SG Series

H2 API PROGRAMMING GUIDE

HAROGIC

Content

1. Version Information **7**

2. Overview **8**

3. Device and API version **9**

4. Overview of Function Categories **10**

5. API Calling Flow **11**

5.1 CW Mode 11

5.2 Real-Time Streaming Mode 11

5.3 Playback Mode 11

6. Device and System Function **12**

6.1 device_list_usb 12

6.2 device_open_usb 13

6.3 device_open_eth 14

6.4 device_close 15

6.5 device_preset 16

6.6 device_query_temperature 16

6.7 device_query_state 17

6.8 device_config_fan 17

6.9 device_query_supply 18

6.10 device_config_gnss 19
6.11 device_query_gnss 19
6.12 device_query_gnss_info 20
6.13 device_query_apiverion 20
6.14 device_config_clock 21
6.15 device_query_clock 22
6.16 device_reset_clock_monitor 22
6.17 device_epoch_to_utc 23
7. Channel Setting Function 25
7.1 channel_preset 25
7.2 channel_start 25
7.3 channel_stop 26
7.4 channel_trigger_bus 26
7.5 channel_config_trigger 27
7.6 channel_query_trigger 27
7.7 channel_config_trigger_out 28
7.8 channel_query_trigger_out 29
7.9 channel_config_lo_mode 30
7.10 channel_query_lo_mode 31
8. Transmission Setting Functions 32
8.1 tx_clear_waveform 32
8.2 tx_download_waveform 32
8.3 tx_config_ffm 33
8.4 tx_query_ffm 33
8.5 tx_config_fscan 34
8.6 tx_query_fscan 35
8.7 tx_config_lscan 36
8.8 tx_query_lscan 38
8.9 tx_config_output 38

8.10 tx_query_output 39
8.11 tx_config_cw 40
8.12 tx_config_stream 41
8.13 tx_send_stream 41
8.14 tx_config_playback 43
9. Modulation Configuration Functions 46
9.1 mod_open 46
9.2 mod_close 46
9.3 mod_init_digitalmod 47
9.4 mod_init_am 47
9.5 mod_init_fm 48
9.6 mod_init_dsss 48
9.7 mod_init_pulse 49
9.8 mod_init_multitone 50
9.9 mod_init_stepsweep 50
9.10 mod_init_rampsweep 51
9.11 mod_init_ofdm 51
9.12 mod_init_awgn 52
9.13 mod_config_digitalmod 52
9.14 mod_config_am 53
9.15 mod_config_fm 54
9.16 mod_config_dsss 54
9.17 mod_config_pulse 55
9.18 mod_config_multitone 56
9.19 mod_config_stepsweep 56
9.20 mod_config_rampsweep 57
9.21 mod_config_ofdm 58
9.22 mod_config_awgn 58
9.23 mod_generate_stream 59

10. Structure Variables 63

10.1 device_info 63

10.2 supply_info 63

10.3 eth_setting 63

10.4 gnss_setting 64

10.5 gnss_info 64

10.6 channel 64

10.7 tx_trigger 65

10.8 trigger_out 65

10.9 digitalmod_profile 66

10.10 am_profile 66

10.11 fm_profile 66

10.12 dsss_profile 67

10.13 pulse_profile 67

10.14 multitone_profile 67

10.15 stepsweep_profile 68

10.16 rampsweep_profile 68

10.17 ofdm_profile 68

10.18 awgn_profile 69

11. Enumeration Variables 70

11.1 state 70

11.2 fan_mode 70

11.3 eth_interface_type 70

11.4 referenceclock_source 70

11.5 lomode 70

11.6 channel_type 70

11.7 trigger_action 70

11.8 trigger_edge 71

11.9 trigger_source 71

11.10 mod_digitalmod_type 71

11.11 mod_filter_type 72

11.12 mod_shape_type 72

Appendix 1: API Return Value Index 73

1. Version Information

Version Update Description Table

Version	Description	Date
V2.0.18	Added: Modulation Configuration Functions section.	05/22/2026
V2.0.17	Add: device_query_state function.	04/30/2026
V2.0.12	Add: device_open_eth, device_config_fan, device_config_clock, device_query_clock, device_reset_clock_monitor, channel_config_trigger_out, channel_query_trigger_out, channel_config_lo_mode, channel_query_lo_mode, tx_config_lscan and tx_query_lscan function.	04/20/2026
V2.0.10	Add: device_list_usb, device_query_supply, device_config_gnss, device_query_gnss, device_query_gnss_info, device_epoch_to_utc, tx_config_fscan, tx_query_fscan and tx_config_stream function.	03/18/2026
V1.0	Initial version.	01/10/2026

2. Overview

This API system is a dynamic link library developed in C, used for programming and controlling the device. The signal generator mainly operates in the following three modes:

Continuous Wave Mode	In this mode, the device transmits a continuous wave, and no data needs to be sent from the user to the device.
Real-Time Streaming Mode	In this mode, the user can send IQ waveform data to the device via the USB interface. The device receives and plays the data in real time. When the user stops sending data, the device will finish playing all received data before stopping. The real-time sampling rate (up to 62.5 MHz) and data throughput are limited by the physical bandwidth of the data interface, while the total data size is not limited.
Playback Mode	This mode uses a pre-stored waveform playback mechanism. The user can first download IQ waveform data into the device memory, and then achieve multi-waveform playback by specifying the waveform index, number of playback loops, sampling rate, and the number of waveforms to be played. Since all waveform data is pre-stored, maximum analog bandwidth output can be achieved without relying on high-speed real-time data transmission. Note that the total amount of storable waveform data is limited by the device memory capacity (up to 128 MB).

3. Device and API version

The system operates through the coordinated execution of multiple software and firmware components. In this system, the involved components include: the main control firmware (MFW/MCU), FPGA firmware (FFW), bus firmware (Bus), and the application programming interface (API).

Version Matching Principle

To ensure stable system operation, the versions of the above components must strictly follow the baseline version alignment principle. Please refer to the table below for version verification and deployment.

Table 1 Software and Firmware Baseline Version Mapping Table

API	FPGA	MCU	Bus
2.0.18	2.0.12	2.0.19	2.0.1
2.0.17	2.0.11	2.0.13	2.0.1

Note:

1. To avoid system abnormalities, do not mix software and firmware from different baseline versions. Ensure that all components are deployed or upgraded strictly according to the versions listed in the table above.
2. Make sure that the API version in use matches the version indicated on the cover of the manual, to prevent inconsistencies between the documented interface descriptions and the actual functionality.

4. Overview of Function Categories

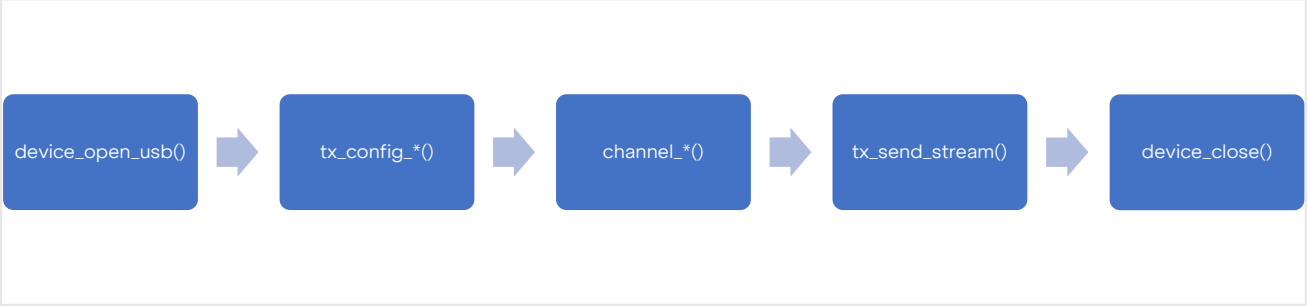
Table 2 API Function Categories

Function Category	Description
Device and System	Global functions that can be called in any operating mode. This category includes device open/close, global settings, retrieving device information, and obtaining device status.
Channel Settings	Channel-level control for managing the device's independent signal paths. Functions allow configuration, enabling, parameter setting, and status querying for specified channels, forming the basis for multi-channel synchronous or independent control.
Transmission Settings	Controls the transmission chain. Functions in this category configure transmission parameters, manage signal transmission processes, handle transmission data, and query transmission status. This is the core of the signal generator functionality.

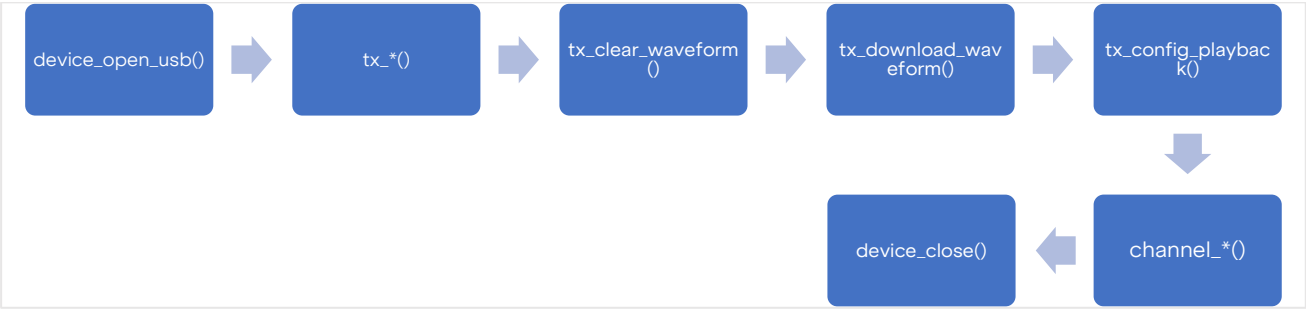
5. API Calling Flow

5.1 CW Mode

5.2 Real-Time Streaming Mode



5.3 Playback Mode



6. Device and System Function

device_open_usb must be called to initialize the device before accessing any hardware-related API functions. Upon completion of the application logic, call device_close to terminate the session and release allocated memory resources.

6.1 device_list_usb

int device_list_usb(device_info* **devInfo,** uint32_t* **count**)	
Description	
Enumerates the list of USB devices currently connected to the host.	
Compatibility	2.0.2 and later.
Parameter description	
[out] **devInfo**	Returns basic information for all currently connected devices. Valid data includes model and uid; refer to the device_info structure definition for details.
[out] **count**	Total number of USB devices currently connected to the host.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	None.
Example	

device_info devInfo;
uint32_t count = 0;
int status = device_list_usb(&devInfo, &count);	

6.2 device_open_usb

int device_open_usb(**void** device,** **uint8_t dnum,** **channel ch_o[],** **uint16_t* chn,** **device_info* dinfo** **)**	
Description	
Initializes the device connection. This function establishes low-level communication and enumerates available channels based on the startup configuration. It must be called before any subsequent API operations. Upon success, it returns the device handle, channel list, channel count, and basic device information.	
Compatibility	2.0.1 and later.
Parameter description	
[out] **device**	Returns the device handle. This handle is used to reference the target device in subsequent API calls.
[in] **dnum**	Specifies the zero-based index of the device to be opened. Used to select the target unit when multiple devices are connected.
[out] **ch_o**	Returns an array of channels supported by the device. Refer to the channel structure definition for details.
[out] **chn**	Returns the total number of channels.
[out] **dinfo**	Returns basic device information. Refer to the device_info structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be invoked before any other API calls and should only be executed once during the initialization phase. Subsequent operations rely on the device handle returned by this function. For every successful device_open_usb call, device_close must be invoked upon completion to release allocated resources.
Example	Refer to the relevant example of the tx_config_cw() function.

6.3 device_open_eth

int device_open_eth(**void** device,** **eth setting settings,** **channel ch_o[],** **uint16_t* chn,** **device info* dinfo** **)**	
Description	
Initializes the ETH device connection. This function establishes low-level communication and enumerates available channels based on the startup configuration. It must be called before any subsequent API operations. Upon success, it returns the device handle, channel list, channel count, and basic device information.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **device**	Returns the device handle. This handle is used to reference the target device in subsequent API calls.
[in] settings	Specifies the ETH device startup configuration. Refer to the eth_setting structure definition for details.
[out] **ch_o**	Returns an array of channels supported by the device. Refer to the channel structure definition for details.
[out] **chn**	Returns the total number of channels.
[out] **dinfo**	Returns basic device information. Refer to the device_info structure definition for details.
Return Value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be invoked before any other API calls and should only be executed once during the initialization phase. Subsequent operations rely on the device handle returned by this function. For every successful device_open_eth call, device_close must be invoked upon completion to release allocated resources.
Example	

int status = 0;
void* device = nullptr;
device_info dinfo;	

```
channel ch[8];  uint16_t chn = 0;
eth_setting settings;
settings.eth_interface = ETH_STANDARD;
settings.eth_is_ipv6 = 0;
settings.eth_ip_address[0] = 192;
settings.eth_ip_address[1] = 168;
settings.eth_ip_address[2] = 1;
settings.eth_ip_address[3] = 100;
settings.eth_remote_port = 5000;
settings.eth_read_timeout = 5000;
status = device_open_eth(&device, settings, ch, &chn, &dinfo);
status = device_close(&device);
```

6.4 device_close

int device_close(void device)**	
Description	
Closes the device connection and releases associated resources.	
Compatibility	1.0.1 and later.
Parameter description	
[in] **device**	Device handle. This handle becomes invalid once the connection is closed.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function should only be invoked at the end of program execution. Upon invocation, the device connection is closed and allocated resources are released. To reuse the device, device_open_* must be called again to re-establish the connection and initialize the hardware.
Example	Refer to the relevant example of tx_config_cw() function.

6.5 device_preset

int device_preset(void device)**	
Description	
Performs a device reset, including a global reset of all available channels.	
Compatibility	1.0.1 and later.
Parameter description	
[in] **device**	Device handle.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	

int status = -1;
void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
status = device_preset(&device);
status = device_close(&device);	

6.6 device_query_temperature

int device_query_temperature(void device, float* temp_o)**	
Description	
Queries the current temperature of the device.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **device**	Device handle.
[out] **temp_o**	Returns the device temperature in degrees Celsius.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.

Example

int status = -1;
void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
float temp_o = 0;
status = device_query_temperature(&device, &temp_o);
status = device_close(&device);

6.7 device_query_state

int device_query_state(void device)**	
Description	
Query the current state of the device.	
Compatibility	1.0.1 and later.
Parameter description	
[in] **device**	Device handle.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.

6.8 device_config_fan

int device_config_fan(**void** device,** **fan_mode mode,** **float auto_threshold** **)**	
Description	
Configures the fan operation settings.	
Compatibility	2.0.12 and later.
Parameter description	
[in] **device**	Device handle.
[in] **mod**	Sets the fan operating mode. Refer to the **fan_mode** enumeration definition for details.
[in] auto_threshold	Sets the temperature thresholds for automatic mode.

Return value	0: Success; Non-zero: Error code. Refer to the **Appendix 1** for details.
Calling constraints	Must be called after device_open_* initialization.

6.9 device_query_supply

int device_query_supply(**void** device,** **supply_info* supplyInfo_o** **)**	
Description	
Queries the current power supply status and electrical parameters of the device.	
Compatibility	2.0.10 and later.
Parameter description	
[in] **device**	Device handle.
[out] **supplyInfo_o**	Returns the device voltage and current parameters. Refer to the supply_info structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	

int status = -1; void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0; uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
supply_info supplyInfo_o;
status = device_query_supply(&device, &supplyInfo_o);
status = device_close(&device);	

6.10 device_config_gnss

int device_config_gnss(**void** device,** **gnss_setting gnss** **)**	
Description	
Configures the GNSS operating parameters for the device.	
Compatibility	2.0.7 and later.
Parameter description	
[in] **device**	Device handle.
[in] **gnss**	GNSS configuration parameters. Refer to the gnss_setting structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of **device_epoch_to_utc()** function.

6.11 device_query_gnss

int device_query_gnss(**void** device,** **gnss_setting* gnss_o** **)**	
Description	
Queries the current GNSS configuration of the device.	
Compatibility	2.0.7 and later.
Parameter description	
[in] **device**	Device handle.
[in] **gnss**	Returns the current GNSS configuration parameters. Refer to the gnss_setting structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of device_epoch_to_utc() function.

6.12 device_query_gnss_info

int device_query_gnss_info(**void** device,** **gnss_info* gnss_o** **)**	
Description	
Queries the current GNSS information of the device.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **device**	Device handle.
[out] **gnss_o**	Returns the GNSS information of the device. Refer to the gnss_info structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of device_epoch_to_utc() function.

6.13 device_query_apiverion

uint32_t device_query_apiverion()	
Description	
Returns the API version information associated with the current library.	
Compatibility	1.0.1 and later.
Return value	API version number. Represented using major, minor, and revision versions. bit[31..16]: major version; bit[15..8]: minor version; bit[7..0]: revision version.
Calling constraints	None.
Example	

uint32_t verion = device_query_apiverion();
int major = 0, minor = 0, rev = 0;
major = (verion >> 16) & 0xffff;
minor = (verion >> 8) & 0xff;
rev = verion & 0xff;	

6.14 device_config_clock

```
int device_query_gnss_info(
    void** device,
    referenceclock_source source,
    double fref,
    state out
)
```

Description	
Configures the reference clock source for the device.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **device**	Device handle.
[in] **source**	Set the reference clock source. Refer to the referenceclock_source structure definition for details.
[in] **fref**	Set the reference clock frequency in Hz.
[in] **out**	Returns the enablement state of the reference clock output.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.

Example

```
int status = -1;
void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
referenceclock_source source = REFCLKSOURCE_EXTERNAL;
double fref = 10e6;
state out = STATE_OFF;
status = device_config_clock(&device, source, fref, out);
state locked_o;
status = device_query_clock(&device, &source, &fref, &out, &locked_o);
status = device_close(&device);
```

6.15 device_query_clock

int device_query_clock(**void** device,** **referenceclock_source* source_o,** **double* fref_o,** **state* out_o,** **state* locked_o** **)**	
Description	
Queries the current reference clock configuration status of the device.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **device**	Device handle.
[out] **source_o**	Returns the reference clock source. Refer to the referenceclock_source structure definition for details.
[out] **fref_o**	Returns the reference clock frequency in Hz.
[out] **out_o**	Returns the enablement state of the reference clock output.
[out] **locked_o**	Returns the lock status of the reference clock monitoring. (The actual reference clock frequency can only be accurately retrieved when the status is Locked).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of device_config_clock() function.

6.16 device_reset_clock_monitor

int device_reset_clock_monitor(void device)**	
Description	
Resets the reference clock monitoring.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **device**	Device handle. This handle becomes invalid once the connection is closed.

Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	

int status = -1;
void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
status = device_reset_clock_monitor(&device);
status = device_close(&device);	

6.17 device_epoch_to_utc

```
void device_epoch_to_utc(
    uint64_t ns_sinceepoch,
    int16_t* year,
    int16_t* mon,
    int16_t* day,
    int16_t* hour,
    int16_t* min,
    int16_t* sec,
    int16_t* ms,
    int16_t* us,
    int16_t* ns
)
```

Description	
Converts a Unix Epoch timestamp (in nanoseconds) into a readable UTC time format.	
Compatibility	2.0.7 and later.
Parameter description	
[in] **ns_sinceepoch**	Elapsed time since Unix Epoch in nanoseconds.
[out] **year**	Returns the year (e.g., 2026).
[out] **mon**	Returns the month (1 to 12).
[out] **day**	Returns the day (1 to 31).
[out] **hour**	Returns the hour (0 to 23).
[out] **min**	Returns the minute (0 to 59).

[out] **sec**	Returns the second (0 to 59).
[out] **ms**	Returns the millisecond (0 to 999).
[out] **us**	Returns the microsecond (0 to 999).
[out] **ns**	Returns the nanosecond (0 to 999).
Return value	None.
Calling constraints	Must be called after the GNSS is Locked and the device_query_gnss_info function has been executed.

Example

```
int status = -1; void* device = nullptr; device_info dinfo;
channel ch[8];
uint16_t chn = 0; uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
gnss_setting gnss;
gnss.antenna = 1;
status = device_config_gnss(&device, gnss);
status = device_query_gnss(&device, &gnss);
gnss_info gnss_info_o;
status = device_query_gnss_info(&device, &gnss_info_o);
int16_t year = 0, mon = 0, day = 0, hour = 0, min = 0, sec = 0, ms = 0, us = 0, ns = 0;
device_epoch_to_utc(gnss_info_o.ns_sinceepoch, &year, &mon, &day, &hour, &min, &sec,
&ms, &us, &ns);
status = device_close(&device);
```

7. Channel Setting Function

7.1 channel_preset

int channel_preset(channel* ch)	
Description	
Performs a reset operation on multiple channels without initiating output.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Channel array. Refer to the channel structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	

int status = -1; void* device = nullptr;
device_info dinfo;
channel ch[8]; uint16_t chn = 0; uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
status = channel_preset(ch);
status = device_close(&device);	

7.2 channel_start

int channel_start(channel* ch)	
Description	
Initiates multi-channel operation. This function must be invoked to apply changes and start the channels after modifying RF or baseband parameters	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Channel array. Refer to the channel structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_cw() function.

7.3 channel_stop

int channel_stop(channel* ch)	
Description	
Terminates multi-channel operation.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Channel array. Refer to the channel structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_cw() function.

7.4 channel_trigger_bus

int channel_trigger_bus(channel* ch)	
Description	
Executes a single bus trigger across multiple channels.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Channel array. Refer to the definition of the channel structure for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Please refer to the relevant example of the tx_config_cw() function.

7.5 channel_config_trigger

int channel_config_trigger(**channel* ch,** **const tx_trigger* trg** **)**	
Description	
Configures the trigger parameters for the specified channels.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[in] **trg**	Trigger configuration parameters. Refer to the tx_trigger structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_cw() function.

7.6 channel_query_trigger

int channel_query_trigger(**channel* ch,** **tx_trigger* trg_o** **)**	
Description	
Queries the current trigger configuration for the specified channels.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[out] **trg_o**	Returns the trigger configuration. Refer to the tx_trigger structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_cw() function.

7.7 channel_config_trigger_out

int channel_config_trigger_out(**channel* ch,** **const trigger_out* trg** **)**	
Description	
Configures the trigger output parameters for the specified channels.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[in] **trg**	Trigger output configuration parameters. Refer to the trigger_out structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	

```
int status = -1;
void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
double start = 1.0e9;
double stop = 2.0e9;
double step = 100e6;
float level = 0.0f;
float dwell = 1e-4;
status = tx_config_fscan(ch, start, stop, step, level, dwell);
tx_trigger trg;
trg.source = TRIGGER_SOURCE_BUS;
trg.action = TRIGGER_ACTION_SWEEP;
trg.count = -1;
status = channel_config_trigger(ch, &trg);
```

```
trigger_out trgout;
trgout.enable = STATE_ON;
trgout.action = TRIGGER_ACTION_HOP;
trgout.edge = TRIGGER_EDGE_RISING;
status = channel_config_trigger_out(ch, &trgout);
status = channel_query_trigger_out(ch, &trgout);
status = tx_config_cw(ch);
status = tx_config_output(ch, STATE_ON, STATE_OFF);
status = channel_start(ch);
status = channel_trigger_bus(ch);
status = device_close(&device);
```

7.8 channel_query_trigger_out

int channel_query_trigger_out(**channel* ch,** **trigger_out* trg_o** **)**	
Description	
Queries the current trigger output configuration for the specified channels.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[out] **trg_o**	Returns the trigger output configuration parameters. See the definition of the trigger_out structure for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of channel_config_trigger_out() function.

7.9 channel_config_lo_mode

int channel_config_lo_mode(**channel* ch,** **lomode mode** **)**	
Description	
Configures the LO mode for the specified channels.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[in] **mode**	LO mode configuration. Refer to the lomode structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	

```
int status = -1;
void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
lomode mode = LOMODE_LOWPHASENOISE;
status = channel_config_lo_mode(ch, mode);
status = channel_query_lo_mode(ch, &mode);
status = device_close(&device);
```

7.10 channel_query_lo_mode

int channel_query_lo_mode(**channel* ch,** **lomode* mode_o** **)**	
Description	
Queries the current LO mode configuration for the specified channels.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[out] **mode_o**	Returns the LO mode configuration. Refer to the lomode structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of channel_config_lo_mode() function.

8. Transmission Setting Functions

8.1 tx_clear_waveform

int tx_clear_waveform(void device)**	
Description	
Clears all waveform data from the device memory.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **device**	Device handle.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_playback() function.

8.2 tx_download_waveform

int tx_download_waveform(**void** device,** **int16_t iq[],** **uint32_t points,** **uint16_t* waveform_o** **)**	
Description	
Downloads waveform data to the device memory.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **device**	Device handle.
[in] **iq**	IQ date array.
[in] **points**	Number of IQ point pairs.
[out] **waveform_o**	Waveform index.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_playback() function.

8.3 tx_config_ffm

int tx_config_ffm(**channel* ch,** **double fc,** **float level** **)**	
Description	
Configures the frequency and amplitude parameters for the specified channels in FFM mode.	
Compatibility	1.0.1 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[in] **fc**	Center frequency.
[in] **level**	Output amplitude.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_cw() function.

8.4 tx_query_ffm

int tx_query_ffm(**channel* ch,** **double* fc_o,** **float* level_o** **)**	
Description	
Queries the current frequency and amplitude parameters for the specified channels in FFM mode.	
Compatibility	1.0.1 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[out] **fc_o**	Returns the center frequency.
[out] **level_o**	Returns the output amplitude.

Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of the tx_config_cw() function.

8.5 tx_config_fscan

int tx_config_fscan(**channel* ch,** **double start,** **double stop,** **double step,** **float level,** **float dwell** **)**	
Description	
Configures the frequency sweep parameters for the specified channels. Performs an incremental sweep from the Start frequency to the Stop frequency using a defined Step size at a constant power level.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[in] **start**	Start frequency of the sweep.
[in] **stop**	Stop frequency of the sweep.
[in] **step**	Frequency step of the sweep.
[in] **level**	Output amplitude.
[in] **dwell**	Dwell time per frequency point (seconds). In TRIGGER_ACTION_HOP mode, the dwell time parameter is ignored, and frequency switching is governed exclusively by the arrival of a trigger event.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	

```
int status = -1;
void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
double start = 1.0e9;
double stop = 2.0e9;
double step = 100e6;
float level = 0.0f;
float dwell = 1e-4;
status = tx_config_fscan(ch, start, stop, step, level, dwell);
status = tx_query_fscan(ch, &start, &stop, &step, &level, &dwell);
tx_trigger trg;
trg.source = TRIGGER_SOURCE_BUS;
trg.action = TRIGGER_ACTION_SWEEP;
trg.count = -1;
status = channel_config_trigger(ch, &trg);
status = tx_config_cw(ch);
status = tx_config_output(ch, STATE_ON, STATE_OFF);
status = channel_start(ch);
status = channel_trigger_bus(ch);
status = device_close(&device);
```

8.6 tx_query_fscan

```
int tx_query_fscan(
    channel* ch,
    double* start_o,
    double* stop_o,
    double* step_o,
    float* level_o,
    float* dwell_o
)
```

Description
Queries the current frequency sweep configuration for the specified channels.

Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[out] **start_o**	Returns the start frequency of the sweep.
[out] **stop_o**	Returns the stop frequency of the sweep.
[out] **step_o**	Returns the frequency step of the sweep.
[out] **level_o**	Returns the output amplitude.
[out] **dwell_o**	Returns the dwell time per frequency point (in seconds).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_fscan() function.

8.7 tx_config_lscan

```
int tx_config_lscan(
    channel* ch,
    double fc,
    float level_start,
    float level_stop,
    float level_step,
    float dwell
)
```

Description	
Configures the amplitude sweep parameters for the specified channels. Performs an incremental sweep from Level Start to Level Stop using a defined Level Step size at a constant center frequency fc.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel definition structure.
[in] **fc**	Center frequency.
[in] **level_start**	Sweep start amplitude.
[in] **level_stop**	Sweep stop amplitude.
[in] **level_step**	Sweep step amplitude.
[in] **dwell**	Dwell time per amplitude point (seconds).

Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	

```
int status = -1;
void *device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
double fc = 1.0e9;
float level_start = -20;
float level_stop = -10;
float level_step = 1;
float dwell = 1e-4;
status = tx_config_lscan(ch, fc, level_start, level_stop, level_step, dwell);
status = tx_query_lscan(ch, &fc, &level_start, &level_stop, &level_step, &dwell);
tx_trigger trg;
trg.source = TRIGGER_SOURCE_BUS;
trg.action = TRIGGER_ACTION_SWEEP;
trg.count = -1;
status = channel_config_trigger(ch, &trg);
status = tx_config_cw(ch);
status = tx_config_output(ch, STATE_ON, STATE_OFF);
status = channel_start(ch);
status = channel_trigger_bus(ch);
status = device_close(&device);
```

8.8 tx_query_lscan

int tx_query_lscan(**channel* ch,** **double* fc_o,** **float* level_start_o,** **float* level_stop_o,** **float* level_step_o,** **float* dwell_o** **)**	
Description	
Queries the current amplitude sweep configuration for the specified channels.	
Compatibility	2.0.11 and later.
Parameter description	
[in] **ch**	Target channels. Refer to the channel structure definition for details.
[out] **fc_o**	Returns the center frequency.
[out] **level_start_o**	Returns the sweep start amplitude.
[out] **level_stop_o**	Returns the sweep stop amplitude.
[out] **level_step_o**	Returns the sweep step amplitude.
[out] **dwell_o**	Returns the dwell time per amplitude point (seconds).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_lscan() function.

8.9 tx_config_output

int tx_config_output(**channel* ch,** **state rfout,** **state mod** **)**	
Description	
Configures the RF and modulation output states for multiple channels.	
Compatibility	2.0.1 and later.
Parameter description	

[in] **ch**	Channel configuration array. Refer to the channel structure definition for details.
[in] **rfout**	RF output state configuration. Refer to the state enumeration structure definition for details.
[in] **mod**	Modulation output state configuration, refer to the state enumeration structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_cw() function.

8.10 tx_query_output

int tx_query_output(**channel* ch,** **state* rfout_o,** **state* mod_o** **)**	
Description	
Queries the current RF and modulation output states for multiple channels.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Channel configuration array. Refer to the channel structure definition for details.
[out] **rfout_o**	Returns the RF output state. Refer to the state enumeration structure definition for details.
[out] **mod_o**	Returns the modulation output state. Refer to the state enumeration structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_config_cw() function.

8.11 tx_config_cw

int tx_config_cw(channel* ch)	
Description	
Enables multi-channel Continuous Wave (CW) transmission.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Channel configuration array. Refer to the channel structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	

```
int status = -1; void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0; uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
double fc = 1.0e9;
float level = 0.0f;
status = tx_config_ffm(ch, fc, level);
status = tx_query_ffm(ch, &fc, &level);
tx_trigger trg;
trg.source = TRIGGER_SOURCE_BUS;
status = channel_config_trigger(ch, &trg);
status = channel_query_trigger(ch, &trg);
status = tx_config_cw(ch);
status = tx_config_output(ch, STATE_ON, STATE_OFF);
state rfout_o, mod_o;
status = tx_query_output(ch, &rfout_o, &mod_o);
status = channel_start(ch);
status = channel_trigger_bus(ch);
std::this_thread::sleep_for(std::chrono::seconds(10));
status = channel_stop(ch);
status = device_close(&device);
```

8.12 tx_config_stream

int tx_config_stream(**channel* ch,** **double sample_rate** **)**	
Description	
Configures the streaming playback parameters for the specified channel.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Channel configuration array. Refer to the channel structure definition for details.
[in] **sample_rate**	Channel sampling rate.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the relevant example of tx_send_stream() function.

8.13 tx_send_stream

int tx_send_stream(**channel* ch,** **int16_t iq[],** **uint32_t points** **)**	
Description	
Performs real-time IQ streaming playback on the specified channel.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Channel configuration array. Refer to the channel structure definition for details.
[in] **iq**	IQ data array.
[in] **points**	Number of IQ point pairs.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.

Example

```
// Simulated waveform: generates a sine wave
std::vector<int16_t> sim_generatewaveform_sine()
{
  constexpr uint32_t TOTAL_INT16_POINTS = 500000;
  constexpr uint32_t IQ_POINTS = TOTAL_INT16_POINTS / 2;
  constexpr int16_t AMP = 30000;
  constexpr uint32_t CYCLES = 100;
  const double twoPi = 2.0 * 3.141592654;
  std::vector<int16_t> iq;
  iq.reserve(TOTAL_INT16_POINTS);
  for (uint32_t n = 0; n < IQ_POINTS; ++n) {
    double phase = twoPi * CYCLES * n / IQ_POINTS;
    iq.push_back(static_cast<int16_t>(AMP * std::cos(phase)));
    iq.push_back(static_cast<int16_t>(AMP * std::sin(phase)));
  }
  return iq;
}

int status = -1;
void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
double samplerate = 50e6;
std::vector<int16_t> data;
data = sim_generatewaveform_sine();
double fc = 1.0e9;
float level = 0.0f;
status = tx_config_ffm(ch, fc, level);
tx_trigger trg;
trg.source = TRIGGER_SOURCE_BUS;
status = channel_config_trigger(ch, &trg);
```

```
status = tx_config_stream(ch, samplerate);
status = tx_config_output(ch, STATE_ON, STATE_ON);
status = channel_start(ch);
status = channel_trigger_bus(ch);
for (int i = 0; i < 100000; ++i) {
    status = tx_send_stream(ch, data.data(), data.size() / 2);
    if (status != STATUS_NOERROR) {
        break;
    }
}
status = device_close(&device);
```

8.14 tx_config_playback

```
int tx_config_playback(
    channel* ch,
    const int16_t waveform[],
    const int32_t repeat[],
    const double sample_rate[],
    int16_t waveforms
)
```

Description	
Configures waveform playback parameters (ARB Mode) for the specified channels. Once configured, a channel_trigger_bus call is required to initiate the transmission via the trigger bus.	
Compatibility	2.0.1 and later.
Parameter description	
[in] **ch**	Channel configuration array. Refer to the channel structure definition for details.
[in] **waveform**	Waveform index array. Specifies the indices of the waveforms to be played.
[in] **repeat**	Waveform repeat count. Specifies the number of playback cycles for each waveform segment within the sequence.
[in] **sample_rate**	Waveform playback sampling rate.
[in] **waveforms**	Number of waveforms to play

Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.

Example

```
std::vector<int16_t> sim_generatewaveform_pulse(double duty_cycle)
{
  constexpr uint32_t TOTAL_IQ_POINTS = 250000;
  constexpr uint32_t TOTAL_INT16_POINTS = TOTAL_IQ_POINTS * 2;
  constexpr int16_t I_VALUE = 32766;
  if (duty_cycle < 0.0) duty_cycle = 0.0;
  if (duty_cycle > 1.0) duty_cycle = 1.0;
  uint32_t active_iq = static_cast<uint32_t>(TOTAL_IQ_POINTS * duty_cycle);
  std::vector<int16_t> iq;
  iq.reserve(TOTAL_INT16_POINTS);
  for (uint32_t i = 0; i < active_iq; ++i) {
    iq.push_back(I_VALUE);  // I
    iq.push_back(0);      // Q
  }
  uint32_t remain_iq = TOTAL_IQ_POINTS - active_iq;
  for (uint32_t i = 0; i < remain_iq; ++i) {
    iq.push_back(0);
    iq.push_back(0);
  }
  return iq;
}
double samplerate = 125e6;
std::vector<int16_t> data;
data = sim_generatewaveform_pulse(0.25));

int status = -1;
void* device = nullptr;
device_info dinfo;
channel ch[8];
uint16_t chn = 0;
uint8_t dnum = 0;
```

```
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
double fc = 1.0e9;
float level = 0.0f;
status = tx_config_ffm(ch, fc, level);
tx_trigger trg;
trg.source = TRIGGER_SOURCE_BUS;
status = channel_config_trigger(ch, &trg);
uint16_t waveid = 0;
status = tx_clear_waveform(&device);
status = tx_download_waveform(&device,  data.data(), data.size() / 2, &waveid);
int16_t waveforms = 1;
const int16_t waveform[] = { waveid };
const int32_t repeat[] = { -1 };
const double srate[] = { samplerate };
status = tx_config_playback(ch, waveform, repeat, srate, waveforms);
status = tx_config_output(ch, STATE_ON, STATE_ON);
status = channel_start(ch);
status = channel_trigger_bus(ch);
std::this_thread::sleep_for(std::chrono::seconds(10));
status = device_close(&device);
```

9. Modulation Configuration Functions

9.1 mod_open

int mod_open(**void** device,** **char version_o[50],** **)**	
Description	
Enables the modulation function. After the device is opened, pass the device information structure to this function for device verification, check whether the modulation library exists, register the function, create the device object, and enable the modulation function.	
Compatibility	2.0.18 and later
Parameter description	
[out] **device**	Returns the device handle. This handle is used to reference the target device in subsequent API calls.
[out] **version_o[50]**	Returns the version number of the modulation library.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called after device_open_* initialization.
Example	Refer to the example of the mod_generate_stream() function.

9.2 mod_close

int mod_close(**void** device** **)**	
Description	
Disable the modulation function.	
Compatibility	2.0.18and later
Parameter description	
[out] **device**	Device handle. The handle becomes invalid after the device is closed.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.

Calling constraints	This function should only be called before the modulation program terminates. After execution, the modulation function will be disabled and all related resources will be released. To enable modulation again, mod_open must be called again.
Example	Refer to the example of the mod_generate_stream() function.

9.3 mod_init_digitalmod

int mod_init_digitalmod(**void** device,** **digitalmod_profile* digitalmod_profile_o** **)**	
Description	
Initialize the digital modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **digitalmod_profile_o**	Digital modulation configuration structure. For details, refer to the definition of the digitalmod_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the example of the mod_generate_stream() function.

9.4 mod_init_am

int mod_init_am(**void** device,** **am_profile* am_profile_o** **)**
Description
Initialize the AM modulation configuration structure and assign default values to all parameters in the structure.

Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **am_profile_o**	AM configuration structure. For details, refer to the definition of the am_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.

9.5 mod_init_fm

int mod_init_fm(**void** device,** **fm_profile* fm_profile_o** **)**	
Description	
Initialize the FM modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **fm_profile_o**	FM configuration structure. For details, refer to the definition of the fm_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.

9.6 mod_init_dsss

int mod_init_dsss(**void** device,** **dsss_profile* dsss_profile_o** **)**
Description
Initialize the direct-sequence spread spectrum configuration structure and assign default values to all parameters in the structure.

Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **dsss_profile_o**	Direct-sequence spread spectrum configuration structure. For details, refer to the definition of the dsss_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.

9.7 mod_init_pulse

int mod_init_pulse(**void** device,** **pulse_profile* pulse_profile_o** **)**	
Description	
Initialize the pulse modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **pulse_profile_o**	Pulse modulation configuration structure. For details, refer to the definition of the pulse_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.

9.8 mod_init_multitone

int mod_init_multitone(**void** device,** **multitone_profile* multitone_profile_o** **)**	
Description	
Initialize the multitone configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **multitone_profile_o**	Multitone configuration structure. For details, refer to the definition of the multitone_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.

9.9 mod_init_stepsweep

int mod_init_stepsweep(**void** device,** **stepsweep_profile* stepsweep_profile_o** **)**	
Description	
Initialize the step sweep configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **stepsweep_profile_o**	Step sweep configuration structure. For details, refer to the definition of the stepsweep_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.

9.10 mod_init_rampsweep

int mod_init_rampsweep(**void** device,** **rampsweep_profile* rampsweep_profile_o** **)**	
Description	
Initialize the ramp sweep configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **rampsweep_profile_o**	Ramp sweep configuration structure. For details, refer to the definition of the rampsweep_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.

9.11 mod_init_ofdm

int mod_init_ofdm (**void** device,** **ofdm_profile* ofdm_profile_o** **)**	
Description	
Initialize the OFDM configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **ofdm_profile_o**	OFDM configuration structure. For details, refer to the definition of the ofdm_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.

9.12 mod_init_awgn

int mod_init_awgn (**void** device,** **awgn_profile* awgn_profile_o** **)**	
Description	
Initialize the Gaussian white noise configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **awgn_profile_o**	Gaussian white noise configuration structure. For details, refer to the definition of the awgn_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.

9.13 mod_config_digitalmod

int mod_config_digitalmod(**void** device,** **const digitalmod_profile* digitalmod_profile_in** **digitalmod_profile* digitalmod_profile_o** **)**	
Description	
Initialize the digital modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[in] **digitalmod_profile_in**	Input digital modulation configuration structure. For details, refer to the definition of the digitalmod_profile structure.

[out] **digitalmod_profile_o**	Output digital modulation configuration structure. For details, refer to the definition of the digitalmod_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the example of the mod_generate_stream() function.

9.14 mod_config_am

```
int mod_config_am(
    void** device,
    const am_profile* am_profile_in
    am_profile* am_profile_o
)
```

Description	
Initialize the AM modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[in] **am_profile_in**	Input AM modulation configuration structure. For details, refer to the definition of the am_profile structure.
[out] **am_profile_o**	Output AM modulation configuration structure. For details, refer to the definition of the am_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_am has been successfully executed.

9.15 mod_config_fm

int mod_config_fm(**void** device,** **const fm_profile* fm_profile_in** **fm_profile* fm_profile_o** **)**	
Description	
Configure FM modulation parameters. If the input configuration is invalid, the function will overwrite the output configuration structure with valid parameters.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[in] **fm_profile_in**	Input FM modulation configuration structure. For details, refer to the definition of the fm_profile structure.
[out] **fm_profile_o**	Input FM modulation configuration structure. For details, refer to the definition of the fm_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_fm has been successfully executed.

9.16 mod_config_dsss

int mod_config _dsss(**void** device,** **const dsss_profile* dsss_profile_in** **dsss_profile* dsss_profile_o** **)**	
Description	
Initialize the direct-sequence spread spectrum configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[in] **dsss_profile_in**	Input direct-sequence spread spectrum configuration structure. For details, refer to the definition of the dsss_profile structure.

[out] **dsss_profile_o**	Output direct-sequence spread spectrum configuration structure. For details, refer to the definition of the dsss_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_dass has been successfully executed.

9.17 mod_config_pulse

int mod_config _pulse(**void** device,** **const pulse_profile* pulse_profile_in** **pulse_profile* pulse_profile_o** **)**	
Description	
Initialize the pulse modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[in] **pulse_profile_in**	Input pulse modulation configuration structure. For details, refer to the definition of the pulse_profile structure.
[out] **pulse_profile_o**	Output pulse modulation configuration structure. For details, refer to the definition of the pulse_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_pulse has been successfully executed.

9.18 mod_config_multitone

int mod_config_multitone(**void** device,** **const multitone_profile* multitone_profile_in** **multitone_profile* multitone_profile_o** **)**	
Description	
Initialize the multitone configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[in] **multitone_profile_in**	Input multitone modulation configuration structure. For details, refer to the definition of the multitone_profile structure.
[out] **multitone_profile_o**	Output multitone modulation configuration structure. For details, refer to the definition of the multitone_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_multitone has been successfully executed.

9.19 mod_config_stepsweep

int mod_config_stepsweep(**void** device,** **const stepsweep_profile* stepsweep_profile_in** **stepsweep_profile* stepsweep_profile_o** **)**	
Description	
Configure step sweep parameters. If the input configuration is invalid, the function will overwrite the output configuration structure with valid parameters.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.

[in] **stepsweep_profile_in**	Input step sweep configuration structure. For details, refer to the definition of the stepsweep_profile structure.
[out] **stepsweep_profile_o**	Output step sweep configuration structure. For details, refer to the definition of the stepsweep_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_stepsweep has been successfully executed.

9.20 mod_config_rampsweep

```
int mod_config_rampsweep(
    void** device,
    const rampsweep_profile* rampsweep_profile_in
    rampsweep_profile* rampsweep_profile_o
)
```

Description	
Configure ramp sweep parameters. If the input configuration is invalid, the function will overwrite the output configuration structure with valid parameters.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[out] **rampsweep_profile_o**	Ramp sweep configuration structure. For details, refer to the definition of the rampsweep_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_rampsweep has been successfully executed.

9.21 mod_config_ofdm

int mod_config_ofdm (**void** device,** **const ofdm_profile* ofdm_profile_in** **ofdm_profile* ofdm_profile_o** **)**	
Description	
Configure OFDM parameters. If the input configuration is invalid, the function will overwrite the output configuration structure with valid parameters.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[in] **ofdm_profile_o**	Input OFDM configuration structure. For details, refer to the definition of the ofdm_profile structure.
[out] **ofdm_profile_o**	Output OFDM configuration structure. For details, refer to the definition of the ofdm_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_ofdm has been successfully executed.

9.22 mod_config_awgn

int mod_config_awgn (**void** device,** **const awgn_profile* awgn_profile_in** **awgn_profile* awgn_profile_o** **)**	
Description	
Initialize the Gaussian white noise configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[in] **awgn_profile_in**	Input Gaussian white noise configuration structure. For details, refer to the definition of the awgn_profile structure.

[out] **awgn_profile_o**	Output Gaussian white noise configuration structure. For details, refer to the definition of the awgn_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_awgn has been successfully executed.

9.23 mod_generate_stream

```
int mod_generate_stream (
    void** device,
    int32_t symbol_length,
    int16_t** iq_o,
    int32_t* length_o,
    double* sample_rate_o
)
```

Description	
After enabling the modulation function, call the initialization function to assign default values, and then use the configuration structure functions to apply parameters and generate signal data.	
Compatibility	2.0.18and later
Parameter description	
[in] **device**	Device handle.
[in] **symbol_length**	Symbol length, used for digital modulation and direct-sequence spread spectrum modulation.
[out] **iq_o**	Output signal data.
[out] **length_o**	Output data length.
[out] **sample_rate_o**	Sampling rate used for generating the current data.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_* has been successfully executed.

Example

```
/* Startup configuration and device information */
int status = 0;
void* device = nullptr;
device_info dinfo;
```

```
channel ch[MAXCHANNELS];
uint16_t chn = 0;
uint8_t dnum = 0;
char version[50] = { "" };
int16_t* iq = nullptr;
int32_t length = 0;
double sampleRate = 0;

/* Device initialization */
status = device_open_usb(&device, dnum, ch, &chn, &dinfo);
if (status != 0) {
   std::cout << "[ERROR] device_open_usb failed! status = " << status << std::endl;
   return status; // If open fails, do not proceed to waveform generation/modulation stage
}

/* Enable modulation */
status = mod_open(&device, version);
digitalmod_profile digitalModProfile_in;
digitalmod_profile digitalModProfile_out;
mod_init_digitalmod(&device, &digitalModProfile_in);

/* Parameter configuration */
digitalModProfile_in.digitalmod_type = DigitalModType_BPSK;
digitalModProfile_in.symbolrate = 5e6;
digitalModProfile_in.sps = 4;
status = mod_config_digitalmod(&device, &digitalModProfile_in, &digitalModProfile_out);

/* Get modulated waveform data */
uint32_t symbol_length = 1U << digitalModProfile_out.pn;
status = mod_generate_stream(&device, symbol_length, &iq, &length, &sampleRate);

// Extend IQ data; both I and Q exceed 65536 points
std::vector<int16_t> vector;
if (length < 65536 * 2) {
   int count = std::ceil(65536.0 * 2 / length);
```

```
  vector.resize(count * length);
  for (int i = 0; i < count; i++) {
    memcpy(vector.data() + i * length, iq, length * sizeof(int16_t));
  }
  length = count * length;
}

/* RF configuration */
double fc = 1.0e9;  // Output frequency Hz
float level = 0.0f; // Output power dBm
status = tx_config_ffm(ch, fc, level); // Configure fixed frequency mode
print_status(status);

/* Trigger configuration */
tx_trigger trg;
trg.source = TRIGGER_SOURCE_BUS;
status = channel_config_trigger(ch, &trg);
print_status(status);

/* Waveform download */
uint16_t waveid = 0;
status = tx_clear_waveform(&device);
int16_t* final_waveform = vector.empty() ? iq : vector.data();
status = tx_download_waveform(&device, final_waveform, length/2, &waveid);
print_status(status);

/* Baseband configuration */
int16_t waveforms = 1;                    // Number of waveforms to play
const int16_t waveform[] = { waveid }; // Waveform IDs to be played
const int32_t repeat[] = { -1 };          // Repeat count for each waveform, -1 means infinite
loop
const double srate[] = { sampleRate }; // Sampling rate for each waveform
status = tx_config_playback(ch, waveform, repeat, srate, waveforms); // Configure playback
print_status(status);
```

```
status = tx_config_output(ch, STATE_ON, STATE_ON); // Enable output
status = channel_start(ch); // Start channel
status = channel_trigger_bus(ch);

std::this_thread::sleep_for(std::chrono::seconds(10));

status = mod_close(&device); // Close modulation
if (status) {
   std::cout << __LINE__ << ", mod_close_status = " << status << std::endl;
}

status = device_close(&device); // Close device
return 0;
```

10. Structure Variables

10.1 device_info

device_info	
uint16_t **model**	Device model.
uint64_t **uid_l64**	Low 64-bit device serial number.
uint32_t **uid_h32**	High 32-bit device serial number.
uint16_t **hardware_version**	Hardware version.
uint32_t **mfw_version**	MCU firmware version.
uint32_t **ffw_version**	FPGA firmware version.
uint32_t **pmu_version**	PMU firmware version.
uint32_t **agu_version**	AGU firmware version.
uint32_t **bus_version**	Bus peripheral firmware version.
uint32_t **eio_version**	EIO firmware version.
uint32_t **bus_bandwidth**	Bus bandwidth, unit MB/s.

10.2 supply_info

supply_info	
float **rf_voltage**	Power port voltage, in V.
float **rf_current**	Power port current, in A.
float **usb_voltage**	USB port voltage, in V.
float **usb_current**	USB port current, in A.

10.3 eth_setting

eth_setting	
eth_interface_type **eth_interface**	Physical interface type. Refer to the eth_interface_type enumeration structure definition for details.
uint8_t **eth_is_ipv6**	ETH protocol type (0: IPv4, 1: IPv6).
uint8_t **eth_ip_address[16]**	IP address of the ETH interface.
uint16_t **eth_remote_port**	Remote port of the ETH interface.
int32_t **eth_read_timeout**	ETH interface read timeout, in ms.

10.4 gnss_setting

gnss_setting	
uint8_t **antenna**	Antenna path selection. 0 for internal antenna, 1 for external antenna.
uint8_t **pps_en**	PPS enable state.
float **pps_x**	PPS frequency (in Hz).
float **pps_delay**	PPS delay (in seconds).

10.5 gnss_info

gnss_info	
state **locked**	GNSS lock status, 0: unlocked, 1: locked. Refer to the state enumeration structure definition for details.
uint8_t **sat_nums**	Current number of GNSS satellites in view.
uint8_t **snr_max**	Maximum SNR of the currently tracked GNSS satellites.
uint8_t **snr_avg**	Average SNR of the currently tracked GNSS satellites.
uint8_t **snr_min**	Minimum SNR of the currently tracked GNSS satellites.
float **latitude**	Latitude: positive for north, negative for south.
float **longitude**	Longitude: positive for east, negative for west.
float **altitude**	Altitude in m.
uint64_t **ns_sinceepoch**	System timestamp (nanoseconds).

10.6 channel

channel	
void** **device**	Device handle to which the channel belongs.
uint16_t **num**	Channel number.
channel_type **type**	Channel type. Refer to the channel_type enumeration structure definition for details.

10.7 tx_trigger

tx_trigger	
trigger_source **source**	Trigger source. Refer to the trigger_source enumeration structure definition for details.
trigger_edge **edge**	Trigger edge. Refer to the trigger_edge enumeration structure definition for details. This parameter is invalid under bus trigger mode.
trigger_action **action**	Trigger action; this parameter is invalid under fixedpoints mode. Refer to the trigger_action enumeration structure definition for details.
int32_t **count**	Number of trigger responses: -1 indicates unlimited responses; >0 specifies a limited number of responses; after responding count times, triggering stops and must be reactivated. Ignored in fixed-frequency mode. When the trigger action is TRIGGER_ACTION_HOP, one trigger switches through count RF states, with the interval between states defined by the configured dwell time. When the trigger action is TRIGGER_ACTION_SWEEP, one trigger performs count sweeps, and the time interval within each sweep is defined by the configured dwell time.

10.8 trigger_out

trigger_out	
state **enable**	Trigger output state. Refer to the state enumeration structure definition for details.
trigger_action **action**	Trigger action. Outputting a trigger signal upon completion of a single hop or a full sweep sequence. Refer to the trigger_action enumeration structure definition for details.
trigger_edge **edge**	Trigger edge. Refer to the trigger_edge enumeration structure definition for details.

10.9 digitalmod_profile

digitalmod_profile	
mod_digitalmod_type digitalmod_type	Trigger output status. Refer to the definition of the mod_digitalmod_type enumeration.
double symbolrate	Symbol rate, in Hz. Range: minimum sampling rate / samples per symbol ~ maximum sampling rate / samples per symbol.
double fskdeviation	FSK frequency deviation, in Hz, maximum is 15 × symbol rate.
int32_t pn	Pseudo-random sequence, range 4 to 32, recommended maximum not exceeding 24.
mod_filter_type filter_type	Filter type. Refer to the definition of the mod_filter_type enumeration.
int32_t filter_symbols	Number of filter symbols, must be an even number. Range: 2 to (sps × filter_symbols ≤ 400), default is 8.
int32_t sps	Samples per symbol, even number from 4 to 32, default is 4.
double alpha	Raised cosine / root raised cosine: 0.025 to 1 Gaussian filter: 0.15 to 2.5 The alpha parameter is only valid for raised cosine, root raised cosine, and Gaussian filtering.

10.10 am_profile

am_profile	
double rate	Modulation rate, in Hz.
double depth	Modulation depth, in %.
mod_shape_type shape	Modulation waveform. Refer to the definition of the mod_shape_type enumeration.

10.11 fm_profile

fm_profile	
double rate	Modulation rate, in Hz.
double deviation	Frequency deviation of the modulation, in Hz.
mod_shape_type shape	Modulation waveform. Refer to the definition of the mod_shape_type enumeration.

10.12 dsss_profile

dsss_profile	
double symbolrate	Symbol rate, in Hz.
mod_digitalmod_type digitalmod_type	Digital modulation type. Refer to the definition of the mod_digitalmod_type enumeration.
int32_t code	Spreading code, PN value, recommended value: 5.
int32_t filter_symbols	Number of filter symbols, must be an even number. Range: 2 to (sps × filter_symbols ≤ 400), default is 8.
int32_t sps	Samples per symbol, even number from 4 to 32, default is 4.
double alpha	Raised cosine / root raised cosine: 0.025 to 1 Gaussian filter: 0.15 to 2.5 The alpha parameter is only valid for raised cosine, root raised cosine, and Gaussian filtering.
int32_t sequenceseed	Initial seed of the pseudo-random sequence, range: 1 to 2,147,483,647.
mod_filter_type filter_type	Filter type. Refer to the definition of the mod_filter_type enumeration.

10.13 pulse_profile

pulse_profile	
double width	Pulse width, in seconds (s).
double period	Period, in seconds (s). The period must be greater than the pulse width.

10.14 multitone_profile

multitone_profile	
double tonephase	Phase, in the range [0, 2*pi].
int32_t tonecount	Number of tones.
double freqspacing	Frequency spacing, in Hz.

10.15 stepsweep_profile

stepsweep_profile	
double center	Center frequency, in Hz.
double span	Span, in Hz.
int32_t points	Number of points.
double dwelltime	Dwell time, in seconds (s).

10.16 rampsweep_profile

rampsweep_profile	
double span	Span, in Hz.
double sweeptime	Sweep time, in seconds (s).
double period	Period, in seconds (s).

10.17 ofdm_profile

ofdm_profile	
double idleinterval	Idle time between transmit bursts, in seconds (s).
int32_t fftsize	Number of FFT points, supporting 16, 32, 64, 128, 256, 512, 1024, and 2048.Additionally, fftsize > guardbandcarriers_left + guardbandcarriers_right.
int32_t symbolcount	Number of symbols.
int32_t guardbandcarriers_left	Number of guard carriers on the left side.
int32_t guardbandcarriers_right	Number of guard carriers on the right side.
int32_t guardinterval	Guard interval, in %.
int32_t window_length	Window length, in %.
uint8_t windowed	Whether windowing is enabled.
mod_digitalmod_type digitalmod_type	Digital modulation type. Note: only the following types are supported: DigitalModType_BPSK、DigitalModType_QPSK、DigitalModType_PSK8、DigitalModType_PSK16、DigitalModType_QAM16、DigitalModType_QAM64、DigitalModType_QAM256
uint8_t nulldc	Whether DC is nulled.
double samplerate	Sampling rate: 195.3125e3 to 125e6, default is 20 MHz.

10.18 awgn_profile

awgn_profile	
double bandwidth	Idle time between transmit bursts, in seconds (s).
int32_t length	Number of FFT points, supporting 16, 32, 64, 128, 256, 512, 1024, and 2048. Additionally, fftsize > guardbandcarriers_left + guardbandcarriers_right.

11. Enumeration Variables

11.1 state

STATE_OFF	Global configuration, 0.
STATE_ON	Global configuration, 1.

11.2 fan_mode

FAN_ON	Fan forced on mode.
FAN_OFF	Fan forced off mode.
FAN_AUTO	Automatically adjust fan state based on temperature

11.3 eth_interface_type

ETH_HIGHSPEED	High-speed Ethernet bus interface
ETH_STANDARD	Standard Ethernet bus interface

11.4 referenceclock_source

REFCLKSOURCE_INTERNAL	Internal reference clock source
REFCLKSOURCE_EXTERNAL	External reference clock source

11.5 lomode

LOMODE_AUTO	In this mode, the instrument automatically selects the optimal operating parameters based on the current output frequency, frequency step, and operating state.
LOMODE_LOWSPUR	This mode optimizes the phase noise performance of the local oscillator by adjusting the PLL control strategy to improve signal phase stability.
LOMODE_LOWPHASENOISE	This mode optimizes the local oscillator output spectrum, effectively suppressing fractional spurs and related spurious components to improve spectral purity.

11.6 channel_type

CHANNELTYPE_RX	Receive channel.
CHANNELTYPE_TX	Transmit channel.
CHANNELTYPE_VX	Vector network analysis channel.

11.7 trigger_action

TRIGGER_ACTION_HOP	Execute a single frequency hop.
TRIGGER_ACTION_SWEEP	Perform a full sweep.

11.8 trigger_edge

TRIGGER_EDGE_RISING	Rising edge trigger.
TRIGGER_EDGE_FALLING	Falling edge trigger.

11.9 trigger_source

TRIGGER_SOURCE_BUS	Bus trigger: under this trigger, trigger edges are ignored.
TRIGGER_SOURCE_EXTERNAL	External trigger.
TRIGGER_SOURCE_XPPS	GNSS 1PPS trigger.
TRIGGER_SOURCE_LEVEL	Level trigger (Rx only).
TRIGGER_SOURCE_FREQMASK	Spectrum template trigger (Rx only).

11.10 mod_digitalmod_type

DigitalModType_BPSK	BPSK
DigitalModType_DBPSK	DBPSK
DigitalModType_QPSK	QPSK
DigitalModType_DQPSK	DQPSK
DigitalModType_OQPSK	OQPSK
DigitalModType_Pi4DQPSK	Pi4DPSK
DigitalModType_PSK8	8PSK
DigitalModType_D8PSK	D8PSK
DigitalModType_PSK16	16PSK
DigitalModType_QAM16	16QAM
DigitalModType_QAM64	64QAM
DigitalModType_QAM256	256QAM
DigitalModType_QAM1024	1024QAM
DigitalModType_ASK2	2ASK
DigitalModType_FSK2	2FSK
DigitalModType_FSK4	4FSK
DigitalModType_FSK8	8FSK
DigitalModType_FSK16	16FSK
DigitalModType_ASK4	4ASK
DigitalModType_ASK8	8ASK
DigitalModType_APSK16	16ASK
DigitalModType_QAM8	8QAM

11.11 mod_filter_type

FilterType_Rectangular	Rectangular filter
FilterType_RaisedCosine	Raised cosine filter
FilterType_RootRaisedCosine	Root raised cosine filter
FilterType_Gaussian	Gaussian filter
FilterType_HalfSine	Half-sine filter

11.12 mod_shape_type

ModShape_Sine	Sine wave
ModShape_Square	Square wave
ModShape_Triangle	Triangle wave
ModShape_Ramp	Sawtooth wave

Appendix 1: API Return Value Index

Error Code	Error/Warning Reason	Type[1]	Handling
0	No error	-	No action required; subsequent operations can proceed normally.
-1	Bus open error	Error	Check device power and data cable connections. Verify that the driver is correctly installed. After resolving the issue, call device_open_usb again to open the device.
-8	Bus transmission error	Error	Reconnect the device. If the issue persists, please contact official technical support.
-29	Transmit power calibration file missing [2]	Error	Check whether the transmit power calibration file is placed in the designated directory. After resolving the issue, call device_open_usb again to open the device.
-43	IQ calibration file missing [2]	Error	Check whether the IQ calibration file is placed in the designated directory. After resolving the issue, call device_open_usb again to open the device.
10	Data transmission timeout	Warning	Wait 10 ms, then retry sending.
42	Using default transmit power calibration file	Warning	Does not affect device operation, but RF output power may deviate.
44	Using default IQ calibration file	Warning	Does not affect device operation, but accuracy may be affected.
98	Parameter out of range, clamped	Warning	The out-of-range parameters are automatically forced back into the device's valid range. The actual values at that time can be queried via the query function. This does not affect continued device operation.
99	Transmit power is too low, but still within limits	Warning	The power level may be inaccurate, but this does not affect continued device operation. Please contact official technical support.
100	Transmit power is too high, but still within limits	Warning	The power level may be inaccurate, but this does not affect continued device operation. Please contact official technical support.
15	System clock unlocked	Warning	The test results may be inaccurate, but this does not affect continued device operation. Please contact official technical support.
16	ADC clock unlocked	Warning	The test results may be inaccurate, but this does not affect continued device operation. Please contact official technical support.
18	RF local oscillator unlocked	Warning	The test results may be inaccurate, but this does not affect continued device operation. Please contact official technical support.

-601	Modulation license file missing	Error	Please place the xx._mod.lic file in the /CalFile/ folder.
-602	Modulation license file verification failed	Error	Usually due to incorrect license content; please contact official technical support.
-603	Device validation failed	Error	Device validation failed when creating the device object. Please check if the device is properly opened and contact official technical support.
-604	Object creation failed	Error	Object creation failed when creating the device object. Please check if the device is properly opened and contact official technical support.
-605	Object release failed	Error	Failed to release the device object. Please contact official technical support.
-606	Modulation type error	Error	When generating a signal, the input modulation type is incorrect. Please correct the modulation type, resend, and regenerate the signal.
-607	Modulation parameter error	Error	When generating a signal, the input modulation parameters are incorrect. Please correct the modulation parameters, resend, and regenerate the signal.
-608	Modulation library missing	Error	The modulation library is not found in the same path as the h2_api library. Please place genSignalWave.dll (Windows) or libgenSignalWave.so (Linux) in the same path as the h2_api library.
-609	Parameter cannot be configured	Error	The current parameter cannot be configured. The last available parameter has been applied.

[1]. If the type is "Error," the issue must be resolved immediately and the device reopened; otherwise, the device cannot continue with subsequent operations. If the type is "Warning," the device can continue with subsequent operations without closing or reopening, but it is still recommended to handle specific return values selectively based on the current application scenario.

[2]. For return values -29 and -43, it is also necessary to confirm that the file path contains only English characters. If the path contains Chinese characters, the API will report a file load failure.

www.harogic.com
info@harogic.com
+65-8299 8857