neoway

Get connected Get smart

N723 OpenCPU

API Notes

Issue 1.0 Date 2022-09-24

Notice

This document provides guide for users to use N723 OpenCPU.

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

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

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

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

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

Sales@neoway.com

Support@neoway.com

Website: http://www.neoway.com

Contents

About This Document

Scope

This document is applicable to N723 OpenCPU.

Audience

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

Change History

Issue	Date	Change	Changed By
1.0	2022-02	Initial draft	Xue Gang1

Conventions

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

1 About N723 OpenCPU

1.1 Overview

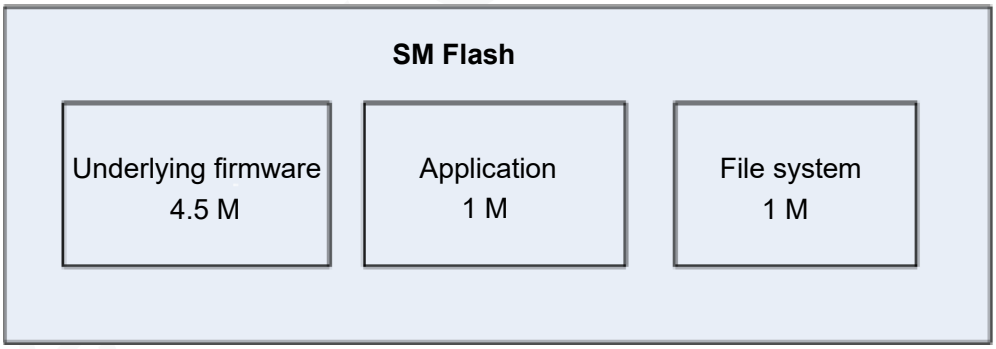
N723 OpenCPU that runs FreeRTOS can provide the following resources:

- ARM Cortex-A5 processor, 500 MHz main frequency
- Supports programming basic on C
- Flash space

  RAM: 1 MB ROM: 1 MB FS: 1 MB
- Supports network modes: GSM/GPRS&LTE, Cat.4
- Supports USB2.0/UART/GPIO

1.2 Flash Space

Figure 1-1 Flash Space



- 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, and generate image files (executable files).
- Operating method

Flash the image file to the flash of the module using SWDownloader and set the file to auto-start upon power-on.

1.3 Basic Functional Modules

- Data dial-up
- Network mode selection, network registration, network status querying
- SMS
- SIM card services
- FOTA
- I/O interfaces including GPIO, UART, and so on
- Virtual AT

1.4 Block Diagram of the Development Process

Process description

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

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

2 APIs

2.1 Power Management

To enable the system to enter sleep mode or wake up from sleep mode, detect the power supply status, and control shutdown or restart of the module.

PM API definitions can be found in **nwy_pm.h**.

nwy_power_off

Format	int nwy_power_off(int option)
Description	To perform shutdown or restart in a normal process according to <option>
Parameter	option: 1: fast shutdown 2: normal startup 3: restart
Return Value	Successful: **NWY_SUCESS** Failed: NWY_ERROR

2.2 I/O Interfaces

2.2.1 UART

To configure UART parameters, and perform UART data read and write function.

UART API definitions can be found in **nwy_uart.h**.

nwy_uart_init

Format	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

Format	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 value
Return Value	Successful: 0 Failed: -1

nwy_uart_get_baud

Format	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: 0 Failed: -1

nwy_uart_set_para

Format	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: 0 Failed: -1

nwy_uart_get_para

Format	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: 0 Failed: -1

nwy_uart_send_data

Format	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_recv_cb

Format	int nwy_uart_reg_recv_cb(uint8_t hd,nwy_uart_recv_callback_t recv_cb);
Description	To receive data via UART
Parameter	recv_cb: receiving callback function
Return Value	Successful: 0 Failed: -1

nwy_uart_deinit

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

2.2.2 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

Format	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

Format	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

Format	int nwy_gpio_set_direction(uint32 gpio_id, nwy_dir_mode_t dirct)
Description	To set the GPIO direction
Parameter	gpio_id: GPIO ID dirct: GPIO direction
Return Value	Successful: NWY_SUCCESS Failed: NWY_ERROR

nwy_gpio_set_value

Format	int nwy_gpio_set_value(uint32 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

Format	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

Format	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

Format	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

Format	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

Format	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

2.3 Service

2.3.1 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

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

nwy_data_release_srv_handle

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

nwy_data_start_call

Format	int nwy_data_start_call(inthndl, nwy_data_start_call_v02_t *param)
Description	To start a 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**nwy_data_get_srv_handle()**.
Parameter	hndl: The specified resource handle of the dial-up connection, obtained from **nwy_data_get_srv_handle()**. para: dial-up parameter
Return Value	Successful: 0 Failed: a value <0

nwy_data_stop_call

Format	int nwy_data_stop_call(inthndl)
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 **nwy_data_get_srv_handle()**.
Parameter	hnd: The specified resource handle of the dial-up connection, obtained from **nwy_data_get_srv_handle()**.
Return Value	Successful: 0 Failed: a value <0

nwy_data_get_ip_addr

Format	int nwy_data_get_ip_addr(inthndl, 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 **nwy_data_get_srv_handle()**. 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 imformation, see the structure definition in **nwy_data_addr_t_info**. len: Number of addresses obtained
Return Value	Successful: 0 Failed: a value <0

nwy_data_set_profile

Format	int nwy_data_set_profile(intprofile_idx, nwy_data_profile_type_tprofile_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 **nwy_data_profile_info_t**.
Return Value	Successful: 0 Failed: a value <0

nwy_data_get_profile

Format	int nwy_data_get_profile(intprofile_idx, nwy_data_profile_type_tprofile_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

Format	char* nwy_ip4addr_ntoa(const nwy_ip4_addr_t *addr)
Description	To switch the IPv4 address format
Parameter	addr: IPv4 address
Return Value	IPv6 address

nwy_ip6addr_ntoa

Format	char *nwy_ip6addr_ntoa(const nwy_ip6_addr_t *addr)
Description	To switch the IPv6 address format
Parameter	addr: IPv4 address
Return Value	IPv6 address

2.3.2 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_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 module
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_RES_OK** Failed: the relevant error code, see **nwy_error.h**.

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 module
Parameter	sim_id: specifies the SIM card ID. Please set to 0 iccid_buf: buffer used to return ICCID, ensure that the buffer length meets the length requirement. buf_len: iccid_buf length
Return Value	Successful: **NWY_RES_OK** Failed: the relevant error code, see **nwy_error.h**.

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 status
Parameter	sim_id: specifies the SIM card ID. Please set to 0
Return Value	Successful: SIM card status; see **nwy_sim_status_t**. Failed: NWY_SIM_UNKNOWN

nwy_sim_verify_pin

Function	int nwy_sim_verify_pin(nwy_sim_id_t sim_id, const char* pin)
Description	To verify PIN code When the obtained UICC status is NWY_SIN_PIN_REQ, this function is required to verify the PIN code.
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_RES_OK** Failed: a value < 0; for details see **nwy_error.h**.

nwy_sim_unblock

Function	int nwy_sim_unblock (nwy_sim_id_t sim_id, const char* puk,const char* new_pin)
Description	To enter PUK When the SIM card is locked by PIN code, that is, the UICC status is NWY_SIM_PUK_REQ. It is required to unlock.
Parameter	sim_id: Specifies an ID number of the card to be operated. puk: character string, the length should comply with 3GPP specifications. new_pin: new PIN code. Character string, the length should comply with 3GPP specifications.
Return Value	Successful: **NWY_RES_OK** Failed: a value < 0; for details see **nwy_error.h**.

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 status of current PIN
Parameter	sim_id: Specifies an ID number of the card to be operated.
Return Value	The current PIN status; for details, see **nwy_sim_pin_mode_t**.

nwy_sim_enable_pin

Function	int nwy_sim_enable_pin(nwy_sim_id_t sim_id, const char* pin)
Description	To enable PIN The function can be called only when the PIN status is **NWY_SIM_PIN_MODE_DISABLED**.
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_RES_OK** Failed: a value < 0; for details see **nwy_error.h**.

nwy_sim_disable_pin

Function	int nwy_sim_disable_pin(nwy_sim_id_t sim_id, const char* pin)
Description	To disable PIN The function can be called only when the PIN status is **NWY_SIM_PIN_MODE_ENABLED**.
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_RES_OK** Failed: a value < 0; for details see **nwy_error.h**.

nwy_sim_change_pin

Function	int nwy_sim_change_pin (nwy_sim_id_t sim_id, const char* old_pin, const char* old_pin)
Description	To modify PIN code This function can be invoked only when the PIN code status is **NWY_SIM_PIN_MODE_ENABLED**.
Parameter	sim_id: Specifies an ID number of the card to be operated. old_pin: old PIN code. character string, the length should comply with 3GPP specifications. new_pin: new PIN code. character string, the length should comply with 3GPP specifications.
Return Value	Successful: **NWY_RES_OK** Failed: a value < 0; for details see **nwy_error.h**.

nwy_sim_get_pin_puk_retry_times

Function	int nwy_sim_get_pin_puk_retry_times (nwy_sim_id_t sim_id, uint8* pin_times, uint8* puk_times)
Description	To obtain the remaining attempts of PIN and PUK
Parameter	sim_id: Specifies an ID number of the card to be operated. pin_times: remaining attempts of PIN puk_times: remaining attempts of PUK
Return Value	Successful: **NWY_RES_OK** Failed: a value < 0; for details see **nwy_error.h**.

nwy_sim_get_msisdn

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

2.3.3 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

Format	nwy_error_e nwy_init_sms_option()
Description	To initialize the SMS parameters
Parameter	N/A.
Return Value	Successful: NWY_SMS_SUCCESS Failed: an enumerated value

nwy_sms_set_storage

Format	nwy_error_e 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: NWY_SMS_SUCCESS Failed: an enumerated value

nwy_sms_get_storage

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

nwy_set_report_option

Format	nwy_error_e 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 to 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: NWY_SMS_SUCCESS Failed: an enumerated value

nwy_sms_set_sca

Format	nwy_error_e 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: NWY_SMS_SUCCESS Failed: an enumerated value
Remarks	Before setting, confirm with the local operator to ensure that the SMS center number is correct.

nwy_sms_get_sca

Format	nwy_error_e nwy_sms_get_sca(nwy_sms_result_type *sca)
Description	To obtain the SMSC number.
Parameter	nwy_sms_result_type *sca
Return Value	Successful: NWY_SMS_SUCCESS Failed: an enumerated value

nwy_sms_send_message

Format	nwy_error_e 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: messgae length msg_contxet: message content msg_format: coding format 0: GSM7 2: UNICODE For details, see **nwy_sms.h**.
Return Value	Successful: NWY_SMS_SUCCESS Failed: an enumerated value
Remarks	Returning value only indicates the function execution is successful not data sent successful.

nwy_sms_recv_message

Format	void nwy_sms_recv_message(nwy_sms_recv_info_type_t *sms_data)

Description	To receive short messages
Parameter	nwy_sms_recv_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 **nwy_sms.h**.
Return Value	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_delete_message

Format	nwy_error_e nwy_sms_delete_message(uint16_t nindex, nwy_sms_storage_type_e nStorage)
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: NWY_SMS_SUCCESS Failed: an enumerated value

nwy_sms_read_message

Format	nwy_error_e nwy_sms_read_message(unsigned nindex, nwy_sms_recv_info_type_t *sms_data);
Description	To read a message
Parameter	nIndex: message index number sms_data: message content
Return Value	Successful: NWY_SMS_SUCCESS Failed: an enumerated value

nwy_sms_send_pdu_message

Format	nwy_error_e nwy_sms_send_pdu_message(char *data, int data_len);
Description	To send a message in PDU mode
Parameter	data: PDU data data_len: PDU data length
Return Value	Successful: NWY_SMS_SUCCESS Failed: an enumerated value

nwy_sms_delete_message_by_type

Format	nwy_error_e nwy_sms_delete_message_by_type(nwy_sms_msg_dflag_e delflag, nwy_sms_storage_type_e nStorage)
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 nStorage: message storage location
Return Value	Successful: NWY_SMS_SUCCESS Failed: an enumerated value

nwy_sms_read_message_list

Format	nwy_error_e nwy_sms_read_message_list(nwy_sms_msg_list_t *smsinfo)
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: NWY_SUCCESS Failed: a negative value
Remarks	N/A.

2.3.4 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.

nwy_nw_get_register_info

Format	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 **nwy_network.h**.
Return Value	Successful: 0 Failed: an error code

nwy_nw_get_network_mode

Format	int nwy_nw_get_network_mode(nwy_nw_mode_type_t *p_mode)
Description	To obtain the current network mode
Parameter	p_mode: network registration information, for details, see the definition in **nwy_network.h**.
Return Value	Successful: 0 Failed: an error code

nwy_nw_set_network_mode

Format	int nwy_nw_set_network_mode(nwy_nw_mode_type_t mode)
Description	To set the current network mode
Parameter	mode: Network information, for the definition, see **nwy_network.h**.
Return Value	Successful: 0 Failed: an error code

nwy_nw_get_operator_name

Format	int nwy_nw_get_operator_name(nwy_nw_operator_name_t *opt_name)
Description	To obtain the operator's information for the currently registered network, including Long EONS (Enhanced Operator Name String), short EONS, MCC, and MNC. For details, see the definition in the **nwy_nw_operator_name_t** data structure.
Parameter	opt_name: Operator information
Return Value	Successful: 0 Failed: an error code

nwy_nw_get_signal_csq

Format	int nwy_nw_get_signal_csq(uint8_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 Failed: an error code

nwy_nw_register_callback_fnuc

Format	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 Failed: an error code

nwy_nw_unregister_callback_fnuc

Format	int nwy_nw_unregister_callback_fnuc()
Description	To de-register the callback function of user-defined networks
Parameter	N/A.
Return Value	Successful: 0 Failed: an error code

nwy_cb_func

Format	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 **nwy_network.h**. ind_struct: structure of the message to be reported
Return Value	Successful: 0 Failed: an error code

nwy_nw_get_forbidden_plmn

Format	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 Failed: an error code

nwy_nw_manual_network_scan

Format	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 Failed: an error code

nwy_nw_manual_network_select

Format	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 Failed: an error code

nwy_nw_band_lock

Format	int nwy_nw_band_lock(nwy_open_net_type net, const uint32 *set_band)
Description	To lock a single frequency or multiple frequencies
Parameter	net: GSM, WCDMA, LTE set_band: band
Return Value	Successful: 0 Failed: an error code

nwy_nw_freq_lock

Format	int nwy_nw_freq_lock(nwy_open_freq_t *freq_info, int n)
Description	To lock single or multiple frequencies, 9 frequencies can be locked simultaneously at most.
Parameter	freq_info: information of the frequency to be locked n: number of frequencies to be locked
Return Value	Successful: 0 Failed: an error code

nwy_nw_get_signal_rssi

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

nwy_nw_get_radio_st

Format	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 Failed: an error code

nwy_nw_set_radio_st

Format	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 Failed: an error code

nwy_nw_get_lacid

Format	int nwy_nw_get_lacid(char *lac,char *cid)
Description	To obtain current network's LAC and CELLID
Parameter	lac: current location area code cid: cell ID of the current network
Return Value	Successful: 0 Failed: an error code

2.3.5 FOTA

To enable a network-connected device to perform app firmware upgrades
The API definitions can be founded in **nwy_fota.h**.

nwy_fota_dm

Format	Int nwy_fota_dm(ota_package_t *ota_pack);
Description	To write an app upgrade package into Flash
Parameter	ota_pack: a structure that indicates the pointer, offset and size of written data. Writing data in sub-packaging method is supported.
Return Value	Successful: 0 Failed: an error code

nwy_package_checksum

Format	int nwy_package_checksum(void);
Description	To verify the app upgrade package after the package is written.
Parameter	N/A.
Return Value	Successful: 0 Failed: an error code

nwy_fota_ua

Format	int nwy_fota_ua (void);
Description	To set the app upgrade verification flag; after the setting, the module will restart automatically.
Parameter	N/A.

Return Value	Successful: 0 Failed: an error code

nwy_get_fota_result

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

2.3.6 Socket

Socket API definitions can be found in **nwy_open_socket.h**.

nwy_sdk_socket_open

Format	int nwy_sdk_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_sdk_socket_send

Format	int nwy_sdk_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 **nwy_open_socket.h**.)

nwy_sdk_socket_recv

Format	int nwy_sdk_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 **nwy_open_socket.h**.)

nwy_sdk_socket_sendto

Format	int nwy_sdk_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 **nwy_open_socket.h**.)

nwy_sdk_socket_recvfrom

Format	int nwy_sdk_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 **nwy_open_socket.h**.)

nwy_sdk_socket_setsockopt

Format	int nwy_sdk_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_sdk_socket_getsockopt

Format	int nwy_sdk_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_sdk_gethostbyname

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

nwy_sdk_gethostbyname1

Format	char* nwy_sdk_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_sdk_socket_close

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

nwy_sdk_socket_connect

Format	int nwy_sdk_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_sdk_socket_bind_lport

Format	int nwy_sdk_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_sdk_socket_bind

Format	int nwy_sdk_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_sdk_socket_listen

Format	int nwy_sdk_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_sdk_socket_accept

Format	int nwy_sdk_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_sdk_socket_select

Format	int nwy_sdk_socket_select(int maxfdp1, fd_set *readset, fd_set *writeset, 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

	writeset: to monitor the write status of file descriptors; a value greater than 0 indicates there are writable files. exceptset: to monitor file abnormalities timeout: timeout period of the select function • NULL: NULL: the select function blocks • 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. • A value greater than 0: timeout period If an event occurs within the period, a value greater than 0 is returned. A value must be returned when the timeout period is over.
Return Value	• The select function encounters abnormalities: a value lower than 0 is returned. • Some files can be read or written: a value greater than 0 is returned. • The function times out and no files can be read or written, or the files are wrong: 0 is returned.
Remarks	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.

nwy_sdk_socket_shutdown

Format	int nwy_sdk_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_sdk_get_default_netif_v4_addr

Format	int nwy_sdk_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

2.3.7 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

Format	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 **nwy_ftp.h**.
Return Value	Successful: 0 Failed: -1

nwy_ftp_get

Format	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 **nwy_ftp.h**.
Return Value	Successful: 0 Failed: -1

nwy_ftp_put

Format	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 nwy_ftp.h.
Return Value	Successful: 0 Failed: -1

nwy_ftp_filesize

Format	int nwy_ftp_filesize(uint16_t channel,const char* filename, uint16 tout)
Description	To query the size of the file in the FTP server

Parameter	uint16_t channel,const char* filename, uint16 tout, see **nwy_ftp.h**.
Return Value	Successful: 0 Failed: -1

nwy_ftp_logout

Format	int nwy_ftp_logout(uint16_t channel,uint16 tout)
Description	To log out from the FTP server
Parameter	uint16 tout time-out period
Return Value	Successful: 0 Failed: -1

2.3.8 HTTP/HTTPS

This section introduces the HTTP/HTTPS-related APIs.

nwy_http_setup

Format	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 1 to 7. const char *url: destination path int port: destination port number httpresultcb cb: callback function of the event notification, for details, see nwy_http.h.
Return Value	Successful: 0 Failed: -1

nwy_https_setup

Format	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 1 to 7. const char *url: destination path int port: destination port number httpresultcb cb: callback function of the event notification, for details, see

	nwy_http.h. nwy_app_ssl_conf_t *ssl_cfg SSL: for the specific definition of configuration information, see nwy_http.h.
Return Value	Successful: 0 Failed: -1

nwy_http_get

Format	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 1 to 7. uint8_t keepalive: whether it is a long connection. int offset: specifies the starting position of the download. int size: specifies the length of the data downloaded. boolean is_https: true https: false For details, see nwy_http.h.
Return Value	Successful: 0 Failed: -1

nwy_http_head

Format	int nwy_http_head(uint16_t channel, uint8_t keepalive, boolean is_https)
Description	To initiate a HEAD request.
Parameter	uint16_t channel: channel number ranges from 1 to 7. uint8_t keepalive: whether it is a long connection. boolean is_https: true https: false For details, see nwy_http.h.
Return Value	Successful: 0 Failed: -1

nwy_http_post

Format	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 1 to 7.

	uint8_t keepalive: whether it is a long connection. uint8_t type: message type const char* data: post data. int len: data length boolean is_https: true https: false For details, see nwy_http.h.
Return Value	Successful: 0 Failed: -1

nwy_http_close

Format	int nwy_http_close(uint16_t channel, boolean is_https)
Description	To proactively close the connection.
Parameter	uint16_t channel: channel number ranges from 1 to 7. boolean is_https: true https: false
Return Value	Successful: 0 Failed: -1

nwy_cert_add

Format	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 /nwy/ directory. const char *data: certificate content int length: certificate length For details, see nwy_http.h.
Return Value	Successful: 0 Failed: -1

nwy_cert_check

Format	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

Format	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 System

2.4.1 Device Management

This section introduces the device management APIs.

The API definitions can be found in **nwy_common.h** and **nwy_dm.h**.

nwy_dm_get_dev_model

Format	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

Format	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

Format	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_imei

Function	int nwy_dm_get_imei(char* imei_buf, size_t buf_len);
Description	To obtain IMEI of the device
Parameter	imsi_buf: buffer used to return IMEI, ensure that the buffer length is greater than 16 bytes. buf_len: imsi_buf length
Return Value	Successful: **NWY_RES_OK** Failed: the relevant error code, see **nwy_error.h**.

nwy_dm_get_hw_version

Format	int nwy_dm_get_hw_version(char *version_buf, int buf_len)
Description	To obatin the hardware version number
Parameter	version_buf: buffer used to return IMEI, ensure that the buffer length is greater than 16 bytes. buf_len: version_buf length.
Return Value	NA

nwy_dm_set_app_version

Format	void nwy_dm_set_app_version(char * version_buf, int buf_len)
Description	To obtain the hardware version number After the app version number is set, you can execute AT+NAPPCHECK? to query the app version number
Parameter	version_buf: app version number content, ensure that the buffer length is greater than 128 bytes. buf_len: version_buf length
Return Value	NA

2.4.2 Message Queue

To perform the message-queue-related functions. The API definitions can be found in **nwy_osi_api.h**.

nwy_msg_queue_create

Format	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_num: number of message queues msg_size: message size
Return Value	Failed: NULL

nwy_msg_queue_delete

Format	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 enumerated value
Remarks	After using the message queue, you must delete the message queue; otherwise, a memory leak will occur.

nwy_msg_queue_send

Format	int nwy_msg_queue_send(nwy_osi_msg_queue_t msg_q,uint32 size,void* msg_ptr,nwy_timeout_type_e timeout)
Description	To send message
Parameter	msg_q: message queue size: message length msg_ptr: message content timeout: timeout setting
Return Value	Successful: **NWY_SUCESS** Failed: another enumerated value

nwy_msg_queue_recv

Format	int nwy_msg_queue_recv(nwy_osi_msg_queue_t msg_q,uint32 size,void*

	msg_ptr,nwy_timeout_type_e timeout)
Description	To receive messages
Parameter	msg_q: message queue size: message length msg_ptr: message content timeout: timeout setting
Return Value	Successful: **NWY_SUCESS** Failed: another enumerated value

nwy_msg_queue_get_pendingevent_cnt

Format	uint32_t 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 returned messages
Return Value	Successful: **NWY_SUCESS** Failed: another enumerated value

2.4.3 File Operation

To perform the message-queue-related functions. The API definitions can be found in **nwy_file.h**.

nwy_sdk_fopen

Format	int nwy_sdk_fopen(const char *path, nwy_file_action_e flags);
Description	To open a file
Parameter	* path: file path flags: refers to **nwy_file_action_e**.
Return Value	Successful: file descriptor Failed: values <0

nwy_sdk_fclose

Format	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

Format	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

Format	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

Format	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 **nwy_fseek_offset_e**.
Return Value	Successful: offset Failed: another value

nwy_sdk_fsize

Format	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

Format	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

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

nwy_sdk_file_unlink

Format	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

Format	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_mkdir

Format	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

Format	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.

2.4.4 Thread

To perform a thread-related operation. The API definitions can be found in **nwy_osi_api.h**.

nwy_create_thread

Format	int *nwy_create_thread(nwy_osi_thread_t *hdl,void *stack,uint32 stack_size,uint8 priority,char *task_name, nwy_task_cb_func cb,void *argv);
Description	To create a thread
Parameter	NA
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_get_current_thread

Format	nwy_osiThread_t *nwy_get_current_thread(nwy_osi_thread_t *hdl);
Description	To get all tasks of the current function
Parameter	NA
Return Value	Successful: NWY_SUCESS

	Failed: another value

nwy_get_thread_priority

Format	int nwy_get_thread_priority(nwy_osi_thread_t hdl, uint8 *priority);
Description	To obtain the priority
Parameter	hdl: thread Priority: priority
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_set_thread_priority

Format	int nwy_set_thread_priority(nwy_osi_thread_t hdl, uint8 new_pri,uint8 *old_pri);
Description	To set the thread priority
Parameter	hdl: thread new_pri: new priority old_pri: prioty set before
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_suspend_thread

Format	void nwy_suspend_thread(nwy_osi_thread_t hdl);
Description	To suspend a task
Parameter	hdl: thread
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_resume_thread

Format	void nwy_resume_thread(nwy_osi_thread_t hdl);
Description	To wake up a task
Parameter	hdl: thread
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_exit_thread

Format	int nwy_exit_thread(nwy_osi_thread_t hdl);
Description	To exit a thread
Parameter	hdl: thread
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_sleep

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

nwy_usleep

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

2.4.5 mutex

To perform a mutex-related operation. The API definitions can be found in **nwy_osi_api.h**.

nwy_create_mutex

Format	int nwy_create_mutex(nwy_osi_mutex_t * mutex);
Description	To initialize a mutually exclusive lock
Parameter	Void
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_lock_mutex

Format	int nwy_lock_mutex(nwy_osi_mutex_t mutex, int time_out);
Description	To add a lock
Parameter	mutex: mutually exclusive lock time_out: time-out period, unit: ms (0: wait forever)
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_unlock_mutex

Format	int nwy_unlock_mutex(nwy_osi_mutex_t mutex);
Description	To unlock
Parameter	mutex: mutually exclusive lock
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_delete_mutex

Format	int nwy_delete_mutex(nwy_osi_mutex_t mutex);
Description	To delete a mutually exclusive lock
Parameter	mutex: mutually exclusive lock
Return Value	Successful: NWY_SUCESS Failed: another value

2.4.6 Semaphores

To perform semaphore-related operations. The API definitions can be found in **nwy_osi_api.h**.

nwy_semaphore_create

Format	int nwy_semaphore_create(nwy_osi_semaphore_t *sem,uint32 sem_count);
Description	To initialize a semaphore
Parameter	sem: semaphore handle sem_count: initial count of the semaphore
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_semaphore_acquire

Format	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_semahpore_release

Format	int nwy_semahpore_release(nwy_osi_semaphore_t sem);
Description	To release a semaphore
Parameter	sem: semaphore
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_semahpore_delete

Format	int nwy_semahpore_delete(nwy_osi_semaphore_t sem);
Description	To delete a semaphore
Parameter	sem: semaphore
Return Value	Successful: NWY_SUCESS Failed: another value

2.4.7 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

Format	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	Set year in the range of 2000-2100.

nwy_get_time

Format	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	Successful: NWY_SUCESS Failed: another value

nwy_get_ms

Format	int64 nwy_get_ms()
Description	To obtain the current time in units of ms
Parameter	N/A
Return Value	time in units of ms

2.4.8 Timer

To perform timer-related operations. The API definitions can be found in **nwy_osi_api.h**.

nwy_timer_create

Format	int nwy_timer_create(nwy_osi_timer_t *timer)
Description	To initialize a timer
Parameter	*timer: timer handle
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_timer_destory

Format	int nwy_timer_destory((nwy_osi_timer_t *timer)
Description	To delete a timer
Parameter	timer

Return Value	Successful: NWY_SUCESS Failed: another value
Remarks	You must call this API after using the timer to prevent a memory leak.

nwy_timer_start

Format	int nwy_timer_start(nwy_osi_timer_t timer,uint32 expert_time,nwy_timer_cb_func_t cb,uint32 timerArgc,nwy_timer_type_e type)
Description	One-shot timer
Parameter	timer: timer expert: time-out period (unit: ms)
Return Value	Successful: NWY_SUCESS Failed: another value

nwy_timer_stop

Format	bool nwy_timer_stop(nwy_osi_timer_t timer)
Description	To stop a timer
Parameter	timer
Return Value	Successful: NWY_SUCESS Failed: another value