API Programming Guide

Programmers Reference Manual

API Version 0.55.27

2024-2-1

HAROGIC

Content

1 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.

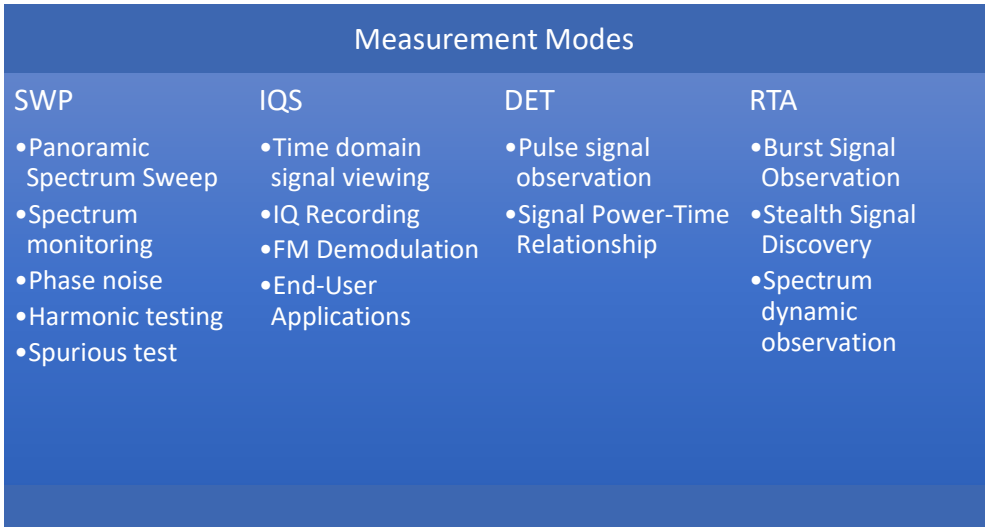

Figure 1 The measurement modes in the API system and the main application scenarios.

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 the SWP mode.

Before starting your development, please read the chapter 6 the callling map. This map can be

the basic framework of your application and it is useful for building a robust and efficient application.

2 The version of the 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 = 0.38.1, FFM version = 0.38.4, API version = 0.38.2, SAStudio4 version = 0.38.23, this combination of all software with y = 38 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.

3 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 returns 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 returns 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 returns 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 returns 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.

ASG	A global function that controls the device or one of the analog signal source options that can be invoked in any measurement mode. This category includes functions to set output mono, sweep, etc.
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.
ASD	Analog demodulation class post-processing functions, independent of hardware state. This category includes functions such as AM demodulation, FM demodulation, etc.

4 Using API in linux

4.1 Install configuration files, header files and so library files

Step 1: Determine the specific information of Linux environment. Open a terminal and use the commands "uname -a", "gcc -v" and "ldd --version" to determine the current Linux environment's system architecture, gcc version and GLIBC version of the current Linux environment, as shown in the following Figure:

Figure 3

Please confirm that the current environment is supported according to the terminal information, .and contact for technical support if it has not been supported.

Table 2 Adapted processors, compilation environments, distributions.

X86 processors	Intel and AMD processors supported.
ARM processors	arrch64 (armv8), armv7 processors, e.g. Raspberry Pi 4b, RK3399, RK3568, RK3588, T507, NIVIDA Jeston TX2.

Compiler environment	gcc 5.3.1, glib 2.21.
Release version	Raspberry Pi 4b custom system, ubantu 18.04, etc.

Step 2: Confirm the contents of the Linux profile folder, which includes:

1) HTRA_C++_Examples(examples、Makefile etc.);

2) Install_HTRA_SDK(driver configuration files, .so library, .h header file, etc.);

3)please read the README file before using it. The details are shown in the following Figure.

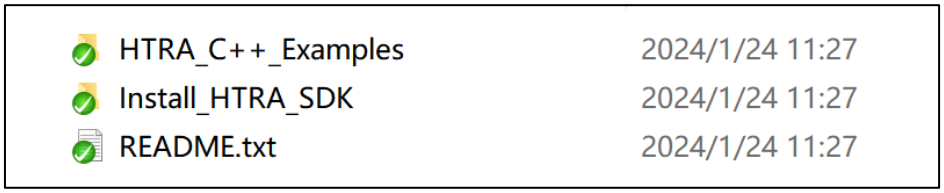

Figure 4

If you do not find the corresponding environment information, please contact for technical support to get it.

Step 3: Copy the Linux driver file to the Linux system, open the Install_HTRA_SDK folder, The details are shown in the following Figure:

1) htraapi(Library file);

2) the install_htraapi_lib.sh file is a configuration script. Please read the README file before using it.

Step 4: Run the.sh file with the command "sudo sh./install_htraapi_lib.sh" to install the required configuration file, the.so library file, and the.h library file. Note: Some versions of Linux systems need to declare after installing the new library that the system correctly knows that the new lib library has been installed through the "sudo ldconfig" command. As shown in the picture below:

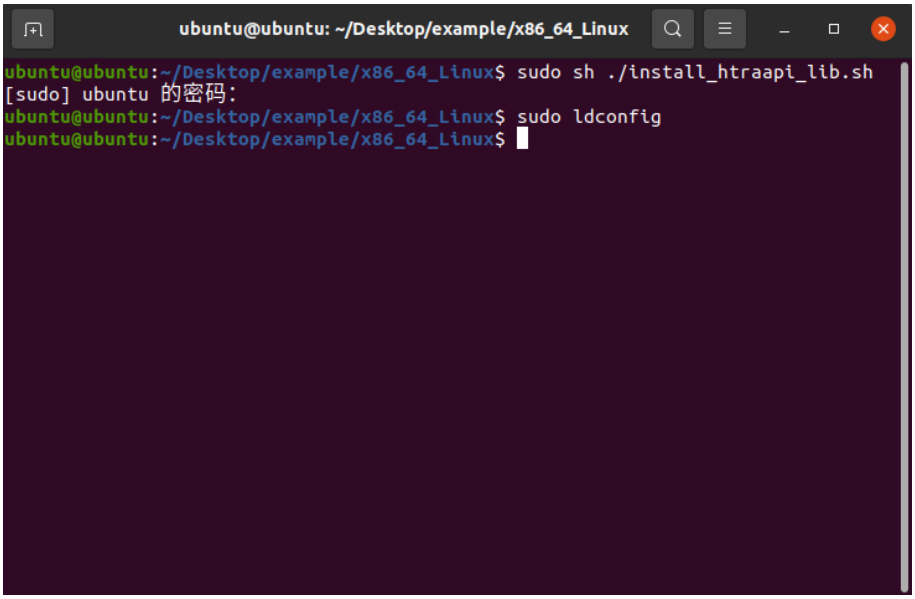

Figure 5

Step 5: After installing the driver and library, connect the device, you can use the command "lsusb" to determine if there is a device connected in the Linux environment. As shown the following Figure, where "ID: 6430 (Or ID: 367f)" is the access device.

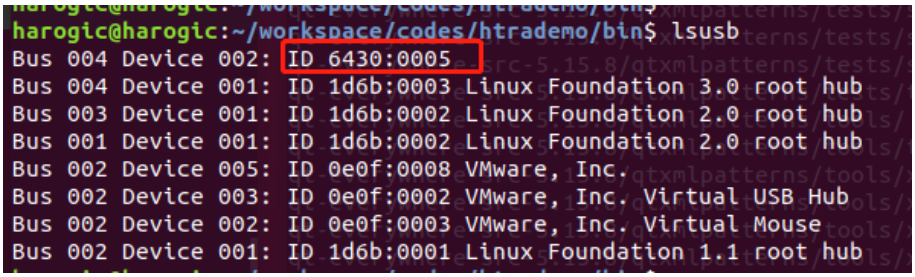

Figure 6

Step 6: Go to the HTRA_C++_Examples folder to compile the examples from the provided Makefile, please read the README.txt file before using it. The specific content is as follows:

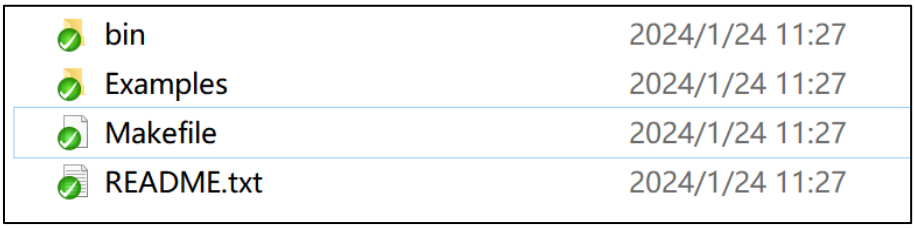

Figure 7

4.2 Using the API and examples in QT

Step 1: Create a Qt project.

Step 2: Configure the .so library file and .h library file in the .pro file, as shown in the following Figure , where the htraapi folder is the .so and header files.

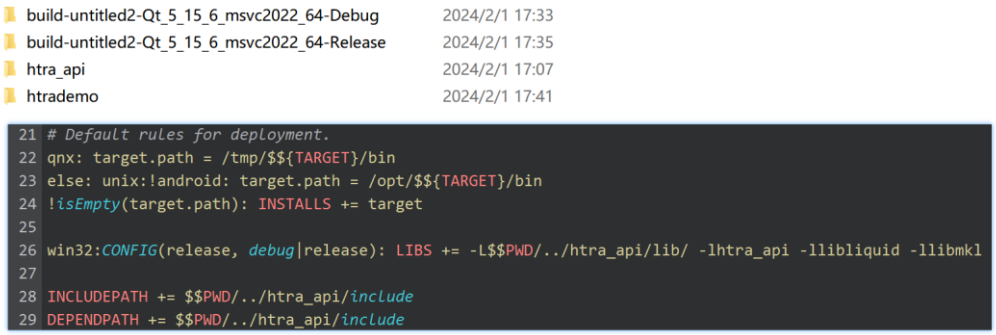

Figure 8

Step3: Place the CalFile folder inside the bin folder and compile the target path of the executable (the CalFile folder must be at the same level as the executable).

Step 4: Execute qmake and rebuild to call the library file normally.

5 Using the API in windows

5.1 Install the device driver

(1) Before installing the driver, please confirm your computer's Windows version and bit number. Then, select the corresponding driver version of your computer to install, as shown in the following Figure below.

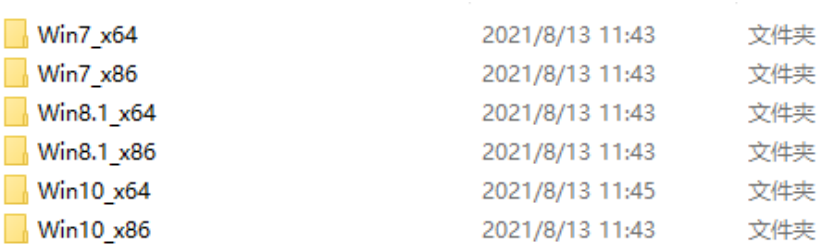

Figure 9 the driver for different Windows version and bit number.

(2) Run the Install_Driver.bat file as administrator, as shown in the following Figure below.

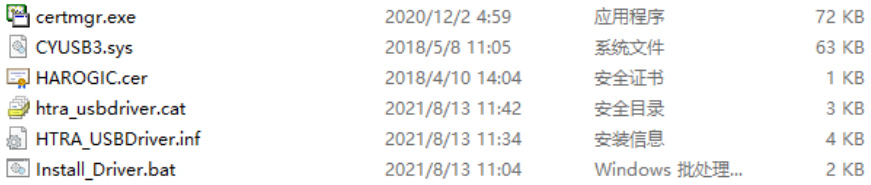

Figure 10

(3) After the driver is successfully installed, the result is as shown in the following Figure below.

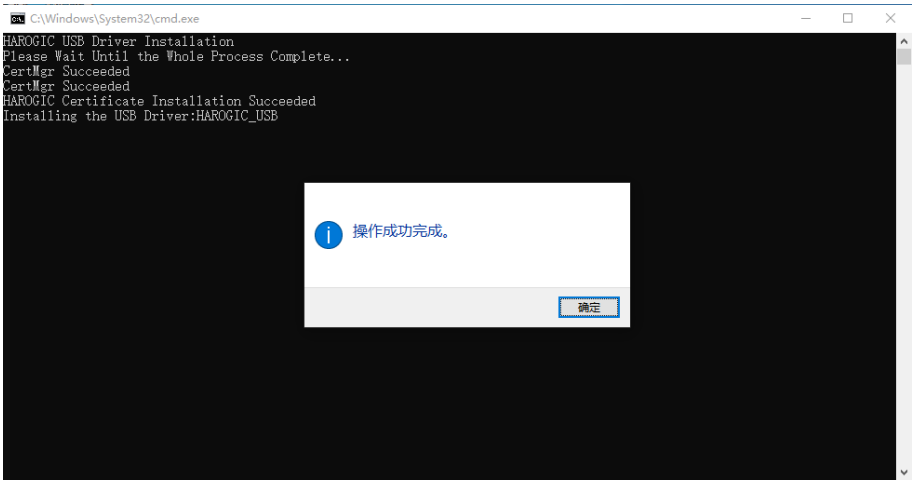

Figure 11

(4) After successful installation, you can view the newly installed devices in Device Manager, as shown in the following Figure:

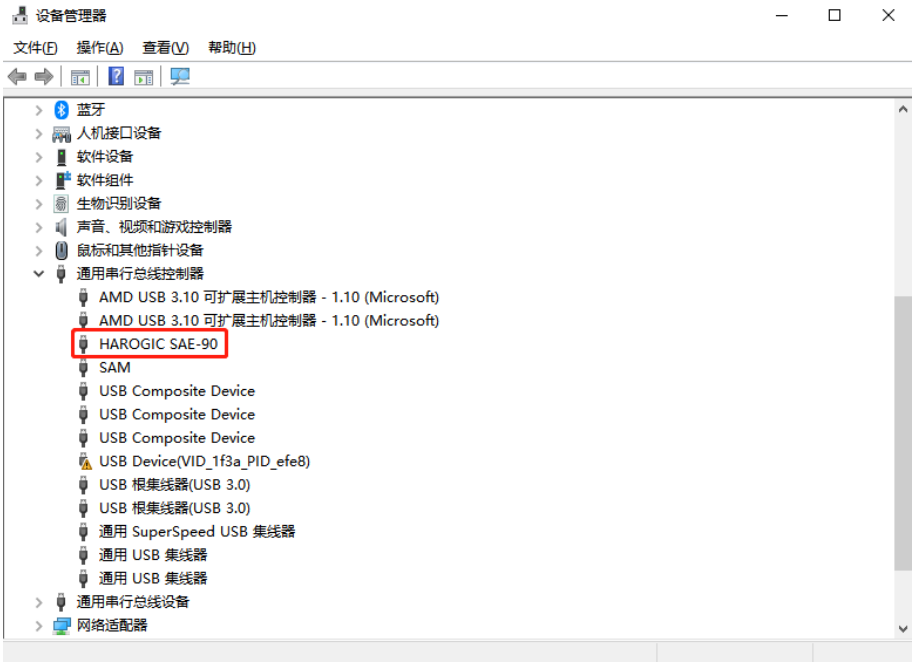

Figure 12

5.2 Using API in C development environment (VS2019)

Step 1: Open Visual Studio 2019 and Create a new Project, as shown in the following Figure below.

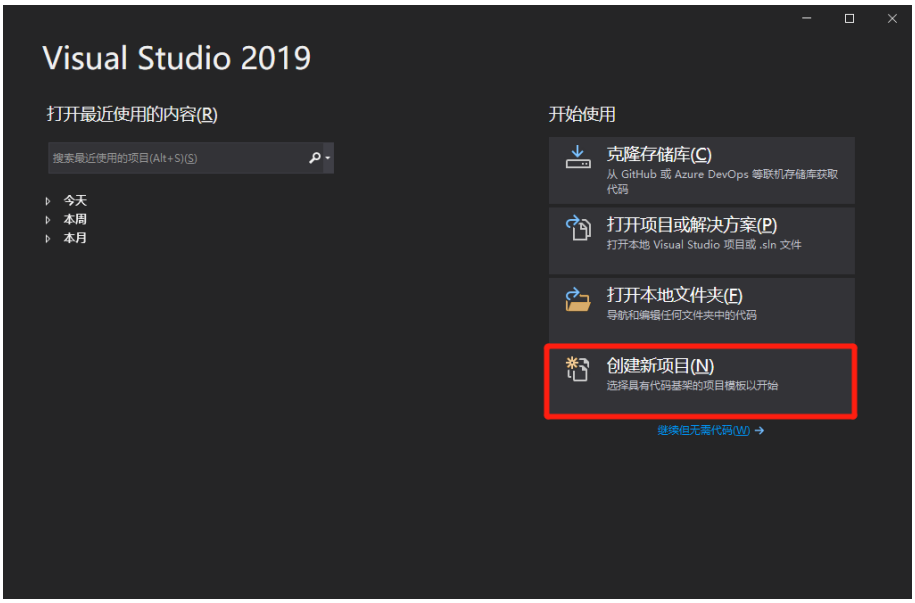

Figure 13

Step 2: Select the empty project and click Next, as shown in the following Figure below.

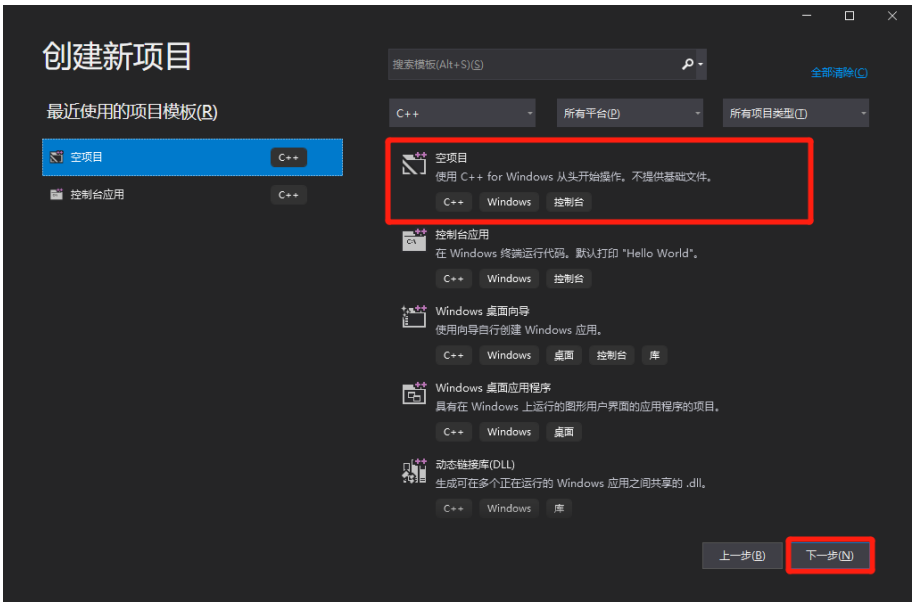

Figure 14

Step 3: Name the project with the desired name, e.g. SWP. Set the project location to the desired storage location, such as Desktop; the solution name can be the same as the project name. Place the solution and project in the same directory, which can be unchecked. Finally click Create, as shown in the following Figure below.

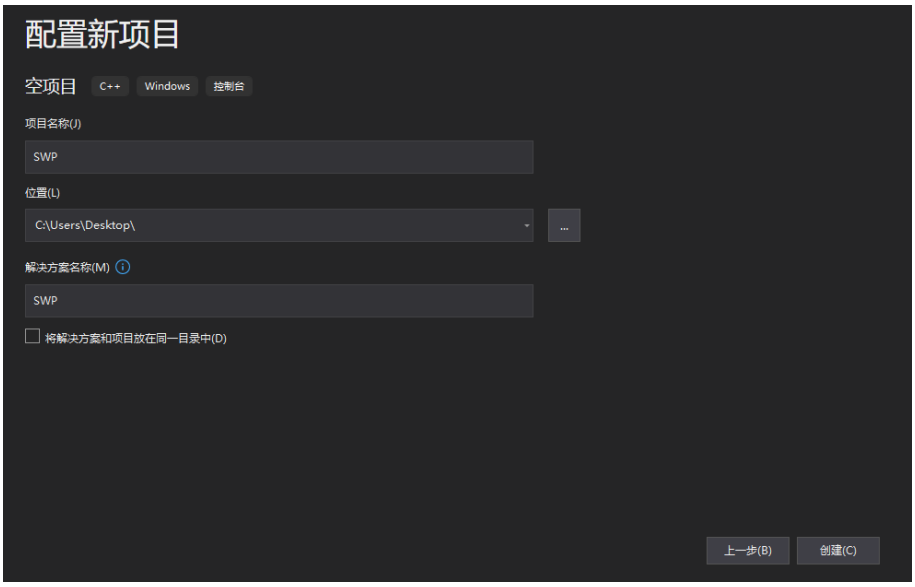

Figure 15

Step 4: After creation, place the \Windows_API\x86\htra_api folder in the USB provided by manufacturer under the project level directory, as shown in the following Figure below.

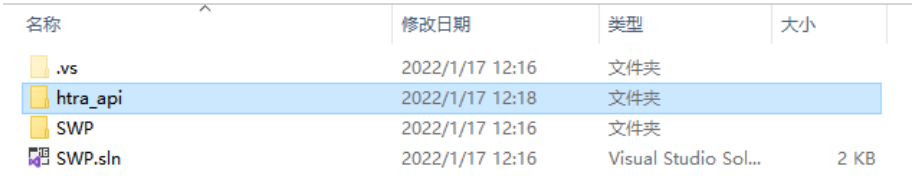

Figure 16

Step 5: Double-click to open SWP.sln, create a new SWP.cpp file in the source file, and then click Project > Properties in the menu bar above, and set the environment in Configuration Properties > Debugging to Path=. \htra_api, as shown in the following Figure below.

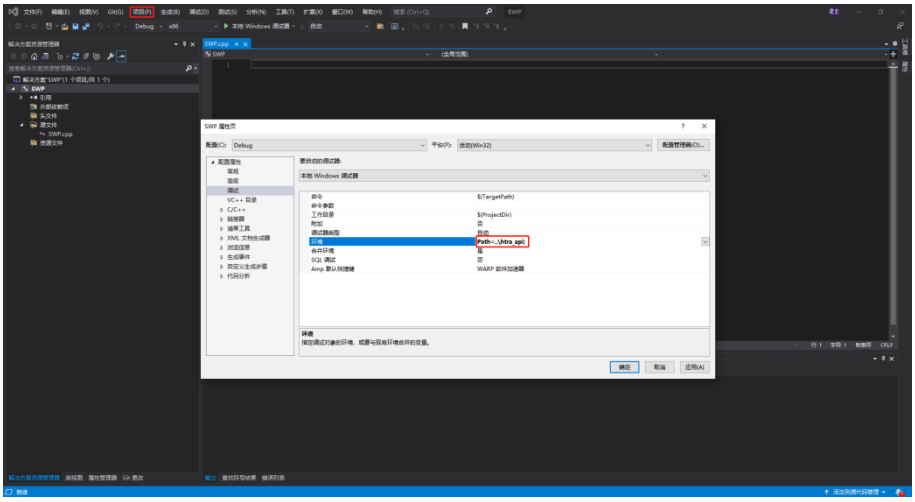

Figure 17

Step 6: Set Additional Include Directories in Configuration Properties > C/C++ > General to . \htra_api, as shown in the following Figure below.

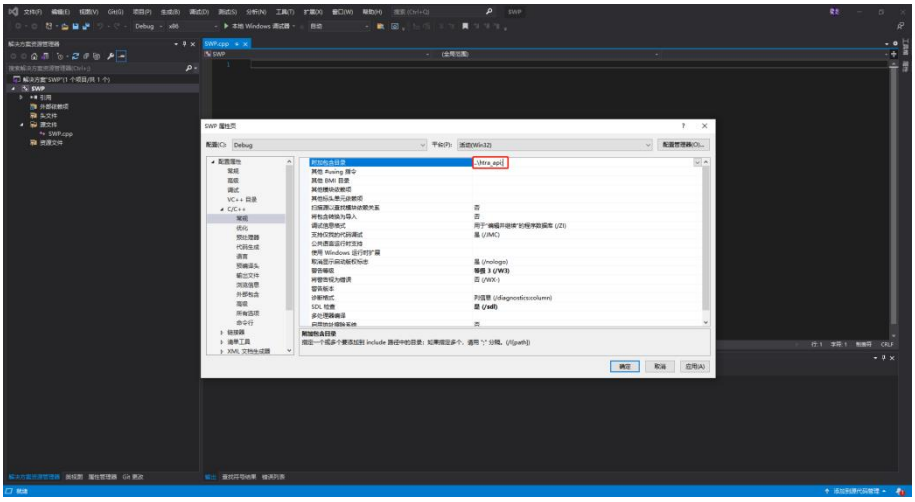

Figure 18

Step 7: Set the additional library directory in Configuration Properties > Linker > General to . \htra_api, as shown in the following Figure below.

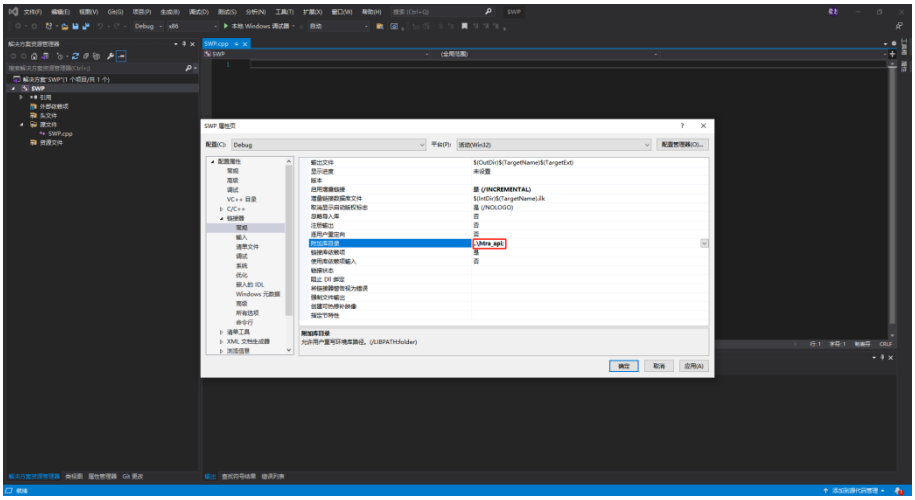

Figure 19

Step 8: Add htra_api.lib to Additional dependencies in Configuration Properties > Linker > Input, as shown in the following Figure below.

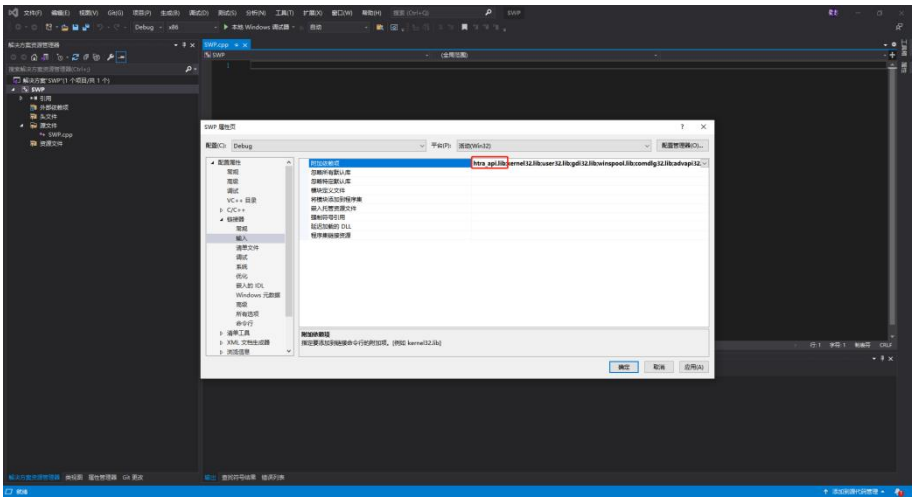

Figure 20

At this point, the C/C++ project is configured and ready for programming development.

5.3 C examples

Multiple examples in 4 modes are provided, each with a separate project that can be used independently.

Sequence	API Example	Description
1	SWPMode_Standard	instrument mode, configure and acquire the spectrum.
2	SWPMode_GUI	Reference example for developing GUI applications by calling the API.
3	IQSMode_FixedPoints	IQS fixed point mode to configure and get IQ data of user specified length.
4	IQSMode_Adaptive	IQS trigger mode, the user configures the trigger start and end signals, and obtains IQ data for the period when the

		trigger signal is valid.
5	DETMode_FixedPoints	DET fixed point mode, configure and get DET data of user specified length.
6	DETMode_Adaptive	DET trigger mode, the user configures the trigger start and end signals, and gets the DET data during the validity of the trigger signal.
7	RTAMode_FixedPoints	RTA fixed points mode, configure and get RTA data of user specified length.
8	RTAMode_Adaptive	RTA trigger mode, the user configures the trigger start and end signals, and gets the RTA data during the validity of the trigger signal.
9	IQSToSpectrum	IQ data conversion to spectrum data example.
10	MPSMode	Example of a multi-configuration mode to acquire data.
11	FMDemod	Example of FM demodulation of IQ data.
12	AMDemod	An example of AM demodulation of IQ data.

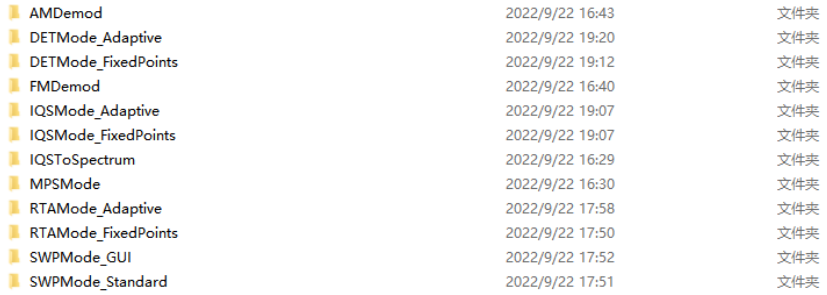

Figure 21

Take SWPMode_Standard project as an example, the htra_api folder contains all the files related to the spectrometer: CalFile folder and .dll, .lib, .h files. The CalFile folder contains the calibration files adapted to the local machine. Users need to place the htra_api files in the project level directory to use the routines, as shown in the following Figure below.

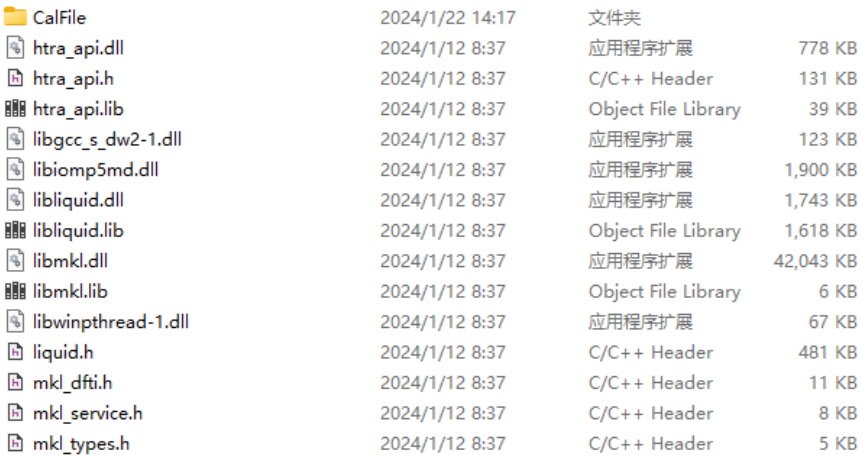
CalFile 2024/1/22 14:17 文件夹
htra_api.dll 2024/1/12 8:37 应用程序扩展 778 KB
htra_api.h 2024/1/12 8:37 C/C++ Header 131 KB
htra_api.lib 2024/1/12 8:37 Object File Library 39 KB
libgcc_s_dw2-1.dll 2024/1/12 8:37 应用程序扩展 123 KB
libiomp5md.dll 2024/1/12 8:37 应用程序扩展 1,900 KB
libliquid.dll 2024/1/12 8:37 应用程序扩展 1,743 KB
libliquid.lib 2024/1/12 8:37 Object File Library 1,618 KB
libmkl.dll 2024/1/12 8:37 应用程序扩展 42,043 KB
libmkl.lib 2024/1/12 8:37 Object File Library 6 KB
libwinpthread-1.dll 2024/1/12 8:37 应用程序扩展 67 KB
liquid.h 2024/1/12 8:37 C/C++ Header 481 KB
mkl_dfti.h 2024/1/12 8:37 C/C++ Header 11 KB
mkl_service.h 2024/1/12 8:37 C/C++ Header 8 KB
mkl_types.h 2024/1/12 8:37 C/C++ Header 5 KB

Figure 22

5.4 Using the API with examples in QT

Step 1: Open Qt Creator, create the project, select the appropriate naming and creation path, as shown in Figure 20; select the appropriate base class to create, then select the corresponding Kits suite, and then create the finished project as shown in the following figure:

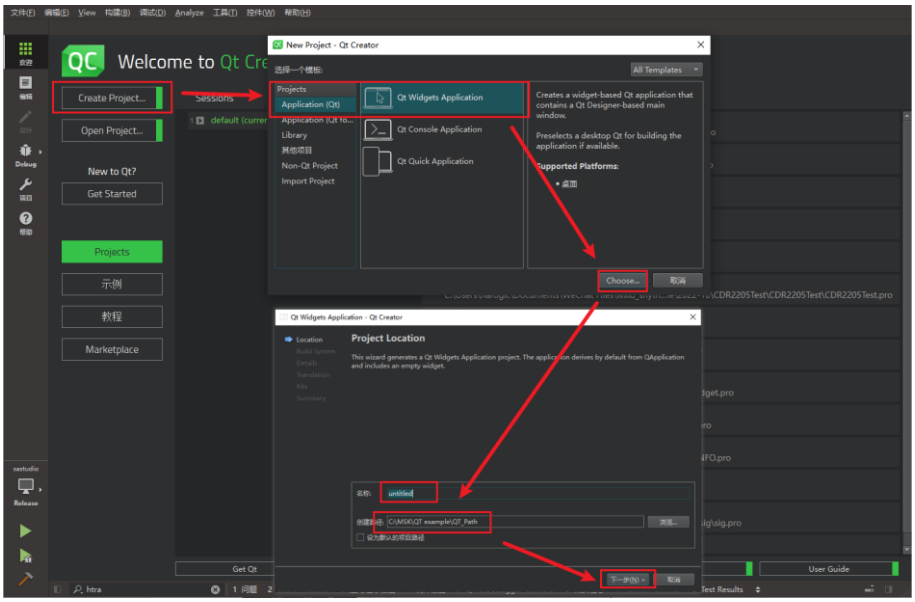

Figure 23

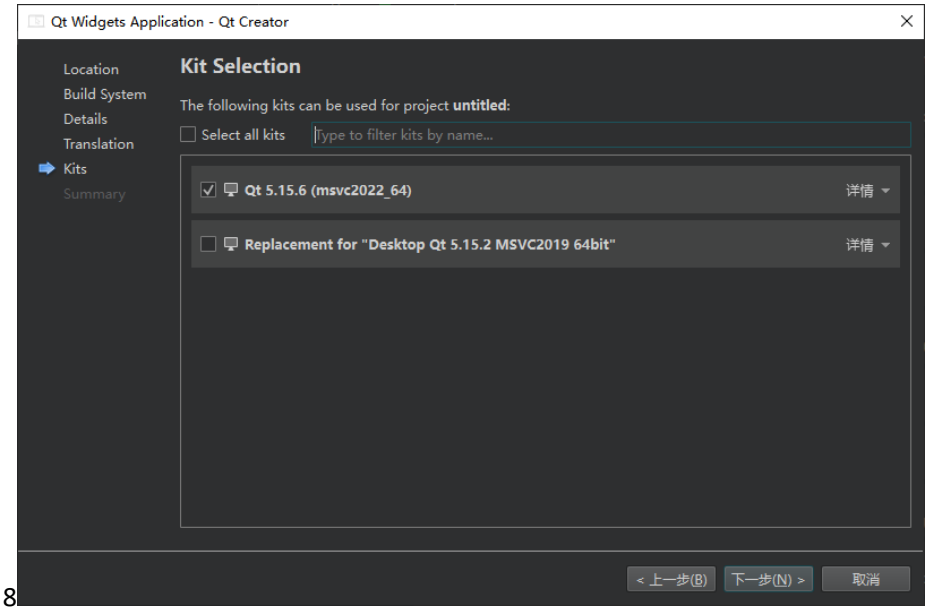

Figure 24

Step 2: Add htra_api and three-party libraries to the Qt project. Take adding x64 bit as an example, the process is shown below:

Create a new folder such as htra_api and create three folders as: dll, lib, include;

Place the corresponding suffix files in the x64 library folder in the U disk into the corresponding folder, where the three dll and CalFile folders are placed in the dll folder. As shown in the figure below;

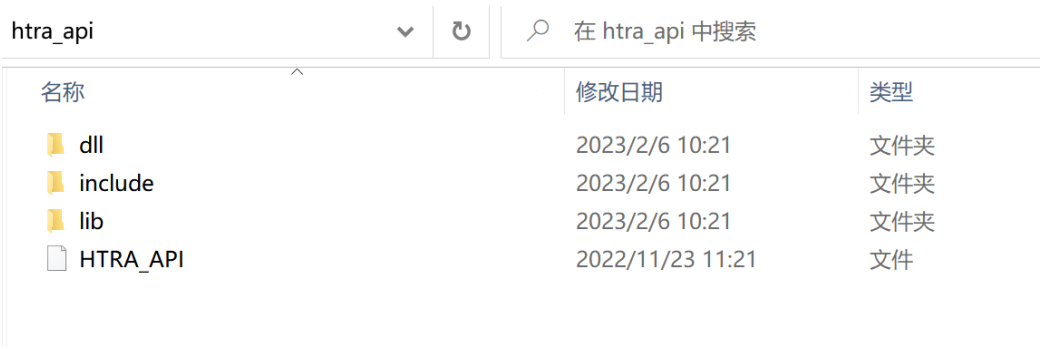

Figure 25

3) Add libraries in Qt > External libraries > Library files > select lib library files, select one and click Open, as shown in the figure below (place the htra_api folder in the Qt project path);

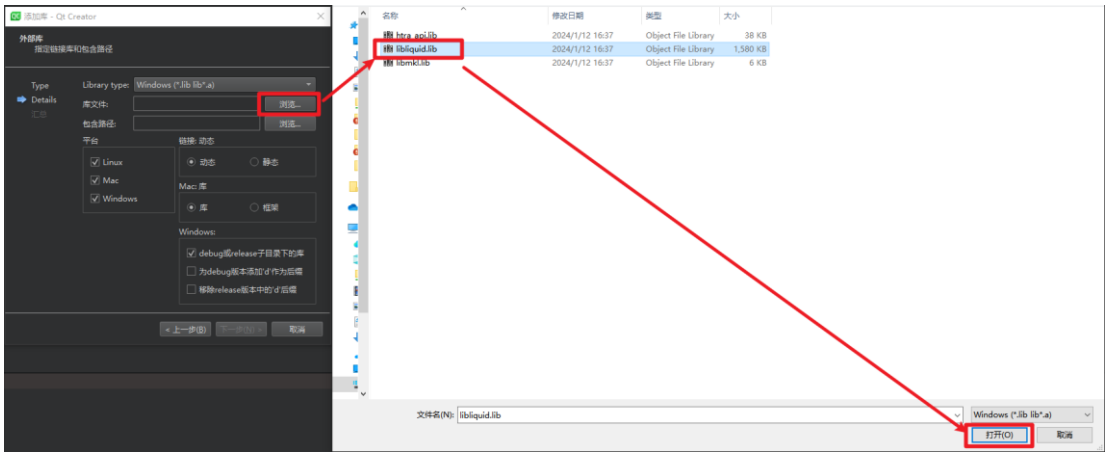

Figure 26

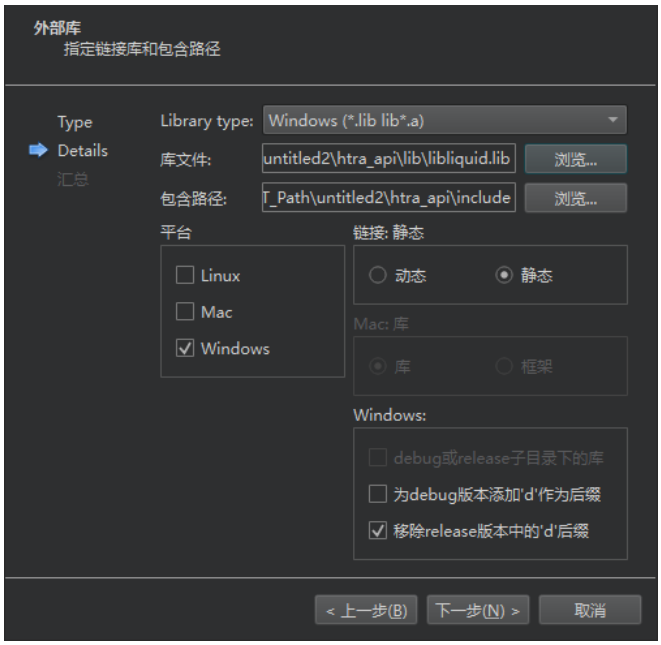

Figure 27

4) Click Next to finish, then go to the .pro file and add the remaining libmkl.lib and ibliquid.lib with reference to the htra_api.lib that has been added, as shown in the following figure after the addition

is complete;

Figure 28

5) Execute qmake and rebuild. Put ... \htra_api\dll folder in the CalFile folder, htra_api.dll, libiomp5md.dll、libmkl.dll and libliquid.dll copy and paste to the same path of the .exe, as shown in the following figure:

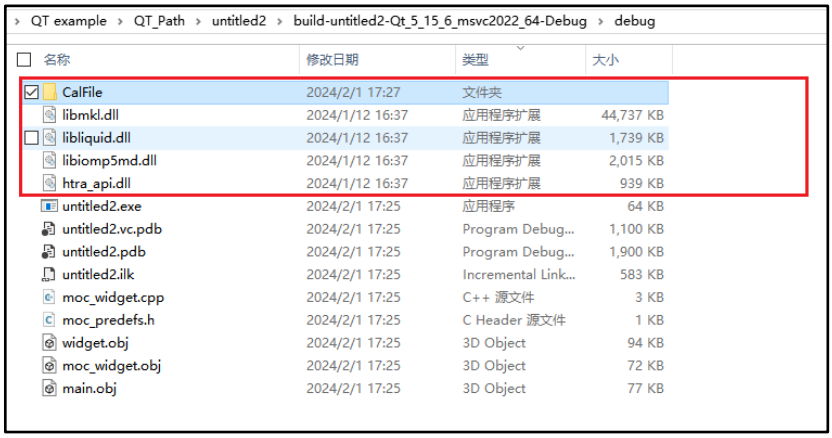

Figure 29

Step 3: At this point the library is added, you can call the library file normally, as shown in the following figure:

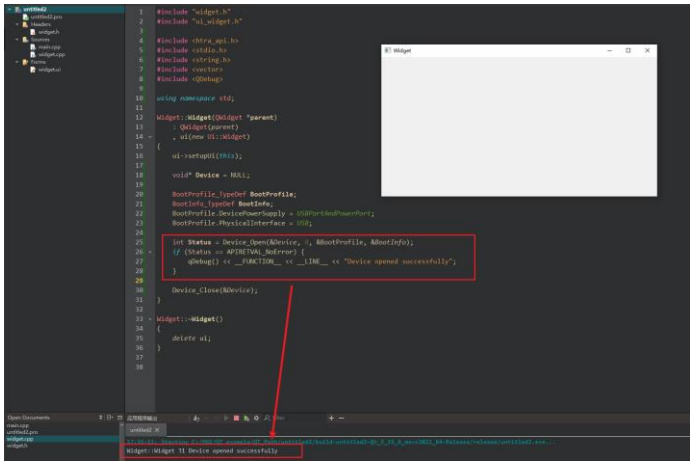

Figure 30

5.5 Using API in the Labview

First, you need to import htra_api.dll related functions in the Labview environment (Labview 2015 as an example).

Step 1: Create a folder and copy the htra_api.dll and htra_api.h in the Windows_APIx86 on the U disk, and then create a folder for placing the htra_api.dll exported function (Note: You need to change the htra_api.h encoding format to utf-8, and change the parameter type in the exported function to uint64_t double, After exporting the function, change the parameter type back to uint64_t type) in vi), as shown in the following Figure:

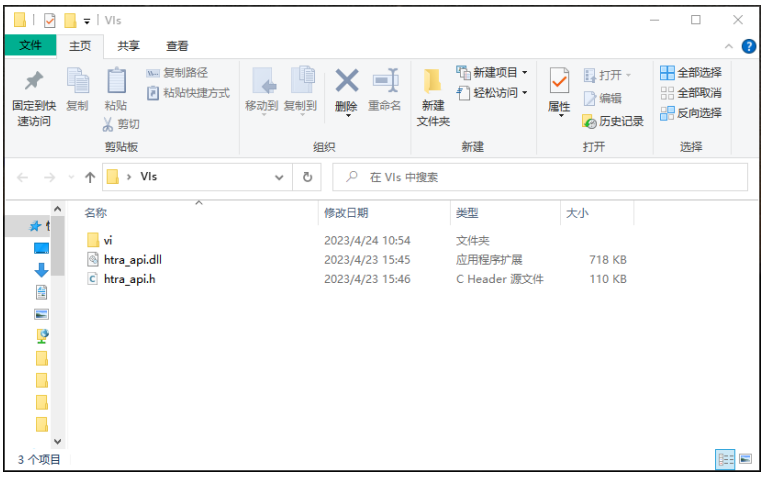

Figure 31

Step 2: Open Labview and export the API function. The following figure shows:

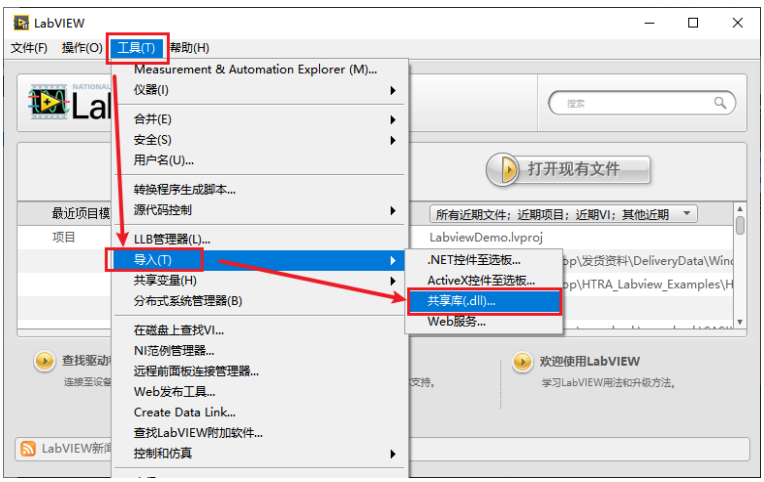

Figure 32

Step 3: Select Create VI for shared library in the new interface, click Next, select the corresponding file in the folder just created by the shared library and header file path, and the header file path can be automatically recognized after selecting the shared library path. The following figure shows:

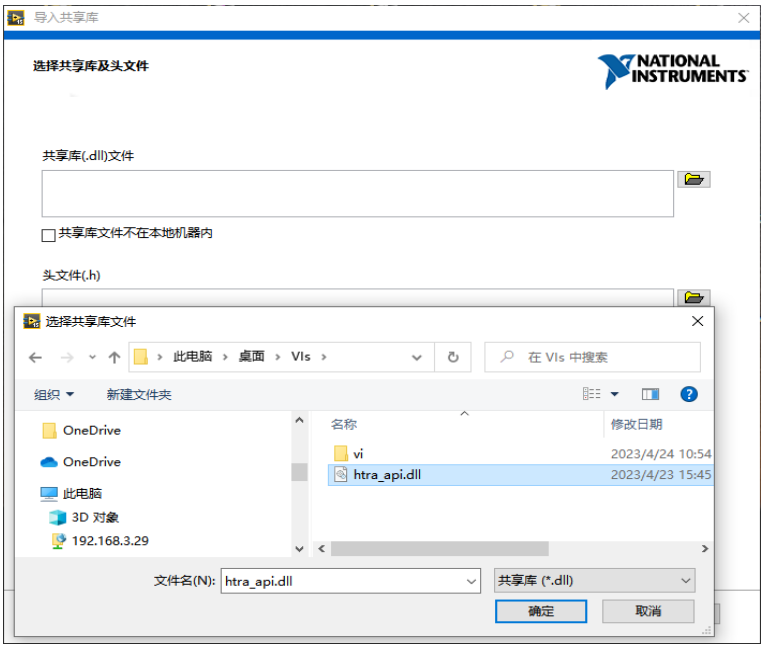

Figure 33

Step 4: Click Next, the configuration includes the path and processing definition, and the preprocessing definition can be left blank. The following figure shows:

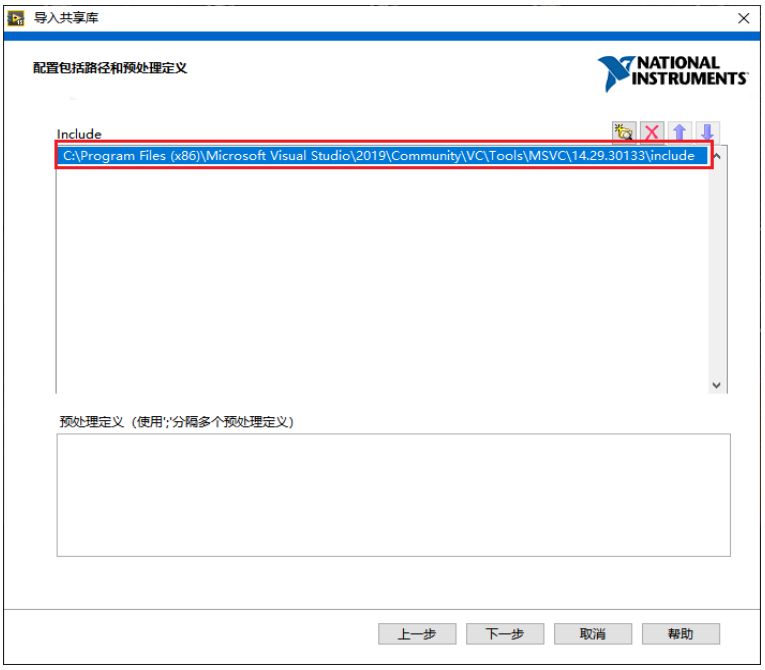

Figure 34

Step 5: Click Next and select the function to be exported (it is recommended to export one mode at a time). The following figure shows:

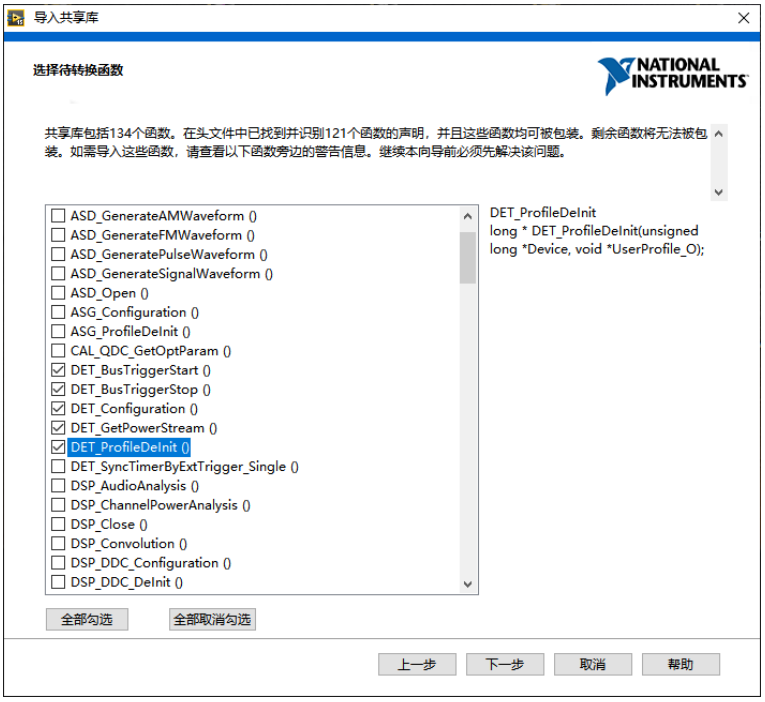

Figure 35

Step 6: Click Next, select the vi folder in the path of the project library to store the shared library and header file, select Default for other configurations, and then select the error handling method to suggest error handling, click Next, as shown in the following figure:

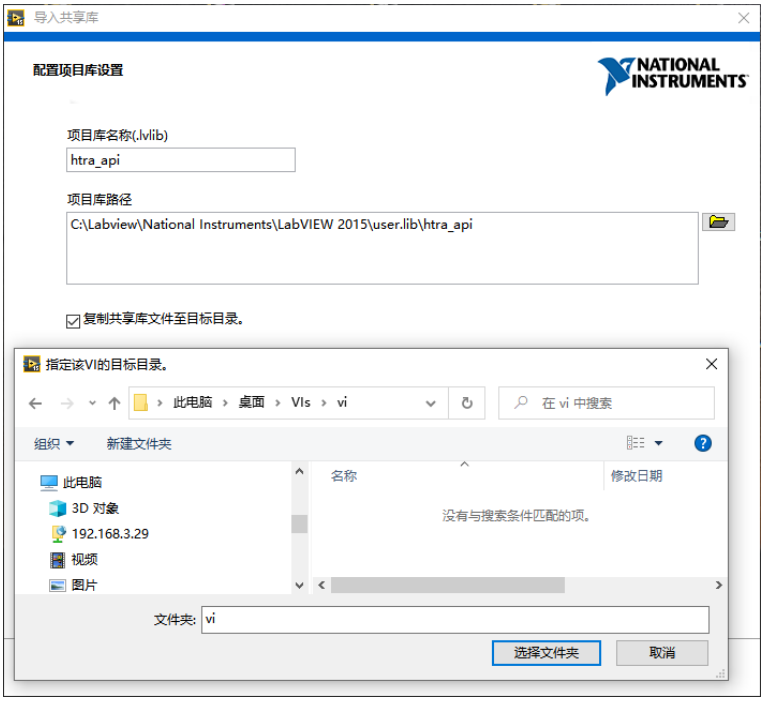

Figure 36

Step 7: The function call library node is set to run in any thread, next, wait for the function to be generated. The following figure shows:

Figure 37

Step 8: Check Open Generation Library, view report can be unchecked, click Finish, delete the exported functions and DLLs from the project in the pop-up htra_api.lvlib, and close the htra_api.lvlib after deletion. The following figure shows:

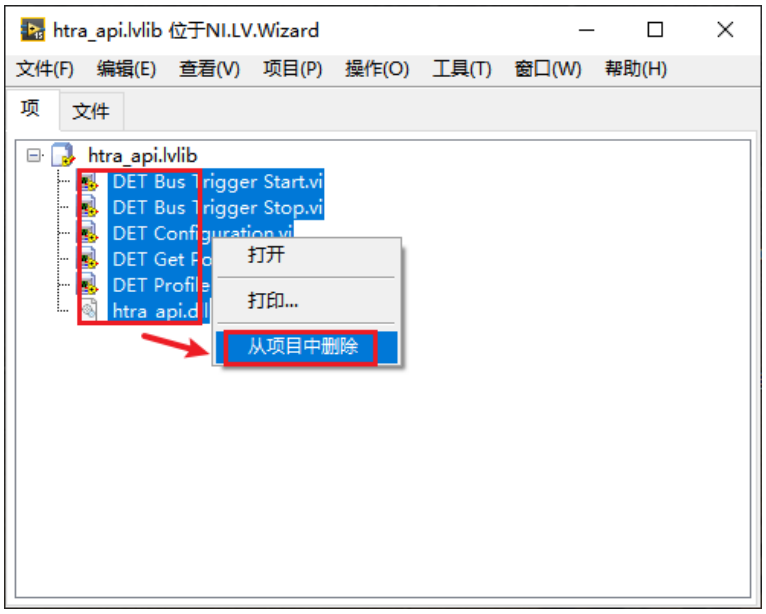

Figure 38

Step 9: After the export is completed, the folder is the API function. The following figure shows:

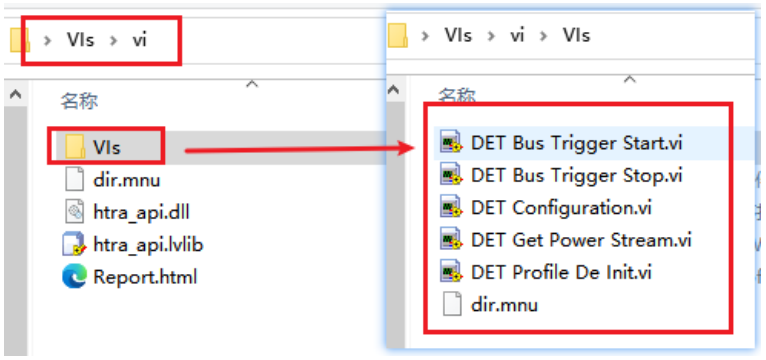

Figure 39

At this point, the export API function in the Labview environment is complete.

Further call the relevant functions of the htra_api.dll in the Labview environment.

Step 1: Open Labview and click Create Project, as shown in the following Figure:

Figure 40

Step 2: Click Done, an empty unnamed project will be created as shown in Figure below:

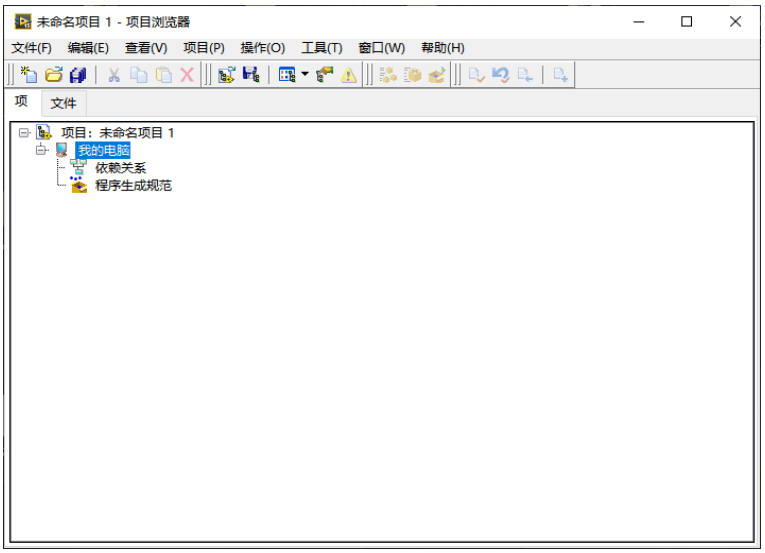

Figure 41

Step 3: Save the project, select the presave path, and give the project a name, and then OK, as shown in Figure below:

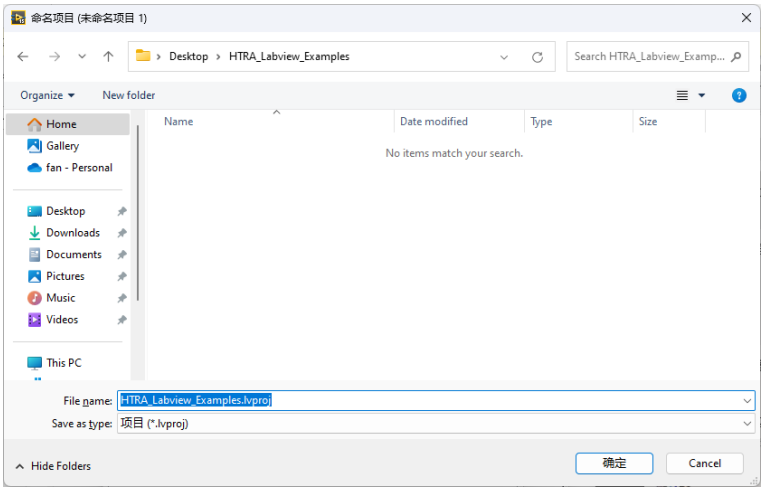

Figure 42

Step 4: Create three folders in the project sibling directory, such as GUI, to store the examples; Create another folder such as htra_api to store htra_api.dll libraries, CalFile folders, and Labview functions htra_api.dll exported; if necessary, create another folder such as Subvi to store subvi. The following figure shows:

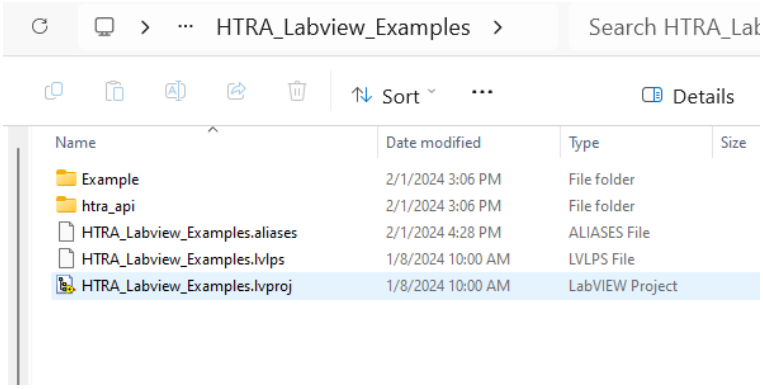

Figure 43

Step 5: Open the vi folder and copy the folder contents to the htra_api folder in the created Labview project; And copy all the contents in the Windows_APIx86 folder on the USB flash drive to the htra_api folder in the Labview project. The following figure shows:

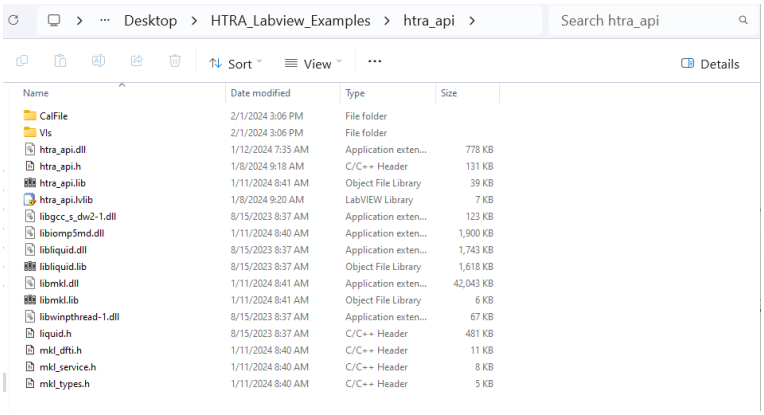

Figure 44

Step 6: Open the project, add the configured GUI folder and htra_api folder to the project, and save:

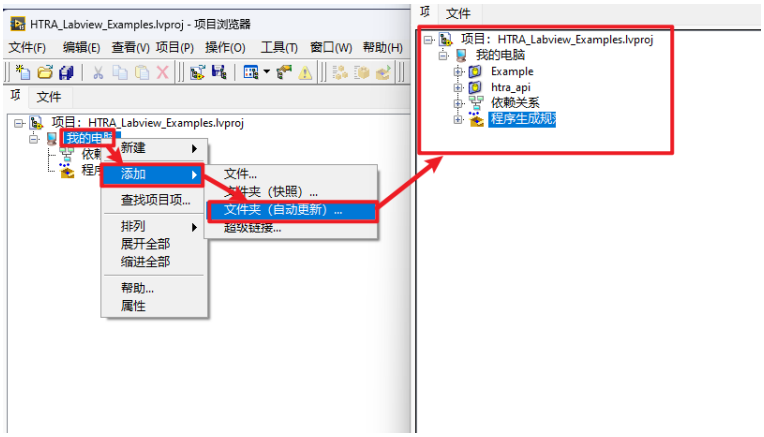

Figure 45

Step 7: Create a VI in the GUI folder, such as named DET_Example. The following figure shows:

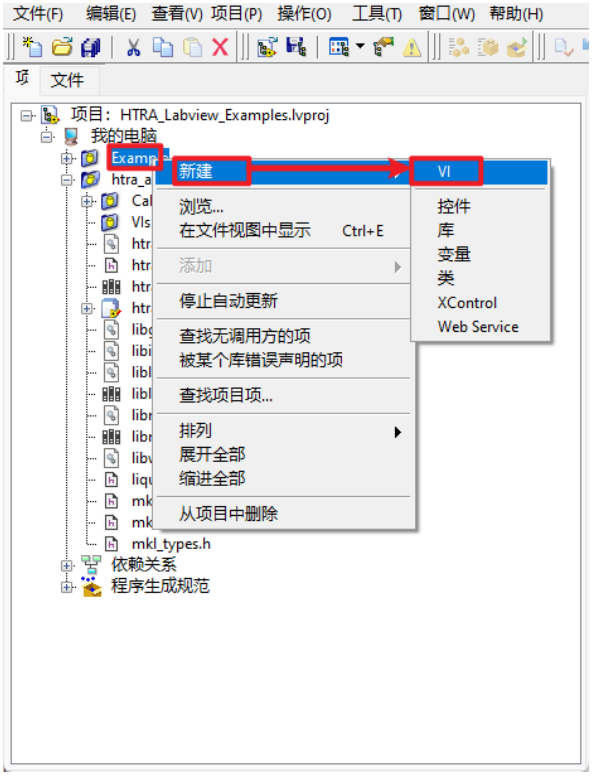

Figure 46

Step 8: You can call the exported Labview API function in the Figure page of the program box, and the call process is consistent with that in the C environment. The following figure shows:

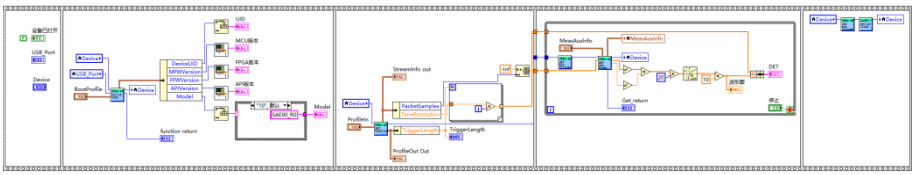

Figure 47

At this point, the API function call in the Labview environment ends.

Note: Since Labview cannot recognize uint64_t type parameters when exporting the DLL library,

you need to change the parameter type in the exported function to uint64_t double, and change the parameter type back to uint64_t type in vi after successfully exporting the function. It can also be modified directly from the Labview example provided. The following figure shows:

Figure 48

After exporting the function, please pay special attention to the number of parameter bits in the structure, Labview exported the structure needs to be supplemented to 64 bits, otherwise, the actual parameters may be serialized and cause the display to read incorrectly, as shown in the following Figure:

Figure 49

5.6 Using API in the Matlab

The following is how to call 64-bit htra_api as an example, 32-bit calling methods are basically the same as 64-bit.

Step 1: Install MSYS2 and mingw-w64 GCC, download and install link: https://www.msys2.org/.

Step 2: Configure Matlab.

(1) Enter in the Matlab terminal: setenv('MW_MINGW64_LOC','D:msys64mingw64'); Note: The address of D:msys64mingw64 is the address where the mingw64 .exe is installed on your computer;

(2) Enter “mex -setup” in the Matlab terminal, and the configuration of the C compiler is complete;

(3) To use the C++ compiler, left-click mex -setup C++ in the terminal.

Figure 50

Step 3： call dll

(1) Load DLL: loadlibrary('.htra_apihtra_api.dll', '.htra_apihtra_api.h'); The file paths of %.dll and .h must be noted, .\ represents the current .m file path.

(2) Determine whether the DLL is loaded: libisloaded('htra_api.dll');

(3) Pointers and struct pointers. For Matlab, there are no pointer-like variables in C. You can create a variable pointer through the libpointer function, and then get the value pointed to by the pointer through the get() function. You can create a struct pointer through the libstruct function, and then get the value that the struct pointer points to via the get() function. The libstruct function can be understood as converting Matlab's structs into C structures.

Variables created by libpointer or libstruct for subsequent use. If libpointer or libstruct is not used, C functions cannot be called normally or values cannot be obtained.

```
BootInfo.DeviceInfo = DeviceInfo;
BootInfo.BusSpeed = 0;
BootInfo.BusVersion = 0;
BootInfo.APIVersion = 0;
BootInfo.ErrorCodes = 1:7;
BootInfo.Errors = 0;
BootInfo.WarningCodes = 1:7;
BootInfo.Warnings = 0;

BootInfo_p = libstruct('BootInfo_TypeDef',BootInfo);

Status = calllib('htra_api', 'Device_Open', Device, DevNum,BootProfile_p,BootInfo_p);

get(BootInfo_p);
```

Figure 51

```
IQ_data = int16(1:StreamInfo_p.PacketSamples * 2);
IQ_data_p = libpointer('int16Ptr',IQ_data);

I_data = 1:StreamInfo_p.PacketSamples;
Q_data = 1:StreamInfo_p.PacketSamples;

for t=1:30

    Status = calllib('htra_api', 'IQS_BusTriggerStart', Device);
    Status = calllib('htra_api', 'IQS_GetIQStream_Data', Device, IQ_data_p);

    for i=1:StreamInfo_p.PacketSamples
        I_data(i)=IQ_data_p.value(2 * i - 1);
        Q_data(i)=IQ_data_p.value(2 * i);
        %fprintf('%d\n',i)
    end

    % figure('Name' , 'IQ');
    plot(1:StreamInfo_p.PacketSamples,I_data,1:StreamInfo_p.PacketSamples,Q_data);
    axis([0, StreamInfo_p.PacketSamples - 1, -50, 50]);
     pause(0.000001);

end
```

Figure 52

(4) When defining pointer variables, be careful to define them as *Ptr, such as uint32Ptr.

```
a=0;
b=0;
c =0;
SamplePoints = libpointer('uint32Ptr',a);
DataByte = libpointer('uint16Ptr',b);
IQSDataStartIndex = libpointer('uint32Ptr',c);


FilePath='.\20230306_194831.iq.wav';
Status = calllib('htra_api', 'Device_GetIQSDataInfo', FilePath, SamplePoints, DataByte, IQSDataStartIndex);
```

Figure 53

6 API call logic and call map

The core steps of HTRA API programming include: open the device, configure the device, acquire data, error and warning handling.

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

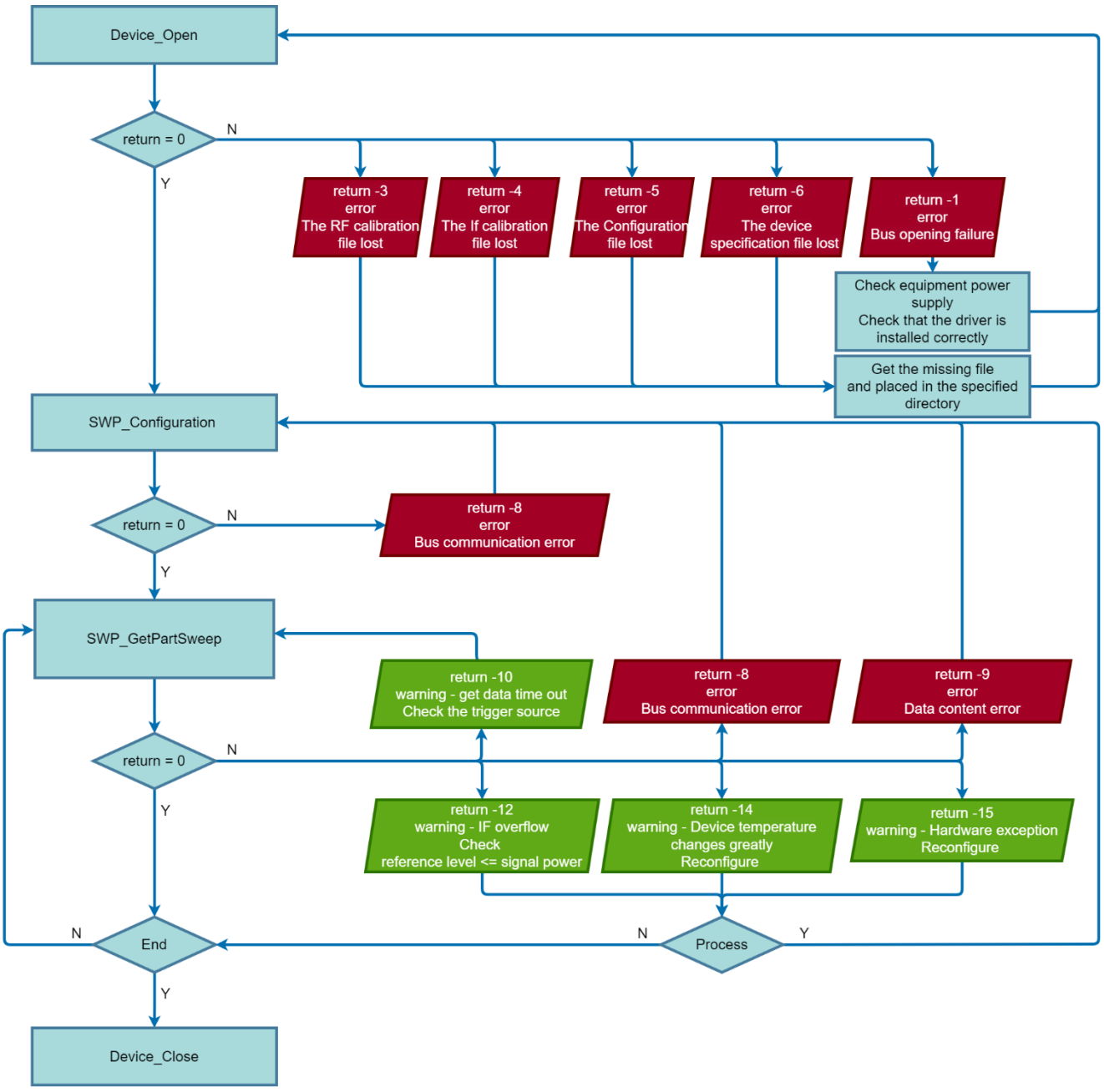

Figure 54 SWP mode call flow chart.

6.2 API call map for IQ Streaming (IQS)

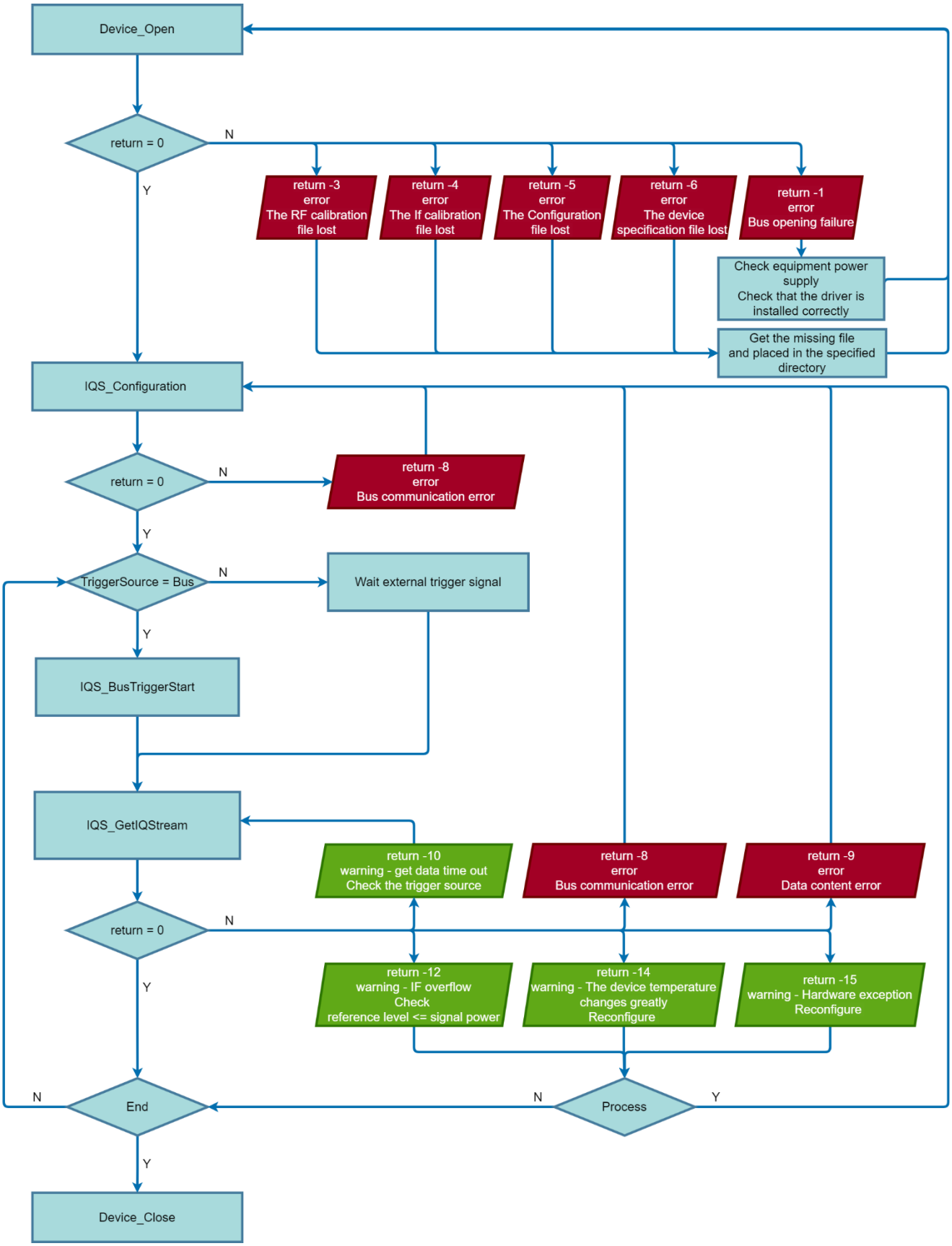

Figure 55 IQS mode call flow chart (trigger mode is Fixed).

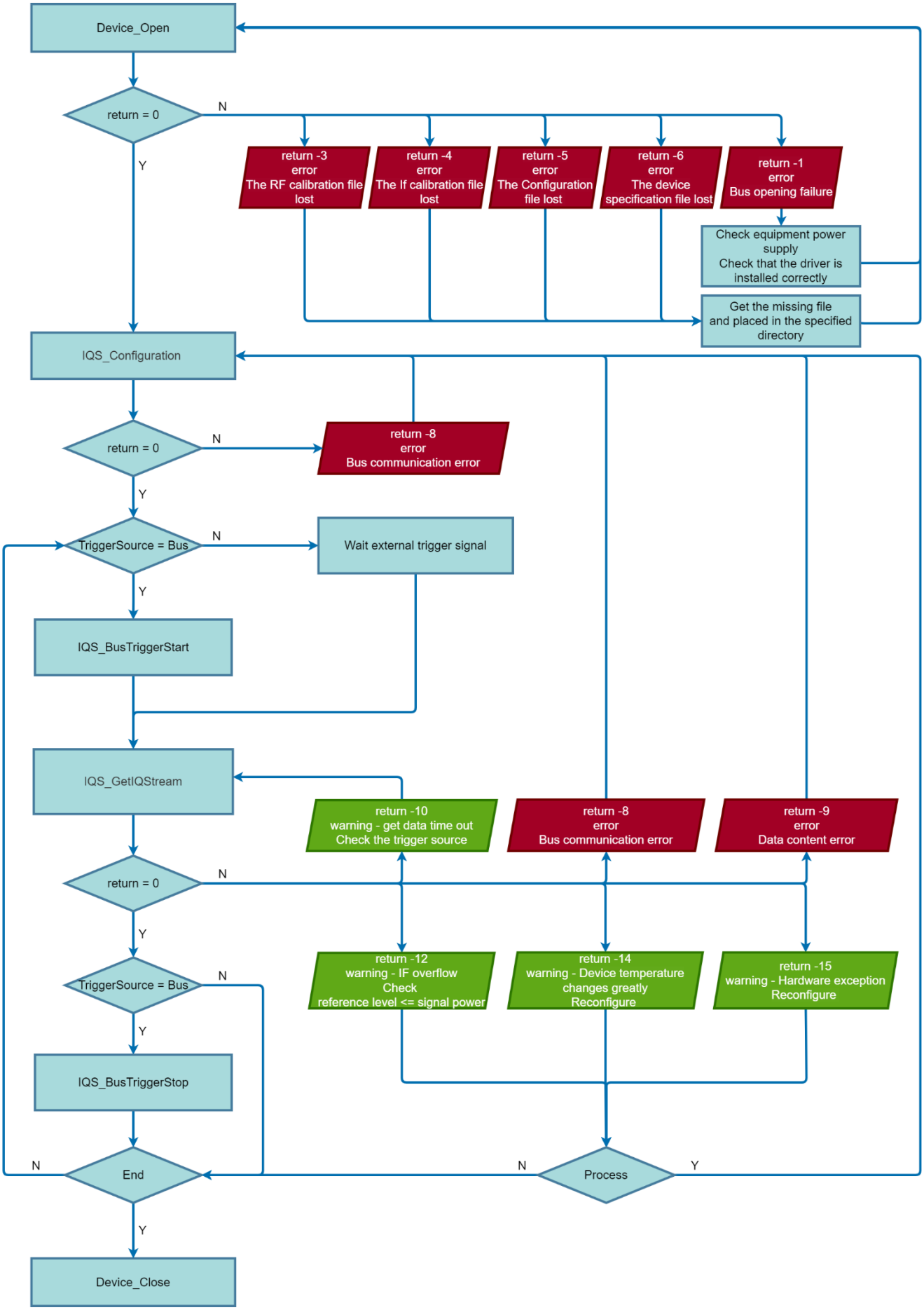

Figure 56 IQS mode call flow chart (trigger mode is Adaptive).

6.3 API call map for detection analysis (DET)

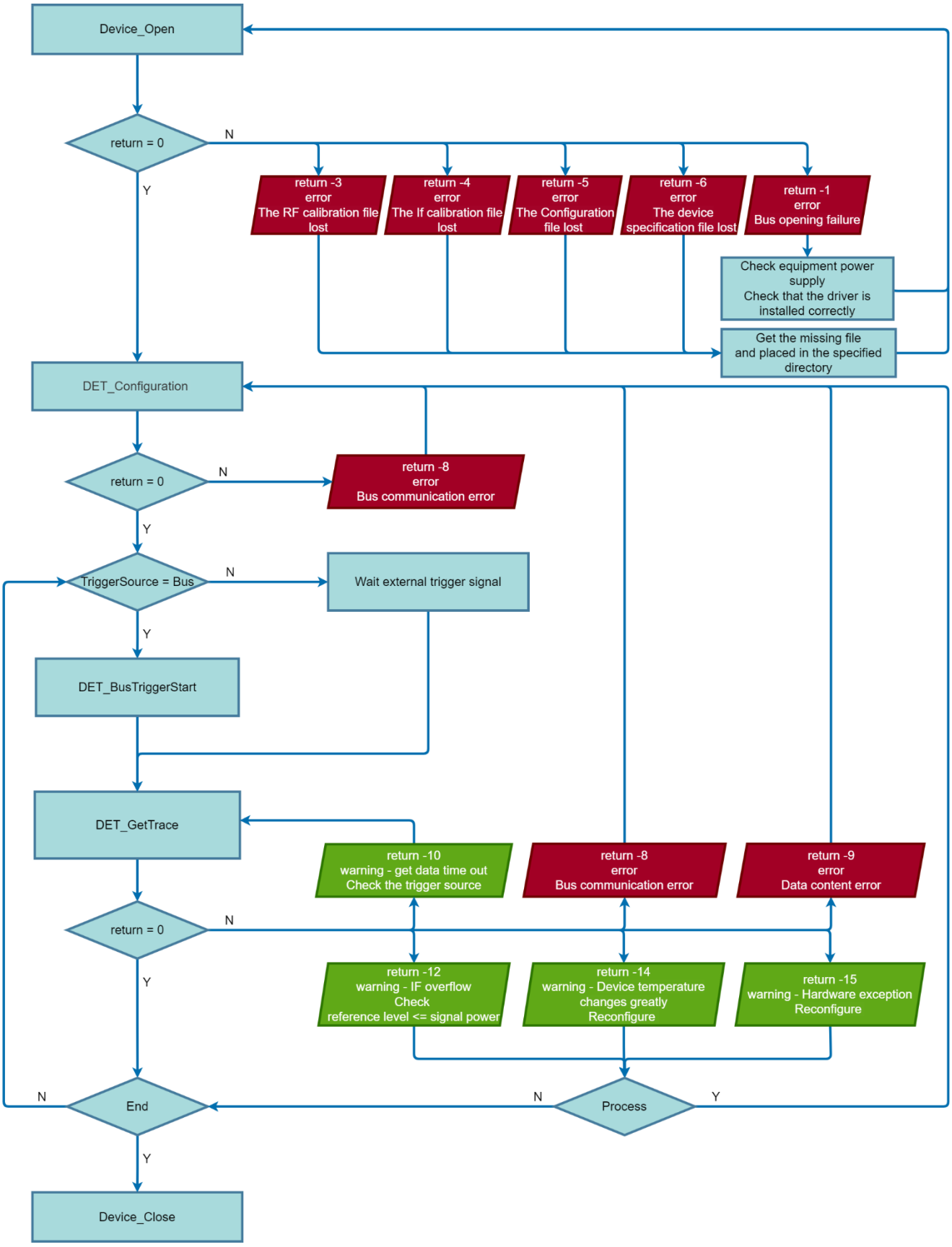

Figure 57 DET mode call flow chart (trigger mode is Fixed).

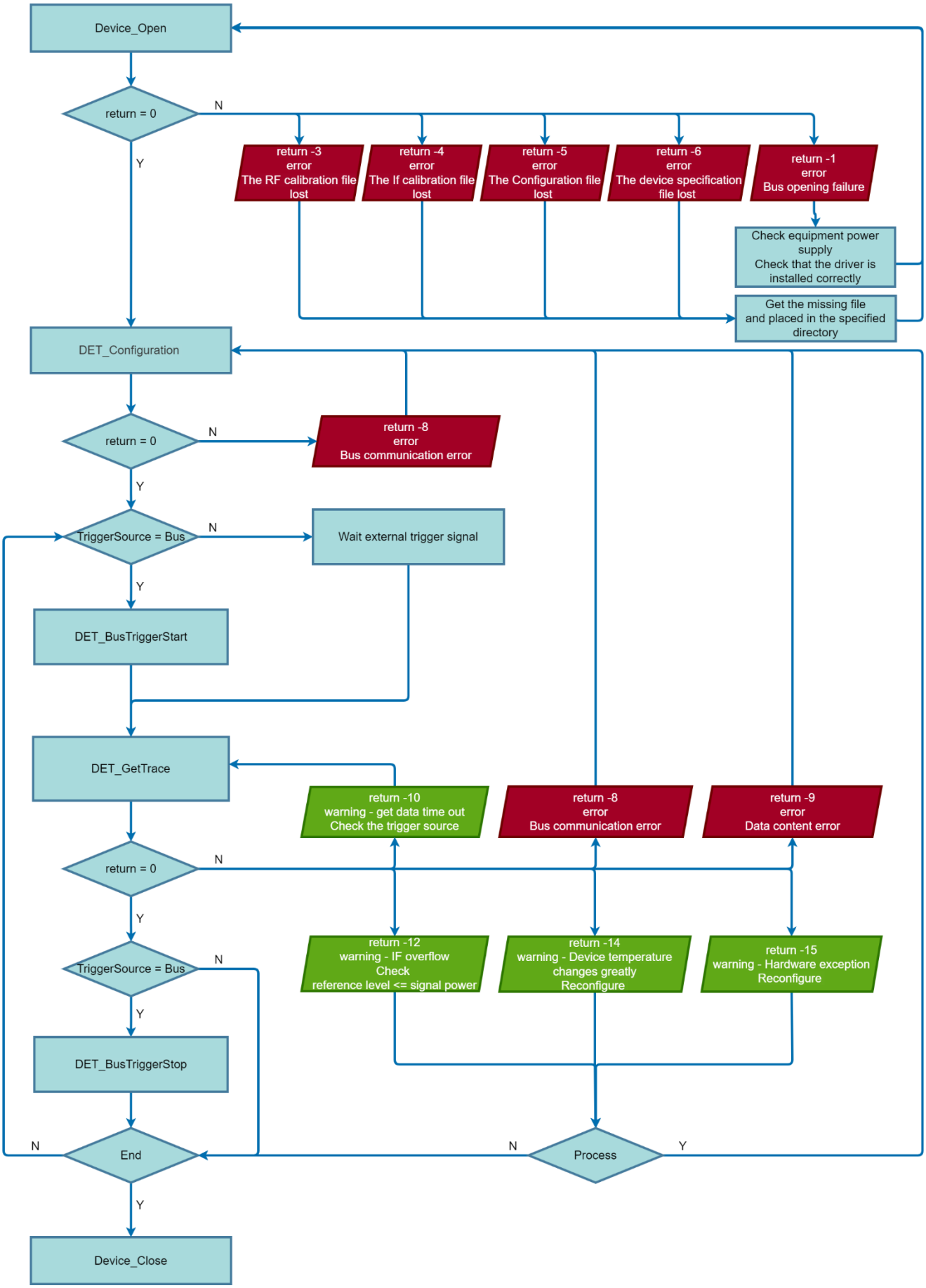

Figure 58 DET mode call flow chart (trigger mode is Adaptive).

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

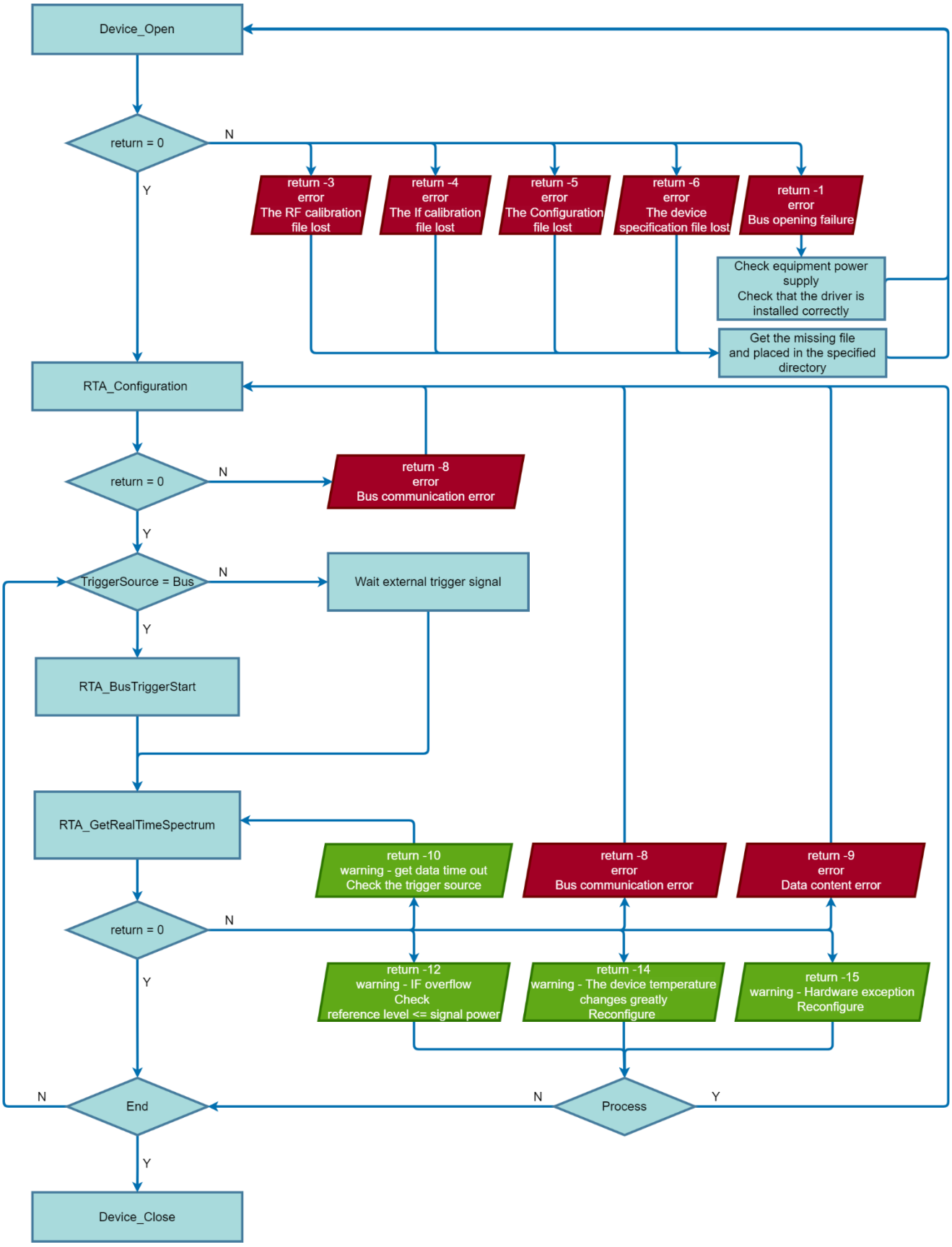

Figure 59 RTA mode call flow chart (trigger mode is Fixed).

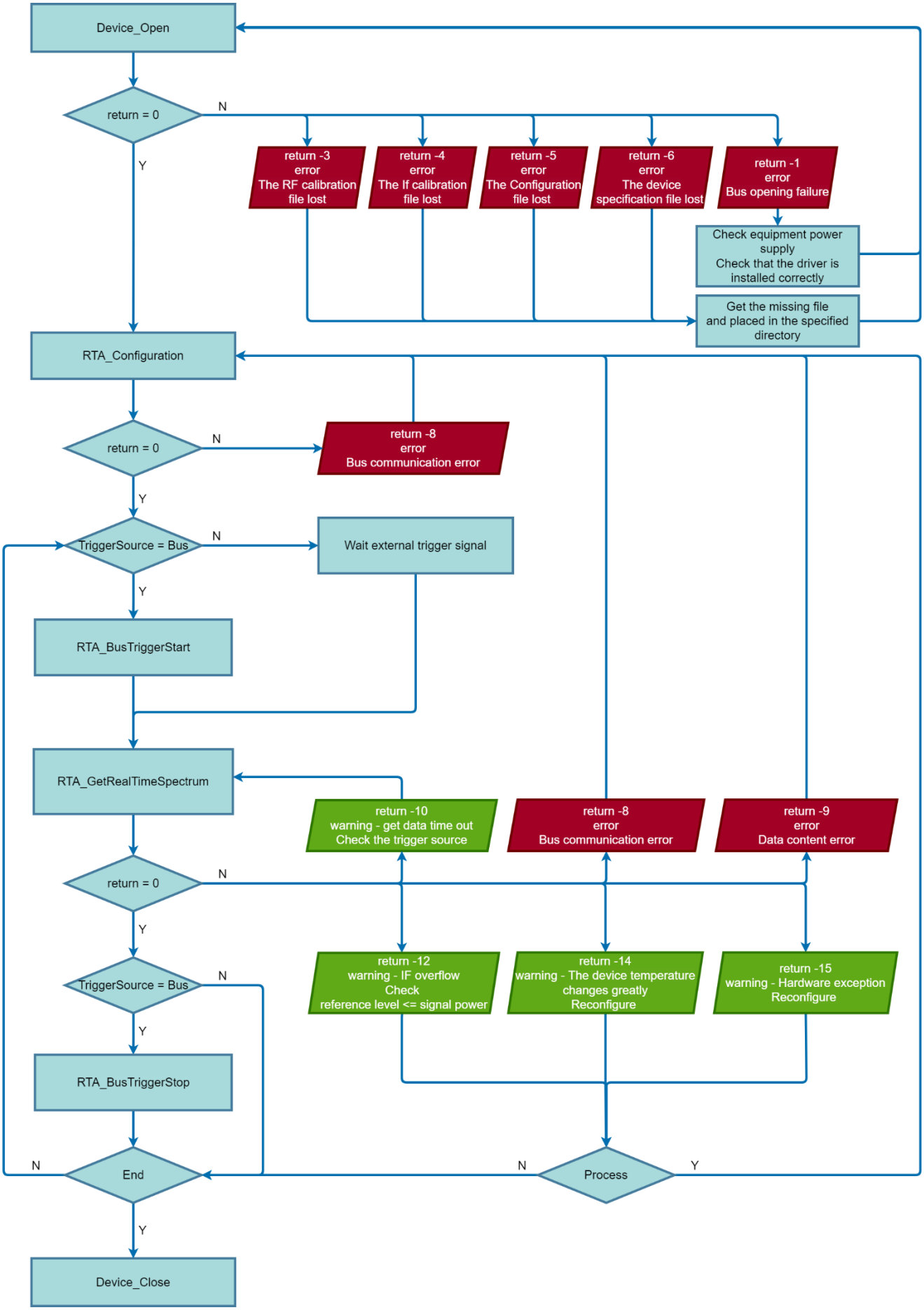

Figure 60 RTA mode call flow chart (trigger mode is Adaptive).

7 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.

7.1 System

Sequence	Parameters	Applicable Mode	Description
1	Device memory pointer void **Device	Device/SWP/IQS/DET/RTA	This parameter is a reference to the memory space required for the device to run. When the API is called, the device must be indexed by this reference.
2	DeviceUID	Device	Each device has a unique device ID, please use this ID to distinguish between different individual devices.

7.2 Amplititude

Sequence	Parameters	Applicable Mode	Description
1	RefLevel_dBm	SWP/IQS/DET/RTA	The system uses auto gain setting by default. Attenuator (Atten) and preamplifier are automatically configured by the system based on RefLevel_dBm. In SWP mode, RefLevel_dBm indicates the maximum input power that the system can maintain accurate results within the set frequency range. For fixed-frequency modes, such as IQS/DET/RTA mode, RefLevel_dBm applies to the set CenterFreq_Hz. The system retains a margin in handling RefLevel_dBm, typically 1~6 dB. Sometimes even if the input power is greater than RefLevel_dBm, the system will still not report a saturation warning.
	Atten		Atten default is automatic (Atten = -1), when the system is completely by RefLevel_dBm specified channel gain. When you need to manually set the channel gain, please set Atten to the specified value.
2	Preamplifier	SWP/IQS/DET/RTA	For devices configured with a preamplifier, the preamplifier is arranged before the attenuator and is located at the front of the system. It has a significant impact on system noise and linearity. Turning on the preamplifier reduces noise, but reduces the maximum

			linear input power and the maximum power loss of the system. The system automatically turns on the preamplifier according to RefLevel_dBm or turns off the preamplifier in any case (to avoid overload damage).
3	AnalogIFBW Grade	SWP/IQS/DET/RTA	For devices configured with multiple analog IF filters, you can choose different IF grades with different characteristics. Different IF grade have different performance in terms of gain, bandwidth, group delay, etc. Please select the appropriate IF grade according to your application needs.
4	IFGainGrade	SWP/IQS/DET/RTA	System IF gain includes RF gain and IF gain, the difference between each IF gain grade is 1dB~3dB. The system allows users to set the IF gain to optimize spurious, linearity and noise level. For example, when the total gain (RefLevel_dBm) remains fixed, increasing IF gain will reduce the receiver's mixer input power, contributing to improved spurious but deteriorated noise performance. Conversely reducing IF gain will increase the receiver's mixer power, which may deteriorate spurious but improve noise performance.

7.3 Frequency

Sequence	Parameters	Applicable Mode	Description
1	FreqAssignment	SWP	SWP mode allows the user to specify the frequency scan range in StartStop mode or CenterSpan mode.
2	StartFreq_Hz StopFreq_Hz CenterFreq_Hz Span_Hz		
3	TracePointStrategy	SWP	Spectrum analysis is operated via employing a fully FFT-based analysis method. The system will return the nearest available points according to the RBW and sweeping range. Moreover, the returned data will be slightly outside the set frequency range, the user can make certain interceptions as needed. When TracePointStrategy = PointsAccuracyPreferred, the system will return the nearest actual available points.
4	TracePoints		
5	TraceBinSize_Hz		
6	TraceAlign		

7.4 Analysis

Sequence	Parameters and Concepts	Applicable mode	Description
1	SpurRejection	SWP	The SpurRejection function can effectively suppress most of the combined component spurious (no optimization effect on the remaining response of the system), but it will reduce the sweep speed and the measurement capability of time-varying signals. For steady-state signal testing (e.g., monophonic signals), turning on SpurRejection can effectively improve the spurious-free dynamic range of the measurement; for fast time-varying signal testing (e.g., modulated signals), turning on SpurRejection may result in probabilistic signal loss or inaccurate power values. By observing the spectrum difference between SpurRejection On and Off mode, it helps to determine whether this function can be turned on for the current test scenario.
2	PowerBalance	SWP	In SWP mode, users can set the PowerBalance parameter to make a trade-off between sweeping speed and power consumption of the device. The system will allow the highest sweeping speed when the PowerBalance is 0; non-zero setting range of power balance is generally 40 ~ 1000, increasing the value leads to lower scanning speed and power consumption. Special attention should be paid to the case of SpurRejection on, PowerBalance control will lead to a further decline in the system's ability to detect time-varying signals. For applications with high time-varying signal detection, this parameter needs to be set carefully.
3	Window	SWP/RTA	When performing FFT-based spectrum analysis, the system provides various windows with different advantages, so please choose according to your needs. For example, the FlatTop window has good amplitude accuracy; the Blackman-Nuttall window has a narrow main flap width and high frequency resolution and faster analysis speed, suitable for high frequency resolution and fast sweeping application.
4	Detector TracePoints	SWP/RTA	Native spectrum traces may have hundreds of thousands of data points, thus detector is used to compress the spectrum traces in a specified way. Due to the limitations of the

			underlying hardware and software in the FFT analysis and frame splicing, the system will achive the nearest available trace points to the desired number of points by the user configuration. Therefore, please pay attention to the return information about trace points to get the actual TracePoints during each configuration.
5	FFTExcutionStrategy	SWP	Users can make the system automatically select between FPGA and CPU calculations based on the RBW or to have only FPGA or CPU calculations. FPGA calculation can significantly reduce the CPU processing power requirement, but in the case of small and medium RBW (RBW ≤ 5kHz) the sweeping speed is slower due to the limitation of single FFT points; CPU calculation allows the system to use more than 64k points of FFT points, obtaining higher sweeping speed than FPGA in the case of small and medium RBW.
6	FrameTimeMultiple	SWP	Sampling duration at a single frequency point is used as the definition of SweepTime. For applications that require a longer SweepTime, you can extend the system's sample duration at a single frequency point by setting SweepTimeMultiple (indicating the multiple of the sample duration relative to the minimum duration). At RBW = VBW, the system will return the result output with thighest total power of multiple FFT frames in the sampling window after extending SweepTime. This feature helps users to obtain accurate signal power when analyzing time-varying signals such as pulse signal.
7	FrameDetector	SWP/RTA	When there are multiple frames of spectrum analysis results at the same local oscillation frequency point, it is allowed to convert the results of multiple frames into the desired output form with user-specified rules (FrameDetector).
8	SweepTime		SweepTime is defined as the total time required to complete a scan from StartFreq_Hz to StopFreq_Hz in this system. When the input sweep time is less than the minimum sweep time of the device, the system will still sweep at the minimum time. Since the time required for the system to sweep a trace is related to several uncertainties (CPU usage, code design, etc.), with the maximum scan speed setting, the system cannot directly specify the total time required to acquire a

			trace and can only give an estimated value. When a larger SweepTime is set (SweepTime > TraceInfo.EstimatedMinSweepTime), the system will control the SweepTime exactly.
9	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.
10	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.

7.5 Default Units

Frequency	Hz
Power	dBm
Voltage	V
Time	s

8 Device (main functions)

8.1 Device_Open

int Device_Open(void Device, int DeviceNum, const BootProfile_TypeDef* BootProfile BootInfo_TypeDef* BootInfo)**	
Description	
The device must be opened first before all future API calls. Calling this function to open a device and a handle will return. When there are multiple devices, open the devices separately by specifying different device number (DeviceNum) and the corresponding device handle can be obtained. Use the device handle to specify which device is to be manipulated in all the following API calls.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle. Use the device handle to specify which device is to be manipulated in the API calls.
int DeviceNum	Device Number. If there are multiple devices, you should open device with a different device number. The device number must accumulate from 0.
BootProfile_TypeDef *BootProfile	Configurations for the device boot.
BootInfo_TypeDef *BootInfo	Feedback information of the devic boot.
BootProfile_TypeDef	
PhysicalInterface_TypeDef PhysicalInterface	Specify the physical interface of the device. The interface must be set correctly, otherwise the driver cannot be opened. A device may be equipped with multiple interfaces. 1) USB: USB interface for data transfer; 2) QSFP: QSFP+ interface for data transfer; 3) ETH: 100M/1000M Ethernet interface for data transfer; 4) HLVDS: LVDS Bus for data transfer; 5) VIRTUAL: virtual Buses for simulation and debugging.
DevicePowerSupply_TypeDef DevicePowerSupply	Specify the power supply of the device. It must be set correctly or the device may not be opened. 1) USBPortAndPowerPort: USB data port and independent power port dual power supply; 2) USBPortOnly: only USB data port power; 3) Others: using a non-USB Bus such as the ETH.
IPVersion_TypeDef ETH_IPVersion	Ethernet IP version: 1) IPv4; 2) IPv6.

uint8_t ETH_IPAddress[16]	When the physical interface is ETH, it is used to specify the IP address. For example, the target IP address is 192.168.1.100, ETH_IPAddress[0] = 192; ETH_IPAddress[1] = 168;ETH_IPAddress[2] = 1;ETH_IPAddress[3] = 100; When the IPv4 is used, only the first 4 numbers need to be filled in, and the rest of the array does not need to be filled in. IPv6 is not supported currently.
uint16_t ETH_RemotePort	Specify the listening port if the physical interface is ETH.
int32_t ETH_ErrorCode	Return the error code of ETH connection if the physical interface is ETH.
int32_t ETH_ReadTimeOut	Specify the time out for data read if the physical interface is ETH. If the Device Configuration & Data functions does not respond within this time, the function will return an error code.
BootInfo_TypeDef	
DeviceInfo_TypeDef DeviceInfo	Device information including device UID, model, Hardware version, etc.
uint32_t BusSpeed	Data bandwidth of the data bus.
uint32_t BusVersion	Firmware version of the data bus.
uint32_t APIVersion	The version of the API under used.
int ErrorCodes[7]	Error code list.
int Errors	Error counts.
int WarningCodes[7]	Warning code list.
int Warning	Warning counts.
DeviceInfo_TypeDef	
uint64_t DeviceUID	Unique ID of the device.
uint16_t Model	Device model.
uint16_t HardwareVersion	Version of the device hardware.
uint16_t MFWVersion	Version of the mcu firmware.
uint16_t FFWVersion	Version of the FPGA firmware.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
```

8.2 Device_Close

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.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL; **BootProfile_TypeDef *BootProfile_IO;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef *BootInfo;** **Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);** **Status = Device_Close(&Device);**	

8.3 Device_QueryDeviceState

int Device_QueryDeviceState(void Device, DeviceState_TypeDef* DeviceState)**	
Description	
get device status information including device temperature, hardware working status, geographic time information (requires optional support), etc. without interruption to data transfer. There may be a small lag in the information.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
DeviceState_TypeDef *DeviceState	Structure pointer to information about the current device state. The information update to the latest value after calling this function.
DeviceState_TypeDef	
int16_t Temperature	The temperature of the device.
uint16_t RFState	The state of the device RF part.
uint16_t BBState	The state of the device base band.
double AbsoluteTimeStamp	The absolute time stamp provided by the insystem GNSS.
float Latitude	The latitude provided by the insystem GNSS.
float Longitude	The longitude provided by the insystem GNSS.
uint16_t GainPattern	The gain pattern of the device.
int64_t RFCFreq	Center frequency.
uint32_t ConvertPattern	Frequency convert pattern.
uint32_t NCOFTW	Frequency tunning word of the NCO.

uint32_t SampleRate	SampleRate = ADCSamlpRate/Decimate factor.
uint16_t CPU_BCFlag	Unavailable.
uint16_t IFOverflow	1) 0: IFOverflow is not detected; 2) 1: IFOverflow is detected.
uint16_t DecimateFactor	Decimate factor.
uint16_t OptionState	The state of the options.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DeviceState_TypeDef *DeviceState;
Status = Device_QueryDeviceInfo(&Device, &DeviceState);
```

8.4 Device_SetIPAddr

int Device_SetIPAddr(void Device, const IPVersion_TypeDef ETH_IPVersion, const uint8_t ETH_IPAddress[], const uint8_t SubnetMask[])**	
Description	
In devices with ETH interface (such as NX series products), call this function to modify the IP address.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
IPVersion_TypeDef ETH_IPVersion	1) IPv4: IPV4 is used; 2) IPv6: IPV6 is used.
const uint8_t ETH_IPAddress[]	IP address in string.
const uint8_t SubnetMask[]	Subnet mask in string.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
```

```
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
const IPVersion_TypeDef  ETH_IPVersion = 0;
const uint8_t ETH_IPAddress[4] = {192,168,1,100};
const uint8_t SubnetMask[4] ={192,168,1,1};
Status = Device_SetIPAddr(&Device, ETH_IPVersion, ETH_IPAddress, SubnetMask);
```

9 Device (the rest)

9.1 Device_QueryDeviceInfo

int Device_QueryDeviceInfo(void Device, DeviceInfo_TypeDef* DeviceInfo)**	
Description	
Get the device information without interruption to data transfer. There may be a small lag in the information.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
DeviceInfo_TypeDef *DeviceInfo	Structure pointer to information about the device information.
DeviceInfo_TypeDef	
uint64_t DeviceUID	Unique ID of the device.
uint16_t Model	Device model.
uint16_t HardwareVersion	Version of the device hardware.
uint16_t MFWVersion	Version of the mcu firmware.
uint16_t FFWVersion	Version of the FPGA firmware.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DeviceInfo_TypeDef DeviceInfo;
Status = Device_QueryDeviceInfo(&Device, &DeviceInfo);
```

9.2 Device_QueryDeviceInfo_Realtime

int Device_QueryDeviceInfo_Realtime(void Device, DeviceInfo_TypeDef* DeviceInfo)**	
Description	
Get the strict real time device information with minor interruption to data transfer.	
Compatibility	0.55.0 and later.
Parameter description	Consistent with function Device_QueryDeviceInfo.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DeviceInfo_TypeDef DeviceInfo;
Status = Device_QueryDeviceInfo_Realtime(&Device, &DeviceInfo);
```

9.3 Device_QueryDeviceState_Realtime

int Device_QueryDeviceState_Realtime(void Device, DeviceState_TypeDef* DeviceState)**	
Description	
Get the strict real time device state with minor interruption to data transfer.	
Compatibility	0.55.0 and later.
Parameter description	Consistent with function Device_QueryDeviceState.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DeviceState_TypeDef *DeviceState;
Status = Device_QueryDeviceInfo_Realtime(&Device, &DeviceState);
```

9.4 Device_UpdateFirmware

int Device_UpdateFirmware(int DeviceNum, const char *path)	
Description	
Update the MCU device firmware.	
Compatibility	0.55.0 and later.
Parameter description	
int DeviceNum	Device Number. If there are multiple devices, you should open each device with a different device number. The device number must accumulate from 0.

const char *path	Specify the address of the file where the firmware is to be written.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL; **char *path = "D:\\ MCUFirmware.bin";** **Status = Device_MCUFirmwareUpdate(&Device, path);**	

9.5 Device_SetSysPowerState

int Device_SetSysPowerState(void **Device, SysPowerMode_TypeDef SysPowerMode)	
Description	
set the power state of the device, including power-on operation, RF partial power-down, RF partial standby, and so on.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
SysPowerMode_TypeDef SysPowerMode	1) PowerOn: power on all the parts of the device; 2) RFPowerOff: power down the RF part of the device; 3) RFStandby: set the RF part into standby.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL; **BootProfile_TypeDef *BootProfile_IO;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef *BootInfo;** **Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);** **SysPowerMode_TypeDef SysPowerMode = PowerON;** **Status = Device_SetSysPowerState(&Device, SysPowerMode);**	

9.6 Device_SetFanState

int Device_SetFanState(void Device, SetFanState_TypeDef SetFanState, float ThreshouldTemperature)**	
Description	
The device fan operating mode.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.

SetFanState_TypeDef **SetFanState**	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: the system fan will be automatically turned on if the threshold temperature is reached.
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 5 celsius below the threshold.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL; **BootProfile_TypeDef *BootProfile_IO;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef *BootInfo;** **Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);** **SetFanState_TypeDef SetFanState = 0;** **float Temperature = 0;** **Status = Device_SetFanState(&Device, SetFanState, Temperature);**	

9.7 Device_CalibrateRefClock

int Device_CalibrateRefClock(void Device, ClkCalibrationSource_TypeDef ClkCalibrationSource, const double TriggerPeriod_s, const uint64_t TriggerCount, const bool RewriteRFCal, double* RefCLKFreq_Hz)**	
Description	
Calibrate the reference clock frequency by the 1PPS of insystem GNSS or the periodic external trigger.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
ClkCalibrationSource_TypeDef **ClkCalibrationSource**	Timing signal source used for calibration: 1) CalibrateByExternal: calibrate by the periodic external trigger; 2) CalibrateByGNSS1PPS: calibrate by the 1PPS of insystem GNSS.
const double TriggerPeriod_s	The period of periodic external trigger. The accuracy of the period will determin the calibration effect.
const uint64_t TriggerCount	Specify the trigger counts used for calibration. Normally, the more triggers used (in espense of more time) the better calibration effect may

	be achieved. If the 1PPS is used, trigger counts larger than 30 is suggested.
const bool RewriteRFCal	1) 0：the calibration result is not written into the calibration file, and the calibration result is invalid after the device is powered down; 2) 1：the calibration results are written to a calibration file and remain calibrated after the device is powered on.
double* RefCLKFreq_Hz	Feedback the new reference clock frequency obtained from this calibration.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	
int Status = -1; int DeviceNum = 0;void *Device = NULL; **BootProfile_TypeDef BootProfile_IO;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef BootInfo;** **Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);** **const double ExtTriggerPeriod_s = 1;** **uint64_t CalibrationTimes = 1 * 60;** **bool RewriteRFCal = false;** **double RefCLKFreq_Hz = 0;** **Status = Device_CalibrateRefClock(&Device, ExtTriggerPeriod_s, CalibrationTimes, RewriteRFCal,& RefCLKFreq_Hz);**	

9.8 Device_Reboot

int Device_Reboot(void Device)**	
Description	
In devices with ETH interface (such as NX series products), call this function to restart the device.	
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.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL; **BootProfile_TypeDef *BootProfile_IO; BootInfo_TypeDef *BootInfo;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);**	

Status = Device_Reboot(&Device);

9.9 Device_RestartNetwork

int Device_RestartNetwork(void Device)**	
Description	
In devices with ETH interface (such as NX series products), call this function to restart the network so that the modified IP address takes effect.	
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.
Invocation constraints	This function needs to be called after Device_Open or after Device_SetIPAddr to change the address.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL; **BootProfile_TypeDef *BootProfile_IO;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef *BootInfo;** **Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);** **const IPVersion_TypeDef ETH_IPVersion = 0;** **const uint8_t ETH_IPAddress[4] = {192,168,1,12};** **const uint8_t SubnetMask[4] ={192,168,1,1};** **Status = Device_SetIPAddr(&Device, ETH_IPVersion, ETH_IPAddress, SubnetMask);** **Status = Device_RestartNetwork(&Device);**	

9.10 Device SetGNSSAntennaState

int Device_SetGNSSAntennaState(void Device, const GNSSAntennaState_TypeDef GNSSAntennaState)**	
Description	
Set GNSS antenna status when using the GPS function.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
GNSSAntennaState_TypeDef	GNSS antenna status;
GNSSAntennaState	GNSS_AntennaExternal: external antenna; GNSS_AntennaInternal: internal antenna;

Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function should be called after Device_Open.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
//If SA series devices are used, the data transfer interface is set to USB
//BootProfile.PhysicalInterface =USB;
BootProfile.PhysicalInterface =ETH;
//if NX series devices are used, the data transfer  interface is set to ETH
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
GNSSAntennaState_TypeDef GNSSAntennaState;
Status = Device_SetGNSSAntennaState (&Device, GNSSAntennaState);
```

9.11 Device_GetGNSSAntennaState

Int Device_GetGNSSAntennaState(void Device, GNSSAntennaState_TypeDef *GNSSAntennaState)**	
Description	
Obtain GNSS antenna status in a non-real-time manner (hardware opt.) without interrupting the data acquisition, but the information is only updated after the data packet is acquired.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
GNSSAntennaState_TypeDef	GNSS antenna status;
GNSSAntennaState	GNSS_AntennaExternal: external antenna; GNSS_AntennaInternal: internal antenna;
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function should be called after Device_Open.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
//If SA series devices are used, the data transfer interface is set to USB
//BootProfile.PhysicalInterface =USB;
BootProfile.PhysicalInterface =ETH;
//if NX series  devices are used, the data transfer  interface is set to ETH
```

```
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
GNSSAntennaState_TypeDef GNSSAntennaState;
Status = Device_GetGNSSAntennaState (&Device, GNSSAntennaState);
```

9.12 Device_GetAntennaState_Realtime

Int Device_GetGNSSAntennaState-Realtime (void Device, GNSSAntennaState_TypeDef *GNSSAntennaState)**	
Description	
obtain the GNSS antenna status in a real-time manner (hardware opt.) that is, but the real-time mode will occupy the data channel for a short time.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
GNSSAntennaState_TypeDef	GNSS antenna status;
GNSSAntennaState	GNSS_AntennaExternal: external antenna; GNSS_AntennaInternal: internal antenna;
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function should be called after Device_Open.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
//If SA series devices are used, the data transfer interface is set to USB
//BootProfile.PhysicalInterface =USB;
BootProfile.PhysicalInterface =ETH;
//if NX series  devices are used, the data transfer  interface is set to ETH
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
GNSSAntennaState_TypeDef GNSSAntennaState;
Status = Device_GetGNSSAntennaState_Realtime(&Device, GNSSAntennaState);
```

9.13 Device_AnysisGNSSTime

int Device_AnysisGNSSTime(double ABSTimestamp, int16_t* hour, int16_t* minute, int16_t* second, int16_t* Year, int16_t* month, int16_t* day)
Description

Obtain GNSS time and date information. (Hardware Opt.)	
Compatibility	0.55.0 and later.
Parameter description	
double ABSTimestamp	Absolute timestamp corresponding to the current data packet
int16_t* hour	Hour in GNSS time and date information
int16_t* minute	Minute in GNSS time and date information
int16_t* second	Second in GNSS time and date information
int16_t* Year	Year in GNSS time and date information
int16_t* month	Month in GNSS time and date information
int16_t* day	Day in GNSS time and date information
GNSSAntennaState_TypeDef	GNSS antenna status;
GNSSAntennaState	GNSS_AntennaExternal: external antenna; GNSS_AntennaInternal: internal antenna;
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function should be called after Device_Open.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
//If SA series devices are used, the data transfer interface is set to USB
//BootProfile.PhysicalInterface =USB;
BootProfile.PhysicalInterface =ETH;
//if NX series  devices are used, the data transfer  interface is set to ETH
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
double ABSTimestamp = 0;
int16_t hour = 0;
 int16_t minute = 0;
int16_t second = 0;
int16_t Year = 0;
int16_t month = 0;
int16_t  day = 0;
Status = Device_AnysisGNSSTime (ABSTimestamp,&hour,&minute, &second, &Year, &month, &day);
```

9.14 Device_GetGNSSA

int Device_GetGNSSAltitude(void Device, int16_t* Altitude)**
Description

obtain the altitude information of the location of the GNSS antenna status in a non-real-time manner (Hardware opt.) that is, without interrupting 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.
int16_t* A	Returns the altitude of the location of the GNSS.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function should be called after Device_Open.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
//If SA series devices are used, the data transfer interface is set to USB
//BootProfile.PhysicalInterface =USB;
BootProfile.PhysicalInterface =ETH;
//if NX series  devices are used, the data transfer  interface is set to ETH
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
int16_t Altitude = 0;
Status = Device_GetGNSSAltitude (&Device, &Altitude);
```

9.15 Device_SetDOCXOWorkMode

int Device_SetDOCXOWorkMode(void Device, const DOCXOWorkMode_TypeDef DOCXOWorkMode)**	
Description	
set DOCXO working mode.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
DOCXOWorkMode_TypeDef	Set the DOCXO antenna state
DOCXOWorkMode	DOCXO_LockMode: disciplining mode; DOCXO_HoldMode: Tracking Mode;
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function should be called after Device_Open.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
```

```
BootProfile_TypeDef *BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
//If SA series devices are used, the data transfer interface is set to USB
//BootProfile.PhysicalInterface =USB;
BootProfile.PhysicalInterface =ETH;
//if NX series  devices are used, the data transfer  interface is set to ETH
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DOCXOWorkMode_TypeDef DOCXOWorkMode ;
Status = Device_SetDOCXOWorkMode (&Device, DOCXOWorkMode);
```

9.16 Device_GetDOCXOWorkMode_Realtime

int Device_GetDOCXOWorkMode_Realtime(void Device, const DOCXOWorkMode_TypeDef *DOCXOWorkMode)**	
Description	
Obtain the DOCXO working status in a real-time manner (Hardware opt.), but the real-time mode will occupy the data channel for a short time.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
DOCXOWorkMode_TypeDef	Set the DOCXO antenna status
DOCXOWorkMode	DOCXO_LockMode: disciplining mode; DOCXO_HoldMode: Tracking Mode;
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function should be called after Device_Open.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
//If SA series devices are used, the data transfer interface is set to USB
//BootProfile.PhysicalInterface =USB;
BootProfile.PhysicalInterface =ETH;
//if NX series  devices are used, the data transfer  interface is set to ETH
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DOCXOWorkMode_TypeDef DOCXOWorkMode ;
Status = Device_ GetDOCXOWorkMode_Realtime (&Device, &DOCXOWorkMode);
```

9.17 Device_GetDOCXOWorkMode

int Device_GetDOCXOWorkMode (void Device, const DOCXOWorkMode_TypeDef *DOCXOWorkMode)**	
Description	
Obtain the DOCXO working status in a real-time manner (Hardware opt.) that is, without interrupting 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.
DOCXOWorkMode_TypeDef	Set the DOCXO antenna status
DOCXOWorkMode	DOCXO_LockMode: disciplining mode; DOCXO_HoldMode: Tracking Mode;
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function should be called after Device_Open.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL; **BootProfile_TypeDef *BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **//If SA series devices are used, the data transfer interface is set to USB** **//BootProfile.PhysicalInterface =USB;** **BootProfile.PhysicalInterface =ETH;** **//if NX series devices are used, the data transfer interface is set to ETH** **BootInfo_TypeDef *BootInfo;** **Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);** **DOCXOWorkMode_TypeDef DOCXOWorkMode ;** **Status = Device_ GetDOCXOWorkMode (&Device, &DOCXOWorkMode);**	

9.18 Device_GetGNSSInfo

int Device_GetGNSSInfo(void Device, GNSSInfo_TypeDef* GNSSInfo)**	
Description	
Obtain the GNSS device status in a non-real-time manner (Hardware opt.) that is, without interrupting 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 obtain the information of the device.
GNSSInfo_TypeDef	
float latitude	Latitude of the GNSS
float longitude	Longitude of the GNSS
int16_t altitude	Altitude of the GNSS
uint8_t SatsNum	Sats number of the GNSS
uint8_t GNSS_LockState	Lock state of the GNSS
uint8_t DOCXO_LockState	Lock state of the DOCXO
DOCXOWorkMode_TypeDef DOCXO_WorkMode	Work mode of the DOCXO
GNSSAntennaState_TypeDef GNSSAntennaState	State of the antenna
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function should be called after Device_Open.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
//If SA series devices are used, the data transfer interface is set to USB
//BootProfile.PhysicalInterface =USB;
BootProfile.PhysicalInterface =ETH;
//if NX series devices are used, the data transfer interface is set to ETH
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
GNSSInfo_TypeDef GNSSInfo;
Status = Device_ GetGNSSInfo (&Device, & GNSSInfo);
```

9.19 Device_GetGNSSInfo_Realtime

int Device_GetGNSSInfo_Realtime(void Device, GNSSInfo_TypeDef* GNSSInfo)**	
Description	
Obtain the GNSS device status in a real-time manner (Hardware opt.) that is, but the real-time mode will occupy the data channel for a short time.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
GNSSInfo_TypeDef* GNSSInfo	GNSS obtain the information of the device.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.

Invocation constraints	This function should be called after Device Open.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
//If SA series devices are used, the data transfer interface is set to USB
//BootProfile.PhysicalInterface =USB;
BootProfile.PhysicalInterface =ETH;
//if NX series devices are used, the data transfer interface is set to ETH
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
GNSSInfo_TypeDef GNSSInfo;
Status = Device_ GetGNSSInfo_Realtime (&Device, & GNSSInfo);
```

10 SWP Mode (main functions)

10.1 SWP_EZProfileDeInit

int SWP_EZProfileDeInit(void Device, SWP_EZProfile_TypeDef* UserProfile_O)**	
Function description	
The function initializes the configuration in basic SWP mode configuration parameter collection. SWP_EZProfile_TypeDef defines all parameters in SWP mode such as frequency, reference level, resolution bandwidth, etc.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
SWP_Profile_TypeDef *UserProfile_O	Pointer to the default profile.
SWP_EZProfile_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 Manual, the parameter is absolute time. When specified as *N, this parameter is 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.
RBWMode_TypeDef RBWMode	RBW update mode: RBW_Manual: Input manually; RBW_Auto: set automatically according to Span; RBW_OneThousandthSpan: mandatory RBW = 0.001 * SPAN; RBW_OnePercentSpan: mandatory RBW = 0.01*SPAN。
VBWMode_TypeDef VBWMode	VBW update mode:

	VBW_Manual: the VBW is set manually; VBW_EqualToRBW: mandatory VBW = RBW; VBW_TenPercentRBW: mandatory VBW = 0.1*RBW; VBW_OnePercentRBW: mandatory VBW = 0.01*RBW; VBW_TenTimesRBW: mandatory VBW = 10*RBW, fully bypass VBW filter.
SweepTimeMode_TypeDef **SweepTimeMode**	Sweep time mode: SWTMode_minSWT: Sweep with minimum sweep time; SWTMode_minSWTx2: Sweep with approximately 2 times of the minimum sweep time. SWTMode_minSWTx4: Sweep with approximately 4 times of the minimum sweep time. SWTMode_minSWTx10: Sweep with approximately 10 times of the minimum sweep time. SWTMode_minSWTx20: Sweep with approximately 20 times of the minimum sweep time. SWTMode_minSWTx50: Sweep with approximately 50 times of the minimum sweep time. SWTMode_minSWTxN: Sweep with approximately N times of the minimum sweep time, N=SweepTimeMultiple. SWTMode_Manual: Sweep at approximately the specified sweep time, which is equal to SweepTime. SWTMode_minSMPxN: A single frequency point is sampled with approximately N times the shortest sampling time, N=SampleTimeMultiple
Detector_TypeDef **Detector**	Detector: Detector_Sample: No inter-frame detection; Detector_PosPeak: MaxHold Frame detection is carried between power spectrum of each frequency point; Detector_Average: Average Frame detection is carried between power spectrum of each frequency point; Detector_NegPeak: MinHold Frame detection is carried between power spectrum of each frequency point; Detector_MaxPower: Each frequency point is sampled for a long time before FFT and frame spectrum with highest power is selected for FFT to capture instantaneous signals such as pulse. (SWP mode only); Detector_RawFrames: Multiple sampling, multile FFT analyses for each frequency, and frame-by-frame output power spectrum (SWP

	mode only); Detector_RMS: RMS Frame detection is carried between power spectrum of each frequency point; Detector_AutoPeak: Aoto Peak Frame detection is carried between power spectrum of each frequency point.
TraceDetectMode_TypeDef TraceDetectMode	Trace detection mode (frequency axis): TraceDetectMode_Auto: automatically selection of a detection; TraceDetectMode_Manual: specification of a detection.
TraceDetector_TypeDef **TraceDetector**	Trace detector: 1) TraceDetector_AutoSample: auto sample detection; 2)TraceDetector_Sample: sample detection; 3)TraceDetector_PosPeak: positive peak detection; 4)TraceDetector_NegPeak: negative peak detection; 5)TraceDetector_RMS: RMS detection; 6)TraceDetector_Bypass: no detection; 7)TraceDetector_AutoPeak: auto peak detection.
uint32_t **TracePoints**	Trace points
RxPort_TypeDef **RxPort**	RF in port: 1)ExternalPort: External port, 2)InternalPort: Internal port, 3)ANT_Port, TR_Port, SWR_Port, INT_Port: only for TRx series.
SpurRejection_TypeDef **SpurRejection**	Specify the strategy for spurious rejection: 1) Bypass: no additional rejection is provided; 2) Standard: a standard rejection is provided; 3) Enhanced: an enhanced rejection is provided; 4) The higher the spurious suppression level, the slower the sweep 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.
SWP_TriggerSource_TypeDef **TriggerSource**	Sweep trigger source for RF receivers: 1) InternalFreeRun: Internal trigger free run; 2) ExternalPerHop: External trigger, each trigger jumps a frequency point; 3) ExternalPerSweep: External triggers, with each trigger refreshing a trace; 4) ExternalPerProfile: External triggers, with each trigger applying a new configuration.
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.
PreamplifierState_TypeDef **Preamplifier**	Set the state of the preamplifier: 1) AutoOn: Automatically enables the preamplifier; 2) ForcedOff: Force to keep the preamplifier off.
SWP_TraceType_TypeDef **TraceType**	Specify the trace type: 1) ClearWrite: Output the normal trace; 2) MaxHold: Output the trace maintained by the maximum value; 3) MinHold: Output the trace maintained by the minimum value; 4) ClearWriteWithIQ: Output time domain data and frequency domain data of current frequency point at the same time.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after Device_Open.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL; **BootProfile_TypeDef BootProfile; BootInfo_TypeDef BootInfo;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);** **SWP_EZProfile_TypeDef ProfileIn;** **Status = SWP_EZProfileDeInit(&Device, &ProfileIn);**	

10.2 SWP_EZConfiguration

int SWP_EZConfiguration(void Device, const SWP_EZProfile_TypeDef* ProfileIn,**

SWP_EZProfile_TypeDef* ProfileOut, SWP_TraceInfo_TypeDef* TraceInfo)	
Description	
configure the spectrometer device to basic SWPMode and related parameters Parameters such as frequency, reference level, resolution bandwidth and other parameters in SWP mode are uniformly encapsulated in the SWP_EZProfile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
SWP_EZProfile_TypeDef *ProfileIn	Configuration set input.
SWP_EZProfile_TypeDef *ProfileOut	Configuration set output.
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 index of the closest point to the SWPProfile.StartFreq_Hz in the trace (Freq_Hz[]).
uint32_t UserStopIndex	The index of the closest point to the SWPProfile.StopFreq_Hz in the trace (Freq_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 of a single hop.
int TraceDetectRatio	Detection ratio of video detection.
int DecimateFactor	Multiple of time domain data extraction.
float FrameTimeMultiple	Frame analysis time multiple: The analysis time of the device at a single frequency = default analysis time (set by the system) * frame time multiple. Increasing the frame time multiple will increase the device's minimum scan time, but is not strictly linear.
double FrameTime	The frame time is the totoal analysis time at a single frequency point.
double EstimateMinSweepTime	The minimum scanning time that can be set under the current configuration (unit: second, the result is mainly affected by Span, RBW, VBW, frame scanning time and other factors).
DataFormat_TypeDef DataFormat	Time domain data format.
Uint64_t SamplePoints	Time domain data sampling length.
uint32_t GainParameter	Gain related parameters, including Space (31 to 24Bit), PreAmplifierState (23 to 16Bit), StartRFBand (15 to 8Bit), StopRFBand (7

	to 0 Bit).
DSPPlatform_Typedef **DSPPlatform**	DSP calculating platform under current configuration: CPU_DSP: CPU; FPGA_DSP: FPGA.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after SWP_EZProfileDeInit.
Example	

```
int Status = -1;
intDeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_EZProfileDeInit(&Device, &ProfileIn);
Status = SWP_EZConfiguration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
Status = Device_Close(&Device);
```

10.3 SWP_ProfileDeInit

int SWP_ProfileDeInit(void Device, SWP_Profile_TypeDef* UserProfile_O)**	
Function description	
The function initializes the configuration in SWP mode configuration parameter collection. SWP_Profile_TypeDef defines all parameters in SWP mode such as frequency, reference level, resolution bandwidth, etc.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
SWP_Profile_TypeDef ***UserProfile_O**	Pointer to the default profile.
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	The total time of the full sweep in seconds. RBW, VBW, and FrameTimeMultiple are first met. When the input sweep time is less than the minimum sweep time of the device, the system will still sweep at the minimum time.
double TraceBinSize_Hz	The frequency interval between adjacent frequency points of the trace in Hz. Only work when TracePointsStrategy = BinSizeAssigned.
Window_TypeDef Window	Specify the window function used for FFT analysis: 1) FlatTop: good amplitude accuracy; 2) Blackman_ Nuttall: good frequency resolution.
RBWMode_TypeDef RBWMode	RBW update mode: RBW_Manual: Input manually; RBW_Auto: set automatically according to Span; RBW_OneThousandthSpan：mandatory RBW = 0.001 * SPAN; RBW_OnePercentSpan：mandatory RBW = 0.01*SPAN。
VBWMode_TypeDef VBWMode	VBW update mode: VBW_Manual: the VBW is set manually; VBW_EqualToRBW: mandatory VBW = RBW; VBW_TenPercentRBW: mandatory VBW = 0.1*RBW; VBW_OnePercentRBW: mandatory VBW = 0.01*RBW; VBW_TenTimesRBW: mandatory VBW = 10*RBW, fully bypass VBW filter.
SweepTimeMode_TypeDef SweepTimeMode	Sweep time mode: SWTMode_minSWT: Sweep with minimum sweep time; SWTMode_minSWTx2: Sweep with approximately 2 times of the minimum sweep time. SWTMode_minSWTx4: Sweep with approximately 4 times of the minimum sweep time. SWTMode_minSWTx10: Sweep with approximately 10 times of the minimum sweep time. SWTMode_minSWTx20: Sweep with approximately 20 times of the minimum sweep time. SWTMode_minSWTx50: Sweep with approximately 50 times of the minimum sweep time. SWTMode_minSWTxN: Sweep with approximately N times of the minimum sweep time，N=SweepTimeMultiple.

	SWTMode_Manual: Sweep at approximately the specified sweep time, which is equal to SweepTime. SWTMode_minSMPxN: A single frequency point is sampled with approximately N times the shortest sampling time, N=SampleTimeMultiple
Detector_TypeDef **Detector**	Detector: Detector_Sample: No inter-frame detection; Detector_PosPeak: MaxHold Frame detection is carried between power spectrum of each frequency point; Detector_Average: Average Frame detection is carried between power spectrum of each frequency point; Detector_NegPeak: MinHold Frame detection is carried between power spectrum of each frequency point; Detector_MaxPower: Each frequency point is sampled for a long time before FFT and frame spectrum with highest power is selected for FFT to capture instantaneous signals such as pulse. (SWP mode only); Detector_RawFrames: Multiple sampling, multile FFT analyses for each frequency, and frame-by-frame output power spectrum (SWP mode only); Detector_RMS: RMS Frame detection is carried between power spectrum of each frequency point; Detector_AutoPeak: Aoto Peak Frame detection is carried between power spectrum of each frequency point.
TraceFormat_TypeDef **TraceFormat**	Trace format: 1)TraceFormat_Standard: equal frequency interval; 2)TraceFormat_PrecisFrq: precise frequency.
TraceDetectMode_TypeDef **TraceDetectMode**	Trace detection mode (frequency axis): TraceDetectMode_Auto: automatically selection of a detection; TraceDetectMode_Manual: specification of a detection.
TraceDetector_TypeDef **TraceDetector**	Trace detector: 1) TraceDetector_AutoSample: auto sample detection; 2)TraceDetector_Sample: sample detection; 3)TraceDetector_PosPeak: positive peak detection; 4)TraceDetector_NegPeak: negative peak detection; 5)TraceDetector_RMS: RMS detection; 6)TraceDetector_Bypass: no detection; 7)TraceDetector_AutoPeak: auto peak detection.

uint32_t **TracePoints**	Trace point
TracePointsStrategy_TypeDef **TracePointsStrategy**	Trace point mapping strategy: 1)SweepSpeedPreferred: Give priority to the fastest sweep speed, and then try to get as close as possible to the target trace points. 2)PointsAccuracyPreferred: the actual trace points are close to the target trace points. 3)BinSizeAssined: trace is generated at the set frequency interval.
TraceAlign_TypeDef **TraceAlign**	Trace alignment: 1)NativeAlign: Natural alignment; 2)AlignToStart: Align to start frequency; 3)AlignToCenter: Align to center frequency.
FFTExcutionStrategy_TypeDef **FFTExcutionStrategy**	Specify the strategy for FFT executing: 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.
RxPort_TypeDef **RxPort**	RF in port: 1)ExternalPort: External port, 2)InternalPort: Internal port, 3)ANT_Port, TR_Port, SWR_Port, INT_Port: only for TRx series.
SpurRejection_TypeDef **SpurRejection**	Specify the strategy for spurious rejection: 1) Bypass: no additional rejection is provided; 2) Standard: a standard rejection is provided; 3) Enhanced: an enhanced rejection is provided; 4) The higher the spurious suppression level, the slower the sweep 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 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.
uint8_t EnableReferenceClockOut	Enable reference clock output.
SystemClockSource_TypeDef **SystemClockSource**	Set the system clock source. The internal system clock is used by default. Please consult our technical support before external source is used: 1) SystemClockSource_Internal: the internal system clock source is used; 2) SystemClockSource_External: the external system clock source is used.
Double ExternalSystemClockFrequency	Set the external system clock frequency in Hz.
SWP_TriggerSource_TypeDef **TriggerSource**	Sweep trigger source for RF receivers: 1) InternalFreeRun: Internal trigger free run; 2) ExternalPerHop: External trigger, each trigger jumps a frequency point; 3) ExternalPerSweep: External triggers, with each trigger refreshing a trace; 4) ExternalPerProfile: External triggers, with each trigger applying a new configuration.
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: No trigger output; 2) PerHop: Output with each completion of frequency hopping; 3) PerSweep: Output with the completion of each scan.; 4) PerProfile: Output with each configuration switch.
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 0-5000, 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	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.
uint8_t EnableDebugMode	unavailable.
uint8_t CalibrationSettings	unavailable.
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.
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**	LO optimization: 1)LOOpt_Auto: LO optimization, auto; 2)LOOpt_Speed: LO optimization, high sweep speed; 3)LOOpt_Spur: LO optimization, low spurious; 4)LOOpt_PhaseNoise: LO optimization, low phase noise.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after Device_Open.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL; **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);
```

10.4 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 SWPMode 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.
SWP_Profile_TypeDef *SWP_ProfileIn	Configuration set input.
SWP_Profile_TypeDef *SWP_ProfileOut	Configuration set output.
SWP_TraceInfo_TypeDef *TraceInfo	The information about the trace under the current configuration.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after SWP_ProfileDeInit.
Example	

```
int Status = -1;
intDeviceNum = 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 ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
Status = Device_Close(&Device);
```

10.5 SWP_AutoSet

int SWP_AutoSet(void Device, SWPApplication_TypeDef Application, const SWP_Profile_TypeDef* ProfileIn, SWP_Profile_TypeDef* ProfileOut, SWP_TraceInfo_TypeDef* TraceInfo, uint8_t ifDoConfig)**	
Description	
This function takes the SWP Mode of the spectrum analyzer and gives the recommended device configuration according to the application goal. The parameters such as frequency, reference level, and resolution bandwidth in SWP mode are encapsulated in the SWP_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
SWPApplication_TypeDef Application	1)SWPPhaseNoiseMeas = 0x00: phase noise measurement; 2)SWPNoiseMeas = 0x01: noise measurement; 3)SWPChannelPowerMeas = 0x02: chanel power measurement.
SWP_Profile_TypeDef *SWP_ProfileIn	Configuration set input.
SWP_Profile_TypeDef *SWP_ProfileOut	Configuration set output.
SWP_TraceInfo_TypeDef *TraceInfo	The information about the trace under the current configuration.
uint8_t ifDoConfig	Returns the result of success or failure
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	When ifDoConfig = 0, it needs to be called before SWP_Configuration; When ifDoConfig =1, the function internally calls the SWP_Configuration function itself, so no additional calls to SWP_Configuration are needed.
Example (ifDoConfig=1)	

```
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 ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
uint8_t ifDoConfig=1;
```

```
SWPApplication_TypeDef Application;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_AutoSet (&Device, Application ,&ProfileIn, &ProfileOut, &TraceInfo, ifDoConfig);
Status = Device_Close(&Device);BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
Status = Device_Close(&Device);
```

10.6 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[]	The frequency array of the measurement data. The array size is equal to the TraceInfo.PartialsweepTracePoints.
float PowerSpec_dBm[]	The power array of the measurement data. The array size is equal to the TraceInfo.PartialsweepTracePoints.
int *HopIndex	Indicate the index of current hop in the whole hopping sequence.
int* FrameIndex	Indicate the index of current frame in all the frames.
MeasAuxInfo_TypeDef *MeasAuxInfo	Auxiliary measurement information.
MeasAuxInfo_TypeDef	
uint32_t MaxIndex	The index of the maximum power point in the packet.
float MaxPower_dBm	The maximum power in the packet.
int16_t Temperature	The temperature of the device. The LSB is 0.01 celsius degree.
uint16_t RFState	The state of the device RF part.
uint16_t BBState	The state of the device baseband.
uint16_t GainPattern	The gain pattern of the device.
uint32_t ConvertPattern	Frequency convert pattern of the device.

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.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after SWP_C.
Example	

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.PartialsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.PartialsweepTracePoints];
int HopIndex = 0;
int FrameIndex = 0;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetPartialSweep(&Device, Frequency, PowerSpec_dBm, &HopIndex,
&FrameIndex,MeasAuxInfo);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

10.7 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 returns 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	Consistent with that in SWP_GetPartialSweep.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after SWP_C.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

11 SWP Mode (other functions)

11.1 SWP_GetPartialSweep_PM1

int SWP_GetPartialSweep_PM1(void Device, SWPTrace_TypeDef* PartialTrace)**	
Description	
The polymorphic form of the SWP_GetPartialSweep, adjusting the form of the returned data on the basis of the original function to strengthen the encapsulation of the data.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
SWPTrace_TypeDef *PartialTrace	Return the top-level structure that contains configuration, return information, and staging data.
SWPTrace_TypeDef	
double* Freq_Hz	The address of the Frequency data that is temporarily stored in the Device pointer.
float* PowerSpec_dBm	The address of the PowerSpec_dBm data that is temporarily stored in the Device pointer.
int HopIndex	The frequency hopping frequency index of the data is used to stitch the spectrum.
int FrameIndex	Frame index of data for applications such as quasi-positive peak detection.
void* AlternIQStream	Address of time domain data (interleaved IQ form).
float ScaletoV	The coefficient from the time domain data to the absolute value of voltage (V).
MeasAuxInfo_TypeDef MeasAuxInfo	Return auxiliary information about the measurement data, as detailed in the SWP_GetPartialSweep description.
SWP_Profile_TypeDef SWP_Profile	Configuration information for the data.
SWP_TraceInfo_TypeDef SWP_TraceInfo	Trace information for the data.
DeviceInfo_TypeDef DeviceInfo	Device information for the data.
DeviceState_TypeDef DeviceState	The device state corresponding to the data (defined in htra_api or in the previous Device_QueryDeviceState function).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after SWP_Configuration.
Example	
int Status = -1; int DeviceNum = 0; void *Device = NULL;	

```
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
SWPTrace_TypeDef PartialTrace;
double *Frequency = new double[TraceInfo.PartialsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.PartialsweepTracePoints];
PartialTrace.Freq_Hz = Frequency;
PartialTrace.PowerSpec_dBm = PowerSpec_dBm;
Status = SWP_GetPartialSweep_PM1(&Device,&PartialTrace);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

11.2 SWP_GetPartialUpdatedFullSweep

int SWP_GetPartialUpdatedFullSweep(void Device, double Freq_Hz[], float PowerSpec_dBm[], int* HopIndex, int* FrameIndex, MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
Capture spectral traces that are full scan in length and fragmented data are refreshed with each call. Suitable for applications such as graphical display of the spectrum.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
double Freq_Hz[]	An array of frequencies returned. The array size is equal to the TracePoints via TraceInfo.FullSweepTracePoints.
float PowerSpec_dBm[]	An array of amplitude returned. The array size is equal to the TracePoints via TraceInfo.FullSweepTracePoints.
int* HopIndex	Returns the frequency hopping frequency sequence number of the data.
int* FrameIndex	Return the frame sequence number of the data.
MeasAuxInfo_TypeDef* MeasAuxInfo	Return auxiliary information about the measurement data, as detailed in the SWP_GetPartialSweep description.

Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after SWP_ProfileDeInit.

Example

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float *PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
int HopIndex = 0;
int FrameIndex = 0;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetPartialUpdatedFullSweep(&Device, Frequency, PowerSpec_dBm,
&HopIndex,&FrameIndex,&MeasAuxInfo);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

11.3 SWP_ResetTraceHold

void SWP_ResetTraceHold(void Device)**	
Description	
TraceType is MaxHold or MinHold, the retained trace data is reset.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
Return value	No return value.
Example	

```
int Status = -1;
int DeviceNum = 0;
```

```
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.PartialsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.PartialsweepTracePoints];
int HopIndex = 0;
int FrameIndex = 0;
MeasAuxInfo_TypeDef *MeasAuxInfo;
Status = SWP_GetPartialSweep(&Device, Frequency, PowerSpec_dBm, &HopIndex,
&FrameIndex,MeasAuxInfo);
SWP_ResetTraceHold(&Device);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

12 IQS Mode (main functions)

IQS mode is IQ Stream Mode, in which users can continuously record time domain data streams.

12.1 IQS_EZProfileDeInit

int IQS_EZProfileDeInit(void Device, IQS_EZProfile_TypeDef* UserProfile_O)**	
Description	
This function initializes the configuration profile of the IQS mode. IQS_EZProfile_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 default profile.
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.
RxPort_TypeDef RxPort	RF in port: 1)ExternalPort: External port, 2)InternalPort: Internal port, 3)ANT_Port, TR_Port, SWR_Port, INT_Port: only for TRx series.
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) FreeRun: automatically triggered once after every IQS_Configuration calling; 2) External: triggered by the external trigger; 3) Bus: triggered by function IQS_BusTriggerStart; 4) Level: The input is continuously monitored and an acquisition will be triggered if the level threshold specified by the TriggerLevel_dBm is reached; 5) Timer: triggered by timer within system; 6) TxSweep: triggered with frequency hopping of the build-in RF generator. 7) MultiDevSyncByExt: synchronized with multiple devices by external

	trigger; 8) MultiDevSyncByGNSS1PPS: synchronized with multiple devices by the 1PPS of the insystem GNSS; 9) SpectrumMask: the specturm of the input is continuously monitored and an acquisition will be triggered if the given spectrum mask is satisfied. SpectrumMask Trigger is only available in RTA mode; 10) GNSS1PPS: triggered by the 1PPS of GNSS within the system.
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: once the device is triggered, the device will keep acquiring data until a stop signal is received. The stop signal may be an external trigger edge or a calling of the function IQS_BusTriggerStop.
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 **TriggerTimer_Period**	Specify the period of timer trigger in second. It is effective when the TriggerSource is set to Timer.
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) Complexfloat: IQ, single channel data 32-bit float format tpye (IQS mode is not available, only by DDC function output data to write back the enumeration variable).
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.
ReferenceClockSource_TypeDef	Specify the referenc clock：

ReferenceClockSource	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.
double NativeIQSampleRate_SPS	For devices with variable sampling rate capability, user can specify the native IQ sampling rate of the device.
uint8_t EnableIFAGC	Enable or disable the IF auto gain control: 1) 0: disabled; 2) 1: enabled; AGC is only available for specific devices.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to 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, DevNum, &BootProfile, &BootInfo);
IQS_EZProfile_TypeDef ProfileIn;
Status = IQS_EZProfileDeInit(&Device, &ProfileIn);
```

12.2 IQS_EZConfiguration

int IQS_EZConfiguration(void Device, const IQS_EZProfile_TypeDef* ProfileIn, IQS_Profile_TypeDef* ProfileOut, IQS_StreamInfo_TypeDef* StreamInfo)**
Description
This function configures the spectrum analyzer into the IQ time domain data stream mode (IQS). In the IQS mode,

the local oscillator frequency is remained unchange and the RF signal with a certain bandwidth centered at the LO frequency is received. At the same time, it can realize the continuous time domain recording function when the decimation multiple is >2 (the PC needs to be in a certain configuration).	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
IQS_EZProfile_TypeDef *ProfileIn	Configuration profile for IQS mode.
IQS_EZProfile_TypeDef *ProfileOut	Feedback profile from the system.
IQS_StreamInfo_TypeDef *StreamInfo	The information about the data stream under the current configuration.
IQS_StreamInfo_TypeDef	
double Bandwidth	Signal bandwidth.
double IQSampleRate	Single-channel sampling rate of IQ corresponding to the current configuration (unit: Sample/second).
uint64_t PacketCount	The total number of data packets corresponding to the current configuration takes effect only in FixedPoints mode.
uint64_t StreamSamples	In Fixedpoints mode, it represents the total number of sampling points corresponding to the current configuration. In Adaptive mode, the value has no physical significance and is 0.
uint64_t StreamDataSize	In Fixedpoints mode, it indicates the total number of bytes of samples corresponding to the current configuration. In Adaptive mode, the value has no physical significance and is 0
uint32_t PacketSamples	Sampling points in packets obtained by each IQS_GetIQStream invocation Sample points contained in each packet.
uint32_t PacketDataSize	The number of valid data bytes to be obtained per call to IQS_GetIQStream.
uint32_t GainParameter	Gain dependent parameters, including Space(31~24Bit), PreAmplifierState(23~16Bit), StartRFBand(15~8Bit), StopRFBand(7~0Bit).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after IQS_EZProfileDeInit.
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);
IQS_EZProfile_TypeDef ProfileIn;
IQS_EZProfile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_EZProfileDeInit(&Device, &ProfileIn);
Status = IQS_EZConfiguration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
```

12.3 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 default profile.
IQS_Profile_TypeDef	
double CenterFreq_Hz	Center frequency in Hz.
double RefLevel_dBm	Reference level in dBm.
uint32_t DecimateFactor	Decimate Factor of time domain.
RxPort_TypeDef RxPort	RF in port: 1)ExternalPort: External port, 2)InternalPort: Internal port, 3)ANT_Port, TR_Port, SWR_Port, INT_Port: only for TRx series.
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) FreeRun: automatically triggered once after every IQS_Configuration calling; 2) External: triggered by the external trigger; 3) Bus: triggered by function IQS_BusTriggerStart; 4) Level: The input is continuously monitored and an acquisition will be triggered if the level threshold specified by the TriggerLevel_dBm is reached; 5) Timer: triggered by insystem timer;

	6) TxSweep: triggered with frequency hopping of the build-in RF generator. 7) MultiDevSyncByExt: synchronized with multiple devices by external trigger; 8) MultiDevSyncByGNSS1PPS: synchronized with multiple devices by the 1PPS of the insystem GNSS; 9) SpectrumMask: the specturm of the input is continuously monitored and an acquisition will be triggered if the given spectrum mask is satisfied. SpectrumMask Trigger is only available in RTA mode; 10) GNSS1PPS: triggered by the 1PPS of the insystem GNSS.
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: once the device is triggered, the device will keep acquiring data until a stop signal is received. The stop signal may be an external trigger edge or a calling of the function IQS_BusTriggerStop.
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.
TriggerOutMode_TypeDef **TriggerOutMode**	Set the trigger output mode: 1) None: trigger out is disabled.
TriggerOutPulsePolarity_TypeDef **TriggerOutPulsePolarity**	Specify the pulse polarity of the output trigger 1) Positive: positive pulse is used; 2) Negative: negative pulse is used.
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**	Once triggered, the trigger action will be executed after this delay in second.
double **PreTriggerTime**	Pre-trigger time, s. Specify the pre-trigger time in second. Once triggered, pre-triggered data with a length of PreTriggerTime will be attached to the trigger data.
TriggerTimerSync_TypeDef	Set the synchronization of the trigger timer:

TriggerTimerSync	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	Enable or disable the retrigger action: Once enabled, system will generate subsequence triggers automatically after the initial trigger. Retrigger is only available when TriggerMode is set to 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) Complexfloat: IQ, single channel data 32-bit float format tpye (IQS mode is not available, only by DDC function output data to write back the enumeration variable).
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.
uint8_t **EnableDebugMode**	unavailable.
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.
SystemClockSource_TypeDef **SystemClockSource**	Set the system clock source. The internal system clock is used by default. Please consult our technical support before external source is used: 1) SystemClockSource_Internal: the internal system clock source is used; 2) SystemClockSource_External: the external system clock source is used.
double **ExternalSystemClockFrequency**	Set the external system clock frequency in Hz.
double **NativeIQSampleRate_SPS**	For devices with variable sampling rate capability, user can specify the native IQ sampling rate of the device.
uint8_t **EnableIFAGC**	Enable or disable the IF auto gain control: 1) 0: disabled;

	2) 1: enabled; AGC is only available for specific devices.
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.
QDCMode_TypeDef **QDCMode**	Set the operating mode of the QDC (quadrature demodulation corrector): 1) QDCOff: the QDC is off; 2) QDCManualMode: QDC is on and manual calibrate is needed; 3) QDCAutoMode: QDC is on and auto coefficients are used.
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 bias of I channel. DCCIOffset is only effective when the DCCancelerMode is set to DCCManualOffsetMode.
int8_t DCCQOffset	Set the DC bias of Q channel. DCCIOffset is only effective when the DCCancelerMode is set to DCCManualOffsetMode.
LOOptimization_TypeDef **LOOptimization**	LO optimization: 1)LOOpt_Auto: LO optimization, auto; 2)LOOpt_Speed: LO optimization, high sweep speed;

	3)LOOpt_Spur: LO optimization, low spurious; 4)LOOpt_PhaseNoise: LO optimization, low phase noise.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to 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, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
```

12.4 IQS_Configuration

int IQS_Configuration(void Device, const IQS_Profile_TypeDef* ProfileIn, IQS_Profile_TypeDef* ProfileOut, IQS_StreamInfo_TypeDef* StreamInfo)**	
Description	
This function configures the spectrometer device into the IQ time domain data stream mode (IQS). In the IQS mode, the local oscillator signal is fixed and the RF signal with a certain bandwidth centered at the local oscillator frequency is received. At the same time, it can realize the continuous time domain recording function when the decimation multiple is >2 (the PC needs to be in a certain configuration).	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
IQS_Profile_TypeDef *IQS_ProfileIn	Configuration profile for IQS mode.
IQS_Profile_TypeDef *IQS_ProfileOut	Feedback profile from the system.
IQS_StreamInfo_TypeDef *StreamInfo	The information about the data stream under the current configuration.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after IQS_ProfileDelnit.
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);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
```

12.5 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.
Invocation constraints	This function needs to be called after IQS_GetIQStream.
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);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef TraceInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
Status = IQS_BusTriggerStart(&Device);
```

12.6 IQS_BusTriggerStop

int IQS_BusTriggerStop(void Device)**

Description	
This function terminates the current bus trigger. When TriggerMode = FixedPoints is configured, the bus trigger terminates itself when it is initiated by the IQS_BusTriggerStart function and reaches the specified trigger length without calling the function.	
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.
Invocation constraints	This function needs to be called after IQS_GetIQStream.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef TraceInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
Status = IQS_BusTriggerStop(&Device);
```

12.7 IQS_GetIQStream

IQS_GetIQStream(void Device, void** AlternIQStream, float* ScaleToV, IQS_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
Get the IQ stream in IQS mode.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
void AlternIQStream**	Pointer to the interleaved IQ data. Data packet is fixed to 64968 bytes. If the data type is int8, both I and Q data are 32484 points and every IQ pair occupys 2 bytes. If the data type is int16, both I and Q data are 16242 points and every IQ pair occupys 4 bytes. If the data type is

	int32, both I and Q data are 8121 points and every IQ pair occupys 8 bytes. The IQ data is organized as IQIQIQ….
Float* ScaleToV	Coefficient for IQ data to the absolute voltage in volt.
IQS_TriggerInfo_TypeDef *TriggerInfo	Trigger information.
MeasAuxInfo_TypeDef *MeasAuxInfo	Auxiliary measurement information.
IQS_TriggerInfo_TypeDef	
uint64_t SysTimerCountOfFirstDataPoint	The corresponding system timer count value for the first data point in the packet.
uint16_t InPacketTriggeredDataSize	The size of triggered data in the packet.
uint16_t InPacketTriggerEdges	The count of trigger edges in the packet.
uint32_t StartDataIndexOfTriggerEdges[25]	StartIndexs for the trigger edges.
uint64_t SysTimerCountOfEdges[25]	The corresponding system timer count value for every trigger edges.
int8_t EdgeType[25]	The polarity of trigger edges.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after IQS_Configuration.
Example	

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

13 IQS Mode (other functions)

13.1 IQS_MultiDevice_WaitExternalSync

int IQS_MultiDevice_WaitExternalSync(void Device, const IQS_Profile_TypeDef* ProfileIn)**	
Description	
Call this function and wait for more synchronous trigger signals.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
const IQS_Profile_TypeDef *ProfileIn	IQS configures the structure pointer as an input variable.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after IQS_GetIQStream, and TriggerSource select MultiDevSyncByExt or MultiDevSyncByGNSS1PPS.
Example	

```
int Status = -1; int DeviceNum0 = 0; int DeviceNum1 = 0;
void *Device0 = NULL; void *Device1 = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device0, DevNum0, &BootProfile, &BootInfo);
Status = Device_Open(&Device1, DevNum1, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn0, ProfileIn1;
IQS_Profile_TypeDef ProfileOut0, ProfileOut1;
IQS_TraceInfo_TypeDef StreamInfo0, StreamInfo1;
Status = IQS_ProfileDeInit(&Device0, &ProfileIn0);
Status = IQS_ProfileDeInit(&Device1, &ProfileIn1);
ProfileIn0.TriggerSource = MultiDevSyncByExt;
ProfileIn1.TriggerSource = MultiDevSyncByExt;
Status = IQS_Configuration(&Device0, &ProfileIn0, &ProfileOut0, &StreamInfo0);
Status = IQS_Configuration(&Device1, &ProfileIn1, &ProfileOut1, &StreamInfo1);
Status = IQS_MultiDevice_Run(&Device0);
Status = IQS_MultiDevice_Run(&Device1);
```

13.2 IQS_MultiDevice_Run

int IQS_MultiDevice_Run(void **Device)	
Description	
Call this function to enable simultaneous operation of multiple machines.	
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.
Invocation constraints	This function needs to be called after IQS_MultiDevice_WaitExternalSync.
Example	

```
int Status = -1; int DeviceNum0 = 0; int DeviceNum1 = 0;
void *Device0 = NULL; void *Device1 = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device0, DevNum0, &BootProfile, &BootInfo);
Status = Device_Open(&Device1, DevNum1, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn0, ProfileIn1;
IQS_Profile_TypeDef ProfileOut0, ProfileOut1;
IQS_TraceInfo_TypeDef StreamInfo0, StreamInfo1;
Status = IQS_ProfileDeInit(&Device0, &ProfileIn0);
Status = IQS_ProfileDeInit(&Device1, &ProfileIn1);
ProfileIn0.TriggerSource = MultiDevSyncByExt;
ProfileIn1.TriggerSource = MultiDevSyncByExt;
Status = IQS_Configuration(&Device0, &ProfileIn0, &ProfileOut0, &StreamInfo0);
Status = IQS_Configuration(&Device1, &ProfileIn1, &ProfileOut1, &StreamInfo1);
Status = IQS_MultiDevice_Run(&Device0);
Status = IQS_MultiDevice_Run(&Device1);
```

13.3 IQS_SyncTimer

int IQS_SyncTimer(void Device)**	
Description	
Call this function to initiate a timer-outer trigger single synchronization.	
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.
Invocation constraints	This function needs to be called after IQS_Configuration, and TriggerSource selects Timer.
Example	

```
int Status = -1; int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.TriggerSource = Timer;
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_SyncTimer(&Device);
```

13.4 IQS_GetIQStream_PM1

int IQS_GetIQStream_PM1(void **Device, IQStream_TypeDef* IQStream)	
Description	
Obtain the time domain data in IQS mode, and the time domain data format is int8_t, int16_t, and int32_t, and you can select the data format according to your needs.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
IQStream_TypeDef *IQStream	IQ data stream, including IQ data and related configuration information.
IQStream_TypeDef	
void *AlternIQStream	Pointer to the interleaved IQ data. Data packet is fixed to 64968 bytes. If the data type is int8, both I and Q data are 32484 points and every IQ pair occupys 2 bytes. If the data type is int16, both I and Q data are 16242 points and every IQ pair occupys 4 bytes. If the data type is int32, both I and Q data are 8121 points and every IQ pair occupys 8

	bytes. The IQ data is organized as IQIQIQ….
float IQS_ScaleToV	The coefficient from the time domain data to the absolute value of voltage (V).
float MaxPower_dBm	The maximum power in the data.
uint32_t MaxIndex	The index of the maximum power in the data.
IQS_Profile_TypeDef IQS_Profile	Configuration information for the data.
IQS_StreamInfo_TypeDef StreamInfo	Format information for the data.
IQS_TriggerInfo_TypeDef TriggerInfo	Trigger information for the data.
DeviceInfo_TypeDef DeviceInfo	Device information for the data.
DeviceState_TypeDef DeviceState	Device status information for data.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.

Example

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device,&ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
IQStream_TypeDef IQStream;
Status = IQS_BusTriggerStart(&Device);
Status = IQS_GetIQStream_PM1(&Device, &IQStream);
```

13.5 IQS_GetIQStream_Data

int IQS_GetIQStream_Data(void Device, int16_t IQ_data[])**	
Description	
Call this function to get the IQ time domain data with a data type of int16_t.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
int16_t IQ_data[]	An array of IQ data that receives 16 bits of single data.

Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.

Example

```
int Status = -1;int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device,&ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_BusTriggerStart(&Device);
int16_t *IQ_data = new int16_t[StreamInfo.PacketSamples];
Status = IQS_GetIQStream_Data(&Device,IQ_data);
delete[] IQ_data;
```

14 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.

14.1 DET_EZProfileDeInit

int DET_EZProfileDeInit(void Device, DET_EZProfile_TypeDef* UserProfile_O)**	
Description	
This function initializes the configuration profile of the DET mode. DET_EZProfile_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 default profile.
DET_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.
RxPort_TypeDef RxPort	RF in port: 1)ExternalPort: External port, 2)InternalPort: Internal port, 3)ANT_Port, TR_Port, SWR_Port, INT_Port: only for TRx series.
uint32_t BusTimeout_ms	Specify the input port of the receiver： 1) ExternalPort: external RF port is linked to the receiver; 2) InternalPort: the output of bult-in signal generator will be connected to the receiver through internal path and the external RF port is isolated.
DET_TriggerSource_TypeDef TriggerSource	Consistent with their defintions in the IQS mode.
TriggerEdge_TypeDef TriggerEdge	
TriggerMode_TypeDef TriggerMode	
uint64_t TriggerLength	
double TriggerLevel_dBm	
double TriggerTimer_Period	

Detector_TypeDef **Detector**	1)Detector_Sample: No inter-frame detection; 2)Detector_PosPeak: MaxHold Frame detection is carried between power spectrum of each frequency point; 3)Detector_Average: Average Frame detection is carried between power spectrum of each frequency point; 4)Detector_NegPeak: MinHold Frame detection is carried between power spectrum of each frequency point; 5)Detector_MaxPower: Each frequency point is sampled for a long time before FFT and frame spectrum with highest power is selected for FFT to capture instantaneous signals such as pulse. (SWP mode only); 6)Detector_RawFrames: Multiple sampling, multiple FFT analyses for each frequency, and frame-by-frame output power spectrum (SWP mode only); 7)Detector_RMS: RMS Frame detection is carried between power spectrum of each frequency point; 8)Detector_AutoPeak: Aoto Peak Frame detection is carried between power spectrum of each frequency point;
uint16_t **DET_TraceDetectRatio**	The DET_TraceDetectRatio indicates how many raw data points are converted to a single output point.
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.
ReferenceClockSource_TypeDef **ReferenceClockSource**	Consistent with their defintions in the IQS mode.
double **ReferenceClockFrequency**	
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to 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, DevNum, &BootProfile, &BootInfo);
DET_EZProfile_TypeDef UserProfile_O;
Status = DET_EZProfileDeInit(&Device, &UserProfile_O);
```

14.2 DET_EZConfiguration

int DET_EZConfiguration(void Device, const DET_EZProfile_TypeDef* ProfileIn, DET_EZProfile_TypeDef* ProfileOut, DET_StreamInfo_TypeDef* StreamInfo);**	
Description	
Set device to the DET mode and configurate it with parameters specified in the DET_EZProfile_TypeDef.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
DET_Profile_TypeDef *DET_ProfileIn	Configuration profile for DET mode.
DET_Profile_TypeDef *DET_ProfileOut	Feedback profile from the system.
DET_StreamInfo_TypeDef *StreamInfo	The information about the data stream under the current configuration.
DET_StreamInfo_TypeDef	
uint64_t PacketCount	Packet count of the power stream. A power stream is generated once the device is triggered.
uint64_t StreamSamples	The total samples in the power stream. When the TriggerMode is set to Adaptive, the StreamSamples is always be 0.
uint64_t StreamDataSize	The data size of the power stream. When the TriggerMode is set to Adaptive, the StreamDataSize is always be 0.
uint64_t PacketSamples	The samples in the packet. It's also the number of samples can be fetched by every calling of function DET_GetPowerStream.
uint64_t PacketDataSize	The data size of the packet. It's also the data size can be fetched by every calling of function DET_GetPowerStream.
double TimeResolution	The time interval in second between the adjacent data points of power stream.
uint32_t GainParameter	Gain related parameters, including Space (31 to 24Bit), PreAmplifierState (23 to 16Bit), StartRFBand (15 to 8Bit), StopRFBand (7 to 0Bit).

Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after DET_EZProfileDeInit.
Example	

```
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);
DET_EZProfile_TypeDef ProfileIn;
DET_EZProfile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_EZProfileDeInit(&Device, &ProfileIn);
Status = DET_EZConfiguration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
```

14.3 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 default profile.
DET_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.
RxPort_TypeDef RxPort	RF in port: 1)ExternalPort: External port, 2)InternalPort: Internal port, 3)ANT_Port, TR_Port, SWR_Port, INT_Port: only for TRx series.
uint32_t BusTimeout_ms	Specify the input port of the receiver： 1) ExternalPort: external RF port is linked to the receiver;

	2) InternalPort: the output of bult-in signal generator will be connected to the receiver through internal path and the external RF port is isolated.
DET_TriggerSource_TypeDef **TriggerSource**	Consistent with their defintions in the IQS mode.
TriggerEdge_TypeDef **TriggerEdge**	
TriggerMode_TypeDef **TriggerMode**	
uint64_t TriggerLength	
TriggerOutMode_TypeDef **TriggerOutMode**	
TriggerOutPulsePolarity_TypeDef **TriggerOutPulsePolarity**	
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	
DETDetector_TypeDef **Detector**	1)Detector_Sample: No inter-frame detection; 2)Detector_PosPeak: MaxHold Frame detection is carried between power spectrum of each frequency point; 3)Detector_Average: Average Frame detection is carried between power spectrum of each frequency point; 4)Detector_NegPeak: MinHold Frame detection is carried between power spectrum of each frequency point; 5)Detector_MaxPower: Each frequency point is sampled for a long time before FFT and frame spectrum with highest power is selected for FFT to capture instantaneous signals such as pulse. (SWP mode only); 6)Detector_RawFrames: Multiple sampling, multiple FFT analyses for each frequency, and frame-by-frame output power spectrum (SWP mode only); 7)Detector_RMS: RMS Frame detection is carried between power

	spectrum of each frequency point; 8)Detector_AutoPeak: Aoto Peak Frame detection is carried between power spectrum of each frequency point;
uint16_t **DET_TraceDetectRatio**	The DET_TraceDetectRatio indicates how many raw data points are converted to a single output point.
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.
uint8_t **EnableDebugMode**	unavailable.
ReferenceClockSource_TypeDef **ReferenceClockSource**	Consistent with their defintions in the IQS mode.
double **ReferenceClockFrequency**	
uint8_t **EnableReferenceClockOut**	
SystemClockSource_TypeDef **SystemClockSource**	
Double **ExternalSystemClockFrequency**	
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**	Consistent with their defintions in the IQS mode.
QDCMode_TypeDef **QDCMode**	
float **QDCIGain**	
float **QDCQGain**	
float **QDCPhaseComp**	
int8_t **DCCIOffset**	
int8_t **DCCQOffset**	
LOOptimization_TypeDef	LO optimization:

LOOptimization	1)LOOpt_Auto: LO optimization, auto; 2)LOOpt_Speed: LO optimization, high sweep speed; 3)LOOpt_Spur: LO optimization, low spurious; 4)LOOpt_PhaseNoise: LO optimization, low phase noise.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to 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, DevNum, &BootProfile, &BootInfo);** **DET_Profile_TypeDef UserProfile_O;** **Status = DET_ProfileDeInit(&Device, &UserProfile_O);**	

14.4 DET_Configuration

int DET_Configuration(void Device, const DET_Profile_TypeDef* ProfileIn, DET_Profile_TypeDef* ProfileOut, DET_StreamInfo_TypeDef* StreamInfo)**	
Description	
Set device to the DET mode and configurate it with parameters specified in the DET_Profile_TypeDef.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
DET_Profile_TypeDef *DET_ProfileIn	Configuration profile for DET mode.
DET_Profile_TypeDef *DET_ProfileOut	Feedback profile from the system.
DET_StreamInfo_TypeDef *StreamInfo	The information about the data stream under the current configuration.
DET_StreamInfo_TypeDef	
uint64_t PacketCount	Packet count of the power stream. A power stream is generated once the device is triggered.
uint64_t StreamSamples	The total samples in the power stream. When the TriggerMode is set to Adaptive, the StreamSamples is always be 0.
uint64_t StreamDataSize	The data size of the power stream. When the TriggerMode is set to

	Adaptive, the StreamDataSize is always be 0.
uint64_t PacketSamples	The samples in the packet. It's also the number of samples can be fetched by every calling of function DET_GetPowerStream.
uint64_t PacketDataSize	The data size of the packet. It's also the data size can be fetched by every calling of function DET_GetPowerStream.
double TimeResolution	The time interval in second between the adjacent data points of power stream.
uint32_t GainParameter	Gain related parameters, including Space (31 to 24Bit), PreAmplifierState (23 to 16Bit), StartRFBand (15 to 8Bit), StopRFBand (7 to 0Bit).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after DET_ProfileDeInit.
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);
DET_Profile_TypeDef ProfileIn;
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
```

14.5 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.

Invocation constraints	This function needs to be called before DET_GetPowerStream.
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);
DET_Profile_TypeDef ProfileIn;
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_BusTriggerStart(&Device);
```

14.6 DET_BusTriggerStop

int DET_BusTriggerStop(void Device)**	
Description	
This function terminates the current bus trigger. When TriggerMode = FixedPoints is configured, the bus trigger terminates on its own when it is initiated by the DET_BusTriggerStart function and reaches the specified trigger length without calling the function.	
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.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn;
```

```
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_BusTriggerStart(&Device);
float NormalizedPowerStream;float ScaleToV;
float NormalizedPowerStream;float ScaleToV;
DET_TriggerInfo_TypeDef  TriggerInfo;
MeasAuxInfo_TypeDef  MeasAuxInfo;
Status = DET_GetPowerStream(&Device, &NormalizedPowerStream, & ScaleToV, &TriggerInfo, &MeasAuxInfo) ;
Status = DET_BusTriggerStop(&Device);
```

14.7 DET_GetPowerStream

int DET_GetPowerStream(void Device, float* NormalizedPowerStream, float* ScaleToV, DET_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
Get the power stream in DET mode.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
float* NormalizedPowerStream	Pointer to the normalized power level ($\sqrt{I^2+Q^2}$).
float *ScaleToV	Coefficient for normalized power level to the absolute voltage in volt.
DET_TriggerInfo_TypeDef *TriggerInfo	Trigger information.
MeasAuxInfo_TypeDef *MeasAuxInfo	Auxiliary measurement information.
DET_TriggerInfo_TypeDef	
uint64_t SysTimerCountOfFirstDataPoint	Consistent with their definitions in the IQS mode.
uint16_t InPacketTriggeredDataSize	
uint16_t InPacketTriggerEdges	
uint32_t StartDataIndexOfTriggerEdges[25]	
uint64_t SysTimerCountOfEdges[25]	
int8_t EdgeType[25]	
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
```

```
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn;
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_BusTriggerStart(&Device);
float NormalizedPowerStream;float ScaleToV;
DET_TriggerInfo_TypeDef  TriggerInfo;
MeasAuxInfo_TypeDef  MeasAuxInfo;
Status = DET_GetPowerStream(&Device, &NormalizedPowerStream, & ScaleToV, &TriggerInfo, &MeasAuxInfo);
```

14.8 DET_SyncTimer

int DET_SyncTimer(void Device)**	
Description	
Synchronize the trigger timer by external trigger or the 1PPS of the insystem GNSS.	
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.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn;
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
```

```
ProfileIn.TriggerSource = Timer;
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_SyncTimer(&Device);
```

15 RTA Mode

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

15.1 RTA_EZProfileDeInit

int RTA_EZProfileDeInit(void Device, RTA_EZProfile_TypeDef* UserProfile_O)**	
Description	
This function initializes the configuration profile of the RTA mode. RTA_EZProfile_TypeDef defines all parameters in RTA mode such as center frequency, reference level, decimate factor, etc.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
RTA_Profile_TypeDef *UserProfile_O	Pointer to the default profile.
RTA_Profile_TypeDef	
double CenterFreq_Hz	Center frequency in Hz.
double RefLevel_dBm	Reference level in dBm.
double RBW_Hz	Resolution bandwidth in Hz.
double VBW_Hz	Video bandwidth in Hz.
RBWMode_TypeDef RBWMode	RBW update mode: RBW_Manual: Input manually; RBW_Auto: set automatically according to Span; RBW_OneThousandthSpan: mandatory RBW = 0.001 * SPAN; RBW_OnePercentSpan: mandatory RBW = 0.01*SPAN。
VBWMode_TypeDef VBWMode	VBW update mode: VBW_Manual: the VBW is set manually; VBW_EqualToRBW: mandatory VBW = RBW; VBW_TenPercentRBW: mandatory VBW = 0.1*RBW; VBW_OnePercentRBW: mandatory VBW = 0.01*RBW; VBW_TenTimesRBW: mandatory VBW = 10*RBW, fully bypass VBW filter.
uint32_t DecimateFactor	The time domain data decimation multiple is set, which determines the bandwidth of RTA
Window_TypeDef Window	Specify the window function used for FFT analysis: 1) FlatTop: good amplitude accuracy; 2) Blackman_ Nuttall: good frequency resolution.
SweepTimeMode_TypeDef SweepTimeMode	Sweep time mode: SWTMode_minSWT: Sweep with minimum sweep time;

	SWTMode_minSWTx2: Sweep with approximately 2 times of the minimum sweep time. SWTMode_minSWTx4: Sweep with approximately 4 times of the minimum sweep time. SWTMode_minSWTx10: Sweep with approximately 10 times of the minimum sweep time. SWTMode_minSWTx20: Sweep with approximately 20 times of the minimum sweep time. SWTMode_minSWTx50: Sweep with approximately 50 times of the minimum sweep time. SWTMode_minSWTxN: Sweep with approximately N times of the minimum sweep time，N=SweepTimeMultiple. SWTMode_Manual: Sweep at approximately the specified sweep time, which is equal to SweepTime. SWTMode_minSMPxN: A single frequency point is sampled with approximately N times the shortest sampling time, N=SampleTimeMultiple
double SweepTime	When the scan time mode is specified as Manual, this parameter is the absolute time; When specified as *N, this parameter is the scan time multiplier
Detector_TypeDef Detector	Detector: Detector_Sample: No inter-frame detection; Detector_PosPeak: MaxHold Frame detection is carried between power spectrum of each frequency point; Detector_Average: Average Frame detection is carried between power spectrum of each frequency point; Detector_NegPeak: MinHold Frame detection is carried between power spectrum of each frequency point; Detector_MaxPower: Each frequency point is sampled for a long time before FFT and frame spectrum with highest power is selected for FFT to capture instantaneous signals such as pulse. (SWP mode only); Detector_RawFrames: Multiple sampling, multile FFT analyses for each frequency, and frame-by-frame output power spectrum (SWP mode only); Detector_RMS: RMS Frame detection is carried between power spectrum of each frequency point; Detector_AutoPeak: Aoto Peak Frame detection is carried between

	power spectrum of each frequency point.
TraceDetectMode_TypeDef TraceDetectMode	Trace detection mode (frequency axis): TraceDetectMode_Auto: automatically selection of a detection; TraceDetectMode_Manual: specification of a detection.
TraceDetector_TypeDef TraceDetector	Trace detector: 1) TraceDetector_AutoSample: auto sample detection; 2)TraceDetector_Sample: sample detection; 3)TraceDetector_PosPeak: positive peak detection; 4)TraceDetector_NegPeak: negative peak detection; 5)TraceDetector_RMS: RMS detection; 6)TraceDetector_Bypass: no detection; 7)TraceDetector_AutoPeak: auto peak detection.
uint32_t **TraceDetectRatio**	The TraceDetectRatio indicates how many raw data points are converted to a single output point.
RxPort_TypeDef **RxPort**	RF in port: 1)ExternalPort: External port, 2)InternalPort: Internal port, 3)ANT_Port, TR_Port, SWR_Port, INT_Port: only for TRx series.
uint32_t **TracePoints**	Trace points
RxPort_TypeDef **RxPort**	RF in port: 1)ExternalPort: External port, 2)InternalPort: Internal port, 3)ANT_Port, TR_Port, SWR_Port, INT_Port: only for TRx series.
uint32_t **BusTimeout_ms**	Sets the data transfer timeout (ms) and returns an error if the device fails to retrieve the data within this time after calling the data fetch function.
RTA_TriggerSource_TypeDef **TriggerSource**	Same as IQS_TriggerSource_TypeDef definition
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: once the device is triggered, the device will keep acquiring data until a stop signal is received. The stop signal may be an external trigger edge or a calling of the function IQS_BusTriggerStop.

double TriggerAcqTime	When the TriggerMode is set to FixedPoints, the TriggerAcqTime specifies the length (in second) of the data to be acquired for each trigger.
double TriggerLevel_dBm	Consistent with their definitions in the IQS mode.
double TriggerTimer_Period	
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.
ReferenceClockSource_TypeDef ReferenceClockSource	Consistent with their definitions in the IQS mode.
double ReferenceClockFrequency	
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after Device_Open.
Example	
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);** **RTA_EZProfile_TypeDef UserProfile_O;** **Status = RTA_EZProfileDeInit(&Device, &UserProfile_O);**	

15.2 RTA_EZConfiguration

int RTA_EZConfiguration(void Device, const RTA_EZProfile_TypeDef* ProfileIn, RTA_EZProfile_TypeDef* ProfileOut, RTA_FrameInfo_TypeDef* FrameInfo)**	
Description	
This function configures the relevant parameters of the RTA mode. The center frequency, reference level, decimation factor and other parameters in RTA mode are encapsulated in the RTA_EZProfile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
RTA_Profile_TypeDef *RTA_ProfileIn	Configuration profile for RTAmode.
RTA_Profile_TypeDef *RTA_ProfileOut	Feedback profile from the system.
RTA_FrameInfo_TypeDef *StreamInfo	The information about the data stream under the current

	configuration.
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.
double TimeResolution	The resolution of timestamp.
double PacketAcqTime	When the TriggerMode is set to FixedPoints, the TriggerAcqTime specifies the length (in second) of the data to be acquired for each trigger.
uint32_t PacketCounts	Packet count of the specturm stream. A specturm stream is generated once the device is triggered.
uint32_t PacketFrames	Frame count of the packet.
uint32_t FFTSize	The FFT size of spectrum analysis.
uint32_t FrameWidth	The point count for per spectrum frame. It's equal to the pixel count of the probability density plot.
uint32_t FrameHeight	The number of amplitude points of the spectral frame, the same as the number of Y-axis points (height) of the probability density plot.
uint32_t PacketSamplePoints	The number of collection points corresponding to each packet of data.
uint32_t PacketValidPoints	The number of valid data points in the frequency domain contained in each packet of data.
uint32_t MaxDensityValue	The maximum value for the pixel of the probability density plot.
uint32_t GainParameter	Gain related parameters, including Space (31 to 24Bit), PreAmplifierState (23 to 16Bit), StartRFBand (15 to 8Bit), StopRFBand (7 to 0Bit).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after RTA_EZProfileDelnit.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
```

```
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
```

15.3 RTA_ProfileDeInit

int RTA_ProfileDeInit(void Device, RTA_Profile_TypeDef* UserProfile_O)**	
Description	
This function initializes the configuration profile of the RTA mode. RTA_Profile_TypeDef defines all parameters in RTA mode such as center frequency, reference level, decimate factor, etc.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
RTA_Profile_TypeDef *UserProfile_O	Pointer to the default profile.
RTA_Profile_TypeDef	
double CenterFreq_Hz	Center frequency in Hz.
double RefLevel_dBm	Reference level in dBm.
double RBW_Hz	Resolution bandwidth in Hz.
double VBW_Hz	Video bandwidth in Hz.
RBWMode_TypeDef RBWMode	RBW update mode: RBW_Manual: Input manually; RBW_Auto: set automatically according to Span; RBW_OneThousandthSpan: mandatory RBW = 0.001 * SPAN; RBW_OnePercentSpan: mandatory RBW = 0.01*SPAN。
VBWMode_TypeDef VBWMode	VBW update mode: VBW_Manual: the VBW is set manually; VBW_EqualToRBW: mandatory VBW = RBW; VBW_TenPercentRBW: mandatory VBW = 0.1*RBW; VBW_OnePercentRBW: mandatory VBW = 0.01*RBW; VBW_TenTimesRBW: mandatory VBW = 10*RBW, fully bypass VBW filter.
uint32_t DecimateFactor	Set the decimation multiple, which determines the bandwidth of the RTA.
Window_TypeDef Window	Specify the window function used for FFT analysis: 1) FlatTop: good amplitude accuracy; 2) Blackman_ Nuttall: good frequency resolution.

SweepTimeMode_TypeDef **SweepTimeMode**	Sweep time mode: SWTMode_minSWT: Sweep with minimum sweep time; SWTMode_minSWTx2: Sweep with approximately 2 times of the minimum sweep time. SWTMode_minSWTx4: Sweep with approximately 4 times of the minimum sweep time. SWTMode_minSWTx10: Sweep with approximately 10 times of the minimum sweep time. SWTMode_minSWTx20: Sweep with approximately 20 times of the minimum sweep time. SWTMode_minSWTx50: Sweep with approximately 50 times of the minimum sweep time. SWTMode_minSWTxN: Sweep with approximately N times of the minimum sweep time，N=SweepTimeMultiple. SWTMode_Manual: Sweep at approximately the specified sweep time, which is equal to SweepTime. SWTMode_minSMPxN: A single frequency point is sampled with approximately N times the shortest sampling time, N=SampleTimeMultiple
double SweepTime	When the scan time mode is specified as Manual, this parameter is the absolute time; When specified as *N, this parameter is the scan time multiplier.
Detector_TypeDef **Detector**	Detector: Detector_Sample: No inter-frame detection; Detector_PosPeak: MaxHold Frame detection is carried between power spectrum of each frequency point; Detector_Average: Average Frame detection is carried between power spectrum of each frequency point; Detector_NegPeak: MinHold Frame detection is carried between power spectrum of each frequency point; Detector_MaxPower: Each frequency point is sampled for a long time before FFT and frame spectrum with highest power is selected for FFT to capture instantaneous signals such as pulse. (SWP mode only); Detector_RawFrames: Multiple sampling, multile FFT analyses for each frequency, and frame-by-frame output power spectrum (SWP mode only); Detector_RMS: RMS Frame detection is carried between power

	spectrum of each frequency point; Detector_AutoPeak: Aoto Peak Frame detection is carried between power spectrum of each frequency point.
TraceDetectMode_TypeDef **TraceDetectMode**	Trace detection mode (frequency axis): TraceDetectMode_Auto: automatically selection of a detection; TraceDetectMode_Manual: specification of a detection.
TraceDetector_TypeDef **TraceDetector**	Trace detector: 1) TraceDetector_AutoSample: auto sample detection; 2)TraceDetector_Sample: sample detection; 3)TraceDetector_PosPeak: positive peak detection; 4)TraceDetector_NegPeak: negative peak detection; 5)TraceDetector_RMS: RMS detection; 6)TraceDetector_Bypass: no detection; 7)TraceDetector_AutoPeak: auto peak detection.
uint32_t **TraceDetectRatio**	The TraceDetectRatio indicates how many raw data points are converted to a single output point.
RxPort_TypeDef **RxPort**	RF in port: 1)ExternalPort: External port, 2)InternalPort: Internal port, 3)ANT_Port, TR_Port, SWR_Port, INT_Port: only for TRx series.
uint32_t **BusTimeout_ms**	Specify the input port of the receiver： 1) ExternalPort: external RF port is linked to the receiver; 2) InternalPort: the output of bult-in signal generator will be connected to the receiver through internal path and the external RF port is isolated.
RTA_TriggerSource_TypeDef **TriggerSource**	Consistent with their definitions in the IQS mode.
TriggerEdge_TypeDef **TriggerEdge**	
TriggerMode_TypeDef **TriggerMode**	
double **TriggerAcqTime**	When the TriggerMode is set to FixedPoints, the TriggerAcqTime specifies the length (in second) of the data to be acquired for each trigger.
DetMode_TypeDef **TraceDetector**	Specify the detector：PosPeak, NegPeak, Sample, Normal, RMS are supported.
TriggerOutMode_TypeDef **TriggerOutMode**	Consistent with their definitions in the IQS mode.
TriggerOutPulsePolarity_TypeDef **TriggerOutPulsePolarity**	
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	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.
uint8_t EnableDebugMode	Set to enable debug mode. Vendor reservation function, please keep 0
ReferenceClockSource_TypeDef ReferenceClockSource	Consistent with their definitions in the IQS mode.
double ReferenceClockFrequency	
SystemClockSource_TypeDef SystemClockSource	
double ExternalSystemClockFrequency	
int8_t Atten	
DCCancelerMode_TypeDef DCCancelerMode	
QDCMode_TypeDef QDCMode	
float QDCIGain	
float QDCQGain	
float QDCPhaseComp	
int8_t DCCIOffset	
int8_t DCCQOffset	

LOOptimization_TypeDef **LOOptimization**	LO optimization: 1)LOOpt_Auto: LO optimization, auto; 2)LOOpt_Speed: LO optimization, high sweep speed; 3)LOOpt_Spur: LO optimization, low spurious; 4)LOOpt_PhaseNoise: LO optimization, low phase noise.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to 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, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef UserProfile_O;
Status = RTA_ProfileDeInit(&Device, &UserProfile_O);
```

15.4 RTA_Configuration

int RTA_Configuration(void Device, const RTA_Profile_TypeDef* ProfileIn, RTA_Profile_TypeDef* ProfileOut, RTA_FrameInfo_TypeDef* FrameInfo)**	
Description	
Set device to the RTA mode and configurate it with parameters specified in the RTA profile.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
RTA_Profile_TypeDef ***RTA_ProfileIn**	Configuration profile for RTAmode.
RTA_Profile_TypeDef ***RTA_ProfileOut**	Feedback profile from the system.
RTA_FrameInfo_TypeDef ***StreamInfo**	The information about the data stream under the current configuration.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after RTA_ProfileDeInit.
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);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
```

15.5 RTA_BusTriggerStart

int RTA_BusTriggerStart(void **Device)	
Description	
Lauch 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.
Invocation constraints	This function needs to be called before 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, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
Status = RTA_BusTriggerStart(&Device);
```

15.6 RTA_BusTriggerStop

int RTA_BusTriggerStop(void **Device)	
Description	
End 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.
Invocation constraints	This function needs to be called after RTA_GetRealTimeSpectrum.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
Status = RTA_BusTriggerStart(&Device);
uint8_t *SpectrumBitmap = new uint8_t[FrameInfo.PacketValidPoints];
uint16_t *SpectrumBitmap = new uint16_t[FrameInfo.FrameHeight * FrameInfo.FrameWidth];
RTA_TriggerInfo_TypeDef TriggerInfo;
RTA_PlotInfo_TypeDef PlotInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = RTA_GetRealTimeSpectrum(&Device, SpectrumTrace, SpectrumBitmap,
&PlotInfo, &TriggerInfo,&MeasAuxInfo);
Status = RTA_BusTriggerStop(&Device);
delete[] SpectrumBitmap; delete[] SpectrumBitmap;
```

15.7 RTA_GetRealTimeSpectrum_Raw

int RTA_GetRealTimeSpectrum_Raw(void **Device, uint8_t SpectrumStream[],

RTA_PlotInfo_TypeDef* PlotInfo, RTA_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo);	
Description	
This function obtains the spectrum data in real-time spectrum mode.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
uint8_t SpectrumStream[]	A spectral stream consisting of consecutive spectral frames is returned, expressed as relative power, LSB = 0.75dB. This array size is equal to the RTA_FrameInfo.PacketValidPoints obtained with the RTA_Configuration function.
RTA_PlotInfo_TypeDef *RTA_PlotInfo	RTA obtains the structure of the drawing information returned after acquisition.
RTA_TriggerInfo_TypeDef *TriggerInfo	Trigger information.
MeasAuxInfo_TypeDef *MeasAuxInfo	Returns auxiliary information on the measured data; see the SWP_GetPartialSweep instructions.
RTA_PlotInfo_TypeDef	
float ScaleTodBm	Absolute power spectrum = SpectrumStream[] * ScaleTodBm + OffsetTodBm.
float OffsetTodBm	
uint64_t SpectrumBitmapIndex	The packet index and frame index to the whole spectrum stream.
RTA_TriggerInfo_TypeDef	
uint64_t SysTimerCountOfFirstDataPoint	Consistent with their definitions in the IQS mode.
uint16_t InPacketTriggeredDataSize	
uint16_t InPacketTriggerEdges	
uint32_t StartDataIndexOfTriggerEdges[25]	
uint64_t SysTimerCountOfEdges[25]	
int8_t EdgeType[25]	
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to 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, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
uint8_t *SpectrumBitmap = new uint8_t[FrameInfo.PacketValidPoints];
RTA_TriggerInfo_TypeDef TriggerInfo;
RTA_PlotInfo_TypeDef PlotInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = RTA_GetRealTimeSpectrum_Raw(&Device, SpectrumTrace, SpectrumBitmap,
&PlotInfo, &TriggerInfo,&MeasAuxInfo);
delete[] SpectrumBitmap;
```

15.8 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	
Get real-time spectrum frames and probability density plot 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[]	Pointer to the probability density plot. It's a bitmap with a size of RTA_FrameInfo_TypeDef.FrameHeight* RTA_FrameInfo_TypeDef.FrameWidth and organized with one demen array.
RTA_PlotInfo_TypeDef *RTA_PlotInfo	RTA obtains the structure of the drawing information returned after acquisition.
RTA_TriggerInfo_TypeDef *TriggerInfo	Trigger information.
MeasAuxInfo_TypeDef *MeasAuxInfo	Auxiliary measurement information.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix

	1.
Invocation constraints	This function needs to be called after RTA_Configuration.
Example	

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

15.9 RTA_SyncTimer

int RTA_SyncTimer(void Device)**	
Description	
Synchronize the trigger timer by external trigger or the 1PPS of the insystem GNSS.	
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.
Invocation constraints	This function needs to be called after RTA_Configuration,and TriggerSource selects the Timer.
Example	

```
int Status = -1;
```

```
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.TriggerSource = Timer;
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
Status = RTA_SyncTimer(Device);
```

16 ASG (Option)

ASG is an auxiliary source function that can output monophonic signal or swept signal.

16.1 ASG_ProfileDeInit

int ASG_ProfileDeInit(void Device, ASG_Profile_TypeDef* Profile)**	
Description	
The function initializes the relevant parameters of the ASG function and initialize the analog signal source.	
Compatibility	0.55.0 and later.
Parameter description	
void ****Device**	Device handle.
ASG_Profile_TypeDef ***Profile**	Pointer to the default profile.
ASG_Profile_TypeDef	
double CenterFreq_Hz	Center frequency in Hz.
double Level_dBm	Output level in dBm.
double StartFreq_Hz	Specify the start frequency of the frequency sweep whe the mode of ASG is set to ASG_FrequencySweep.
double StopFreq_Hz	Specify the stop frequency of the frequency sweep whe the mode of ASG is set to ASG_FrequencySweep.
double StepFreq_Hz	Specify the step frequency of the frequency sweep whe the mode of ASG is set to ASG_FrequencySweep.
double StartLevel_dBm	Specify the start level of the level sweep whe the mode of ASG is set to ASG_PowerSweep.
double StopLevel_dBm	Specify the stop level of the level sweep whe the mode of ASG is set to ASG_PowerSweep.
double StepLevel_dBm	Specify the step level of the level sweep whe the mode of ASG is set to ASG_PowerSweep.
double DwellTime_s	Specify the dwell time in second for power or frequency sweep.
double ReferenceClockFrequency	Specify the reference clock frequency in Hz. It allows user to adjust frequency accuracy manually.
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.
ASG_Port_TypeDef **Port**	Specify the output port of generator: 1) ASG_Port_External: external; 2) ASG_Port_Internal: internal； 3) ASG_Port_ANT: ANT Port (TRx series only)； 4) ASG_Port_T: TR Port (TRx series only); 5) ASG_Port_SWR: SWR Port (TRx series only); 6) ASG_Port_INT: INT Port (TRx series only).
ASG_Mode_TypeDef **Mode**	Specify the woring mode of generator: 1) ASG_Mute: mute; 2) ASG_FixedPoint: Fixed point; 3) ASG_FrequencySweep: Frequency sweep; 4) ASG_PowerSweep: Timer trigger.
ASG_TriggerSource_TypeDef **TriggerSource**	Specify the trigger source: 1) ASG_TriggerIn_FreeRun: free running; 2) ASG_TriggerIn_External: external trigger; 3) ASG_TriggerIn_Bus: timer triggering.
ASG_TriggerInMode_TypeDef **TriggerInMode**	Specify SG trigger in mode: 1) ASG_TriggerInMode_Null: free running; 2) ASG_TriggerInMode_SinglePoint: single point trigger (trigger a single frequency or power configuration); 3) ASG_TriggerInMode_SingleSweep: single sweep trigger (trigger a sweep that takes one cycle at a time); 4) ASG_TriggerInMode_Continous: continuous scan trigger (trigger a continuous signal).
ASG_TriggerOutMode_TypeDef **TriggerOutMode**	Specify SG trigger out mode: 1) ASG_TriggerOutMode_Null: trigger out is disabled; 2) ASG_TriggerOutMode_SinglePoint: a tirgger pluse will be sent once a frequency hop is completed; 3) ASG_TriggerOutMode_SingleSweep: a tirgger pluse will be sent once a full trace sweep is completed;
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to 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, DevNum, &BootProfile, &BootInfo);** **ASG_Profile_TypeDef *Profile;** **Status= ASG_ProfileDeInit(&Device, Profile);**

16.2 ASG_Configuration

int ASG_Configuration(void **Device, ASG_Profile_TypeDef *ProfileIn, ASG_Profile_TypeDef *ProfileOut,ASG_Info_TypeDef *ASG_Info)	
Description	
Configure the device to ASGMode and related parameters such as center frequency, power, and dwell time.	
Compatibility	0.55.0 and later.
Parameter description	
void **Device	Device handle.
ASG_Profile_TypeDef *ProfileIn	Configuration profile for ASG mode.
ASG_Profile_TypeDef *ProfileOut	Feedback profile from the system.
ASG_Info_TypeDef *ASG_Info	The information about ASG under current configuration.
ASG_Info_TypeDef	
uint32_t SweepPoints	Number of sweeping points.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	
int Status = -1; **int DeviceNum = 0;** **void *Device = NULL;** **BootProfile_TypeDef *BootProfile_IO;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef *BootInfo;** **Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);** **ASG_Profile_TypeDef ProfileIn;** **ASG_Profile_TypeDef ProfileOut;**	

```
ASG_Info_TypeDef ASG_Info;
ProfileIn. CenterFreq_Hz = 1e9;
Status = ASG_Configuration(&Device, &ProfileIn, &ProfileOut, &ASG_Info);
```

17 ASD

The ASD function is used for demodulation of analog modulated signals such as AM and FM signals.

17.1 ASD_Open

void ASD_Open (void **AnalogMod)	
Description	
This function activates the analog demodulation functions and allocates memory space for the relevant data. This function must be called before other functions of Analog are called. When a function needs to be called at the same time, it can be operated by adding several more Analog Pointers to it.	
Compatibility	0.55.0 and later.
Parameter description	
void **Analog	Pointer to the Analog memory space.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	
int Status = -1; **void *Analog = NULL;** **Status = ASD_Open(&Analog);**	

17.2 ASD_Close

void ASD_Close (void AnalogMod)**	
Description	
Close ASD function and free the relevant memory.	
Compatibility	0.55.0 and later.
Parameter description	
void **Analog	Pointer to the Analog memory space.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	Simply call this function at the end of the program execution, after calling this function the Analog function is turned off and the memory space is freed. If the Analog function needs to be invoked again, the Analog function is turned on by calling ASD_Open again.
Example	
int Status = -1; void *Analog = NULL; **Status = ASD_Open(&Analog);** **Status = ASD_Close(&Analog);**	

17.3 ASD_Demodulate_FM

int ASD_Demodulate_FM(void AnalogMod, const IQStream_TypeDef* IQStreamIn, bool reset, float result[], double* carrierOffsetHz)**	
Description	
Demodulate IQ data with FM demodulator.	
Compatibility	0.55.0 and later.
Parameter description	
void **Analog	Pointer to the Analog memory space.
IQStream_TypeDef *IQStreamIn	Pointer to the IQStream to be demodulated. The IQStream datatype includs IQ data and related configuration information.
bool reset	For a continuousdemodulation, it needs to be set to 1 for the first-time function calling, and 0 for subsequent calling. 0: No reset for the cache. 1: Reset the cache.
float result[]	Pointer to demodulation resut. The array size of result is same with that of input IQ data.
double carrierOffsetHz	The frequency offset of the carrier in Hz.
Invocation constraints	This function needs to be called after ASD_Open.
Example	

```
int Status = -1;
void *Analog = NULL;
Status = ASD_Open(&Analog);
bool reset = 1;
double carrierOffsetHz = 0;
float *result = new float[IQStreamIn.StreamInfo.PacketSamples];
Status = ASD_Demodulate_FM(&Analog, &IQStream, reset, result, &carrierOffsetHz);
delete[] result;
Status = ASD_Close(&Analog);
```

17.4 ASD_Demodulate_AM

int ASD_Demodulate_AM(const IQStream_TypeDef* IQStreamIn, float result[])	
Description	
Demodulate IQ data with AM demodulator.	
Compatibility	0.55.0 and later.
Parameter description	

IQStream_TypeDef *IQStreamIn	Pointer to the IQStream to be demodulated.
float result[]	Pointer to demodulation resut. The array size of result is same with that of input IQ data.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after ASD_Open.
Example	

```
int Status = -1;
void *Analog = NULL;
Status = ASD_Open(&Analog);
bool reset = 1;
double carrierOffsetHz = 0;
float *result = new float[IQStreamIn.StreamInfo.PacketSamples];
Status = ASD_Demodulate_AM(&Analog, &IQStream，result);
delete[] result;
Status = ASD_Close(&Analog);
```

18 Digital signal processing (Trace analysis)

18.1 DSP_TraceAnalysis_IM3

int DSP_TraceAnalysis_IM3(const double Freqs[], const float PowerSpec_dBm[], 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	
double Freqs[]	Pointer to the frequency array.
float PowerSpec_dBm[]	Pointer to the power array.
uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
TraceAnalysisResult_IP3_TypeDef *IM3Result	Returns the measurement result of IM3.
TraceAnalysisResult_IP3_TypeDef	
double LowToneFreq	Frequency of the low tone in Hz
double HighToneFreq	Frequency of the high tone in Hz.
double LowIM3PFreq	Frequency of the low intermodulation product.
double HighIM3PFreq	Frequency of the high intermodulation product.
float LowTonePower_dBm	Power of the low tone in dBm.
float HighTonePower_dBm	Power of the high tone in dBm.
float TonePowerDiff_dB	Power difference between high tone and low tone in dB.
float LowIM3P_dBc	LowIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - LowTonePower_dBm, the strength of the low intermodulation product relative to the strongest tone.
float HighIM3P_dBc	HighIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - HighTonePower_dBm, the strength of the high product relative to the strongest tone.
float IP3_dBm	IP3 results in dBm.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	
int Status = -1; **int DeviceNum = 0;** **void *Device = NULL;** **BootProfile_TypeDef BootProfile_IO;**	

```
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
TraceAnalysisResult_IP3_TypeDef IM3Result;
Status = DSP_TraceAnalysis_IM3(Frequency, PowerSpec_dBm, TraceInfo.FullsweepTracePoints, &IM3Result);
delete[] Frequency;delete[] PowerSpec_dBm;
```

18.2 DSP_TraceAnalysis_IM2

int DSP_TraceAnalysis_IM2(const double Freqs[], const float PowerSpec_dBm[], 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	
double Freqs[]	Pointer to the frequency array.
float PowerSpec_dBm[]	Pointer to the power array.
uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
TraceAnalysisResult_IP2_TypeDef *IM2Result	Returns the measurement result of IP2.
TraceAnalysisResult_IP2_TypeDef	
double LowToneFreq	Frequency of the low tone in Hz
double HighToneFreq	Frequency of the high tone in Hz.
double IM2PFreq	Frequency of the IM2 product in Hz.
float LowTonePower_dBm	Power of the low tone in dBm.
float HighTonePower_dBm	Power of the high tone in dBm.
float TonePowerDiff_dB	Power difference between the high tone and low tone in dB..
float IM2P_dBc	IM2P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) -

	IM2P_dBm, the strength of the IM2 product relative to the strongest tone.
float IP2_dBm	IP2 results in dBm.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.

Example

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
TraceAnalysisResult_IP2_TypeDef IM2Result;
Status = DSP_TraceAnalysis_IM2(Frequency, PowerSpec_dBm, TraceInfo.FullsweepTracePoints, &IM2Result);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

18.3 DSP_TraceAnalysis_PhaseNoise

int DSP_TraceAnalysis_PhaseNoise(const double Freq_Hz[], const float PowerSpec_dBm[], const double* OffsetFreqs, uint32_t TracePoints, uint32_t OffsetFreqsToAnalysis, double* CarrierFreqOut, float* PhaseNoiseOut_dBc)

Description

This function analyzes the phase noise of a spectrum trace.

Compatibility	0.55.0 and later.

Parameter description

double Freq_Hz[]	Frequency of the low tone in Hz
float PowerSpec_dBm[]	Frequency of the high tone in Hz.

double* OffsetFreqs	Pointer to the array of offset frequencies to be analyzed.
uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
uint32_t OffsetFreqsToAnalysis	The size of the OffsetFreqs array.
double* CarrierFreqOut	The calculated frequency of carrier.
float* PhaseNoiseOut_dBc	Phase noise results of the given OffsetFreqs
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.

Example

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
int32_t OffsetFreqsToAnalysis = 3;
double *OffsetFreqs = new double[OffsetFreqsToAnalysis];
OffsetFreqs[0  = 1e6;
OffsetFreqs[1] = 100e3;
OffsetFreqs[2] = 10e3;
double CarrierFreqOut = 0;
float *PhaseNoiseOut_dBc = new float[OffsetFreqsToAnalysis];
Status=DSP_TraceAnalysis_PhaseNoise(Frequency,PowerSpec_dBm,OffsetFreqs,
TraceInfo.FullsweepTracePoints,OffsetFreqsToAnalysis, &CarrierFreqOut,PhaseNoiseOut_dBc);
delete[] Frequency;
delete[] PowerSpec_dBm;
delete[] OffsetFreqs ;
delete[] PhaseNoiseOut_dBc;
```

18.4 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	
double Freq_Hz[]	Pointer to the frequency array.
float PowerSpec_dBm[]	Pointer to the power array.
uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
double CenterFrequency	Specify the channel center frequency in Hz.
double AnalysisSpan	Specify the channel bandwidth in Hz.
double RBW	Specify the resolution bandwidth in Hz.
DSP_ChannelPowerInfo_TypeDef *ChannelPowerResult	Channel power result.
DSP_ChannelPowerInfo_TypeDef	
float ChannelPower_dBm	Channel Power in dBm.
float PowerDensity	Power density in dBm/Hz.
float ChannelPeakIndex	The peak index within the channel.
double ChannelPeakFreq_Hz	Peak frequency within the channel in Hz.
float ChannelPeakPower_dBm	Peak power within the channel in dBm.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
double CenterFrequency = 1e9;
double AnalysisSpan = 50e6;
DSP_ChannelPowerInfo_TypeDef ChannelPowerResult;
Status=DSP_TraceAnalysis_ChannelPower(Frequency,PowerSpec_dBm,TraceInfo.FullsweepTracePoints,CenterFrequency, AnalysisSpan,&ChannelPowerResult);
delete[] Frequency;delete[] PowerSpec_dBm;
```

18.5 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	
double Freq_Hz[]	Pointer to the frequency array.
float PowerSpec_dBm[]	Pointer to the power array.
uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
float XdB	Specify X dB for analysis.
TraceAnalysisResult_XdB_TypeDef* XdBResult	Result for XdB anlaysis.
TraceAnalysisResult_XdB_TypeDef	
double XdBBandWidth_Hz	XdB bandwidth in Hz.
double CenterFreq_Hz	Center frequency in Hz of the signal.
double StartFreq_Hz	Start frequency in Hz of the signal.
double StopFreq_Hz	The stop frequency in Hz of the signal.
float StartPower_dBm	The power in dBm corresponding to the start frequency of the signal
float StopPower_dBm	The power in dBm corresponding to the stop frequency of the signal
uint32_t PeakIndex	Peak index within XdB bandwidth.
double PeakFreq_Hz	Peak frequency in Hz over XdB bandwidth.
float PeakPower_dBm	Peak power in dBm over XdB bandwidth.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the

	appendix 1.

Example

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float* PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
float XdB = 3;
TraceAnalysisResult_XdB_TypeDef XdBResult
Status=DSP_TraceAnalysis_XdBBW(Frequency,PowerSpec_dBm,TraceInfo.FullsweepTracePoints,XdB,
&XdBResult);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

18.6 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	
double Freq_Hz[]	Pointer to the frequency array.
float PowerSpec_dBm[]	Pointer to the power array.
uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
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	Bandwidth in Hz for given OccupiedRatio.
double CenterFreq_Hz	Center frequency in Hz of the signal.
double StartFreq_Hz	Start frequency in Hz of the signal.
double StopFreq_Hz	The stop frequency in Hz of the signal.
float StartPower_dBm	The power in dBm corresponding to the start frequency of the signal
float StopPower_dBm	The power in dBm corresponding to the stop frequency of the signal
float StartRatio	The proportion of power corresponding to the starting frequency of the occupied bandwidth.
float StopRatio	The proportion of power corresponding to the termination frequency that occupies the bandwidth.
uint32_t PeakIndex	Peak indexes within the consumed bandwidth
double PeakFreq_Hz	Peak frequency in Hz within the occupied bandwidth.
float PeakPower_dBm	Peak power in dBm within the occupied bandwidth.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float* PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
float OccupiedRatio = 0.99;
TraceAnalysisResult_OBW_TypeDef OBWResult;
```

```
Status=DSP_TraceAnalysis_OBW(Frequency,PowerSpec_dBm,TraceInfo.FullsweepTracePoints,OccupiedRatio,
&OBWResult);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

18.7 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	
double Freq_Hz[]	Pointer to the frequency array.
float PowerSpec_dBm[]	Pointer to the power array.
uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo	Specify the needed frequency information for ACPR analysis.
TraceAnalysisResult_ACPR_TypeDef* ACPRResult	Result of the ACPR analysis.
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	Power of the main channel in dBm.
uint32_t MainChPeakIndex	The peak index of the primary channel.
double MainChPeakFreq_Hz	The peak frequency in Hz of the primary channel.
float MainChPeakPower_dBm	Primary channel peak power in dBm.
double L_AdjChCenterFreq_Hz	Center frequency of the left adjacent channel in Hz.
double L_AdjChBW_Hz	Bandwidth of the left adjacency in Hz.
float L_AdjChPower_dBm	Power of the left adjacent channel in dBm.
float L_AdjChPowerRatio	Power ration of the left adjacent channel to the main channel.

float L_AdjChPowerDiff_dBc	Left adjacent channel power difference (left adjacent channel power - main channel power dBc).
float L_AdjChPeakIndex	The peak index of the left adjacent road.
double L_AdjChPeakFreq_Hz	The peak frequency in Hz of the left channel.
float L_AdjChPeakPower_dBm	Power of the left adjacent channel in dBm.
double R_AdjChCenterFreq_Hz	Center frequency of the right adjacent channel in Hz.
double R_AdjChBW_Hz	Bandwidth of the right adjacency in Hz.
float R_AdjChPower_dBm	Power of the right adjacent channel in dBm.
float R_AdjChPowerRatio	Power ration of the right adjacent channel to the main channel.
float R_AdjChPowerDiff_dBc	Right adjacent channel power difference (right adjacent channel power - main channel power dBc).
float R_AdjChPeakIndex	The peak index of the right adjacent road.
double R_AdjChPeakFreq_Hz	The peak frequency in Hz of the right channel.
float R_AdjChPeakPower_dBm	The value power in dBm of the right adjacent channel.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Example	

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &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);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float* PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo;
ACPRFreqInfo.RBW = ProfileOut.RBW_Hz;
ACPRFreqInfo.MainChCenterFreq_Hz = 1e9;
ACPRFreqInfo.MainChBW_Hz = 50e6;
```

```
ACPRFreqInfo.AdjChSpace_Hz = 50e6;
ACPRFreqInfo.AdjChPair = 1;
TraceAnalysisResult_ACPR_TypeDef ACPRPowerInfo;
Status=DSP_TraceAnalysis_ACPR(Frequency,PowerSpec_dBm,TraceInfo.FullsweepTracePoints,ACPRFreqInfo,
&ACPRPowerInfo);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

19 Digital signal processing (processing of streaming)

19.1 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	Memory space reference required to run the DSP. After calling this function, the function returns the memory address of the currently open DSP function. Later calls to other apis must use this reference to index this address.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function must be called before other DSP functions are called, and it only needs to be called once at the beginning. Other functions can perform related operations according to the device memory address returned by this function. For any non-exceptional call to DSP_Open, the DSP_Close function must be called to free the memory after the entire functionality has been used.
Example	

```
int Status = -1;
void *DSP = NULL;
Status = DSP_Open(&DSP);
```

19.2 DSP_Close

int DSP_Close(void DSP)**

Description	
This function turns off the DSP function and frees up memory space.	
Compatibility	0.55.0 and later.
Parameter description	
void **DSP	DSP handle
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	Just call this function at the end of the program execution. After calling this function, the DSP function is shut down and the memory space is released. If necessary, the DSP function is called again. The DSP functions need to be turned on again by calling DSP_Open.
Example	
int Status = -1; **void *DSP = NULL;** **Status = DSP_Open(&DSP);** **Status = DSP_Close(&DSP);**	

19.3 DSP_FFT_DeInit

int DSP_FFT_DeInit(DSP_FFT_TypeDef* IQToSpectrum)	
Description	
This function initializes the parameters for configuring FFT mode. The parameters including FFT points, window type, and decimate factor in FFT mode are uniformly encapsulated in the DSP_FFT_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
DSP_FFT_TypeDef ***IQToSpectrum**	Pointer to the default profile.
DSP_FFT_TypeDef	
uint32_t **FFTSize**	FFT analysis points.
uint32_t **SamplePts**	The number of valid sampling points.
Window_TypeDef **Window**	the window functions employed in the FFT process: 1) FlatTop; 2) Blackman_Nuttall; 3) Blackman: (Not available); 4) Hamming: (Not available); 5) Hanning: (Not available);
DetMode_TypeDef **Detector**	Type of detector during video detection: 1)raceDetector_AutoSample: auto sample detection. 2)TraceDetector_Sample: sample detection. 3)TraceDetector_PosPeak: positive peak detection.

	4)TraceDetector_NegPeak: negative peak detection. 5)TraceDetector_RMS: RMS detection. 6)TraceDetector_Bypass: no detection.
uint32_t DetectionRatio	Trace detection ratio.
float Intercept	Output spectrum interception. For example, Intercept = 0.8 to output 80% of the spectrum result.
bool Calibration	Whether calibration is performed.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called before DSP_FFT_Configuration.
Example	
int Status = -1; **DSP_FFT_TypeDef UserProfile_O;** **Status = DSP_FFT_DeInit(&UserProfile_O);**	

19.4 DSP_FFT_Configuration

int DSP_FFT_Configuration(void DSP, const DSP_FFT_TypeDef* ProfileIn, DSP_FFT_TypeDef* ProfileOut, uint32_t* TracePoints, double* RBWRatio)**	
Description	
This function configures the parameters related to the FFT mode. The parameters such as FFT points, window type, and decimate factor in FFT mode are uniformly encapsulated in the DSP_FFT_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void **DSP	DSP handle
DSP_FFT_TypeDef *ProfileIn	Configuration profile for DSP_FFT mode
DSP_FFT_TypeDef *ProfileOut	Feedback profile from the system.
uint32_t TracePoints	The number of spectrum points that can be obtained under the current DSP_FFT configuration.
double *RBWRatio	Returns the ratio of RBW to the sample rate. RBW = RBWRatio * IQSampleRate
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after DSP_FFT_D.
Example	
int Status = -1; void *DSP = NULL; uint32_t TracePoints = 0; DSP_FFT_TypeDef ProfileIn,ProfileOut; **Status = DSP_Open(&DSP);** **Status = DSP_FFT_DeInit(&ProfileIn);** **Status = DSP_FFT_Configuration(&DSP, &ProfileIn, &ProfileOut, &TracePoints);**	

19.5 DSP_FFT_IQSToSpectrum

int DSP_FFT_IQSToSpectrum(void DSP_FFT, const IQStream_TypeDef* IQStream, double* Frequency, float* PowerSpec_dBm)**	
Description	
Convert IQ data into spectrum data, including frequency and power.	
Compatibility	0.55.0 and later.
Parameter description	
void **DSP	DSP handle
IQStream_TypeDef *IQStream	Pointer to the default profile. Information about IQ streaming, including IQ data and related configuration information.
double *Frequency	The frequency array of the spectrum data. The array size is equal to the DSP_FFT_Configuration() function.
float *PowerSpec_dBm	The power array for the spectrum data. The array size is equal to the DSP_FFT_Configuration() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after DSP_FFT_Configuration.
Example	

```
int Status = -1;void *DSP = NULL;Status = DSP_Open(&DSP); uint32_t TracePoints = 0;
DSP_FFT_TypeDef ProfileIn,ProfileOut;
Status = DSP_FFT_DeInit(&ProfileIn);
Status = DSP_FFT_Configuration(&DSP, &ProfileIn, &ProfileOut, &TracePoints);
double *Frequency = new double[TracePoints];
float * PowerSpec_dBm = new float[TracePoints];
Status = DSP_FFT_IQSToSpectrum(&Device, &IQStream, Frequency, PowerSpec_dBm);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

19.6 DSP_DDC_DeInit

int DSP_DDC_DeInit(DSP_DDC_TypeDef* DDC_ProfileIn)	
Description	
This function initializes the parameters related to the DDC mode. The complex mixing and resampling parameters in DDC mode are packaged in a DSP_DDC_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
DSP_DDC_TypeDef *DDC_ProfileIn	Configuration profile for DSP_DDC mode
DSP_DDC_TypeDef	

double DDCOffsetFrequency	The frequency offset value for the multimix.
double SampleRate	The sample rate of the multimix, it needs to be the same as the IQ data sampling rate.
float DecimateFactor	The re-sampling decimate factor, 1 ~ 2^{16}.
uint64_t SamplePoints	The number of sample points for multimixing.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called before DSP_DDC_Configuration.
Example	
int Status = -1; **DSP_DDC_TypeDef DDC_ProfileIn;** **Status = DSP_DDC_DeInit(&DDC_ProfileIn);**	

19.7 DSP_DDC_Configuration

int DSP_DDC_Configuration(void DSP, const DSP_DDC_TypeDef* DDC_ProfileIn, DSP_DDC_TypeDef* DDC_ProfileOut)**	
Description	
This function configures the parameters related to the DDC mode. The complex mixing and resampling parameters in DDC mode are packaged in a DSP_DDC_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void **DSP	DSP handle
DSP_DDC_TypeDef *ProfileIn	Configuration profile for DSP_DDC mode.
DSP_DDC_TypeDef *ProfileOut	Feed back from the system.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after DSP_DDC_DeInit.
Example	
int Status = -1;void *DSP = NULL;Status = DSP_Open(&DSP); **DSP_DDC_TypeDef ProfileIn,ProfileOut;** **Status = DSP_DDC_DeInit(&ProfileIn);** **uint32_t DDC_Points = 0;** **Status = DSP_DDC_Configuration(&DSP, &ProfileIn, &ProfileOut, &DDC_Points);**	

19.8 DSP_DDC_Reset

void DSP_DDC_Reset(void DSP)**	
Description	
This function resets the cache in the DDC.	
Compatibility	0.55.0 and later.

Parameter description	
void **DSP	DSP handle
Return value	No return value.
Invocation constraints	This function needs to be called after DSP_Open.
Example	
int Status = -1; **void *DSP = NULL;** **Status = DSP_Open(&DSP);** **Status = DSP_DDC_Reset(&DSP);**	

19.9 DSP_DDC_Excute

int DSP_DDC_Excute(void DSP, const IQStream_TypeDef* IQStreamIn, IQStream_TypeDef* IQStreamOut)**	
Description	
This function digitally downconverts IQ data.	
Compatibility	0.55.0 and later.
Parameter description	
void **DSP	DSP handle
IQStream_TypeDef *IQStreamIn	Relevant information of the IQ streaming, including IQ data and related configuration information.
IQStream_TypeDef *IQStreamOut	Output the relevant information of the IQ streaming, including IQ data and related configuration information.
Invocation constraints	This function needs to be called after DSP_DDC_Configuration.
Example	
int Status = -1; **void *DSP = NULL;** **Status = DSP_Open(&DSP);** **DSP_DDC_TypeDef ProfileIn,ProfileOut;** **Status = DSP_DDC_DeInit(&ProfileIn);** **uint32_t DDC_Points = 0;** **Status = DSP_DDC_Configuration(&DSP, &ProfileIn, &ProfileOut);** **IQStream_TypeDef IQStreamOut;** **Status = DSP_DDC_Excute(&DSP, &IQStreamIn, &IQStreamOut);**	

19.10 DSP_AudioAnalysis

void DSP_AudioAnalysis(const double Audio[], const uint64_t SamplePoints, const double SampleRate, DSP_AudioAnalysis_TypeDef* AudioAnalysis)

Description	
This function analyzes the audio voltage (V), audio frequency (Hz), Sinnard (dB), and total harmonic distortion (%) parameters of the audio.	
Compatibility	0.55.0 and later.
Parameter description	
double Audio[]	The array of audio signals.
uint64_t SamplePoints	The length of the audio signal array.
double SampleRate	The sample rate of the audio signal.
DSP_AudioAnalysis_TypeDef* AudioAnalysis	Returns the measurement results for audio analysis.
DSP_AudioAnalysis_TypeDef	
double AudioVoltage	Audio voltage in V.
double AudioFrequency	Audio frequency in Hz.
double SINDA	Cinnard in dB.
double THD	Total harmonic distortion (%).
Return value	No return value.

Example

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device,&ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_BusTriggerStart(&Device);
void* AlternIQStream = NULL;
float ScaleToV = 0;
IQS_TriggerInfo_TypeDef TriggerInfo;
int32_t  I_MaxValue = 0;
uint32_t  I_MaxIndex=0;
IQS_GetIQStream(&Device, AlternIQStream,&ScaleToV,&TriggerInfo,&I_MaxValue,&I_MaxIndex);
double*Audio= new double[StreamInfo.PacketSamples];
```

```
int16_t* IQ = (int16_t*)AlternIQStream;
for(uint64_t i = 0;i<StreamInfo.PacketSamples;i++){
    Audio[i] = ((double)IQ[2*i])*ScaleToV ;
}
DSP_AudioAnalysis_TypeDef AudioAnalysis;
DSP_AudioAnalysis(Audio,StreamInfo.PacketSamples,StreamInfo.IQSampleRate,&AudioAnalysis);
delete[] Audio;
```

19.11 DSP_LPF_DeInit

void DSP_LPF_DeInit(Filter_TypeDef* LPF_ProfileIn)	
Description	
This function initializes the relevant parameters of LPF mode. The number of filter taps, cutoff frequency, stopband attenuation and other parameters in LPF mode are packaged in the Filter_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
Filter_TypeDef ***LPF_ProfileIn**	The pointer to the default profile.
Filter_TypeDef	Low-pass filter structure.
int n	Set the number of filter taps (n > 0).
float fc	Set the cutoff frequency (cutoff frequency/sample rate 0 < fc < 0.5).
float As	Set the stopband attenuation (As > 0, [dB]).
float mu	Set the fractional sampling offset (-0.5 < mu < 0.5).
uint32_t SamplePts	Set the number of Samples > 0.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called before DSP_LPF_Configuration.
Example	
Filter_TypeDef **LPF_ProfileIn;** **DSP_LPF_DeInit(&LPF_ProfileIn);**	

19.12 DSP_LPF_Configuration

void DSP_LPF_Configuration(void DSP, const Filter_TypeDef* LPF_ProfileIn, Filter_TypeDef* LPF_ProfileOut)**	
Description	
This function configures the parameters related to the LPF mode. The number of filter taps, cutoff frequency, stopband attenuation and other parameters in LPF mode are packaged in the Filter_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void **DSP	The pointer to the DSP memory space.

Filter_TypeDef *LPF_ProfileIn	The pointer to the default profile.
Filter_TypeDef *LPF_ProfileOut	The feed back from the system.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after DSP_LPF_DeInit.
Example	
int Status = -1; void *DSP = NULL; **Status = DSP_Open(&DSP);** **Filter_TypeDef LPF_ProfileIn;** **Filter_TypeDef LPF_ProfileOut;** **DSP_LPF_DeInit(&LPF_ProfileIn);** **DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileIOut);**	

19.13 DSP_LPF_Reset

void DSP_LPF_Reset(void DSP)**	
Description	
This function resets the cache in LPF.	
Compatibility	0.55.0 and later.
Parameter description	
void **DSP	The pointer to the memory space.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after DSP_Open.
Example	
int Status = -1; **void *DSP = NULL;** **Status = DSP_Open(&DSP);** **DSP_LPF_Reset(&DSP);**	

19.14 DSP_LPF_Execute_Real

void DSP_LPF_Execute_Real(void DSP, float* Signal, float* LPF_Signal)**	
Description	
This function performs low-pass filtering on the real signal.	
Compatibility	0.55.0 and later.
Parameter description	
void **DSP	The pointer to the memory space.
float *Signal	Input signal.
float *LPF_Signal	Low-pass filtered signal.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.

Invocation constraints	This function needs to be called after DSP_LPF_Configuration.
Example	

```
int Status = -1;
void *DSP = NULL;
Status = DSP_Open(&DSP);
Filter_TypeDef LPF_ProfileIn;
Filter_TypeDef LPF_ProfileOut;
DSP_LPF_DeInit(&LPF_ProfileIn);
LPF_ProfileIn.SamplePts = 16242;
DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileIOut);
float* LPF_Signal = new float[LPF_ProfileIn.SamplePts];
DSP_LPF_Execute_Real(&DSP, Signal, LPF_Signal);
delete[] LPF_Signal;
```

19.15 DSP_LPF_Execute_Complex

void DSP_LPF_Execute_Complex(void DSP, const IQStream_TypeDef* IQStreamIn, IQStream_TypeDef* IQStreamOut)**	
Description	
This function performs low-pass filtering on IQ signals.	
Compatibility	0.55.0 and later.
Parameter description	
void **DSP	The pointer to the memory space.
IQStream_TypeDef *IQStreamIn	The pointer to the default profile.
IQStream_TypeDef *IQStreamOut	The feedback from the system.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the appendix 1.
Invocation constraints	This function needs to be called after DSP_LPF_Configuration.
Example	

```
int Status = -1; void *DSP = NULL;
Status = DSP_Open(&DSP);
Filter_TypeDef LPF_ProfileIn,LPF_ProfileOut; IQStream_TypeDef IQStreamOut;
DSP_LPF_DeInit(&LPF_ProfileIn);
DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileIOut);
DSP_LPF_Execute_Complex(&DSP, &IQStreamIn, &IQStreamOut);
```

20 Appendix 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 Chinese characters, the API call will also indicate file loading failure.