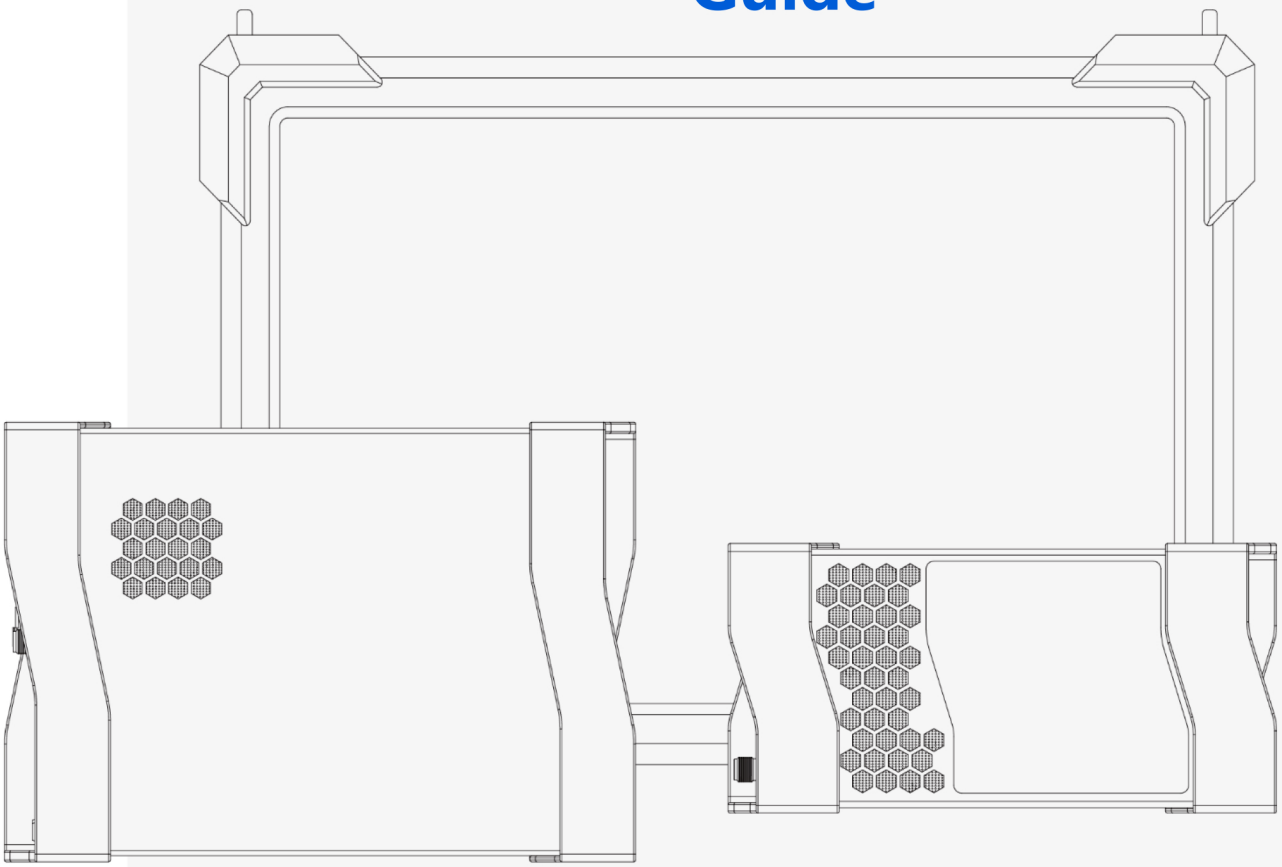
APPLICATION GUIDE

API Programming Guide



PX Series

SA Series

NX Series

V 0.55.73

2026-01-04

Content

1 Version Information

Version	Content	Time
0.55.61	1. Added: MPS mode; 2. Added: Add the AGC function in SWP/IQS/DET/RTA mode; 3. Added: Device_SetGNSSxpps, RTA_GetIQStream, RTA_TriggerStart, RTA_SetLookBackCmd, RTA_SetDataFormat functions; 4. Modified: The number of IQ points per packet retrieved by IQ_GetIQStream has been changed from 16242 to 16240.	2025-08-15
0.55.61	1. Added: MeasAuxInfo_TypeDef structure adds new IFAGCGain parameter.	2025-08-19
0.55.67	1. Added: DeviceState_TypeDef structure, MeasAuxInfo_TypeDef structure, MSCAN_Data_Typedef structure adds new nsSinceEpoch parameter. 2. Added: Device_SetIFAGCTarget, Device_ConvertEpochToReadablefunctions;	2025-09-30
0.55.72	1. Added: Add a delay parameter to Device_SetGNSSxpps; 2. Modified: Optimize the MSCAN function; 3. Added: Compatible with N400 series. 4. Modified: Content revision. 5. Modified: Modify RTA_BusTriggerStart to RTA_Trigger Start	2025-12-23
0.55.73	1. Added: Add Kaiser Window 2. Added: Add Fan Control	2026-01-04

2 Overview

This API system consists of dynamic linked libraries and header file. It is used for the secondary development of the real-time spectrum and receiver.

The measurement mode is a core concept of the API system. Different measurement modes have different test behavior and capability. As the first step of development, an appropriate measurement mode should be selected according to the task. The measurement modes of the the API system include the sweep mode (SWP), the IQ streaming mode (IQS), the power detection mode (DET), and the real time analysis mode (RTA). Fully understanding the mechanism of the measurement modes will help to maximize the devic's performance and obtain accurate measurement results.

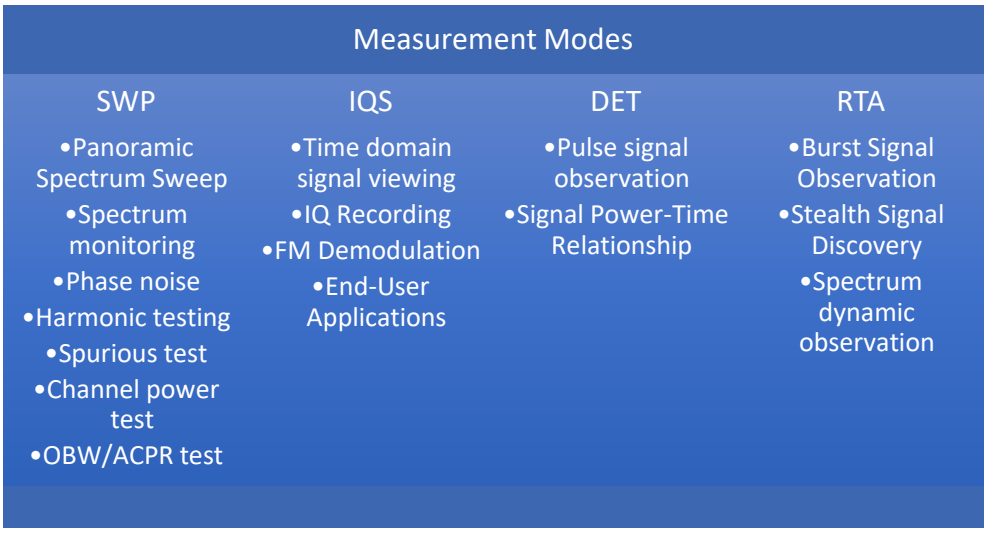


Figure 1 The Measurement Modes in API System and Main Application Scenarios

The basic flow of API calls consists of five steps: 1. open the device; 2. configurate the device to the specified measurement mode with appropriate parameter values; 3. obtain the measurement data; 4. execute user-defined process; 5. close the device.

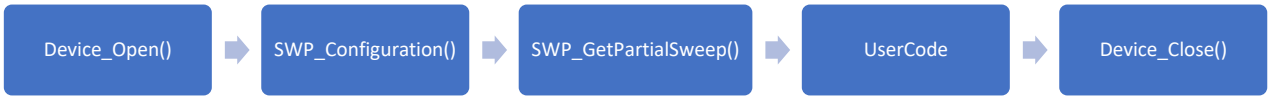


Figure 2 Typical Calling Steps for SWP Mode

Before starting your development, please read the chapter 5 API Call Logic and Call Map. This map can be the basic framework of your application and it is useful for building a robust and efficient application.

3 The Version of Device and API

The system is a joint operation of multiple softwares. In this system, the software involved includes 1) device master firmware (MFM), 2) FPGA firmware (FFM), 3) API, 4) SAStudio4; 5) other applications depend on user's requirements.

This system uses a unified software version naming rule to manage all the above software. We use the format of x.y.z to name software versions, where x is the primary version number, y is the secondary version number, and z is the sub-version number. The secondary version number y represents the compatibility mark of the software, and all software in the system must have the same y secondary version number to operate strictly and correctly.

For Example: MFM version = 0.55.1, FFM version = 0.55.2, API version = 0.55.4, SAStudio4 version = 1.55.45, this combination of all software with y = 55 will run matchingly.

For Example, if MFM version = 0.38.1, FFM version = 0.38.4, API version = 0.54.2, SAStudio4 version = 0.38.23, this combination where the API is an overrun version y = 54 and does not match the versions of MFM, FFM, SAStudio4, it will give an incorrect result.

Please note that the version of the API used corresponds to the version identified on the cover page of this brochure, so as to avoid any inconsistency between the description of this brochure and the actual API.

4 API Introduction of Function Categories

Table 1 Function Categories

Function Category	Description
Device and system	Global function, the functions under this category can be called in any measurement mode. including functions to turn on/off the device, set global settings, get device information and get device status.
SWP	In this mode, the receiver achieves the goal of frequency scanning performs via frequency hopping. Afterwards the baseband performs spectrum analysis on the time domain data within the analysis bandwidth acquired at each frequency point, and return the spectrum results to the user. SWP mode is suitable for frequency trace-oriented measurement and analysis applications. This category includes functions for configuration of SWP mode, acquisition of spectrum data, trigger control, etc.
IQS	In this mode, set the center frequency point to the specified frequency and keep the receiver state such as the local oscillation frequency fixed. The baseband acquires and Return time domain data within the analysis bandwidth to the user based on the specified trigger signal. IQS mode is suitable for signal recording, demodulation analysis, and simultaneous multi-dimensional analysis applications. This category includes functions for configuration of IQS mode, acquisition of spectrum data, and trigger control.
DET	In this mode, set the center frequency to the specified frequency and keeps the receiver state such as the local oscillation frequency fixed. The baseband performs (DET) on the time domain signal in the analysis bandwidth and return the power result to the user according to the specified trigger signal. DET mode is suitable for applications that are related with in-band power-time relationships, such as pulse parameter measurements. This category includes functions for configuring the DET mode, acquiring spectral data, and trigger control.
RTA	In this mode, the receiver sets the center frequency to the specified frequency and keeps the receiver state fixed such as the local oscillation frequency. The baseband performs continuous spectrum analysis of the time domain signal within the analysis bandwidth and return the spectrum results to the user. RTA mode is suitable for applications that focus on transient and burst signals, such as interference exclusion and identification of characteristic signals in complex electromagnetic environments. This category includes functions such as configuration of RTA mode, acquisition of spectrum data, and trigger control.

DSP	Generic post-processing functions, independent of hardware state. This category includes functions such as DDC, FFT analysis, and video detector for IQ data; and measurement and analysis of spectral traces, such as IM3, phase noise, channel power, and occupied bandwidth.
MSCAN	In MSCAN mode, the spectrum analyzer scans each frequency band sequentially and outputs the measurement results. It is suitable for application scenarios such as multi-band monitoring, interference detection, and automated testing. MSCAN is a multi-segment spectrum scanning function used to scan multiple frequency intervals in a single measurement process. Each scan segment can be independently configured with relevant measurement parameters such as Refercen Level, Decimation Factor, FFTSize, etc., allowing users to simultaneously observe the spectrum and acquire IQ data, thereby improving measurement efficiency.

5 API Call Logic and Call Map

5.1 API call map for standard sweep frequency analysis (SWP)

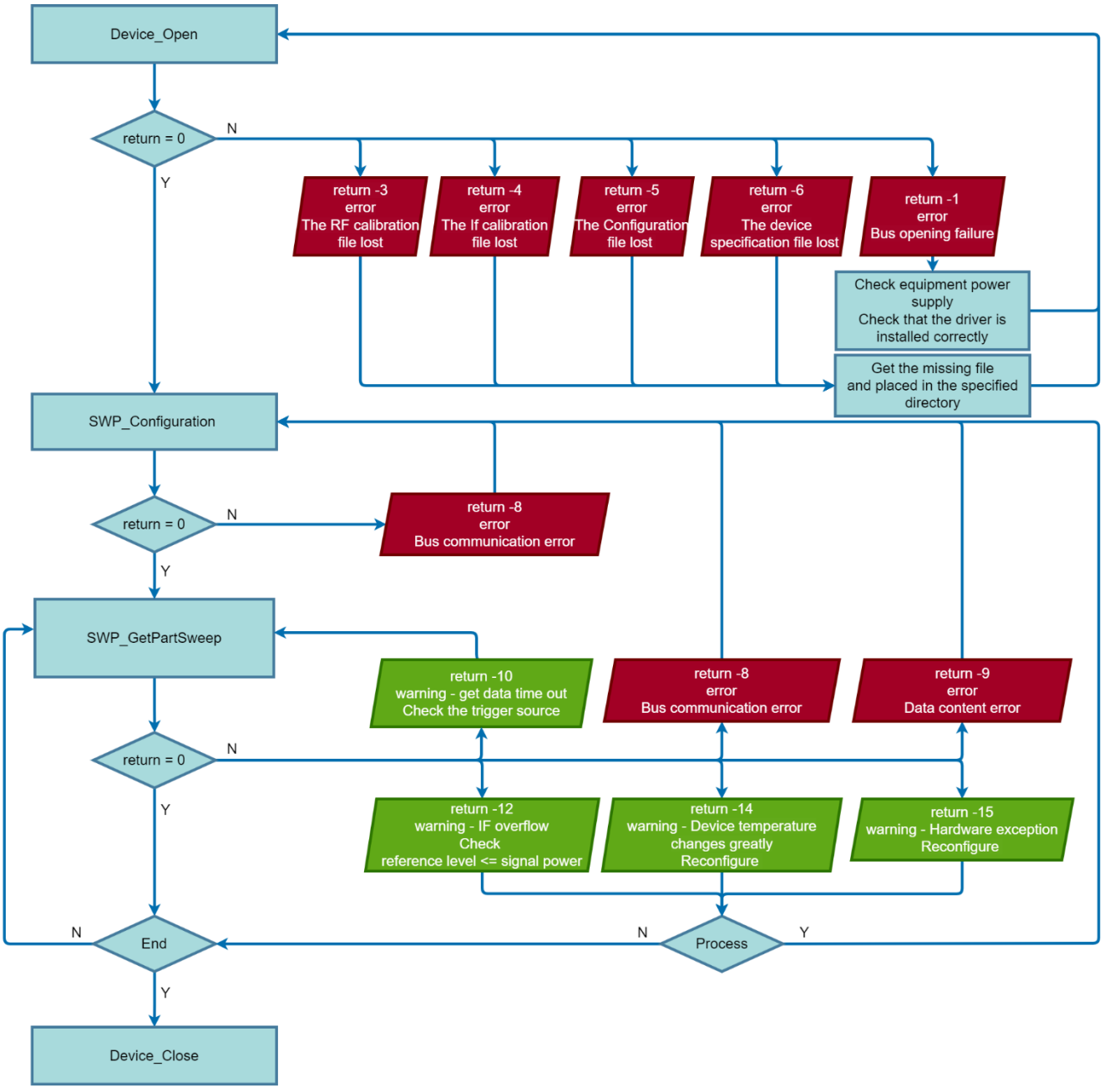


Figure 3 SWP Mode Call Flow Chart

5.2 API call map for IQ Streaming (IQS)

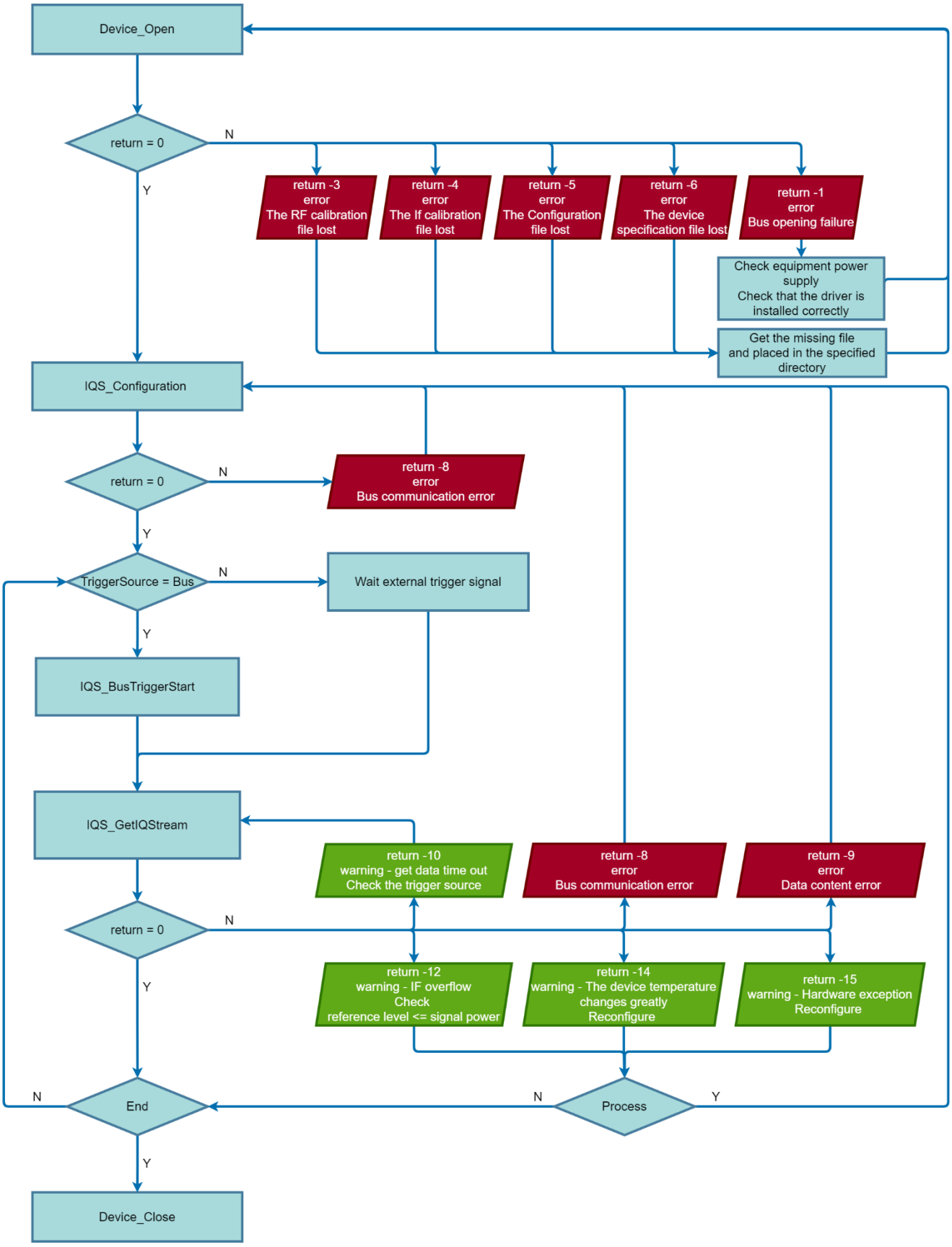


Figure 4 IQS mode Call Flow Chart (Trigger Mode is Fixed)

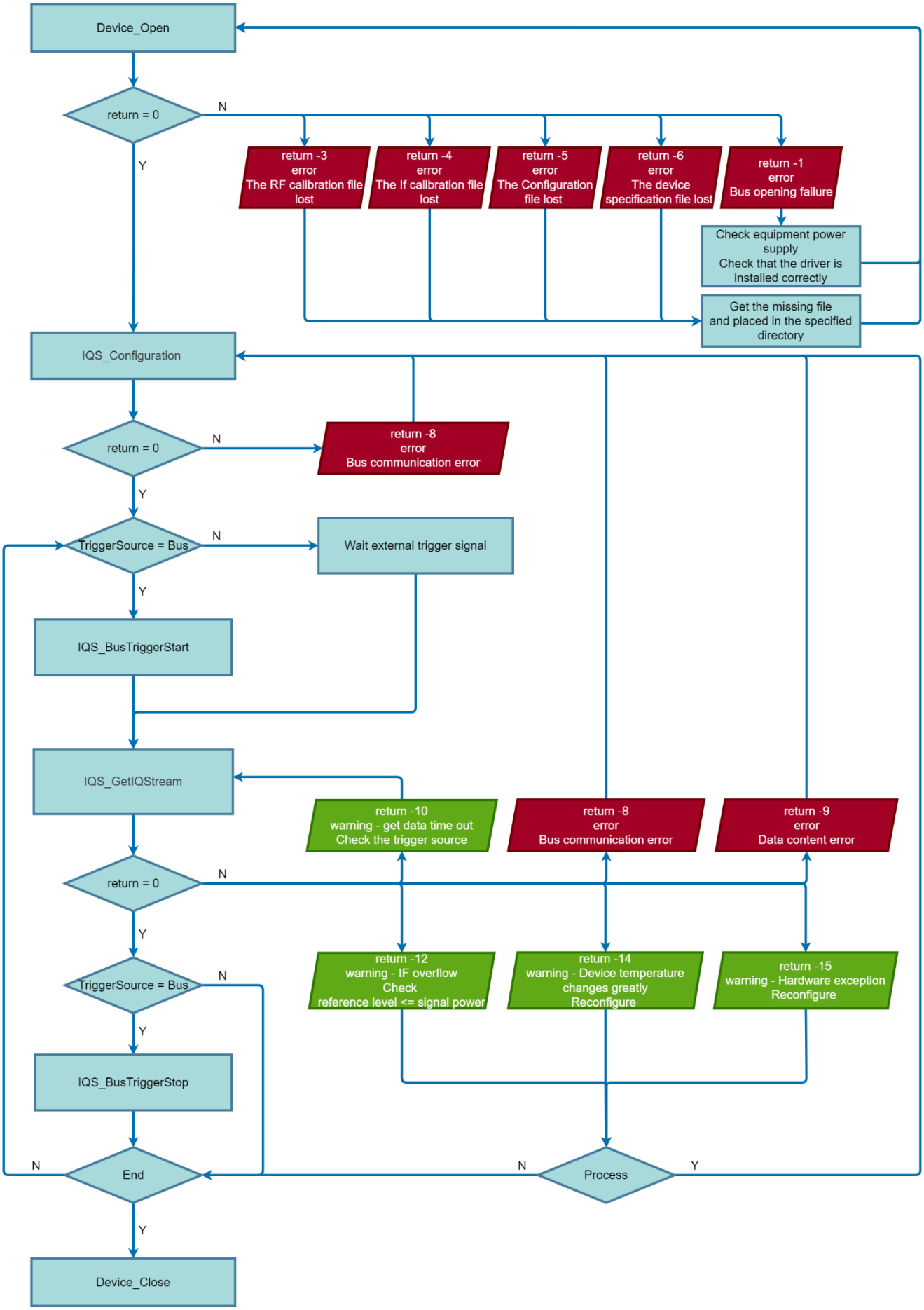


Figure 5 IQS Mode Call Flow Chart (Trigger Mode is Adaptive)

5.3 API call map for detection analysis (DET)

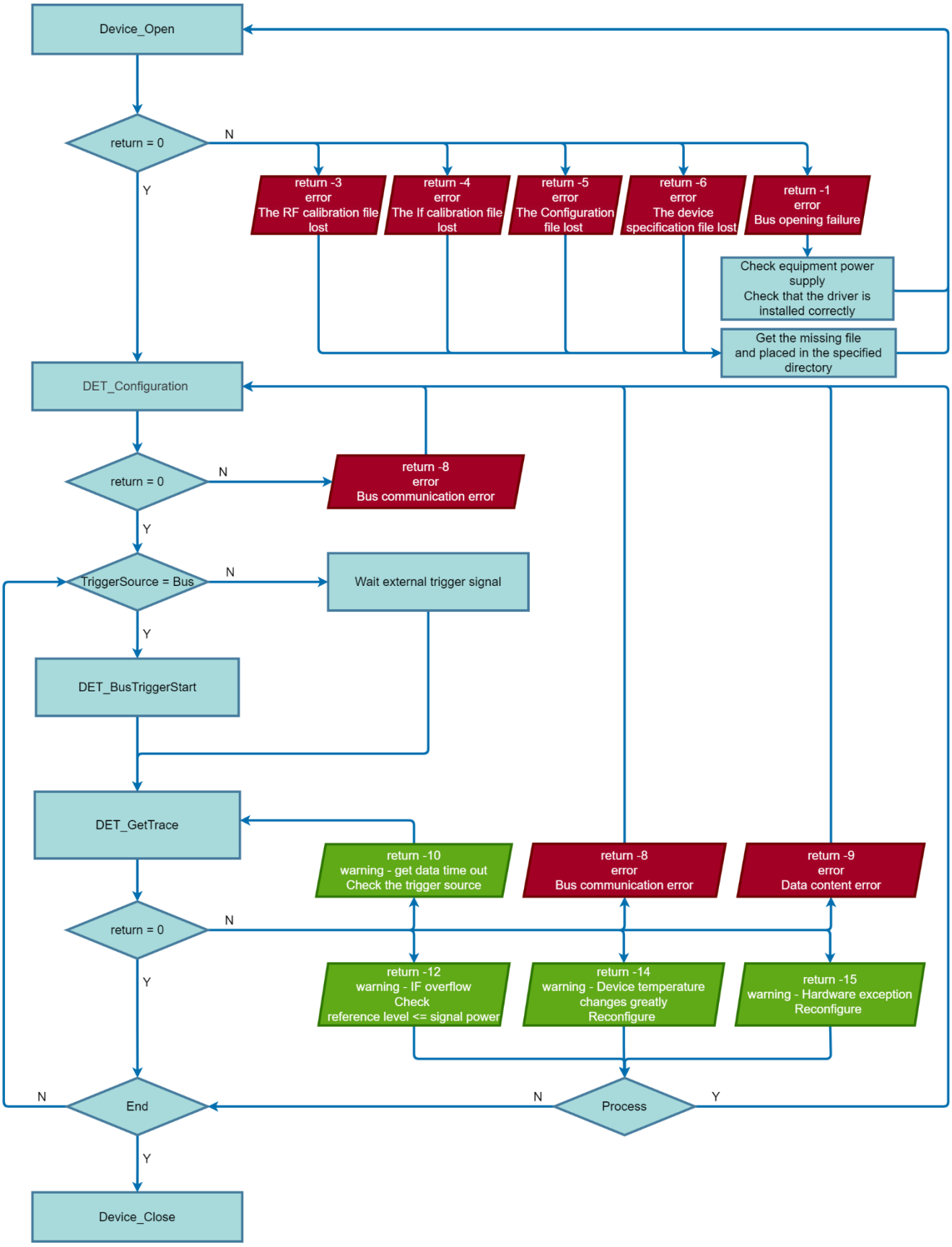


Figure 6 DET Mode Call Flow Chart (Trigger Mode is Fixed)

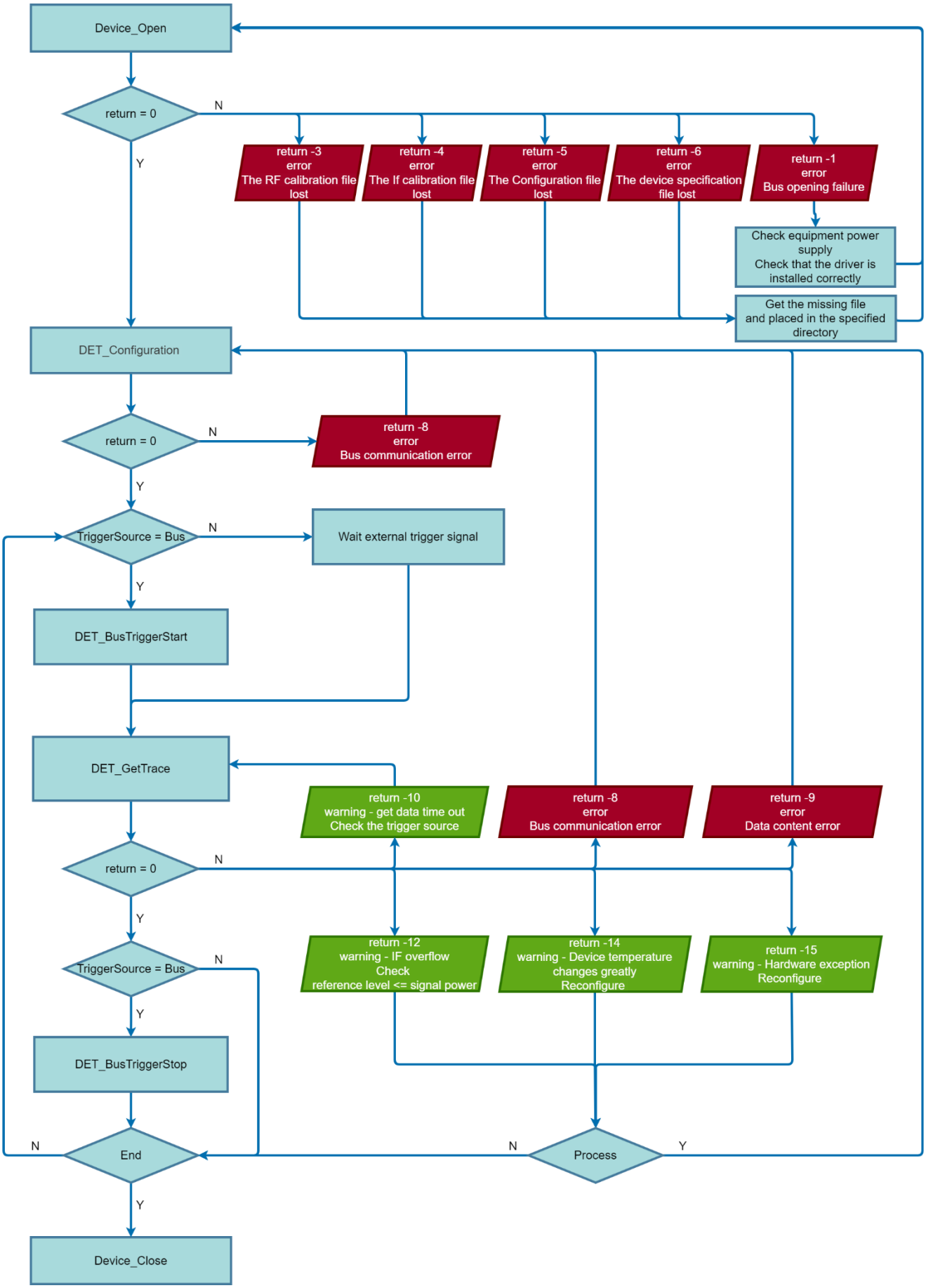


Figure 7 DET Mode Call Flow Chart (Trigger Mode is Adaptive)

5.4 API call map for Real-time Analysis (RTA)

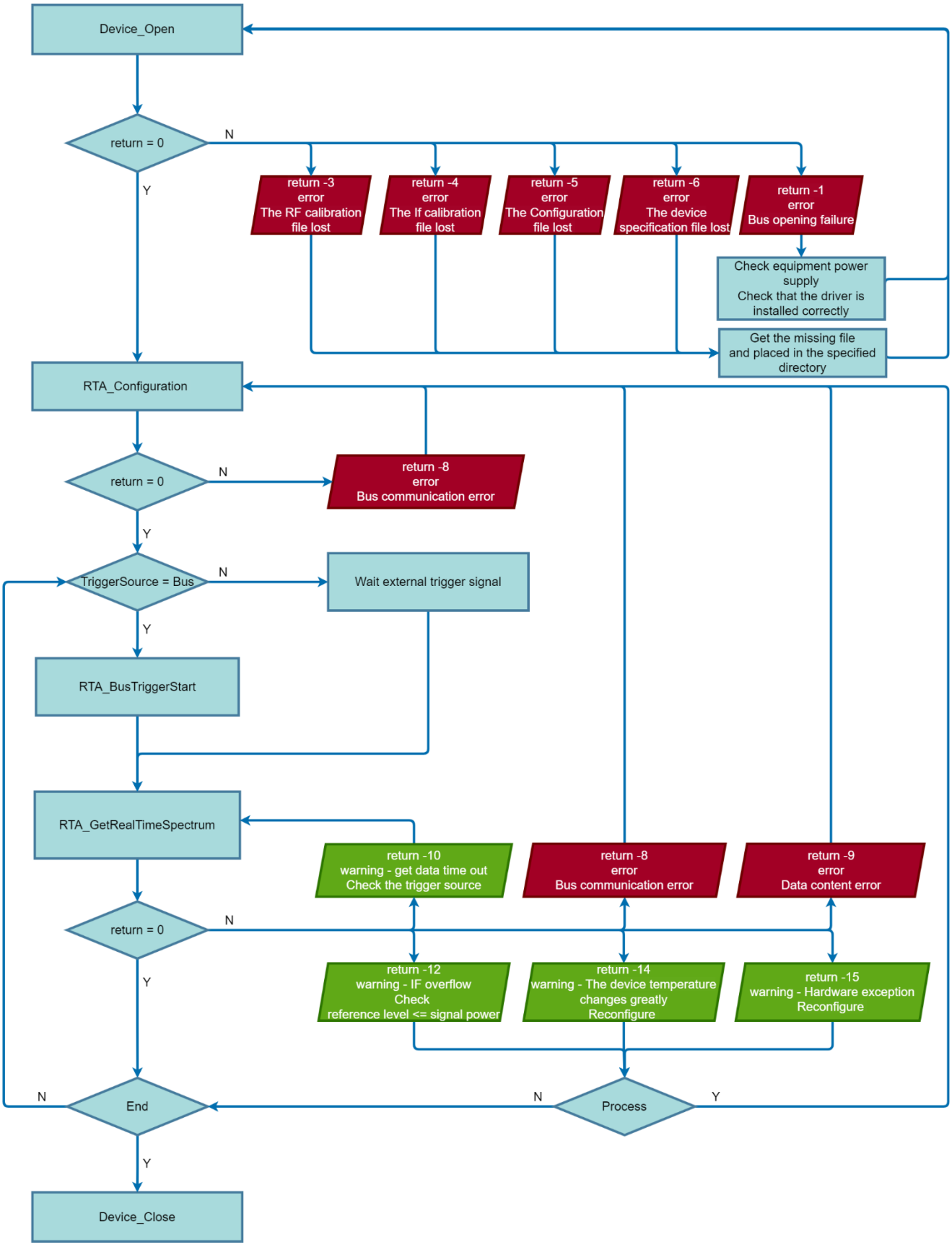


Figure 8 RTA Mode Call Flow Chart (Trigger Mode is Fixed)

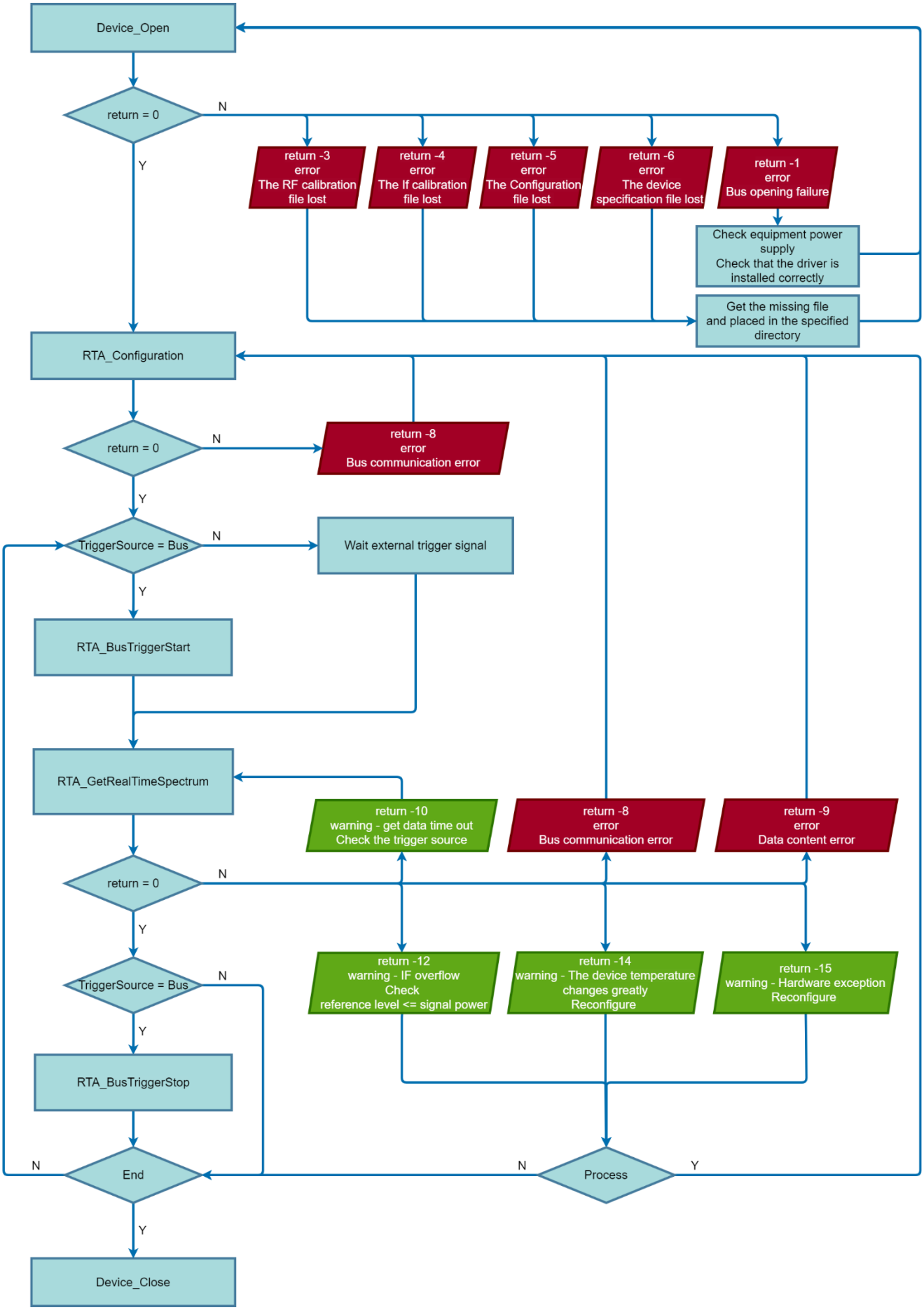


Figure 9 RTA Mode Call Flow Chart (Trigger Mode is Adaptive)

6 Important Variables Definition Reference

This section lists some of the important parameters involved in the spectrum analyzer/receiver devices. A good understanding of these parameters is important for proper use of the devices. They are summarized here for easy reference in case of preview or problems.

6.1 System

Table 2 System Parameters and Related Concepts Information Explanation Table

Sequence	Parameters	Applicable Mode	Description
1	Device memory pointer void** Device	Device/SWP/IQS/DET/ RTA	This parameter references the memory space required for device operation. When calling the API, you must use this reference to index the currently opened device.
2	DeviceUID	Device	Each device has a unique device ID, please use this ID to distinguish between different individual devices.

6.2 Amplititude

Table 3 Amplitude Parameters and Related Concepts Information Explanation Table

Sequence	Parameters	Applicable Mode	Description
1	RefLevel_dBm	SWP/IQS/DET/RTA	By default, the system automatically configures the attenuator and preamplifier based on the reference level. The reference level can be simply understood as the maximum input power that the system can accept without saturating. The system will keep a certain margin in handling the reference level, usually 1 - 6dB, so it is normal that the system will not report a saturation warning at some frequencies even if the input power is greater than the reference level. Setting the reference level according to the expected input power, so that the reference level is slightly higher than the expected maximum power of the input signal, you can generally get a good dynamic range. For example, if the expected signal is a monotone signal at -3dBm, setting the reference level to 0dBm will result in good observed dynamics.
2	Atten	SWP/IQS/DET/RTA	The default attenuation mode is automatic (Atten = -1), where the system determines channel attenuation solely based on the reference level. To manually set channel

			attenutation, specify a desired value for Atten.
3	Preamplifier	SWP/IQS/DET/RTA	For devices equipped with a preamplifier, enabling/disabling it significantly impacts system noise performance and linearity: 1) Enabling the preamplifier: reduces system noise; decreases maximum linear input power and damage threshold; 2) Disabling the preamplifier: increases maximum tolerable input power; results in higher system noise. The system typically supports: 1) Automatic preamp control based on reference level; 2) Manual override to force-disable the preamp (preventing overload damage).
4	AnalogIFBW Grade	SWP/IQS/DET/RTA	For devices equipped with multiple analog IF filters, the system provides multiple IF channels with different characteristics. Each analog IF bandwidth option exhibits distinct performance in out-of-band rejection, in-band flatness, and group delay. Select the appropriate IF bandwidth setting based on application requirements.
5	IFGainGrade	SWP/IQS/DET/RTA	The system allows users to adjust IF gain to optimize spurious performance, linearity, and noise levels. Higher gain settings provide greater IF amplification, typically in 1 dB to 3dB increments per step. Total system gain = RF gain + IF gain. When maintaining a constant reference level (fixed total gain): 1) Increasing IF gain -> Reduces mixer input power -> Imporves spurious suppression and linearity -> Degrades noise performance; 2) Decreasing IF gain -> Increases mixer input power -> Worsens spurious/linearity -> Improves noise performance. Special cases: 1) At maximum RF gain(e.g., ref. level = -60 dBm): Further IF gain increase boosts total gain, potentially improving noise performance; 2) Below maximum RF gain(e.g., ref. level = 0 dBm): Higher IF -> Better spurious/linearity, worse noise; LowerIF gain -> Worse spurious/linearity, better noise.

6.3 Frequency

Table 4 Frequency Parameters and Related Concepts Information Explanation Table

Sequence	Parameters	Applicable Mode	Description
1	FreqAssignment	SWP	In SWP mode, this variable allows users to define the frequency scan range either in StartStop or CenterSpan format.
2	StartFreq_Hz StopFreq_Hz CenterFreq_Hz Span_Hz		
3	TracePointStrategy	SWP	In SWP mode, the spectrum analysis method is determined by TracePointStrategy: 1) When TracePointStrategy = BinSizeAssined: The system uses sweep-based analysis, where the trace point count is explicitly defined by TracePoints. In this mode, TraceAlign settings are ignored; 2) For other TracePointStrategy values: The system employs FFT-based analysis. Due to underlying software implementation and trace detection mechanisms, the native ananlysis frequencies cannot be perfectly aligned with the configured start/stop frequencies. The returned trace data points will slightly exceed the specified. User-adjustable handling methods: 1) Truncate endpoint data to match the desired frequency range; 2) Set TraceAlign = AlignToStart to force alignment at the start frequency (end frequency still requires truncation). Note for FFT mode: While users can specify a desired trace point count (TracePoints), hardware limitations typically prevent exact matches. The system returns the closest achievable point count.
4	TracePoints		
5	TraceAlign		

6.4 Analysis

Table 5 Analysis Parameters and Related Concepts Information Explanation Table

Sequence	Parameters and Concepts	Applicable mode	Description
1	SpurRejection	SWP	The system provides three spurious suppression modes: Off, Standard, and Enhanced. This feature effectively suppresses most composite spurious components but does not improve system residual responses. It also reduces sweep speed and time-varying signal measurement capability. 1) For steady-state signals (e.g., CW tones): Enabling spurious suppression significantly improves spurious-free dynamic range; 2) For fast time-varying signals (e.g., modulated signals): Activation may cause intermittent signal loss or inaccurate power measurements. Use with caution. Recommendation: Toggle the feature while observing spectral changes to determine if spurious suppression is suitable for your test scenario.
2	PowerBalance	SWP	In SWP mode, users can balance scan speed and power consumption by configuring the Power Consumption Balance parameter: 1) Power Consumption Blance = 0: The system operates at maximum sweep speed (highest power draw); 2) Power Consumption Balance = 40-1000 (typical range): Higher values reduce scan speed and lower power consumption. Critical Note: Higher Power Consumption Balance values significantly degrade time-varying signal detection capability. Use caution when testing highly dynamic signals.
3	Window	SWP/RTA	When performing FFT-based spectrum analysis, the system provides multiple window functions, each with distinct advantages. Please select the appropriate window based on your testing requirements: 1) FlatTop Window: Provides excellent amplitude accuracy; Significantly reduces amplitude errors caused by the picket-fence effect; Ideal for high-precision amplitude measurements;

			2) Blackman-Nuttall Window: Features a narrow main lobe for high frequency resolution; Enables faster scanning speeds than FlatTop at the same RBW setting; Suitable for high-frequency-resolution and fast-sweep testing scenarios; 3) LowSideLobe Window: Features extremely low sidelobe levels, effectively suppressing interference from strong signals to adjacent frequencies. Ideal for test scenarios requiring high dynamic range or coexistence of strong and weak signals.
4	FFTExcutionStrategy	SWP	In standard spectrum analysis mode, users can select the signal processing method: 1) Auto mode: The system automatically selects FPGA or CPU processing based on RBW; 2) FPGA-Only: Significantly reduces CPU load; Slower sweep speeds ata RBW ⩽ 5KHz due to FFT size limitations; 3) CPU-Only: Supports FFT sizes > 64K points, enabling faster sweeps ata narrow RBW (⩽5KHz).
5	SweepTime	SWP/RTA	In this system, the sweep time is defined as the total time required to ccomplete one full scan from the start frequency to the stop frequency. When SweepTime = SWTMode_Manual, this parameter represents the absolute time; when *N is specified, this parameter is the scan time multiplier, i.e., scanning is performed at N times the minimum scan time.
6	DecimateFactor	IQS/DET/RTA	In IQS/DET/RTA mode, the system uses DecimateFactor to realize variable analysis bandwidth. Analysis bandwidth = Analysis bandwidth (DecimateFactor = 1) / DecimateFactor. Due to the limitations, the system will achieve the nearest available value for the DecimateFactor according to the desired DecimateFactor for configuration and feedback to the user.
7	BusTimeOut	IQS/DET/RTA	BusTimeOut sets an upper limit of execution time for functions related to fetching data, and the process will be ended if valid data cannot be fetched within that time so that the system does not wait indefinitely. In IQS/DET/RTA mode, this parameter needs to be set.

6.4.1 Detectors and Trace Detectors

Detector: Under the same local frequency point, the detector ratio frame data is collected, and according to the characteristics of the detector, the multi-frame data is detected frequency by frequency point, and the eigenvalue frame is finally generated. The following figure takes PosPeak Detector as an example to introduce the process of positive peak detection, where Frame 0, Frame 1 and Frame 2 are the data collected at different moments, and After PosPeak Detector is the data after positive peak detection.

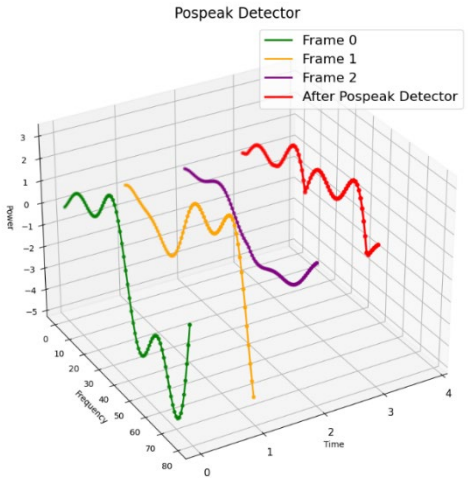


Figure 10 Positive peak detection

Trace Detector: According to the selected trace detector, the entire spectrum trace is detected in steps of trace detection ratio to generate the eigenvalue trace. The following figure takes PosPeak TraceDetector as an example to introduce the process of Positive Peak Trace Detector, where Before PosPeak Trace Detector is the data before Positive Peak Trace Detector and After PosPeak Trace Detector is the data after Positive Peak Trace Detector.

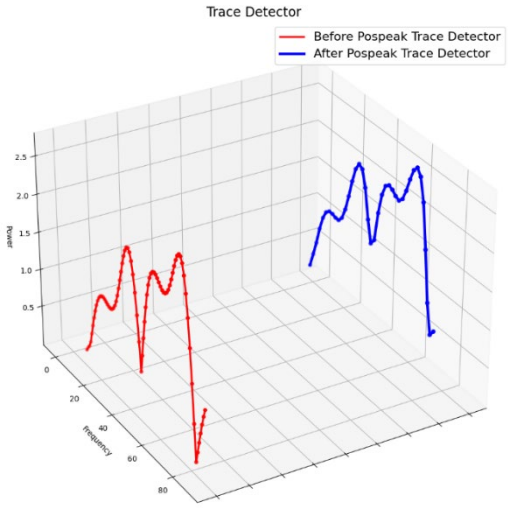


Figure 11 Positive peak trace detectors

6.5 Default Units

Table 6 Summary of Main Variables and Units

Variables	Unit
Frequency	Hz
Power	dBm
Voltage	V
Time	s

7 Device and System (main functions)

Before calling any API function associated with device hardware, you must call the Device_Open function to open the device. After completing your application's operations, call the Device_Close function to close the device and release memory space.

7.1 Device_Open

int Device_Open(void Device, int DeviceNum, const BootProfile_TypeDef* BootProfile, BootInfo_TypeDef* BootInfo)**	
Description	
The device must be opened before any API calls. Opening a device returns a handle. For multiple devices, open each with a different DeviceNum to get its handle, which is used in all subsequent API calls.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle. Required by all API calls to identify the target instrument.
int DeviceNum	Specifies the device index. When multiple instruments are connected, this index selects which device to open. The index starts at 0 and increments for each detected device.
BootProfile_TypeDef* BootProfile	Pointer to the boot configuration structure. Used to set the device's startup profile before initialization.
BootInfo_TypeDef* BootInfo	Pointer to the boot information structure. Returns status and diagnostic information after the device startup process.
BootProfile_TypeDef	
PhysicalInterface_TypeDef PhysicalInterface	Specifies the physical interface used to open the device. The correct interface must be selected; otherwise, the device cannot be accessed. A single instrument may support multiple interfaces. USB: Connects to the device via USB.
DevicePowerSupply_TypeDef DevicePowerSupply	Specifies the device's power supply mode. Must be correctly set to successfully open the device. 1) USBPortAndPowerPort: Powered via both USB data port and a separate power port 2) USBPortOnly: Powered only via the USB data port.
BootInfo_TypeDef	
DeviceInfo_TypeDef DeviceInfo	Returns the device information.
uint32_t BusSpeed	Indicates the bus speed or bandwidth of the connected device.
uint32_t BusVersion	Indicates the firmware version of the device's communication bus.

uint32_t APIVersion	Indicates the current API version. Encoded as major, minor, and revision: 1) Bits[31..16]: Major version 2) Bits[15..8]: Minor version 3) Bits[7..0]: Revision number
int ErrorCodes[7]	List of error codes generated during the device startup process. Each element represents a specific startup error or status.
int Errors	Total number of errors encountered during the device startup process.
int WarningCodes[7]	List of warning codes generated during the device startup process. Each element represents a specific startup warning.
int Warnings	Total number of warnings encountered during the device startup process.
DeviceInfo_TypeDef	
uint64_t DeviceUID	Unique identifier for the device.
uint16_t Model	Device model number.
uint16_t HardwareVersion	Hardware version of the device.
uint32_t MFWVersion	MCU firmware version of the device.
uint32_t FFWVersion	FPGA firmware version of the device.
uint16_t PMWVersion	PMU firmware version of the device.
uint16_t AGUVersion	AGU firmware verision of the device.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	This function must be called once before any other functions. The returned device memory address is used for subsequent operations. Under normal conditions, Device_Close must be called after use to release memory.
Example	Please refer to the Device_QueryDeviceState () function for related examples.

7.2 Device_Close

int Device_Close(void Device)**	
Description	
Shut down the device, and needs to be called to turn off the USB device and free up the memory space opened by the device in API calling.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Only need to call this function at the very end of program execution. After this function is called, the USB device connection will be closed and memory space will be released. If you need to use the device again, you'll

	need to re-establish the USB connection and open the device by calling Device_Open.
Example	Please refer to the Device_QueryDeviceState () function for related examples.

7.3 Device_QueryDeviceState

int Device_QueryDeviceState(void Device, DeviceState_TypeDef* DeviceState)**	
Description	
Get the device status information of the current spectrometer, including device temperature, hardware working status, geographic time information (option support is required), etc. In a non-real-time way, it does not interrupt the data acquisition, but the information is only updated after acquiring the data packet.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
DeviceState_TypeDef* DeviceState	Pointer to a structure containing the current device state information. After calling the function, the structure is updated with the latest device status.
DeviceState_TypeDef	
int16_t Temperature	Device temperature. The actual temperature in Celsius is calculated as 0.01 * Temperature.
double AbsoluteTimeStamp	Absolute timestamp representing the current time in seconds.
float Latitude	Latitude coordinate. Positive values indicate north latitude; negative values indicate south latitude.
float Longitude	Longitude coordinate. Positive values indicate east longitude; negative values indicate west longitude.
uint64_t nsSinceEpoch	Nanosecond-resolution timestamp of the current packet.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	
int Status = -1; int DeviceNum = 0; void* Device = NULL; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef BootInfo;** **Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);** **DeviceState_TypeDef DeviceState;** **Status = Device_QueryDeviceState (&Device, &DeviceState);**	

Status = Device_Close(&Device);

7.4 Device_SetGNSSxpps

int Device_SetGNSSxpps (void Device, uint8_t enableout, double xpps, double delay)**	
Description	
Sets the GNSS second pulse.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
uint8_t enableout	Enable/disable second pulse. 1) enableout=1: turn on second pulse; 2) enableout=0: turn off second pulse.
double xpps	Set the period of the second pulse. The range is: 0.25 to 10M.
double delay	Delay.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	
int Status = -1; int DeviceNum = 0; void* Device = NULL; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef BootInfo;** **Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);** **uint8_t Enableout = 1;double xpps= 1; double delay =0;** **Status = Device_SetGNSSxpps (&Device, Enableout, xpps, delay);** **Status = Device_Close(&Device);**	

7.5 Device_GetGNSSInfo

int Device_GetGNSSInfo(void Device, GNSSInfo_TypeDef* GNSSInfo)**	
Description	
When using the GNSS functionality, calling this function allows you to acquire the GNSS device status in a non-real-time manner (requires option support). The non-real-time method does not interrupt the data acquisition, but the information is only updated after the packet is acquired.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
GNSSInfo_TypeDef* GNSSInfo	GNSS to acquire device information.

GNSSInfo_TypeDef	
float latitude	Returns the latitude of the GNSS antenna.
float longitude	Returns the longitude of the GNSS antenna.
int16_t altitude	Returns the altitude of the GNSS antenna.
uint8_t SatsNum	Returns the number of satellites currently in use by the GNSS antenna.
uint8_t GNSS_LockState	Returns the GPS lock status.
uint8_t DOCXO_LockState	Returns the DOCXO lock status.
DOCXOWorkMode_TypeDef **DOCXO_WorkMode**	Returns the DOCXO operating status. 1) DOCXO_LockMode: Taming Mode; 2) DOCXO_HoldMode: Tracking Mode.
GNSSAntennaState_TypeDef **GNSSAntennaState**	Returns the antenna status. 1) GNSS_AntennaExternal:External antenna; 2) GNSS_AntennaInternal:Internal antenna.
int16_t hour	Returns the hour component from the GNSS time and date information.
int16_t minute	Returns the minute component from the GNSS time and date information.
int16_t second	Returns the second component from the GNSS time and date information.
int16_t Year	Returns the year component from the GNSS time and date information.
int16_t month	Returns the month component from the GNSS time and date information.
int16_t day	Returns the day component from the GNSS time and date information.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	
int Status = -1; int DeviceNum = 0; void* Device = NULL; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface =USB;** **BootInfo_TypeDef BootInfo;** **Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);** **GNSSInfo_TypeDef GNSSInfo;** **Status = Device_GetGNSSInfo (&Device, & GNSSInfo);** **Status = Device_Close(&Device);**	

7.6 Device_SetIFAGCTarget

int Device_SetIFAGCTarget (void Device, double* Target)**
Description
Set the IFAGC target power: Range: –30 to 0 dBFs. "Target" is the dBFs margin to ADC saturation.The new value takes effect only after xxx_Configuration is called again.

Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
double* Target	Setting the target power of IFAGC. Range: -30 to 0 dBFs.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.

7.7 Device_ConvertEpochToReadable

int Device_ConvertEpochToReadable (uint64_t nsSinceEpoch, 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	
Convert nsSinceEpoch to year, month, day, hour, minute, second, millisecond, microsecond, nanosecond.	
Compatibility	0.55.0 and later.
Parameter description	
uint64_t nsSinceEpoch	The nanosecond timestamp of the current packet.
int16_t* year	Pointer to the year component.
int16_t* mon	Pointer to the month component.
int16_t* day	Pointer to the day of the month.
int16_t* hour	Pointer to the hour.
int16_t* min	Pointer to the minute.
int16_t* sec	Pointer to the second.
int16_t* ms	Pointer to milliseconds.
int16_t* us	Pointer to microseconds.
int16_t* ns	Pointer to remaining nanoseconds.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.

7.8 Device_SetFanState

int Device_SetFanState(void Device, const FanState_TypeDef FanState, const float ThreshouldTemperature)**	
Description	
Controls the device fan operating mode.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.

const FanState_TypeDef FanState	Fan operating mode: 1) FAN_FORCED_ON: the system fan is forced on; 2) FAN_FORCED_OFF: the system fan is forced off; 3) FAN_AUTO: automatic mode.
const float ThreshouldTemperature	Threshold temperature for fan auto control. When the FanState is set to FAN_AUTO, the system fan will be automatically turned on if the threshold temperature is reached. The system fan will also be turned off if temperature is 10 celsius below the threshold.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	
int Status = -1; int DeviceNum = 0; void* Device = NULL; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef BootInfo;** **Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);** **FanState_TypeDef FanState = FanState_On;** **float Temperature = 0;** **Status = Device_SetFanState(&Device, FanState, Temperature);** **Status = Device_Close(&Device);**	

8 SWP Mode (main functions)

8.1 SWP_ProfileDeInit

int SWP_ProfileDeInit(void Device, SWP_Profile_TypeDef* UserProfile_O)**	
Function description	
Initializes the configuration parameter set for SWP mode (SWP_Profile_TypeDef). SWP_Profile_TypeDef defines all parameters of the device in SWP mode, including frequency, reference level, resolution bandwidth, and more.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
SWP_Profile_TypeDef* UserProfile_O	Outputs the default configuration parameter set.
SWP_Profile_TypeDef	
double StartFreq_Hz	Start frequency in Hz.
double StopFreq_Hz	Stop frequency in Hz.
double CenterFreq_Hz	Center frequency in Hz.
double Span_Hz	Frequency span in Hz.
double RefLevel_dBm	Reference level in dBm.
double RBW_Hz	Resolution bandwidth in Hz.
double VBW_Hz	Video bandwidth in Hz.
double SweepTime	When the sweep time mode is set to Manual, this parameter represents the absolute time. When it is set to *N, this parameter represents the sweep time multiplier.
SWP_FreqAssignment_TypeDef FreqAssignment	Specify the frequency assignment of the sweep: StartStop or CenterSpan. 1) StartStop: Specify sweep range by start and stop frequency; 2) CenterSpan: Specify sweep range by center frequency and span.
Window_TypeDef Window	Specify the window function used for FFT analysis: 1) FlatTop: good amplitude accuracy; 2) Blackman_ Nuttall: good frequency resolution; 3) LowSideLobe: low sidelobe window. 4) Rect: Rectangular window. 5) Kaiser: Kaiser window.
RBWMode_TypeDef RBWMode	Configure RBW Update Method: 1) RBW_Manual: Manually set the RBW value; 2) RBW_Auto: Automatically adjusts RBW based on SPAN; 3) RBW_OneThousandthSpan: enforces RBW = 0.001*SPAN;

	4) RBW_OnePercentSpan: enforces RBW = 0.01 * SPAN.
VBWMode_TypeDef VBWMode	Configure VBW Update Method: 1) VBW_Manual: manually specify the VBW value; 2) VBW_EqualToRBW: enforces VBW = RBW; 3) VBW_TenpercentRBW: enforces VBW = 0.1 * RBW; 4) VBW_OnePercentRBW: enforces VBW = 0.01 * RBW; 5) VBW_TenTimesRBW: enforces VBW = 10 * RBW, effectively bypassing the VBW filter.
SweepTimeMode_TypeDef SweepTimeMode	Sweep Time Mode Configuration: 1) SWTMode_minSWT: Performs sweep with the minimun sweep time; 2) SWTMode_minSWTx2: Performs sweep with approximately *2 the minimum sweep time; 3) SWTMode_minSWTx4: Performs sweep with approximately *4 the minimum sweep time; 4) SWTMode_minSWTx10: Performs sweep with approximately *10 the minimum sweep time; 5) SWTMode_minSWTx20: Performs sweep with approximately *20 the minimum sweep time; 6) SWTMode_minSWTx50: Performs sweep with approximately *50 the minimum sweep time; 7) SWTMode_minSWTxN: Performs sweep with approximately *N the minimum sweep time, where N = SweepTimeMultiple; 8) SWTMode_Manual: Performs sweep with the manually specified sweep time, where sweep time = SweepTime; 9) SWTMode_minSMPxN: Performs sampling ata each individual frequency point with a duration approximately N times the minimum sampling time, where N = SampleTimeMultiple.
Detector_TypeDef Detector	Set Detector 1) No inter-frame detection is applied. Each frequency bin outputs its raw power spectrum; 2) Detector_PosPeak: Performs inter-frame detection for each frequency bin and outputs one frame using MaxHold across frames; 3) Detector_Average: Performs inter-frame detection for each frequency bin and outputs one frame using the average across frames; 4) Detector_NegPeak: Performs inter-frame detection for each frequency bin and outputs one frame using MinHold across frames; 5) Detector_MaxPower: Prior to FFT, long-duration sampling is performed for each frequency bin. The frame with the highest power is

	selected for FFT. Used for capturing pulses and other transient signals (SWP mode only). 6) Detector_RawFrames: Multiple samples and FFT analyses are performed for each frequency bin, and power spectra are output frame by frame (SWP mode only); 7) Detector_RMS: Performs inter-frame detection for each frequency bin and outputs one frame using RMS across frames.
TraceFormat_TypeDef TraceFormat	Trace Format Configuration: 1) TraceFormat_Standard: frequency is evenly spaced; 2) TraceFormat_PrecisFrq: frequency is precise.
TraceDetectMode_TypeDef TraceDetectMode	Trace Detection Mode Configuration (Frequency Axis): 1) TraceDetectMode_Auto: Automatically selects the optimal trace detection mode; 2) TraceDetectMode_Manua: Uses the manually specified trace detection mode.
TraceDetector_TypeDef TraceDetector	Trace Detector Type Configuration: 1) TraceDetector_AutoSample: Automatic sample detection; 2) TraceDetector_Sample: Sample detection; 3) TraceDetector_PosPeak: Positive peak detection; 4) TraceDetector_NegPeak: Negative peak detection; 5) TraceDetector_RMS: Root means square detection; 6) TraceDetector_Bypass: No detection performed; 7) TraceDetector_AutoPeak: Automatic peak detection mode; 8) TraceDetector_Normal: Normal detection mode.
uint32_t TracePoints	The total points of sweep trace. The system will return the nearest available points according to the RBW and sweeping range.
TracePointsStrategy_TypeDef TracePointsStrategy	The strategy for setting the number of trace points: 1) SweepSpeedPreferred: Priority is given to faster sweep rate and return trace point is as close as possible to target trace points; 2) PointsAccuracyPreferred: Priority is given to ensuring that the actual number of trace points is close to the target trace points; 3) BinSizeAssigned: The priority is to ensure that the trace is generated at the set frequency interval.
TraceAlign_TypeDef TraceAlign	The strategy for setting trace aligning: 1) NativeAlign：Natural frequency alignment; 2) AlignToStart: Aligns to start frequency.
FFTExecutionStrategy_TypeDef	Specify the strategy for FFT executing:

FFTExecutionStrategy	1) Auto: automatically choose whether to use the CPU or FPGA; 2) Auto_CPUPreferred: automatically choose whether to use the CPU or FPGA, CPU first; 3) Auto_FPGAPreferred: automatically choose whether to use the CPU or FPGA for FFT calculations, FPGA first; 4) CPUOnly_LowResOcc: Mandatory use of CPU computing, low resource usage, Maximum number of FFT points: 256K; 5) CPUOnly_MediumResOcc: Mandatory use of CPU computing, medium resource usage, Maximum number of FFT points: 1M; 6) CPUOnly_HighResOcc: Mandatory use of CPU computing, high resource usage, Maximum number of FFT points: 4M; 7) FPGAOnly: Mandatory use of FPGA computing.
SpurRejection_TypeDef SpurRejection	Specify the strategy for spurious rejection: 1) Bypass: No spur suppression; 2) Standard: Medium-level spur suppression; 3) Enhanced: High-level spur suppression. The higher the spur suppression level, the slower the scan rate.
ReferenceClockSource_TypeDef ReferenceClockSource	Specify the referenc clock: 1) ReferenceClockSource_Internal: the internal reference clock (10MHz as default) is used; 2) ReferenceClockSource_External: the external reference clock (10MHz as default) is used, and if the external reference clock can not be locked, it will automatically switch to the internal reference clock; 3) ReferenceClockSource_Internal_Premium: the internal high quality (DOCXO or OCXO) clock source is used; 4) ReferenceClockSource_External_Forced: the external reference clock (10MHz as default) is used and will not change to the internal source even if it can not be locked.
double ReferenceClockFrequency	Specify the reference clock frequency in Hz. It allows user to adjust frequency accuracy manually.
uint8_t EnableReferenceClockOut	Specify the reference clock output enable.
SWP_TriggerSource_TypeDef TriggerSource	Sweep trigger source for RF receivers: 1) InternalFreeRun: Internal trigger free run; 2) ExternalPerHop: External trigger, each trigger jumps one frequency point; 3) ExternalPerSweep: External trigger, each trigger refreshes a trace.
TriggerEdge_TypeDef TriggerEdge	Specify the trigger edge:

	1) RisingEdge: rising edge is effective; 2) FallingEdge: falling edge is effective; 3) DoubleEdge: both rising edge and falling edge are effective.
TriggerOutMode_TypeDef **TriggerOutMode**	Set the trigger output mode: 1) None: trigger out is disabled; 2) PerHop: a tirgger pluse will be sent once a frequency hop is completed; 3) PerSweep: a tirgger pluse will be sent once a full trace sweep is completed; 4) PerProfile: swtich output with each configuration change.
TriggerOutPulsePolarity_TypeDef **TriggerOutPulsePolarity**	Specify the pulse polarity of the output trigger: 1) Positive: positive pulse is used; 2) Negative: negative pulse is used.
uint32_t PowerBalance	Set dynamic power consumption control in SWP mode. The typical range is 0 to 5000, increasing this value will reduce the power consumption of the device but also slow down the sweep speed.
GainStrategy_TypeDef GainStrategy	Specify the gain strategy of the receiver: 1) LowNoisePreferred: optimized for low noise; 2) HighLinearityPreferred: optimized for high linearity.
PreamplifierState_TypeDef **Preamplifier**	Set the state of the preamplifier: 1) AutoOn: the preamplifier will be automatically enabled according to the reference level and atten; 2) ForcedOff: the preamplifer will be kept off regardless the reference level and atten.
uint8_t AnalogIFBWGrade	Set the analog intermediate frequency (IF) bandwidth grade.
uint8_t IFGainGrade	Set the intermediate frequency (IF) gain grade.
int8_t Atten	Set the attenuation of receiver in dB. If the atten is set to -1, it will be ignored and the receiver will set gain state according to the reference level only. For value larger than -1, it has a higher priority than the reference level.
uint8_t EnableIFAGC	Enable or disable the IF AGC: 1) 0: disabled; manual gain control (MGC) is used. 2) 1: enabled; reduces IF gain when saturation occurs to prevent clipping.
SWP_TraceType_TypeDef TraceType	Specify the trace type: 1) ClearWrite: the trace will be updated with clear and write method; 2) MaxHold: the trace will be updated with max hold method; 3) MinHold: the trace will be updated with max hold method; 4) ClearWriteWithIQ: the trace will be updated with clear and write

	method. The corresponding IQ data of each spectrum frame will also be attached.
LOOptimization_TypeDef Looptimization	Specify LO optimization: 1) LOOpt_Auto: Automatic LO optimization; 2) LOOpt_Speed: High sweep speed optimization; 3) LOOpt_Spur: Low spurious optimization; 4) LOOpt_PhaseNoise: Low phase noise optimization.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the SWP_GetPartialSweep() function for related examples.

8.2 SWP_Configuration

int SWP_Configuration(void Device, const SWP_Profile_TypeDef* ProfileIn, SWP_Profile_TypeDef* ProfileOut, SWP_TraceInfo_TypeDef* TraceInfo)**	
Description	
Configure the spectrometer device to SWP working mode and related parameters Parameters such as frequency, reference level, resolution bandwidth and other parameters in SWP mode are uniformly encapsulated in the SWP_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const SWP_Profile_TypeDef* SWP_ProfileIn	Configuration Profile Input. Refer to the detailed definition of the structure parameter used in the SWP_ProfileDeInit() function.
SWP_Profile_TypeDef* SWP_ProfileOut	Configuration Profile Output. Refer to the detailed definition of the structure parameter used in the SWP_ProfileDeInit() function.
SWP_TraceInfo_TypeDef* TraceInfo	The information about the trace under the current configuration.
SWP_TraceInfo_TypeDef	
int FullsweepTracePoints	The points count of a full trace.
int PartialsweepTracePoints	The points count of each trace segment which can be fetched by single SWP_GetPartialSweep calling.
int TotalHops	The total hops for a full trace.
uint32_t UserStartIndex	The array index in the trace corresponding to the user-specified StartFreq_Hz. When HopIndex = 0, Freq[UserStartIndex] is the frequency point closest to SWPProfile.StartFreq_Hz.
uint32_t UserStopIndex	The array index in the trace corresponding to the user-specified

	StopFreq_Hz. When HopIndex = TotalHops - 1, Freq[UserStopIndex] is the frequency point closest to SWPProfile.StopFreq_Hz.
double TraceBinBW_Hz	The interval frequency between two points of the trace.
double StartFreq_Hz	The frequency of the first point in the trace.
double AnalysisBW_Hz	Analysis bandwidth for each frequency point.
int TraceDetectRatio	The Detection Ratio for Video Detector
int DecimateFactor	The decimation factor for time-domain data.
float FrameTimeMultiple	Frame analysis time multiplier. The analysis time at a single frequency point = default analysis time (set by the system) * frame time multiplier. Increasing the multiplier will increase the device's minimum sweep time, though not in a strictly linear manner.
double FrameTime	Frame sweep time: The signal duration (in second) used for a single-frame FFT analysis.
double EstimateMinSweepTime	The EstimateMinSweepTime is estimated minimum time for completing a full sweep. It is mainly affected by Span, RBW, VBW, frame time.
DataFormat_TypeDef DataFormat	Time-domain data format. Output data format for IQ data: 1) Complex8bit: complex data in 8-bit format; 2) Complex16bit: complex data in 16-bit format; 3) Complex32bit: complex data in 32-bit format; 4) Real16bit: Real, single-channel data 16-bit; 5) Real8bit: Real, single-channel data-8bit; 6) Realfloat: Real, single-channel data 32bit float.
uint64_t SamplePoints	Time-domain data sampling length.
uint32_t GainParameter	Gain-related parameters include Space (31 - 24 Bit), PreAmplifierState (23 - 16 Bit), StartRFBand (15 - 8 Bit), and StopRFBand (7 - 0 Bit).
DSPPlatform_Typedef DSPPlatform	DSP Processing Platform Configuration: 1) CPU_DSP: Computation performed on CPU; 2) FPGA_DSP: Computation performed on FPGA.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Msut be called after SWP_ProfileDelnit.
Example	Please refer to the SWP_GetPartialSweep() function for related examples.

8.3 SWP_GetPartialSweep

int SWP_GetPartialSweep(void Device, double Freq_Hz[], float PowerSpec_dBm[], int***

HopIndex, int* FrameIndex, MeasAuxInfo_TypeDef* MeasAuxInfo)	
Description	
Obtain the spectral results obtained at each frequency hopping point in SWP mode, and return the frequency hopping point sequence number, frame sequence number and auxiliary information of the measurement data, so that the user can stitch the results of multiple scans into the entire spectrum curve and return the function status.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
double Freq_Hz[]	Frequency Array Returned. The array size is equal to the TraceInfo.PartialSweepTracePoints.
float PowerSpec_dBm[]	Amplitude Array Returned. The array size is equal to the TraceInfo.PartialSweepTracePoints.
int* HopIndex	Returns the frequency hop index of the data.
int* FrameIndex	Returns the frame index of the data.
MeasAuxInfo_TypeDef* MeasAuxInfo	Returns the auxiliary information of the data.
MeasAuxInfo_TypeDef	
uint32_t MaxIndex	The index of the maximum power value within the data packet.
float MaxPower_dBm	The maximum power in the packet.
int16_t Temperature	The temperature of the device. Degree Celsius = 0.01 * Temperature.
double SysTimeStamp	System timestamp in second provided by the insystem timer.
double AbsoluteTimeStamp	Absolute timestamp provided by the insystem GNSS.
float Latitude	The latitude provided by the insystem GNSS.
float Longitude	The longitude provided by the insystem GNSS.
float Altitude	Altitude corresponding to the current data packet.
float SATHealth	Health status (SNR) of the current GNSS positioning satellites.
double IFAGCGain	Current IF AGC gain, in dB. Positive values indicate amplification, while negative values indicate attenuation.
double RefclkFreqOffset	The unit is ppm, representing the offset of the current device reference clock frequency, with the GNSS frequency as the reference.
uint64_t nsSinceEpoch	The nanosecond timestamp of the current packet.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Msut be called after SWP_Configuration.
Example	
int Status = -1; int DeviceNum = 0; void* Device = NULL; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;**	

```
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn,SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.StartFreq_Hz = 9e3;
SWP_ProfileIn.StopFreq_Hz = 6.35e9;
SWP_ProfileIn.RBWMode = RBW_Manual;
SWP_ProfileIn.RBW_Hz = 200e3;
Status = SWP_Configuration(&Device, &SWP_ProfileIn, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
int HopIndex = 0, FrameIndex = 0; MeasAuxInfo_TypeDef MeasAuxInfo;
for (int i = 0; i < TraceInfo.TotalHops; i++) {
Status = SWP_GetPartialSweep(&Device, Frequency.data() + i * TraceInfo.PartialsweepTracePoints,
PowerSpec_dBm.data() + i * TraceInfo.PartialsweepTracePoints, &HopIndex, &FrameIndex, &MeasAuxInfo); }
Device_Close(&Device);
```

8.4 SWP_GetFullSweep

int SWP_GetFullSweep(void Device, double Freq_Hz[], float PowerSpec_dBm[], MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
Obtain the results of an entire spectrum in SWP mode and auxiliary information about the measurement data, and return the function status at the same time.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
double Freq_Hz[]	The frequency array of the measurement data. The array size is equal to the TraceInfo.FullsweepTracePoints.
float PowerSpec_dBm[]	The power array of the measurement data. The array size is equal to the TraceInfo.FullsweepTracePoints.
MeasAuxInfo_TypeDef* MeasAuxInfo	Returns the auxiliary information of the data. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after SWP_Configuration.
Example	

```
int Status = -1;int DeviceNum = 0;void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
Status = Device_Close(&Device);
```

9 IQS Mode (main functions)

9.1 IQS_ProfileDeInit

int IQS_ProfileDeInit(void Device, IQS_Profile_TypeDef* UserProfile_O)**	
Description	
This function initializes the configuration profile of the IQS mode. IQS_Profile_TypeDef defines all parameters in IQS mode such as center frequency, reference level, decimate factor, etc.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
IQS_Profile_TypeDef* UserProfile_O	Pointer to the IQS configuration structure, used as an input/output variable.
IQS_Profile_TypeDef	
double CenterFreq_Hz	Center frequency in Hz.
double RefLevel_dBm	Reference level in dBm.
uint32_t DecimateFactor	Decimate factor. Effective analysis bandwidth = raw analysis bandwidth /decimate factor.
uint32_t BusTimeout_ms	Set the data transfer timeout in ms. The funcitions for data fetching will return an error if it fails to fetch data within the ButTimeOut.
IQS_TriggerSource_TypeDef TriggerSource	Specify the trigger source in the IQS mode: 1) External: Triggered by a physical signal connected to a trigger input port outside the device. 2) Bus: Triggered by means of a function (instruction); 3) Level: The device detects the input signal according to the set level threshold, and triggers automatically when the input exceeds the threshold; 4) Timer: Use the device internal timer to automatically trigger the set time period; 5) TxSweep: Data acquisition triggered by scanning from the device's internal signal source (ASG option required) When this trigger source is selected, the acquisition process will be triggered by the output trigger signal of the signal source scanning; 6) MultiDevSyncByExt: When the external trigger signal arrives, multiple machines perform synchronous trigger behavior; 7) MultiDevSyncByGNSS1PPS: On the arrival of GNSS-1PPS, the multicomputer does a synchronized trigger behavior (GNSS module option

	required); 8) GNSS1PPS: triggered by the 1PPS of the insystem GNSS (GNSS module option required).
TriggerEdge_TypeDef TriggerEdge	Specify the trigger edge: 1) RisingEdge: rising edge is effective; 2) FallingEdge: falling edge is effective; 3) DoubleEdge: both rising edge and falling edge are effective.
TriggerMode_TypeDef TriggerMode	Specify the trigger mode: 1) FixedPoint: data with a length specified by the TriggerLength will be acquired once the device is triggered; 2) Adaptive: After triggering, data acquisition continues until a termination signal is received (externally triggered termination edge or bus-triggered termination command).
uint64_t TriggerLength	When TriggerMode is set to FixedPoints, the trigger length specifies the total points of the data to be acquired for each trigger.
double TriggerLevel_dBm	When the TriggerSource is set to Level. The TriggerLevel specifies the threshold level of the trigger in dBm.
double TriggerLevel_SafeTime	When the TriggerSource is set to Level. The TriggerLevel_SafeTime specifies the saft time for a trigger in second. If the trigger signal can not maintain the active level for this time, the trigger will not be executed.
double TriggerDelay	Set the trigger delay in seconds. After a trigger occurs, the trigger action will be executed after this delay.
double PreTriggerTime	Set the pre-trigger time in seconds. After a trigger occurs, the data within this pre-trigger duration before the trigger action will also be included in the triggered data.
TriggerTimerSync_TypeDef TriggerTimerSync	Set the synchronization of the trigger timer: 1) NoneSync: no synchronization; 2) SyncToExt_RisingEdge: the timer will be continuously synchronized by the rising edge of external trigger; 3) SyncToExt_FallingEdge: the timer will be continuously synchronized by the falling edge of external trigger; 4) SyncToExt_SingleRisingEdge: after a calling of functioin IQS_SyncTimer_Single, the timer will be ready to be synchronizated and will be synchronized for a once by the rising edge of external trigger; 5) SyncToExt_SingleFallingEdge: after a calling of functioin IQS_SyncTimer_Single, the timer will be ready to be synchronizated and will be synchronized for a once by the falling edge of external trigger;

	6) SyncToGNSS1PPS_RisingEdge: the timer will be continuously synchronized by the rising edge of the 1PPS from the insystem GNSS; 7) SyncToGNSS1PPS_FallingEdge: the timer will be continuously synchronized by the falling edge of the 1PPS from the insystem GNSS; 8) SyncToGNSS1PPS_SingleRisingEdge: after a calling of functioin IQS_SyncTimer_Single, the timer will be ready to be synchronizated and will be synchronized for a once by the rising edge of the 1PPS from the insystem GNSS; 9) SyncToGNSS1PPS_SingleFallingEdge: after a calling of functioin IQS_SyncTimer_Single, the timer will be ready to be synchronizated and will be synchronized for a once by the falling edge of the 1PPS from the insystem GNSS.
double TriggerTimer_Period	Specify the period of timer trigger in second. It is effective when the TriggerSource is set to Timer.
uint8_t EnableReTrigger	Auto retrigger: Enables the device to perform subsequent timed self-triggers after the initial trigger source is activated. For example, after each external trigger, the device can automatically trigger three more times at 1 ms intervals. This function is effective only when TriggerMode = FixedPoint.
double ReTrigger_Period	Specify the retrigger period in second.
uint16_t ReTrigger_Count	Specify the counts of the retrigger after initial trigger.
DataFormat_TypeDef DataFormat	Output data format for IQ data: 1) Complex8bit: complex data in 8-bit format; 2) Complex16bit: complex data in 16-bit format; 3) Complex32bit: complex data in 32-bit format; 4) Real16bit: Real, single-channel data 16-bit; 5) Real8bit: Real, single-channel data-8bit; 6) Realfloat: Real, single-channel data 32bit float.
GainStrategy_TypeDef GainStrategy	Specify the gain strategy of the receiver: 1) LowNoisePreferred: optimized for low noise; 2) HighLinearityPreferred: optimized for high linearity.
PreamplifierState_TypeDef Preamplifier	Set the state of the preamplifier: 1) AutoOn: the preamplifier will be automatically enabled according to the reference level and atten; 2) ForcedOff: the preamplifer will be kept off regardless the reference level and atten.
uint8_t AnalogIFBWGrade	Specify the grade of analog IF bandwidth.

uint8_t IFGainGrade	Specify the gain grade of the IF. Larger grade leads to a higher IF gain.
ReferenceClockSource_TypeDef ReferenceClockSource	Specify the referenc clock: 1) ReferenceClockSource_Internal: the internal reference clock (10MHz as default) is used; 2) ReferenceClockSource_External: the external reference clock (10MHz as default) is used, and if the external reference clock can not be locked, it will automatically switch to the internal reference clock; 3) ReferenceClockSource_Internal_Premium: the internal high quality (DOCXO or OCXO)clock source is used; 4) ReferenceClockSource_External_Forced: the external reference clock (10MHz as default) is used and will not change to the internal source even if it can not be locked. 5) ReferenceClockSource_External_SysClock: External system clock. Uses the external clock input directly as the system clock, bypassing the internal reference PLL.
double ReferenceClockFrequency	Specify the reference clock frequency in Hz.
uint8_t EnableReferenceClockOut	Specify the reference clock output enable.
double NativeIQSampleRate_SPS	Set the native IQ sampling rate. The device can adjust the sampling rate by modifying this parameter. If changing this parameter has no effect, it indicates that the device does not support adjusting the sampling rate.
uint8_t EnableIFAGC	Enable or disable the IF AGC: 1) 0: disabled; manual gain control (MGC) is used. 2) 1: enabled; reduces IF gain when saturation occurs to prevent clipping.
int8_t Atten	Set the attenuation of receiver in dB. If the atten is set to -1, it will be ignored and the receiver will set gain state according to the reference level only. For value larger than -1, it has a higher priority than the reference level.
DCCancelerMode_TypeDef DCCancelerMode	Set the operating mode of DC canceler: 1) DCCOff: the DC canceler if off; 2) DCCHighPassFilterMode: canceler is on and high-pass filter method is applied, which has a good rejection of DC offset but also rejection for signal from DC to 100kHz; 3) DCCManualOffsetMode: canceler is on and manual bias method is applied, for which manual calibration is required but has no damge to the signal in DC; 4) DCCAutoOffsetMode: canceler is on and auto bias method is applied, for which manual calibration is not needed and has no damge to the signal in

	DC. If the device has a DC component at the centre frequency, please enable this function to suppress the DC component, if there is no DC component, there is no need to set this parameter.
QDCMode_TypeDef **QDCMode**	Set the IQ amplitude/phase correction mode: 1) QDCOff: the QDC is off; 2) QDCManualMode: QDC is on and manual calibrate is needed. In this mode, the QDCIGain, QDCQGain, and QDCPhaseComp are configured based on user-defined manual settings; 3) QDCAutoMode: QDC is on and auto coefficients are used. In this mode, the QDCIGain, QDCQGain, and QDCPhaseComp are automatically configured to default values. If the amplitude-phase characteristics of the IQ data remain unchanged after enabling the QDC function, then this device does not require QDC activation.
float QDCIGain	Set the normalized linear gain of I channel. The setting range is 0.8 - 1.2 and 1.0 stands for no gain. QDCIGain is only effective when the QDCMode is set to QDCManualMode.
float QDCQGain	Set the normalized linear gain of Q channel. The setting range is 0.8 - 1.2 and 1.0 stands for no gain. QDCQGain is only effective when the QDCMode is set to QDCManualMode.
float QDCPhaseComp	Set the phase compensation coefficient of the QDC. The setting range is -0.2 - 0.2. QDCPhaseComp is only effective when the QDCMode is set to QDCManualMode.
int8_t DCCIOffset	Set the DC offset of the I channel, in LSB. Effective only when DCCancelerMode is set to DCCManualOffsetMode.
int8_t DCCQOffset	Set the DC offset of the Q channel, in LSB. Effective only when DCCancelerMode is set to DCCManualOffsetMode.
LOOptimization_TypeDef **LOOptimization**	Set LO Optimization: 1) LOOpt_Auto: Auto LO optimization; 2) LOOpt_Speed: High-speed LO optimization; 3) LOOpt_Spur: LO-spur LO optimization; 4) LOOpt_PhaseNoise: LO-phase-noise LO optimization.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the IQS_GetIQStream() function for related examples.

9.2 IQS_Configuration

int IQS_Configuration(void Device, const IQS_Profile_TypeDef* ProfileIn, IQS_Profile_TypeDef* ProfileOut, IQS_StreamInfo_TypeDef* StreamInfo)**	
Description	
Set device to the IQS mode and configurate it with parameters specified in the IQS profile. In IQS mode, the LO signal remains fixed, and the system receives RF signals with a certain bandwidth centered at the LO frequency. Additionally, when the decimation factor is greater than 2, continuous time-domain recording can be achieved (requires a PC with specific configurations or higher).	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const IQS_Profile_TypeDef* IQS_ProfileIn	Configuration profile for IQS mode. Refer to the detailed definition of the structure parameter used in the IQS_ProfileDeInit() function.
IQS_Profile_TypeDef* IQS_ProfileOut	Feedback profile from the system. Refer to the detailed definition of the structure parameter used in the IQS_ProfileDeInit() function.
IQS_StreamInfo_TypeDef* StreamInfo	The information about the data stream under the current configuration.
IQS_StreamInfo_TypeDef	
double Bandwidth	Bandwith of the receiver's physical channel or digital signal processing under current configutation.
double IQSampleRate	IQ single-channel sampling rate under current configuration (unit: Sample/second).
uint64_t PacketCount	Total packet count under current configuration (effective only in FixedPoints mode).
uint64_t StreamSamples	In FixedPoints mode: represents the total sampling points under current configuration. In Adaptive mode: physically meaningless (value fixed to 0).
uint64_t StreamDataSize	In FixedPoints mode: indicates the total bytes to sample under current configuration. In Adaptive mode: no physical meaning (value fixed to 0).
uint32_t PacketSamples	Number of sampling points per data packet retrieved by IQS_GetIQStream.
uint32_t PacketDataSize	Number of meaningful data bytes returned by each IQS_GetIQStream() invocation.

uint32_t GainParameter	Gain-related parameters, including: 1) Space (Bits 31 - 24); 2) PreAmplifierState (Bits 23 - 16); 3) StartRFBand (Bits 15 - 8); 4) StopRFBand (Bits 7 - 0).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_ProfileDelnit.
Example	Please refer to the IQS_GetIQStream() function for related examples.

9.3 IQS_BusTriggerStart

int IQS_BusTriggerStart(void Device)**	
Description	
Lanuch a bus trigger.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called before IQS_GetIQStream.
Example	Please refer to the IQS_GetIQStream() function for related examples.

9.4 IQS_GetIQStream

int IQS_GetIQStream(void Device, void** AlternIQStream, float* ScaleToV, IQS_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
This function Return time-domain data, measurement parameters, and trigger details when operating in IQS mode.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
void AlternIQStream**	Memory address of time-domain data (interleaved IQ format). Each data packet has a fixed size of 64960 bytes. 1) When using int8_t IQ data type: I/Q channels each contain 32480 samples; each sample occupies 1byte; 2) When using int16_t IQ data type: I/Q channel each contain 16240 samples; each sample occupies 2 bytes; 3) When using int32_t IQ data type: I/Q channels each contain 8120 samples; each sample occupies 4 bytes; The IQ data is stored in interleaved format: IQIQ… .

float* ScaleToV	Coefficient for IQ data to the absolute voltage in volt.
MeasAuxInfo_TypeDef* **MeasAuxInfo**	Return auxiliary measurement information. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep () function.
IQS_TriggerInfo_TypeDef* **TriggerInfo**	Trigger-related information for the IQ data stream.
TriggerInfo_TypeDef	
uint64_t SysTimerCountOfFirstDataPoint	System timestamp of the first data point in the current packet.
uint16_t InPacketTriggeredDataSize	Number of bytes of valid triggered data in the current packet.
uint16_t InPacketTriggerEdges	Number of trigger edges contained in the current packet.
uint32_t **StartDataIndexOfTriggerEdges[25]**	Data position of each trigger edge within the current packet.
uint64_t SysTimerCountOfEdges[25]	System timestamp of each trigger edge in the current packet.
int8_t EdgeType[25]	Polarity of each trigger edge in the current packet.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_Configuration.
Example	

```
int Status = -1;int DeviceNum = 0;void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn, ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_BusTriggerStart(&Device);
void* AlternIQStream = NULL;float ScaleToV = 0;
TriggerInfo_TypeDef TriggerInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = IQS_GetIQStream(&Device, &AlternIQStream, &ScaleToV, &TriggerInfo, &MeasAuxInfo);
Status = Device_Close(&Device);
```

10 DET Mode

DET is a detection analysis mode that performs power detection on signals within a certain bandwidth to help users observe the level of the signal.

10.1 DET_ProfileDeInit

int DET_ProfileDeInit(void Device, DET_Profile_TypeDef* UserProfile_O)**	
Description	
This function initializes the configuration profile of the DET mode. DET_Profile_TypeDef defines all parameters in DET mode such as center frequency, reference level, decimate factor, etc.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
DET_Profile_TypeDef* UserProfile_O	Pointer to the DET configuration structure (input/output parameter).
DET_Profile_TypeDef	
double CenterFreq_Hz	Refer to the detailed definition of the structure parameter used in the IQS_ProfileDeInit() function.
double RefLevel_dBm	
uint32_t DecimateFactor	
uint32_t BusTimeout_ms	
DET_TriggerSource_TypeDef TriggerSource	
TriggerEdge_TypeDef TriggerEdge	
TriggerMode_TypeDef TriggerMode	
uint64_t TriggerLength	
double TriggerLevel_dBm	
double TriggerLevel_SafeTime	
double TriggerDelay	
double PreTriggerTime	
TriggerTimerSync_TypeDef TriggerTimerSync	
double TriggerTimer_Period	
uint8_t EnableReTrigger	
double ReTrigger_Period	
uint16_t ReTrigger_Count	

GainStrategy_TypeDef GainStrategy	
Detector_TypeDef Detector	Set Detector 1) No inter-frame detection is applied. Each frequency bin outputs its raw power spectrum; 2) Detector_PosPeak: Performs inter-frame detection for each frequency bin and outputs one frame using MaxHold across frames; 3) Detector_Average: Performs inter-frame detection for each frequency bin and outputs one frame using the average across frames; 4) Detector_NegPeak: Performs inter-frame detection for each frequency bin and outputs one frame using MinHold across frames; 5) Detector_RMS: Performs inter-frame detection for each frequency bin and outputs one frame using RMS across frames.
uint16_t DetectRatio	Set DET trace detection ratio. The detector detects the power trace, and detects 1 output trace point for every DetectRatio raw data point.
PreamplifierState_TypeDef Preamplifier	Please refer to the function with same name in the IQS_ProfileDeInit() section.
uint8_t AnalogIFBWGrade	
uint8_t IFGainGrade	
ReferenceClockSource_TypeDef ReferenceClockSource	
double ReferenceClockFrequency	
uint8_t EnableReferenceClockOut	
int8_t Atten	
uint8_t EnableIFAGC	
DCCancelerMode_TypeDef DCCancelerMode	
QDCMode_TypeDef QDCMode	
float QDCIGain	
float QDCQGain	
float QDCPhaseComp	
int8_t DCCIOffset	
int8_t DCCQOffset	
LOOptimization_TypeDef Looptimization	
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.

Example	Please refer to the DET_GetPowerStream() function for related examples.

10.2 DET_Configuration

int DET_Configuration(void Device, const DET_Profile_TypeDef* ProfileIn, DET_Profile_TypeDef* ProfileOut, DET_StreamInfo_TypeDef* StreamInfo)**	
Description	
The center frequency, reference level, decimation factor, and other parameters in DET mode are encapsulated in the DET_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const DET_Profile_TypeDef* DET_ProfileIn	DET configuration structure pointer (input parameter). Refer to the detailed definition of the structure parameter used in the DET_ProfileDeInit() function.
DET_Profile_TypeDef* DET_ProfileOut	DET configuration structure pointer (output parameter). Refer to the detailed definition of the structure parameter used in the DET_ProfileDeInit() function.
DET_StreamInfo_TypeDef* StreamInfo	Relevant information of DET data in DET mode.
DET_StreamInfo_TypeDef	
uint64_t PacketCount	Total number of data packets per single trigger. It is valid when the TriggerMode= Fixedpoints. It is invalid when TriggerMode= Adaptive, and value is set as 0.
uint64_t StreamSamples	The total samples in the power stream. It is valid when the TriggerMode= Fixedpoints. It is invalid when TriggerMode= Adaptive, and value is set as 0.
uint64_t StreamDataSize	The data size of the power stream. It is valid when the TriggerMode= Fixedpoints. It is invalid when TriggerMode= Adaptive, and value is set as 0.
uint32_t PacketSamples	The total samples in the packet. It's also the number of samples can be obtained by every calling of function DET_GetPowerStream.
uint32_t PacketDataSize	The data size of the packet. It's also the data size can be obtained by every calling of function DET_GetPowerStream.
double TimeResolution	The time interval between adjacent data points, s.
uint32_t GainParameter	Gain parameter information: 1) Bits 31-24: Gain range; 2) Bits 23-16: Preamplifier status;

	3) Bits 15-0: Frequency band information.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DET_ProfileDeInit.
Example	Please refer to the DET_GetPowerStream() function for related examples.

10.3 DET_BusTriggerStart

int DET_BusTriggerStart(void Device)**	
Description	
Lanuch a bus trigger.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DET_GetPowerStream.
Example	Please refer to the DET_GetPowerStream() function for related examples.

10.4 DET_GetPowerStream

int DET_GetPowerStream(void Device, float NormalizedPowerStream[], float* ScaleToV, DET_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
Retrieves DET mode detection data, returning the integer-to-absolute amplitude scale factor and trigger-related information, where NormalizedPowerStream represents the value of ($\sqrt{I^2+Q^2}$).	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
float NormalizedPowerStream[]	The value of ($\sqrt{I^2+Q^2}$).
float* ScaleToV	Coefficient for normalized power level to the absolute voltage (V).
DET_TriggerInfo_TypeDef* TriggerInfo	Trigger information of DET data. Refer to the detailed definition of the structure parameter used in the IQS_GetIQStream() function.
MeasAuxInfo_TypeDef* MeasAuxInfo	Auxiliary measurement information. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.

Calling constraints	Must be called after DET_Configuration.

Example

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn, ProfileOut;
DET_StreamInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
vector<float> NormalizedPowerStream(StreamInfo.PacketSamples);
float ScaleToV;
DET_TriggerInfo_TypeDef  TriggerInfo;
MeasAuxInfo_TypeDef  MeasAuxInfo;
Status = DET_BusTriggerStart(&Device);
Status = DET_GetPowerStream(&Device, NormalizedPowerStream.data(), &ScaleToV, &TriggerInfo, &MeasAuxInfo);
Status = Device_Close(&Device);
```

11 RTA Mode

RTA is a real-time spectrum analysis mode that helps users observe frequency hopping or transient burst signals.

11.1 RTA_ProfileDeInit

int RTA_ProfileDeInit(void Device, RTA_Profile_TypeDef* UserProfile_O)**	
Description	
Initialize and configure the related parameters of RTA mode. The center frequency, reference level, decimation factor, and other parameters in the RTA mode are uniformly encapsulated in the RTA_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
RTA_Profile_TypeDef* UserProfile_O	RTA configuration structure pointer, used as input/output variable.
RTA_Profile_TypeDef	
double CenterFreq_Hz	Please refer to parameters with same name in the SWP_ProfileDeInit() section.
double RefLevel_dBm	
double RBW_Hz	
double VBW_Hz	
RBWMode_TypeDef RBW_Mode	
VBWMode_TypeDef VBW_Mode	
uint32_t DecimateFactor	Set decimation factor.
Window_TypeDef Window	Please refer to parameters with same name in the SWP_ProfileDeInit() section.
SweepTimeMode_TypeDef SweepTimeMode	
double SweepTime	
Detector_TypeDef Detector	
TraceDetectMode_TypeDef TraceDetectMode	
TraceDetector_TypeDef TraceDetector	
uint32_t TraceDetectRatio	Trace detection ratio. The trace detector outputs 1 spectrum data point for every TraceDetectRatio original spectrum data points in the original spectrum trace.
uint32_t BusTimeout_ms	Please refer to parameters with same name in the IQS_ProfileDeInit () section.
RTA_TriggerSource_TypeDef TriggerSource	
TriggerEdge_TypeDef TriggerEdge	

TriggerMode_TypeDef TriggerMode	
double TriggerAcqTime	Sets the sampling time after input trigger, effective only in FixedPoints mode.
double TriggerLevel_dBm	Please refer to parameters with same name in the IQS_ProfileDeInit() section.
double TriggerLevel_SafeTime	
double TriggerDelay	
double PreTriggerTime	
TriggerTimerSync_TypeDef TriggerTimerSync	
double TriggerTimer_Period	
uint8_t EnableReTrigger	
double ReTrigger_Period	
uint16_t ReTrigger_Count	
GainStrategy_TypeDef GainStrategy	
PreamplifierState_TypeDef Preamplifier	
uint8_t AnalogIFBWGrade	
uint8_t IFGainGrade	
ReferenceClockSource_TypeDef ReferenceClockSource	
double ReferenceClockFrequency	
uint8_t EnableReferenceClockOut	
int8_t Atten	
uint8_t EnableIFAGC	
DCCancelerMode_TypeDef DCCancelerMode	
QDCMode_TypeDef QDCMode	
float QDCIGain	
float QDCQGain	
float QDCPhaseComp	
int8_t DCCIOffset	
int8_t DCCQOffset	
LOOptimization_TypeDef LOOptimization	
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.

Example	Please refer to the RTA_GetRealTimeSpectrum() function for related examples.

11.2 RTA_SetDataFormat

int RTA_SetDataFormat (void Device, DataFormat_TypeDef* DataFormat)**	
Description	
RTA mode to set the data type of the real-time spectrum.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
DataFormat_TypeDef* DataFormat	Streaming mode data format type. Refer to the detailed definition of the structure parameter used in the IQS_ProfileDeInit() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called before RTA_Configuration.
Example	Please refer to the RTA_GetIQStream () function for related examples.

11.3 RTA_SetLookBackCmd

int RTA_SetLookBackCmd (void Device, LookBack_TypeDef* LookBackCmd)**	
Description	
RTA mode, set the real-time spectrum LookBack state, after enabling LookBack, the device will cache the IQ data corresponding to the spectrum for the user to read if needed.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
LookBack_TypeDef* LookBackCmd	LookBack status: 1) LookBack_Off: turn off the lookback function and do not upload the original IQ data; 2) LookBack_On: turn on the lookback function and upload the original IQ data.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called before RTA_Configuration.
Example	Please refer to the RTA_GetIQStream () function for related examples.

11.4 RTA_Configuration

int RTA_Configuration(void Device, const RTA_Profile_TypeDef* ProfileIn, RTA_Profile_TypeDef* ProfileOut, RTA_FrameInfo_TypeDef* FrameInfo)**

Description	
Configure the related parameters of the RTA mode. Parameters such as center frequency, reference level, and decimation factor in RTA mode are uniformly encapsulated in the RTA_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const RTA_Profile_TypeDef* RTA_ProfileIn	Pointer to the RTA configuration structure, used as an input variable. Refer to the detailed definition of the structure parameter used in the RTA_ProfileDeInit() function.
RTA_Profile_TypeDef* RTA_ProfileOut	Pointer to the RTA configuration structure, used as an output variable. Refer to the detailed definition of the structure parameter used in the RTA_ProfileDeInit() function.
RTA_FrameInfo_TypeDef* StreamInfo	Related information of RTA data in RTA mode.
RTA_FrameInfo_TypeDef	
double StartFrequency_Hz	Start frequency in Hz.
double StopFrequency_Hz	Stop frequency in Hz.
double POI	Minimum signal duration time with 100% probability of interception in s.
double TraceTimestampStep	The timestamp step of each trace in each packet of data. (The overall packet timestamp is SysTimerCountOfFirstDataPoint in TriggerInfo)
double TimeResolution	The sampling time of each time-domain data, which is also the resolution of the timestamp
double PacketAcqTime	The acquisition time corresponding to each data packet
uint32_t PacketCount	Packet count of the specturm stream. A specturm stream is generated once the device is triggered.
uint32_t PacketFrame	The number of valid frames in each data packet
uint32_t FFTSize	The number of points in the FFT of each frame
uint32_t FrameWidth	The number of points after FFT frame capture, which is also the number of points per Trace in the data packet, and can be used as the number of X-axis points (width) in the probability density map.
uint32_t FrameHeight	The spectral amplitude range corresponding to the FFT frame, which can be used as the number of Y-axis points (height) in the probability density map.
uint32_t PacketSamplePoints	The number of sampling points corresponding to each data packet.
uint32_t PacketValidPoints	The number of valid frequency domain data points contained in each data packet.

uint32_t MaxDensityValue	The upper limit of the element value at a single site in the probability density bitmap.
uint32_t GainParameter	Gain-related parameters, including: 1) Space (31 - 24Bit); 2) PreAmplifierState (23 - 16Bit); 3) StartRFBand (15 - 8Bit); 4) StopRFBand (7 - 0Bit).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after RTA_ProfileDeInit.
Example	Please refer to the RTA_GetRealTimeSpectrum() function for related examples.

11.5 RTA_TriggerStart

int RTA_TriggerStart(void Device)**	
Description	
Lauch a trigger. When the trigger type is set to bus trigger, calling this interface starts a single triggered acquisition. When the trigger type is non-bus trigger, the system waits for an external trigger and then starts the acquisition.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called before RTA_GetRealTimeSpectrum.
Example	Please refer to the RTA_GetIQStream () function for related examples.

11.6 RTA_GetRealTimeSpectrum

int RTA_GetRealTimeSpectrum(void Device, uint8_t SpectrumStream[], uint16_t SpectrumBitmap[], RTA_PlotInfo_TypeDef* PlotInfo, RTA_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
Obtain the real-time spectrum in RTA mode.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
uint8_t SpectrumStream[]	Pointer to the spectrum stream. The spectrum stream is consisted of contiguous spectral frames. The spectrum is in normalized relative power with LSB equals to 0.75dB. The array size is equal to the RTA_FrameInfo.PacketValidPoints.

uint16_t SpectrumBitmap[]	Return the probability density map bitmap. The size of this array equals RTA_FrameInfo.FrameHeight * RTA_FrameInfo.FrameWidth obtained via the RTA_Configuration function.
RTA_PlotInfo_TypeDef* **RTA_PlotInfo**	Structure of plotting information returned after RTA acquisition.
RTA_TriggerInfo_TypeDef* **TriggerInfo**	Trigger-related information of RTA data. Refer to the detailed definition of the structure parameter used in the IQS_GetIQStream() function.
MeasAuxInfo_TypeDef* **MeasAuxInfo**	Return auxiliary information of the measurement data. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
RTA_PlotInfo_TypeDef	
float ScaleTodBm **float OffsetTodBm**	Scale factor and bias value of the amplitude to dBm of the spectral frame. The spectrum frame amplitude absolute power is equal to SpecturmStream[] * ScaleTodBm + OffsetTodBm
uint64_t SpectrumBitmapIndex	The current probability density picture segment then the ordinal number in the complete trigger data.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after RTA_Configuration.

Example

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn, ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
vector<uint8_t> SpectrumTrace(FrameInfo.PacketValidPoints);
vector<uint16_t> SpectrumBitmap(FrameInfo.FrameHeight* FrameInfo.FrameWidth);
RTA_TriggerInfo_TypeDef TriggerInfo;
RTA_PlotInfo_TypeDef PlotInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = RTA_TriggerStart(&Device);
```

Status = RTA_GetRealTimeSpectrum(&Device, SpectrumTrace.data(), SpectrumBitmap.data(), &PlotInfo, &TriggerInfo, &MeasAuxInfo); **Status = Device_Close(&Device);**

11.7 RTA_GetIQStream

int RTA_GetIQStream(void Device, void* AlternIQStream, float* ScaleToV, TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
RTA mode, get the real-time IQStream and trigger related information in RTA mode.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
void* AlternIQStream	Time-domain data (interleaved IQ format). A single packet can contain up to 8192 IQ points, with the total byte count determined by the InPacketTriggeredDataSize parameter in the TriggerInfo_TypeDef structure. The IQ data is stored in an IQIQ... interleaved arrangement. 1) When the data type is int16_t, the total number of data points equals InPacketTriggeredDataSize/2; 2) When the data type is int8_t, the total number of data points equals InPacketTriggeredDataSize; 3) When the data type is int32_t, the total number of data points equals InPacketTriggeredDataSize/4;
float* ScaleToV **TriggerInfo_TypeDef* TriggerInfo**	Refer to the detailed definition of the structure parameter used in the IQS_GetIQStream () function.
MeasAuxInfo_TypeDef* MeasAuxInfo	Auxiliary measurement information. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after RTA_GetRealTimeSpectrum.
Example	
int Status = -1; int DeviceNum = 0; void* Device = NULL; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef BootInfo;**	

```
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef RTA, RTA_Feedback;
RTA_FrameInfo_TypeDef FrameInfo;
RTA_PlotInfo_TypeDef RTA_PlotInfo;
RTA_TriggerInfo_TypeDef TriggerInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
float ScaleToV; int32_t pointsize = 1;
vector<uint8_t> AlternIQStream(131072);
DataFormat_TypeDef DataFormat = Real16bit;
LookBack_TypeDef LookBackCmd = LookBack_On;
RTA_ProfileDeInit(&Device, &RTA);
RTA.DecimateFactor = 32;
RTA.RBWMode = RBW_Manual;
RTA.RBW_Hz = 1e6;
RTA.SweepTime = 0.0001;
RTA.TriggerAcqTime = 0.1;
if (DataFormat == Real8bit) pointsize = 1;
else if (DataFormat == Real16bit) pointsize = 2;
Status = RTA_SetDataFormat(&Device, &DataFormat);
Status = RTA_SetLookBackCmd(&Device, &LookBackCmd);
Status = RTA_Configuration(&Device, &RTA, &RTA_Feedback, &FrameInfo);
vector<uint8_t> SpectrumTrace(FrameInfo.PacketValidPoints * pointsize);
vector<uint16_t> SpectrumBitmap(FrameInfo.FrameHeight * FrameInfo.FrameWidth);
while (1) {
   RTA_TriggerStart(&Device);
   for (int i = 0; i < FrameInfo.PacketCount; i++){
      Status = RTA_GetRealTimeSpectrum(&Device, SpectrumTrace.data(), SpectrumBitmap.data(),
&RTA_PlotInfo, &TriggerInfo, &MeasAuxInfo);
   }
   for (int i = 0; i < FrameInfo.PacketCount; i++){
   Status = RTA_GetIQStream(&Device, (void*)AlternIQStream.data(), &ScaleToV, &TriggerInfo, &MeasAuxInfo);
      if (Status == APIRETVAL_LastPacket) {
      break;
      }
   }
}
Device_Close(&Device);
```

12 MSCAN Mode

12.1 MSCAN_ProfileDeinit

int MSCAN_ProfileDeinit(void Device, MSCAN_Profile_TypeDef* MSCAN_Profiles, int32_t* Elements);**	
Description	
Initialize MSCAN_Profiles.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
MSCAN_Profile_TypeDef* MSCAN_Profiles	Pointer to an array of MSCAN profile structures.
int32_t* Elements	Number of MSCAN_Profiles.
MSCAN_Profile_TypeDef	
double CenterFreq_Hz	Center frequency in Hz.
double RefLevel_dBm	Reference level in dBm.
double DwellTime	Dwell time at each frequency point.
uint32_t DecimateFactor	Decimation factor for IQ data.
uint32_t FFTSize	Number of points for FFT.
uint32_t DetectCount	Number of detection.
Detector_TypeDef Detector	Detection type. Refer to the detailed definition of the structure parameter used in the SWP_ProfileDeInit() function.
IFAGC_TypeDef IFAGC	Whether to enable IFAGC: 1) IFAGC_Off: disable IFAGC; 2) IFAGC_On: enable IFAGC.
XPPSTrigger_TypedDef XPPSTrigger	Whether to enable XPPSTrigger: 1) XPPSTrigger_Off: disable XPPSTrigger; 2) XPPSTrigger_On: enable XPPSTrigger.
IQPlayBack_TypedDef IQPlayBack	Whether to enable IQPlayBack: 1) IQPlayBack_Off: diable IQPlayBack; 2) IQPlayBack_On: enable IQPlayBack.
Window_TypeDef Window	Refer to the detailed definition of the structure parameter used in the SWP_ProfileDeInit() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.

Example	Please refer to the MSCAN_GetData() function for related examples.

12.2 MSCAN_Configuration

int MSCAN_Configuration(void Device, MSCAN_Profile_TypeDef* ProfilesIn, MSCAN_Profile_TypeDef* ProfilesOut, MSCAN_Info_Typedef *MSCAN_Info, int32_t *Elements, int64_t *Repeatitions, PreamplifierState_TypeDef* Preamplifier);**	
Description	
Close ListMode.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
MSCAN_Profile_TypeDef* ProfilesIn	Refer to the detailed definition of the structure parameter used in the MSCAN_ProfileDeinit() function.
MSCAN_Profile_TypeDef* ProfilesOut	
MSCAN_Info_Typedef* MSCAN_Info	Return the MSCAN spectrum frame counts, spectrum points, and IQ data points.
int32_t* Elements	Number of MSCAN_Profiles. The maximum value is 128; values exceeding 128 are overwritten as 128.
Int64_t* Repeatitions	Pointer to the number of repetitions for each MSCAN profile in the scanning list.
PreamplifierState_TypeDef* Preamplifier	Refer to the detailed definition of the structure parameter used in the SWPProfileDeInit() function.
MSCAN_Info_Typedef	
int SpectrumFrames	The number of frames in the spectrum.
int SpectrumPoints	Number of points in each frame of the spectrum.
int IQStreamPoints	The number of points in the IQ data.
double CenterFreq_Hz	Center frequency in Hz.
double Span_Hz	Frequency span in Hz.
double IQSampleRate	IQ sample rate under current configuration.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after MSCAN_ProfileDeinit.
Example	Please refer to the MSCAN_GetData() function for related examples.

12.3 MSCAN_Start

int MSCAN_Start(void Device)**	
Description	
Start list scan, scan according to the number of repetitions specified by the user, and automatically stop after the scan is completed.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after MSCAN_Configuration.
Example	Please refer to the MSCAN_GetData() function for related examples.

12.4 MSCAN_Stop

int MSCAN_Stop(void Device)**	
Description	
Stop list scan.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after MSCAN_GetData.
Example	Please refer to the MSCAN_GetData() function for related examples.

12.5 MSCAN_GetData

int MSCAN_GetData(void Device,MSCAN_Data_Typedef* MSCAN_Data);**	
Description	
Get the scan results.	
Compatibility	0.55.61 and later.
Parameter description	
void Device**	Device handle.
MSCAN_Data_Typedef* MSCAN_Data	Pointer to a structure where the MSCAN scan results will be stored. After calling MSCAN_GetData, this structure contains the relevant measurement data for each scanned profile.
MSCAN_Data_Typedef	
int64_t RepeatIndex	Indicates how many times the current element has been repeated

int32_t ElementIndex	Index of the current element within the profile.
int Status	Return status of the element processing.
uint32_t SpectrumFrames	Actual number of spectral frames acquired.
uint32_t SpectrumPoints	Number of points per spectral frame.
uint32_t IQStreamPoints	Number of IQ data points actually acquired.
float ScaleTodBm	Scaling factor for converting linear power to dBm. Absolute power = SpectrumStream[] * ScaleTodBm + OffsetTodBm.
float OffsetTodBm	Offset applied when converting relative power to absolute power in dBm.
float ScaleToV	Scaling factor for converting int16 IQ data to absolute voltage (V).
uint8_t* SpectrumStream	Spectrum data (the user needs to allocate space, with the size of the space being the frame count × number of points).
int16_t* IQStream	IQ data (the user needs to allocate space, with the size of the space referring to the number of IQ data points).
double Temperature	Device temperature. Actual value in Celsius = 0.01 * Temperature.
double SysTimeStamp	System timestamp corresponding to the current data packet, in seconds.
double AbsoluteTimeStamp	Absolute timestamp corresponding to the current data packet.
double Latitude	Latitude coordinate. Positive values indicate north latitude; negative values indicate south latitude.
double Longitude	Longitude coordinate. Positive values indicate east longitude; negative values indicate west longitude.
double Altitude	Altitude corresponding to the current data packet, in meters.
double SATHealth	Health of the GNSS satellites (e.g., SNR) used for positioning.
double RefClkFreqOffset	Reference clock frequency offset in ppm, relative to GNSS reference frequency.
uint64_t nsSinceEpoch	Nanosecond-resolution system timestamp corresponding to the current data packet.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after MSCAN_Configuration.

Example

```
int Status = 0;int DeviceNum = 0;void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
std::cout << "Device_Open: " << Status << std::endl;
if (Status) {
```

```
    return 0;
}
int32_t Elements = 95
int64_t Repeatitions = 100;
std::vector<MSCAN_Profile_TypeDef> MSCAN_ProfileIn(Elements), MSCAN_ProfileOut(Elements);
std::vector<MSCAN_Info_Typedef> MSCAN_Info(Elements);
Status = MSCAN_ProfileDeinit(&Device, MSCAN_ProfileIn.data(), &Elements);

for (uint32_t i = 0; i < Elements; i++) {
  MSCAN_ProfileIn[i].CenterFreq_Hz = 50e6 + i * 100e6;
  MSCAN_ProfileIn[i].DwellTime = 16.384e-6;
  MSCAN_ProfileIn[i].DetectCount = 1;
  MSCAN_ProfileIn[i].IQPlayBack = IQPlayBack_Off;
  MSCAN_ProfileIn[i].FFTSize = 2048;
}
PreamplifierState_TypeDef Preamplifier = AutoOn;
Status = MSCAN_Configuration(&Device, MSCAN_ProfileIn.data(), MSCAN_ProfileOut.data(),
MSCAN_Info.data(), &Elements, &Repeatitions, &Preamplifier);

//Allocate buffer sized for the largest element.
uint32_t SpectrumFrames_max = 0;
uint32_t SpectrumPoints_max = 0;
uint32_t IQStreamPoints_max = 0;
for (uint32_t i = 0; i < Elements; i++) {
  if (SpectrumFrames_max < MSCAN_Info[i].SpectrumFrames) {
    SpectrumFrames_max = MSCAN_Info[i].SpectrumFrames;
  }
  if (SpectrumPoints_max < MSCAN_Info[i].SpectrumPoints) {
    SpectrumPoints_max = MSCAN_Info[i].SpectrumPoints;
  }
  if (IQStreamPoints_max < MSCAN_Info[i].IQStreamPoints) {
    IQStreamPoints_max = MSCAN_Info[i].IQStreamPoints;
  }
}
std::vector<uint8_t> SpectrumStream(SpectrumFrames_max * SpectrumPoints_max);
std::vector<double> Frequency(SpectrumPoints_max);
std::vector<int16_t> IQStream(IQStreamPoints_max * 2);
MSCAN_Data_Typedef MSCAN_Data;
```

```
MSCAN_Data.SpectrumStream = SpectrumStream.data();
MSCAN_Data.IQStream = IQStream.data();
while (1) {
  MSCAN_Start(&Device);
  auto start = std::chrono::high_resolution_clock::now();
  for (uint32_t i = 0; i < Repeatitions; i++) {
    for (uint32_t t = 0; t < Elements; ) {
      Status = MSCAN_GetData(&Device, &MSCAN_Data);
      if (Status) {
        //std::cout << cnt << ", MSCAN_GetData: " << Status << std::endl;
      }
      else {
        if (MSCAN_Data.Status) {
          //std::cout << "Repeatitions: " << i << ", Elements: " << t << ", MSCAN_Data.Status: " <<
MSCAN_Data.Status << std::endl;
        }
        t = MSCAN_Data.ElementIndex + 1;
      }
      if (MSCAN_Data.ElementIndex == (Elements - 1) && MSCAN_Data.RepeatIndex == (Repeatitions - 1)) {
        break;
      }
    }
    if (MSCAN_Data.ElementIndex == (Elements - 1) && MSCAN_Data.RepeatIndex == (Repeatitions - 1)) {
      break;
    }
  }
  auto stop = std::chrono::high_resolution_clock::now();
  auto time = std::chrono::duration_cast<std::chrono::duration<double, std::ratio<1, 1000>>>(stop - start);
  std::cout << "95 elements take time: " << time.count() / Repeatitions << "ms" << std::endl;
}
Device_Close(&Device);
```

13 Digital Signal Processing (Trace analysis)

13.1 DSP_Open

int DSP_Open(void DSP)**	
Description	
This function initializes inner variables and allocates memory space that is needed to excute DSP processing. This function must be called before the DSP handle to be used. Enable this function for DSP processing that requires different configurations.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space. After calling this function, it returns the memory address of the currently enabled DSP function. Subsequent API calls must use this reference to access the corresponding memory allocation.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	This function must be called once before calling any other DSP functions. Subsequent operations should use the returned device memory address. For every successful call to DSP_Open, you must call DSP_Close after completing all operations to release memory resources.
Example	
int Status = -1;void* DSP = NULL;void* Device = NULL; int DeviceNum = 0; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef BootInfo;** **Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);** **Status = DSP_Open(&DSP);** **Status = Device_Close(&Device);**	

13.2 DSP_TraceAnalysis_IM3

int DSP_TraceAnalysis_IM3(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, TraceAnalysisResult_IP3_TypeDef* IM3Result)
Description
This function analyzes the IM3 result of a trace.

Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
TraceAnalysisResult_IP3_TypeDef* IM3Result	Return the measurement result of IP3.
TraceAnalysisResult_IP3_TypeDef	
double LowToneFreq	Low-tone signal frequency. Unit depends on the data source.
double HighToneFreq	High-tone signal frequency. Unit depends on the data source.
double LowIM3PFreq	Low-frequency IM3 product frequency. Unit depends on the data source.
double HighIM3PFreq	High-frequency IM3 product frequency. Unit depends on the data source.
float LowTonePower_dBm	Power of the low-tone signal, in dBm.
float HighTonePower_dBm	Power of the high-tone signal, in dBm.
float TonePowerDiff_dB	Power difference: LowTonePower_dBm − HighTonePower_dBm.
float LowIM3P_dBc	Relative level of the low-frequency IM3 product, referenced to the stronger tone. LowIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) − LowIM3P_dBm.
float HighIM3P_dBc	Relative level of the high-frequency IM3 product, referenced to the stronger tone. HighIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) − HighIM3P_dBm.
float IP3_dBm	Third-order intercept point (IP3), in dBm.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	
int Status = -1; int DeviceNum = 0; void* Device = NULL; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef BootInfo;** **Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);** **SWP_Profile_TypeDef ProfileIn, ProfileOut;** **SWP_TraceInfo_TypeDef TraceInfo;**	

```
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
TraceAnalysisResult_IP3_TypeDef IM3Result;
Status = DSP_TraceAnalysis_IM3(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints,
&IM3Result);
Status = Device_Close(&Device);
```

13.3 DSP_TraceAnalysis_IM2

int DSP_TraceAnalysis_IM2(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, TraceAnalysisResult_IP2_TypeDef* IM2Result)	
Description	
This function analyzes the IM2 parameters of a trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
TraceAnalysisResult_IP2_TypeDef* IM2Result	Return the measurement result of IP2.
TraceAnalysisResult_IP2_TypeDef	
double LowToneFreq	Frequency of the lower-tone input signal. The unit depends on the source data.
double HighToneFreq	Frequency of the higher-tone input signal. The unit depends on the source data.
double IM2PFreq	Frequency of the second-order intermodulation (IM2) product. The unit depends on the source data.
float LowTonePower_dBm	Measured power of the lower-tone signal, in dBm.
float HighTonePower_dBm	Measured power of the higher-tone signal, in dBm.
float TonePowerDiff_dB	Power difference between the two tones, defined as LowTonePower_dBm – HighTonePower_dBm.
float IM2P_dBc	IM2P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - IM2P_dBm, the strength of the low freqeuncy product relative to the strongest tone.
float IP2_dBm	The second-order intercept point (IP2), expressed in dBm.

Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn, ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
uint8_t IfDoConfig = 1;
SWP_AutoSet(&Device, SWPChannelPowerMeas, &ProfileIn, &ProfileOut, &TraceInfo, IfDoConfig);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
TraceAnalysisResult_IP2_TypeDef IM2Result;
Status = DSP_TraceAnalysis_IM2(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints,
&IM2Result);
Status = Device_Close(&Device);
```

13.4 DSP_TraceAnalysis_PhaseNoise

int DSP_TraceAnalysis_PhaseNoise(const double Freq_Hz[], const float PowerSpec_dBm[], const double OffsetFreqs[], const uint32_t TracePoints, const uint32_t OffsetFreqsToAnalysis, double CarrierFreqOut[], float PhaseNoiseOut_dBc[])	
Description	
Analyse the phase noise parameters of the traces.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	An array of input frequencies.
const float PowerSpec_dBm[]	An array of Input power.
const double OffsetFreqs[]	Frequency Offset Array.
const uint32_t TracePoints	The number of trace points, i.e. the length of the two input arrays.
const uint32_t	The number of frequencies offset points to be analyzed.

OffsetFreqsToAnalysis	
double CarrierFreqOut[]	The actual carrier frequency.
float PhaseNoiseOut_dBc[]	The phase noise corresponding to the frequency offset array.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Chapter 6 and Appendix 1.
Calling constraints	None.

Example

```
int Status = 0;void* Device = NULL; int DevNum = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.CenterFreq_Hz = 1e9;
ProfileIn.FreqAssignment = CenterSpan;
SWP_Profile_TypeDef ProfileSetOut;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
uint8_t IfDoConfig = 1;
SWP_AutoSet(&Device, SWPPhaseNoiseMeas, &ProfileIn, &ProfileSetOut, &TraceInfo, IfDoConfig);
vector<double>Frequency(TraceInfo.FullsweepTracePoints);
vector<float>PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
SmoothMethod_TypeDef SmoothMethod;
SmoothMethod = MovingAvrage;
unsigned int IndexOffset = 5; unsigned int PolynomialOrder = 0;
size_t maxPowerIndex = 0; float maxValue = PowerSpec_dBm[0];
for (size_t i = 1; i < TraceInfo.FullsweepTracePoints; ++i) {
   if (PowerSpec_dBm[i] > maxValue) {
      maxValue = PowerSpec_dBm[i];
      maxPowerIndex = i;
   }
}
size_t newSize = TraceInfo.FullsweepTracePoints - maxPowerIndex;
```

```
unsigned int WindowLength = PERCENTAGE(ProfileIn.TracePoints, 1);
vector<double>NewFrequencyArray(newSize);
vector<float>NewPowerSpec_dBm(newSize);
for (size_t i = 0; i < newSize; ++i) {
   NewFrequencyArray[i] = Frequency[maxPowerIndex + i];
}
for (size_t i = 0; i < newSize; ++i) {
   NewPowerSpec_dBm[i] = PowerSpec_dBm[maxPowerIndex + i];
}
DSP_TraceSmooth(NewPowerSpec_dBm.data(), newSize, SmoothMethod, IndexOffset, WindowLength, PolynomialOrder);
const double OffsetFreqs[] = { 100,1000,10000,100000,1000000,10000000 };
const uint32_t OffsetFreqsToAnalysis = 6;
double CarrierFreqOut[OffsetFreqsToAnalysis] = { 0 };
float PhaseNoiseOut_dBc[OffsetFreqsToAnalysis] = { 0 };
Status = DSP_TraceAnalysis_PhaseNoise(NewFrequencyArray.data(), NewPowerSpec_dBm.data(), OffsetFreqs, newSize, OffsetFreqsToAnalysis, CarrierFreqOut, PhaseNoiseOut_dBc);
Device_Close(&Device);
```

13.5 DSP_TraceAnalysis_ChannelPower

int DSP_TraceAnalysis_ChannelPower(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, const double CenterFrequency, const double AnalysisSpan, const double RBW, DSP_ChannelPowerInfo_TypeDef* ChannelPowerResult)	
Description	
This function analyzes the channel power of a trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
const double CenterFrequency	Specify the channel center frequency in Hz.
const double AnalysisSpan	Specify the channel bandwidth in Hz.
const double RBW	Specify the resolution bandwidth in Hz.
DSP_ChannelPowerInfo_TypeDef* ChannelPowerResult	Return the channel power measurement result.
DSP_ChannelPowerInfo_TypeDef	
float ChannelPower_dBm	Represents the total channel power in dBm, integrated over

	the specified channel bandwidth
float PowerDensity	Indicates the power spectral density within the channel, expressed in dBm/Hz
float ChannelPeakIndex	Specifies the index of the peak bin within the channel. This index refers to the FFT bin or measurement point where the maximum power occurs.
double ChannelPeakFreq_Hz	Provides the frequency, in hertz, corresponding to the peak power detected within the channel.
float ChannelPeakPower_dBm	Indicates the peak power level within the channel, expressed in dBm.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn, ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
uint8_t IfDoConfig = 1;
SWP_AutoSet(&Device, SWPChannelPowerMeas, &ProfileIn, &ProfileOut, &TraceInfo, IfDoConfig);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
double CenterFrequency = 1e9; double AnalysisSpan = 50e6;
DSP_ChannelPowerInfo_TypeDef ChannelPowerResult; double RBW ;
Status = DSP_TraceAnalysis_ChannelPower(Frequency.data(), PowerSpec_dBm.data(),
TraceInfo.FullsweepTracePoints, CenterFrequency, AnalysisSpan, ProfileOut.RBW_Hz, &ChannelPowerResult);
Status = Device_Close(&Device);
```

13.6 DSP_TraceAnalysis_XdBBW

int DSP_TraceAnalysis_XdBBW(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, const float XdB, TraceAnalysisResult_XdB_TypeDef* XdBResult)	
Description	
This function analyzes the XdB bandwidth of the trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
const float XdB	The dB value by which the peak power needs to be reduced.
TraceAnalysisResult_XdB_TypeDef* XdBResult	Returns the measurement result at XdB.
TraceAnalysisResult_XdB_TypeDef	
double XdBBandWidth_Hz	XdB bandwidth in Hz.
double CenterFreq_Hz	Center frequency of the XdB bandwidth (Hz).
double StartFreq_Hz	Start frequency of the XdB bandwidth (Hz).
double StopFreq_Hz	Stop frequency of the XdB bandwidth (Hz).
float StartPower_dBm	Power at the start frequency of the XdB bandwidth (dBm).
float StopPower_dBm	Power at the stop frequency of the XdB bandwidth (dBm).
uint32_t PeakIndex	Peak index within the XdB bandwidth.
double PeakFreq_Hz	Peak frequency within the XdB bandwidth (Hz).
float PeakPower_dBm	Peak power within the XdB bandwidth (dBm).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn, ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
```

```
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
TraceAnalysisResult_XdB_TypeDef XdBResult; float XdB = 3;
Status = DSP_TraceAnalysis_XdBBW(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints,
XdB, &XdBResult);
Status = Device_Close(&Device);
```

13.7 DSP_TraceAnalysis_OBW

int DSP_TraceAnalysis_OBW(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, const float OccupiedRatio, TraceAnalysisResult_OBW_TypeDef* OBWResult)	
Description	
This function analyzes the occupied bandwidth of the trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
const float OccupiedRatio	Specify the occupied bandwidth, usually set as 0.99.
TraceAnalysisResult_OBW_TypeDef* OBWResult	Results for occupation bandwidth analysis.
TraceAnalysisResult_OBW_TypeDef	
double OccupiedBandWidth	Occupied bandwidth (Hz).
double CenterFreq_Hz	Center frequency of the occupied bandwidth (Hz).
double StartFreq_Hz	Start frequency of the occupied bandwidth (Hz).
double StopFreq_Hz	Stop frequency of the occupied bandwidth (Hz).
float StartPower_dBm	Power at the start frequency of the occupied bandwidth (dBm).
float StopPower_dBm	Power at the stop frequency of the occupied bandwidth (dBm).
float StartRatio	Power ratio at the start frequency of the occupied bandwidth.
float StopRatio	Power ratio at the stop frequency of the occupied bandwidth.
uint32_t PeakIndex	Peak index within the occupied bandwidth.
double PeakFreq_Hz	Peak frequency within the occupied bandwidth (Hz).
float PeakPower_dBm	Peak power within the occupied bandwidth (dBm).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn, ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
TraceAnalysisResult_OBW_TypeDef OBWResult; float OccupiedRatio = 0.99;
Status = DSP_TraceAnalysis_OBW(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints,
OccupiedRatio, &OBWResult);
Status = Device_Close(&Device);
```

13.8 DSP_TraceAnalysis_ACPR

int DSP_TraceAnalysis_ACPR(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, const DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo, TraceAnalysisResult_ACPR_TypeDef* ACPRResult)	
Description	
This function analyzers the adjacent channel power ratio (ACPR) of a trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
const DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo	Specify the needed frequency information for ACPR analysis.
TraceAnalysisResult_ACPR_TypeDef* ACPRResult	Return the ACPR measurement result.
DSP_ACPRFreqInfo_TypeDef	
double RBW	Specify the resolution bandwidth in Hz for ACPR analysis.
double MainChCenterFreq_Hz	Specify the center frequency of main channel in Hz.

double MainChBW_Hz	Specify the bandwidth of main channel in Hz.
double AdjChSpace_Hz	Specify the channel space in Hz. Channel space is the frequency interval between the center frequency of the main channel and that of the adjacent channel.
uint32_t AdjChPair	1: one pair adjacent channels will be analyzed; 2: two pairs adjacent channels will be analyzed.
TraceAnalysisResult_ACPR_TypeDef	
float MainChPower_dBm	Main channel power (dBm).
uint32_t MainChPeakIndex	Peak index of the main channel.
double MainChPeakFreq_Hz	Peak frequency of the main channel (Hz).
float MainChPeakPower_dBm	Peak power of the main channel (dBm).
double L_AdjChCenterFreq_Hz	Center frequency of the left adjacent channel (Hz).
double L_AdjChBW_Hz	Bandwidth of the left adjacent channel (Hz).
float L_AdjChPower_dBm	Power of the left adjacent channel (dBm).
float L_AdjChPowerRatio	Power ratio of the left adjacent channel (left adjacent channel power / main channel power).
float L_AdjChPowerDiff_dBc	Power difference of the left adjacent channel (left adjacent channel power – main channel power, dBc).
float L_AdjChPeakIndex	Peak index of the left adjacent channel.
double L_AdjChPeakFreq_Hz	Peak frequency of the left adjacent channel (Hz).
float L_AdjChPeakPower_dBm	Peak power of the left adjacent channel (dBm).
double R_AdjChCenterFreq_Hz	Center frequency of the right adjacent channel (Hz).
double R_AdjChBW_Hz	Bandwidth of the right adjacent channel (Hz).
float R_AdjChPower_dBm	Power of the right adjacent channel (dBm).
float R_AdjChPowerRatio	Power ratio of the right adjacent channel (right adjacent channel power / main channel power).
float R_AdjChPowerDiff_dBc	Power difference of the right adjacent channel (right adjacent channel power – main channel power, dBc).
float R_AdjChPeakIndex	Peak index of the right adjacent channel.
double R_AdjChPeakFreq_Hz	Peak frequency of the right adjacent channel (Hz).
float R_AdjChPeakPower_dBm	Peak power of the right adjacent channel (dBm).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	
int Status = -1; int DeviceNum = 0; void* Device = NULL; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;**	

```
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn, ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo;
ACPRFreqInfo.RBW = ProfileOut.RBW_Hz;
ACPRFreqInfo.MainChCenterFreq_Hz = 1e9;
ACPRFreqInfo.MainChBW_Hz = 50e6;
ACPRFreqInfo.AdjChSpace_Hz = 50e6;
ACPRFreqInfo.AdjChPair = 2;
TraceAnalysisResult_ACPR_TypeDef ACPRPowerInfo;
vector<TraceAnalysisResult_ACPR_TypeDef> ACPRResult(ACPRFreqInfo.AdjChPair);
Status = DSP_TraceAnalysis_ACPR(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints,
ACPRFreqInfo, ACPRResult.data());
Status = Device_Close(&Device);
```

13.9 DSP_AudioAnalysis

void DSP_AudioAnalysis(const double Audio[], const uint64_t SamplePoints, const double SampleRate, DSP_AudioAnalysis_TypeDef* AudioAnalysis)	
Description	
Analyze the audio parameters: audio voltage (V), audio frequency (Hz), signal-to-noise ratio (dB), and total harmonic distortion (%).	
Compatibility	0.55.0 and later.
Parameter description	
const double Audio[]	An array of audio signals.
const uint64_t SamplePoints	The length of the audio signal array.
const double SampleRate	The sample rate of the audio signal.
DSP_AudioAnalysis_TypeDef* AudioAnalysis	Returns the measurement results of the audio analysis.
DSP_AudioAnalysis_TypeDef	

double AudioVoltage	Measured audio voltage in volts (V).
double AudioFrequency	Measured audio frequency in hertz (Hz).
double SINDA	Signal to noise and distortion ratio (SINAD) in dB.
double THD	Total harmonic distortion (THD) expressed as a percentage.
Return value	None.
Calling constraints	None.

Example

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn, ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
IQStream_TypeDef IQStream;
void* AnalogMod = NULL;
ASD_Open(&AnalogMod);
bool reset = 1;
vector<float> result(StreamInfo.PacketSamples);
FM_DemodParam_TypeDef FM_DemodParam;
void* DSP = NULL;
DSP_Open(&DSP);
Status = IQS_BusTriggerStart(&Device);
DSP_AudioAnalysis_TypeDef  AudioAnalysis;
Status = IQS_GetIQStream_PM1(&Device, &IQStream);
Status = ASD_FMDemodulation(&AnalogMod, &IQStream, reset, result.data(), &FM_DemodParam);
vector<double> Audio(result.begin(), result.end());
DSP_AudioAnalysis(Audio.data(), StreamInfo.PacketSamples, StreamInfo.IQSampleRate, &AudioAnalysis);
Status = IQS_BusTriggerStop(&Device);
DSP_Close(&DSP);
ASD_Close(&AnalogMod);
Status = Device_Close(&Device);
```

14 Appendix 1: API Return Value Index

Error Code	Cause of error/warning	Type[1]	Handling
0	No error	-	No processing is required, subsequent processes can be executed normally.
-1	Bus open error	Error	Check the device power supply, data line connection and check if the driver is installed correctly. After troubleshooting the error, you need to call Device_Open again to open the device.
-3	RF calibration file is missing[2]	Error	Check if the RF calibration file is placed in the specified directory. After troubleshooting the error, you need to call Device_Open again to open the device.
-4	IF calibration file is missing[2]	Error	Check if the IF calibration file is placed in the specified directory. After eliminating the error, you need to call Device_Open again to open the device.
-5	Device configuration information is missing[2]	Error	Check if the RF calibration file used is correct with the IF calibration file. After removing the error, you need to call Device_Open again to open the device.
-6	Device specification file is missing[2]	Error	Check that the device specification file (if required) is placed in the specified directory.
-7	Update Strategy failed	Error	Re-call Device_Open to open the device.
-8	Bus communication error	Error	Re-call the configuration function in the current mode.
-9	Data content error	Error	Re-call the configuration function in the current mode.
-10	Data not retrieved within specified time	Warning	Check whether the trigger source outputs the trigger signal normally, if there is no abnormality, continue to call the current function until the data is obtained.
-11	Configuration error via bus down	Warning.	Re-call the Configuration function to configure the device.
-12	Input signal amplitude exceeds the rated range in the current configuration	Warning	The current function gets to reduce the input signal amplitude or increase RefLevel_dBm as appropriate.
-14	The temperature has changed significantly since the last	Warning	The device temperature has changed significantly since the last configuration, it is recommended to re-call the Configuration function to configure the device for best

	configuration		performance.
-15	There is a locking exception in the local oscillator or clock	Warning	It is recommended to re-call the Configuration function to configure the device to try to restore the normal state.

[1] Type is "Error", you need to troubleshoot the problem immediately and turn the device back on, otherwise the device cannot continue to run subsequent processes. If the type is "warning", the device can continue the process without shutting down or reopening the device. However, it is still recommended to deal with the specific return value and the current application scenario selectively.

[2] For the return value of -3, -4, -5, or -6, you also need to confirm whether the file storage path is a full English path. If the path contains non-English characters, the API call will also indicate file loading failure.