic_msg)
Description	To publish messages.
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: another value
--------------	--

## IOT\_MQTT\_Unsubscribe

Function	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

Function	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: another value

## IOT\_MQTT\_Yield

Function	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
Remarks	After an IoT instance is created and the module connects to the server through the instance, you should create a thread to call IOT_MQTT_Yield which is used to receive network messages.

## 2.4.14 Voice

Voice API definitions can be found in **nwy\_voice.h**. The APIs can implement voice call-related features.

## nwy\_voice\_call\_start

Function	int nwy_voice_call_start(uint8_t sim_id, char *phone_num)
Description	To make a phone call
Parameter	sim_id: Currently only SIM 1 is supported. phone_num: Destination number
Return Value	Successful: <b>NWY_RES_OK</b> Failed: <b>nwy_error_t</b>
Remarks	When entering a phone number, please note the phone number format.

## nwy\_voice\_call\_end

Function	int nwy_voice_call_end(uint8_t sim_id)
Description	To hang up a phone call
Parameter	sim_id: Currently only SIM 1 is supported.
Return Value	Successful: <b>NWY_RES_OK</b> Failed: <b>nwy_error_t</b>

## nwy\_voice\_call\_autoanswer

Function	int nwy_voice_call_autoanswer()
Description	To answer a phone call
Parameter	N/A.
Return Value	Successful: <b>NWY_RES_OK</b> Failed: <b>nwy_error_t</b>

## nwy\_voice\_setvolte

Function	int nwy_voice_setvolte(uint8_t sim_id, uint8_t setvolte)
Description	To set VoLTE voice
Parameter	sim_id: Currently only SIM 1 is supported. setvolte: To open IMS
Return Value	Successful: <b>NWY_RES_OK</b> Failed: other values
Remarks	Ensure that the used SIM card supports VoLTE.

## nwy\_voice\_add\_statehandler

Function	Int nwy_voice_add_statehandler(nwy_voice_statehandlerfunc state_handle)
Description	To register the URCs related to voice, including caller ID display and callee status.
Parameter	<pre>typedef void (*nwy_voice_statehandlerfunc) (     int          call_id,     char*        phone_num,     nwy_vc_call_state_t  state,     void         *contextPtr ); typedef struct nwy_voice_handle {     nwy_voice_statehandlerfunc func; }nwy_voice_handle_t; callback function used to notify the underlying</pre>
Return Value	Successful: NWY_RES_OK

## 2.5 System

To perform device related operations. The API definitions can be found in **nwy\_dm.h**.

### 2.5.1 Device Management

## nwy\_dm\_get\_dev\_model

Function	int nwy_dm_get_dev_model(char *model_buf, int buf_len);
Description	To obtain device information
Parameter	model_buf: device information buf_len: buffer length
Return Value	Successful: NWY_SUCESS Failed: another value

## nwy\_dm\_get\_inner\_version

Function	int nwy_dm_get_inner_version(char *version_buf, int buf_len)
Description	To query the firmware version
Parameter	version_buf: version number buf_len: buffer length

## Return Value

Successful: NWY\_SUCESS  
Failed: another value

## nwy\_dm\_get\_open\_sdk\_version

## Function

int nwy\_dm\_get\_open\_sdk\_version(char \*version\_buf, int buf\_len)

## Description

To query the Open SDK version

## Parameter

version\_buf: version number  
buf\_len: buffer length

## Return Value

Successful: NWY\_SUCESS  
Failed: another value

## nwy\_dm\_get\_device\_version

## Function

void nwy\_dm\_get\_device\_version(char\* dev\_ver, char\* lin\_ver);

## Description

To query the device version number and the baseline version number

## Parameter

dev\_ver: device version number  
lin\_ver: baseline version number

## Return Value

NA

## nwy\_dm\_get\_rftemperature

## Function

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

## Function

void nwy\_dm\_get\_hw\_version(char \*hw\_ver, int buf\_len)

## Description

To obtain the hardware version number

## Parameter

hw\_ver: the hardware version buffer that is stored.  
buf\_len: buffer length

## Return Value

NA



## nwy\_dm\_get\_imei

Function	int nwy_dm_get_imei( char *imei, int buf_len)
Description	To obtain the device's IMEI number
Parameter	device's IMEI number
Return Value	Successful: NWY_SUCCESS Failed: the corresponding error code, see <b>nwy_common.h</b>

## 2.5.2 Message Queue

To perform the message-queue-related functions. The API definitions can be found in **nwy\_osi\_api.h**.

## nwy\_msg\_queue\_create

Function	int nwy_msg_queue_create(nwy_osi_msg_queue_t *msg_q, char *name, uint32 msg_size, uint32 msg_num)
Description	To initialize the message queue
Parameter	msg_q: the created message queue name: message queue name, optional and not supported msg_size: message size msg_num: length of the message queue
Return Value	Successful: NWY_SUCCESS Failed: another value
Remarks	Please use message queues carefully because when the message queue is used, calling <b>nwy_wait_thread_event</b> to receive events is forbidden, which may result in a program exception since most of the underlying processing in the system, such as HTTP, FTP, timer, etc., are driven by the received events. You can use an event instead of using a message queue. For the usage of event, see the reference design in 2.5.4 .  When you are designing a message queue, in order to prevent the message queue from being blocked due to too small definition, please define the size of the message queue reasonably according to size of your code.

## nwy\_msg\_queue\_delete

Function	int nwy_msg_queue_delete(nwy_osi_msg_queue_t msg_q)
Description	To delete a message queue
Parameter	msg_q: message queue

Return Value	Successful: NWY_SUCESS Failed: another value
Remarks	After using the message queue, you must delete the message queue; otherwise a memory leak will occur.

## nwy\_msg\_queue\_send

Function	int nwy_msg_queue_send(nwy_osi_msg_queue_t msg_q, uint32 size, const void *msg_ptr, uint32 timeout)
Description	To send message
Parameter	msg_q: message queue size: message length msg_ptr: message content timeout: timeout period (unit: ms) 0: always waiting. When timeout is 0xffffffff, there will be a return value immediately.
Return Value	Successful: NWY_SUCESS Failed: another value
Remarks	0: always waiting. 0xffffffff: there will be a return value immediately.

## nwy\_msg\_queue\_rcv

Function	int nwy_msg_queue_rcv(nwy_osi_msg_queue_t msg_q, uint32 size, void *msg_ptr, uint32 timeout)
Description	To receive messages
Parameter	msg_q: message queue size: message length msg_ptr: message content timeout: timeout period (unit: ms) 0: always waiting. When timeout is 0xffffffff, there will be a return value immediately.
Return Value	Successful: NWY_SUCESS Failed: another value
Remarks	Generally, creating a thread independently to receive messages is required; then, you process the message after it is received.

### nwy\_msg\_queue\_get\_pendingevent\_cnt

Function	int nwy_msg_queue_get_pendingevent_cnt(nwy_osi_msg_queue_t msg_q, uint32 *size)
Description	To obtain the number of messages stored in the queue
Parameter	msg_q: message queue size: number of the obtained messages
Return Value	Successful: NWY_SUCESS Failed: another value

### nwy\_msg\_queue\_get\_spaceevent\_cnt

Function	int nwy_msg_queue_get_spaceevent_cnt(nwy_osi_msg_queue_t msg_q, uint32 *size)
Description	To obtain the number of idle messages in the queue
Parameter	msg_q: message queue size: number of the obtained idle messages
Return Value	Successful: NWY_SUCESS Failed: another value

## 2.5.3 File Operation

To perform the message-queue-related functions. The API definitions 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: refers to <b>nwy_file_action_e</b> .
Return Value	Successful: file descriptor Failed: values <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: another value

## 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: data bytes
Return Value	Successful: data bytes Failed: another value

## nwy\_sdk\_fwrite

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

## nwy\_sdk\_fseek

Function	int nwy_sdk_fseek(int fd, int offset, nwy_fseek_offset_e mode);
Description	To set a file pointer
Parameter	fd: file descriptor offset: offset mode: refers to <b>nwy_fseek_offset_e</b> .
Return Value	Successful: offset Failed: another value

## nwy\_sdk\_fsize

Function	long nwy_sdk_fsize(const char *path);
----------	---------------------------------------

Description	To obtain the file size
Parameter	path: file path
Return Value	Successful: file size in bytes Failed: another value

## nwy\_sdk\_fsize\_fd

Function	long nwy_sdk_fsize_fd(int fd);
Description	To obtain the file size according to the file descriptor.
Parameter	fd: file descriptor
Return Value	Successful: file size in bytes Failed: -1

## nwy\_sdk\_fexist

Function	bool nwy_sdk_fexist(const char *path);
Description	To check if the file exists
Parameter	path: file path
Return Value	True: existence False: no existence

## nwy\_sdk\_get\_stat\_path

Function	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

Function	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

Function	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

Function	int nwy_sdk_ftrunc_fd(int fd, long len)
Description	To set the size of a file to a specific length based on the file descriptor. If the size of the source file is larger than the specified size, the excess part will be truncated.
Parameter	fd: file descriptor len: length of the specified file.
Return Value	Successful: 0 Failed: -1

## nwy\_sdk\_ftrunc\_path

Function	int nwy_sdk_ftrunc_path(const char *path, long len);
Description	To set the size of a file with a specific name to a specific length. If the size of the source file is larger than the specified size, the excess part will be truncated.
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: another value

## nwy\_sdk\_frename

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

## nwy\_sdk\_vfs\_opendir

Function	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

Function	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

Function	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 of the folder Failed: -1

## nwy\_sdk\_vfs\_seekdir



Function	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 folder in the directory
Return Value	NA

### nwy\_sdk\_vfs\_rewinddir

Function	void nwy_sdk_vfs_rewinddir(nwy_dir *pdir);
Description	To read the location offset to the position where the directory stream starts
Parameter	pdir: directory flow
Return Value	NA

### nwy\_sdk\_vfs\_closedir

Function	int nwy_sdk_vfs_closedir(nwy_dir *pdir)
Description	To close the folder directory stream
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: directory path
Return Value	Successful: NWY_SUCESS Failed: another value

### nwy\_sdk\_vfs\_rmdir

Function	int nwy_sdk_vfs_rmdir(const char *name);
Description	To delete a directory
Parameter	name: directory path
Return Value	Successful: NWY_SUCESS

	Failed: another value
Remarks	Non-empty folders cannot be deleted.

### nwy\_sdk\_vfs\_ls

Function	int nwy_sdk_vfs_ls(void);
Description	To retrieve the available space of the user memory.
Parameter	void
Return Value	The file system mount point

### nwy\_sdk\_vfs\_free\_size

Function	int nwy_sdk_vfs_free_size(const char *path);
Description	To retrieve the available space of the file system.
Parameter	Path: File system mounting point.
Return Value	Size of remaining space in bytes.

### nwy\_sdk\_sfile\_init

Function	int nwy_sdk_sfile_init(const char *path);
Description	To initialize the security file operation (power-off 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 read the security file
Parameter	path: file name Dst: buffer used to read data Size: size of the buffer that stores data
Return Value	Successful: actual size of the read data Failed: -1

## nwy\_sdk\_sfile\_write

Function	long nwy_sdk_sfile_write(const char *path, void *data, long size);
Description	To write data to the security file
Parameter	path: file name Dst: buffer to write data to Size: buffer size
Return Value	Successful: actual size of the written data Failed: -1

## nwy\_sdk\_sfile\_size

Function	long nwy_sdk_sfile_size(const char *path);
Description	To obtain the size of the security file
Parameter	path: file name
Return Value	Successful: file size Failed: -1

## nwy\_sdk\_fread\_path

Function	long nwy_sdk_fread_path(const char *path, void *dst, long size);
Description	To read file according to the file name
Parameter	pPath: file name Dst: buffer to write data to Size: buffer size
Return Value	Successful: actual size of the read data 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 write data to Size: buffer size
Return Value	Successful: actual size of the written data

Failed: -1
------------

## nwy\_sdk\_vfs\_rmdir\_recursive

Function	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
Remarks	To delete a non-empty file folder, use this function

## 2.5.4 Thread

To perform a thread-related operation. The API definitions can be found in **nwy\_osi\_api.h**.

### nwy\_create\_thread

Function	int nwy_create_thread(nwy_osi_thread_t *hdl, uint32 stack_size, uint8 priority, char *task_name, nwy_task_cb_func_t func, void *argv, uint32 event_count)
Description	To create a thread
Parameter	nwy_osi_thread_t *hdl: thread handle uint32 stack_size: thread stack size uint8 priority: thread priority char *task_name: thread name nwy_task_cb_func_t func: function used to process threads void *argv: parameter of the function uint32 event_count: the maximum number of events that a thread can handle
Return Value	Successful: NWY_SUCCESS Failed: another value
Remarks	During thread creation, stack_size is measured in byte, which will be converted to word internally. In the Cat.1 platform, a word is equal to 4 bytes. When setting stack_size, you need to calculate the required stack size roughly based on the size of the local variable in the entity function of the thread, and define the required stack size reasonably. If stack_size is too small, stack overflow may occur. If you need to use large variables, it is recommended that you use global variables or malloc. When using malloc, you must remember to use the free function to release the memory. event_count refers to the maximum number of events processed by the task. You need to define an appropriate event_count

as required. It is recommended that you set event\_count to 32. If event\_count is too small, the events will be full and other threads of the system may fail to run normally. As long as an event in the system is blocked, the system will fail to run normally.

## nwy\_create\_thread\_withstack

Function	int nwy_create_thread_withstack(nwy_osi_thread_t *hdl, void *stack, uint32 stack_size, uint8 priority, char *task_name, nwy_task_cb_func_t func, void *argv, uint32 event_count)
Description	To create a thread
Parameter	nwy_osi_thread_t *hdl: thread handle void *stack: address of the thread stack uint32 stack_size: thread stack size uint8 priority: thread priority char *task_name: thread name nwy_task_cb_func_t func: function used to process threads void *argv: parameter of the function uint32 event_count: the maximum number of events that a thread can handle
Return Value	Successful: NWY_SUCESS Failed: another value
Remarks	During thread creation, stack_size is measured in byte, which will be converted to word internally. In the Cat.1 platform, a word is equal to 4 bytes. When setting stack_size, you need to calculate the required stack size roughly based on the size of the local variable in the entity function of the thread, and define the required stack size reasonably. If stack_size is too small, stack overflow may occur. If you need to use large variables, it is recommended that you use global variables or malloc. When using malloc, you must remember to use the free function to release the memory. event_count refers to the maximum number of events processed by the task. You need to define an appropriate event_count as required. It is recommended that you set event_count to 32. If event_count is too small, the events will be full and other threads of the system may fail to run normally. As long as an event in the system is blocked, the system will fail to run normally.

## nwy\_get\_current\_thread

Function	int nwy_get_current_thread(nwy_osi_thread_t *hdl)
Description	To get all tasks of the current function
Parameter	nwy_osi_thread_t *hdl: thread handle

Return Value	Successful: NWY_SUCESS Failed: another value
--------------	---

### nwy\_set\_thread\_priority

Function	int nwy_set_thread_priority(nwy_osi_thread_t hdl, uint32_t new_pri, uint32_t *old_pri)
Description	To set the priority
Parameter	nwy_osi_thread_t *hdl: thread handle new_pri: the set priority uint32_t *old_pri: this parameter is not used
Return Value	Successful: NWY_SUCESS Failed: another value

### nwy\_get\_thread\_priority

Function	int nwy_get_thread_priority(nwy_osi_thread_t hdl, uint32_t *priority)
Description	To obtain the thread priority
Parameter	nwy_osi_thread_t hdl: thread handle uint32_t *priority: priority
Return Value	Successful: NWY_SUCESS Failed: another value

### nwy\_suspend\_thread

Function	int nwy_suspend_thread(nwy_osi_thread_t hdl)
Description	To suspend a task
Parameter	nwy_osi_thread_t hdl: thread handle
Return Value	Successful: NWY_SUCESS Failed: another value

### nwy\_resume\_thread

Function	void nwy_resume_thread(nwy_osiThread_t thread);
Description	To wake up a task
Parameter	nwy_osi_thread_t hdl: thread handle

Return Value	Successful: NWY_SUCESS Failed: another value
--------------	---

## nwy\_send\_thread\_event

Function	bool nwy_send_thread_event(nwy_osi_thread_t hdl, nwy_event_msg_t *event, uint32 timeout)
Description	To send an event to the task
Parameter	thread: thread used to receive events event: event Timeout: timeout period, 0 indicates the value will be returned immediately.
Return Value	Successful: true Failed: false
Remarks	Threads in the 8910 platform system is driven by events. To perform an action, send the corresponding event to the task.

## nwy\_wait\_thread\_event

Function	bool nwy_wait_thread_event(nwy_osiThread_t thread, nwy_osiEvent_t *event, uint32 timeout)
Description	Task receiving event
Parameter	thread: current event event: event timeout: timeout period; 0 indicates wait forever
Return Value	Successful: true Failed: false
Remarks	This function does not occupy CPU resources during waiting.

## nwy\_exit\_thread\_self

Function	int nwy_exit_thread_self()
Description	To exist from the current thread (Call this function in the thread processing function.)
Parameter	Void
Return Value	Successful: NWY_SUCESS Failed: another value
Remarks	When exiting a thread is required, you must execute the thread exit function. Otherwise, the module may crash.

### nwy\_get\_thread\_pendingevent\_cnt

Function	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 handle
Return Value	The number of events stored in the current thread.

### nwy\_get\_thread\_spaceevent\_cnt

Function	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

Function	void nwy_sleep(uint32 ms);
Description	Sleep delay
Parameter	sleep: delay time, unit: ms
Return Value	NA
Remarks	Part of the CPU resource will be released during the sleep process. The accuracy is ticks, and 1ms is equal to 20 ticks.

### nwy\_usleep

Function	void nwy_usleep(uint32 ms);
Description	Sleep delay
Parameter	sleep: delay time, unit: $\mu$ s
Return Value	NA

### nwy\_get\_thread\_info

Function	int nwy_get_thread_info(nwy_osi_thread_t hdl, nwy_thread_info_t *thread_info)
----------	---



Description	To obtain the thread information, including the thread status, name, and available stack size. For details, see the <code>nwy_thread_info_t</code> structure. The available stack size must be converted. <code>thread_stack_highwatermark</code> indicates the available stack size, and it is measured in word. To convert its unit to a byte, use the following formula: $(\text{thread\_stack\_highwatermark} \times 4) - 3$ .
Parameter	<b>thread</b> : thread handle <b>thread_info</b> : the obtained thread information
Return Value	Successful: <code>NWY_SUCCESS</code> Failed: non <code>NWY_SUCCESS</code>
Remarks	As stated in the description, you need to pay attention to unit conversion when using this API to obtain the available stack size.

### `nwy_get_thread_list`

Function	<code>uint32_t nwy_get_thread_list(char *thread_list)</code>
Description	To obtain the thread information in the system
Parameter	<b>thread_list</b> : thread information list
Return Value	Successful: <code>NWY_SUCCESS</code> Failed: non <code>NWY_SUCCESS</code>
Remarks	The displayed information includes the name, status, priority, available stack size, and serial number. The stack size is in words, and one word is equal to 4 bytes. It is recommended that you pass <code>thread_list</code> in 1024 bytes.

### `nwy_get_thread_runtime_stats`

Function	<code>uint32_t nwy_get_thread_runtime_stats(char *stats_buf)</code>
Description	To obtain the thread CPU utilization in the system.
Parameter	<b>stats_buf</b> : CPU information of the thread.
Return Value	Successful: <code>NWY_SUCCESS</code> Failed: non <code>NWY_SUCCESS</code>
Remarks	The displayed information includes the name, running count, and CPU utilization. It is recommended that you pass <code>stats_buf</code> in 1024 bytes.

### Sample code of Event:

```
typedef struct nwy_info
{
    int len;
    char buf[1024];
}nwy_info;
```

```
int nwy_send_event_sample()
{
    bool ret = false;
    nwy_osiEvent_t event;
    nwy_info *info;
    info = (nwy_info *)malloc(sizeof(nwy_info));
    memset(&event, 0, sizeof(event));
    event.id = EVENT_ID;
    event.param1 = info;
    ret = nwy_send_thread_event(thread, &event, 0);
    return ret;
}

int nwy_function_call(int addr)
{
    nwy_info *msg_info = NULL;
    msg_info = (nwy_info *)addr;

    //TODO

    free(addr);
}

int nwy_thread_process()
{
    nwy_osiEvent_t event;
    //receiver
    while(1)
    {
        memset(&event, 0, sizeof(event));
        nwy_wait_thread_event(thread, &event, 0);
        switch(event.id)
        {
            case EVENT_ID:
            {
                nwy_function_call(event.param1);
                break;
            }
            case EVENT_ID_2:
                .....;
            default:
                break;
        }
    }
    nwy_exit_thread();
}
```

## 2.5.5 mutex

To perform a mutex-related operation. The API definitions can be found in **nwy\_osi\_api.h**.

### nwy\_create\_mutex

Function	int nwy_create_mutex(nwy_osi_mutex_t *mutex)
Description	To initialize a mutually exclusive lock
Parameter	nwy_osi_mutex_t *mutex: the returned mutually exclusive lock
Return Value	Successful: NWY_SUCESS Failed: other values

## nwy\_lock\_mutex

Function	int nwy_lock_mutex(nwy_osi_mutex_t mutex, int time_out)
Description	To add a lock
Parameter	mutex: mutually exclusive lock timeout: time-out period, unit: ms (0: no timeout, indicating wait forever)
Return Value	Successful: NWY_SUCCESS Failed: other values

## nwy\_unlock\_mutex

Function	int nwy_unlock_mutex(nwy_osiMutex_t mutex);
Description	To unlock
Parameter	mutex: mutually exclusive lock
Return Value	Successful: NWY_SUCCESS Failed: other values

## nwy\_delete\_mutex

Function	int nwy_delete_mutex(nwy_osiMutex_t mutex);
Description	To delete a mutually exclusive lock
Parameter	mutex: mutually exclusive lock
Return Value	Successful: NWY_SUCCESS Failed: other values

## 2.5.6 pipe

To perform a pipe-related operation. The API definitions can be found in **nwy\_osi\_api.h**.

## nwy\_osi\_pipe\_create

Function	int nwy_osi_pipe_create(nwy_pipe_t *pipe_hdl,unsigned size)
Description	To initialize a pipe
Parameter	pipe_hdl: to store a created pipe handle size: pipe buffer size
Return Value	Successful: NWY_SUCCESS Failed: other values

### nwy\_osi\_pipe\_delete

Function	void nwy_osi_pipe_delete(nwy_pipe_t pipe)
Description	To delete a pipe
Parameter	pipe: the created pipe
Return Value	NA

### nwy\_osi\_pipe\_read

Function	int nwy_osi_pipe_read(nwy_pipe_t pipe, void *buf, unsigned size)
Description	To read data from pipe
Parameter	pipe: the created pipe buf: buffer used to read data size: size of the data buff
Return Value	Number of data bytes read

### nwy\_osi\_pipe\_write

Function	int nwy_osi_pipe_write(nwy_pipe_t pipe, void *buf, unsigned size)
Description	To write data to pipe
Parameter	pipe: the created pipe buf: buffer to write data to size: size of the data buff
Return Value	Number of data bytes written

## 2.5.7 Semaphores

To perform semaphore-related operations. The API definitions can be found in **nwy\_osi\_api.h**.

### nwy\_semaphore\_create

Function	int nwy_semaphore_create(nwy_osi_semaphore_t *sem, uint32 max_count, uint32 init_count)
Description	To initialize a semaphore
Parameter	sem: stores the created sem max_count: maximum count of the semaphore init_count: initial count of the semaphore

## Return Value

Successful: NWY\_SUCESS  
Failed: another value

## nwy\_semaphore\_acquire

## Function

int nwy\_semaphore\_acquire(nwy\_osi\_semaphore\_t sem, int time\_out)

## Description

To request a semaphore

## Parameter

sem: semaphore  
timeout: time-out period, unit: ms (0: wait forever)

## Return Value

Successful: NWY\_SUCESS  
Failed: another value

## nwy\_semaphore\_release

## Function

void nwy\_semaphore\_release(nwy\_osiSemaphore\_t \*sem);

## Description

To release a semaphore

## Parameter

sem: semaphore

## Return Value

Successful: NWY\_SUCESS  
Failed: another value

## nwy\_semaphore\_delete

## Function

void nwy\_semaphore\_delete(nwy\_osiSemaphore\_t \*sem);

## Description

To delete a semaphore

## Parameter

sem: semaphore

## Return Value

Successful: NWY\_SUCESS  
Failed: another value

## 2.5.8 Time

Time API definitions can be found in **nwy\_api.h** and the relevant header file is **nwy\_common.h**. These APIs are used to read and set the date.

## nwy\_set\_time

## Function

void nwy\_set\_time(nwy\_time\_t \*julian\_time, char timezone)

Description	To set a time interface
Parameter	julian_time: time (input parameter) timezone: timezone information (input parameter)
Return Value	N/A.
Remarks	The year parameter ranges from 2000 to 2100.

### nwy\_get\_time

Function	int nwy_get_time(nwy_time_t *julian_time, char *timezone)
Description	To obtain the time interface
Parameter	julian_time: time (output parameter) timezone: timezone information (output parameter)
Return Value	0: failed Successful: 1

### nwy\_updatetime\_ntp

Function	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, 1 - 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	Successful: NWY_SUCESS Failed: another value

### nwy\_get\_time\_zone\_switch

Function	int nwy_get_time_zone_switch(nwy_time_zone_switch_e *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: another value
--------------	---

### nwy\_set\_time\_zone\_switch

Function	int nwy_set_time_zone_switch(nwy_time_zone_switch_e 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: another value

### nwy\_get\_up\_time\_us

Function	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: $\mu$ s

## 2.5.9 Timer

To perform timer-related operations. The API definitions can be found in **nwy\_osi\_api.h**.

### nwy\_sdk\_timer\_create

Function	int nwy_sdk_timer_create(nwy_osi_timer_t *timer, nwy_osi_thread_t hdl, nwy_timer_cb_func_t cb, nwy_timer_cb_para_t ctx)
Description	To initialize a timer
Parameter	timer: stores the created timer hdl: thread used to process timer messages cb: callback function of the timer; the cb should be consistent with the cb in nwy_sdk_timer_start ctx: context of the callback function
Return Value	Successful: NWY_SUCESS Failed: another value
Remarks	The timer needs to call nwy_wait_thead_event in the entity processing of the

thread it creates. Otherwise the callback function cannot be triggered after the timer expires. This is because the timer is implemented through the event, and the timer will send a system event to the thread after the timer expires. If `nwy_wait_thread_event` is not called, the thread cannot receive the timer event and therefore cannot call the callback function.

## nwy\_sdk\_timer\_create\_ex

Function	<code>int nwy_sdk_timer_create_ex(nwy_osi_timer_t *timer, nwy_timer_para_t *para);</code>
Description	To initialize a timer
Parameter	<p>timer: stores the created timer  para: the parameter struct required to initialize the timer</p> <pre>typedef struct nwy_timer_para_t {     int expired_time;     nwy_timer_type_e type;     nwy_osi_thread_t thread_hdl;     nwy_timer_cb_func_t cb;     nwy_timer_cb_para_t cb_para; }nwy_timer_para_t;</pre>
Return Value	<p>Successful: <code>NWY_SUCCESS</code>  Failed: another value</p>
Remarks	The timer needs to call <code>nwy_wait_thread_event</code> in the entity processing of the thread it creates. Otherwise the callback function cannot be triggered after the timer expires. This is because the timer is implemented through the event, and the timer will send a system event to the thread after the timer expires. If <code>nwy_wait_thread_event</code> is not called, the thread cannot receive the timer event and therefore cannot call the callback function.

## nwy\_sdk\_timer\_destroy

Function	<code>int nwy_sdk_timer_destroy(nwy_osi_timer_t timer)</code>
Description	To delete a timer
Parameter	timer
Return Value	<p>Successful: <code>NWY_SUCCESS</code>  Failed: another value</p>
Remarks	You must call this API after using the timer to prevent a memory leak.

## nwy\_sdk\_timer\_start



Function	<code>int nwy_sdk_timer_start(nwy_osi_timer_t timer, uint32 experi_time, nwy_timer_cb_func_t cb, nwy_timer_cb_para_t ctx, nwy_timer_type_e type)</code>
Description	To start a timer
Parameter	timer timeout: time-out period (unit: ms) cb: callback function of the timer; the cb should be consistent with the cb in <code>nwy_sdk_timer_start</code> ctx: context of the callback function type: timer execution type; start once or periodically
Return Value	Successful: true Failed: false

### `nwy_sdk_timer_start_ex`

Function	<code>int nwy_sdk_timer_start_ex(nwy_osi_timer_t timer, nwy_timer_para_t *para);</code>
Description	To start a timer
Parameter	timer para: the parameter struct required to initialize the timer typedef struct nwy_timer_para_t { int expired_time; nwy_timer_type_e type; nwy_osi_thread_t thread_hdl; nwy_timer_cb_func_t cb; nwy_timer_cb_para_t cb_para; }nwy_timer_para_t;
Return Value	Successful: true Failed: false

### `nwy_sdk_timer_stop`

Function	<code>int nwy_sdk_timer_stop(nwy_osi_timer_t timer)</code>
Description	To stop a timer
Parameter	timer
Return Value	Successful: true Failed: false