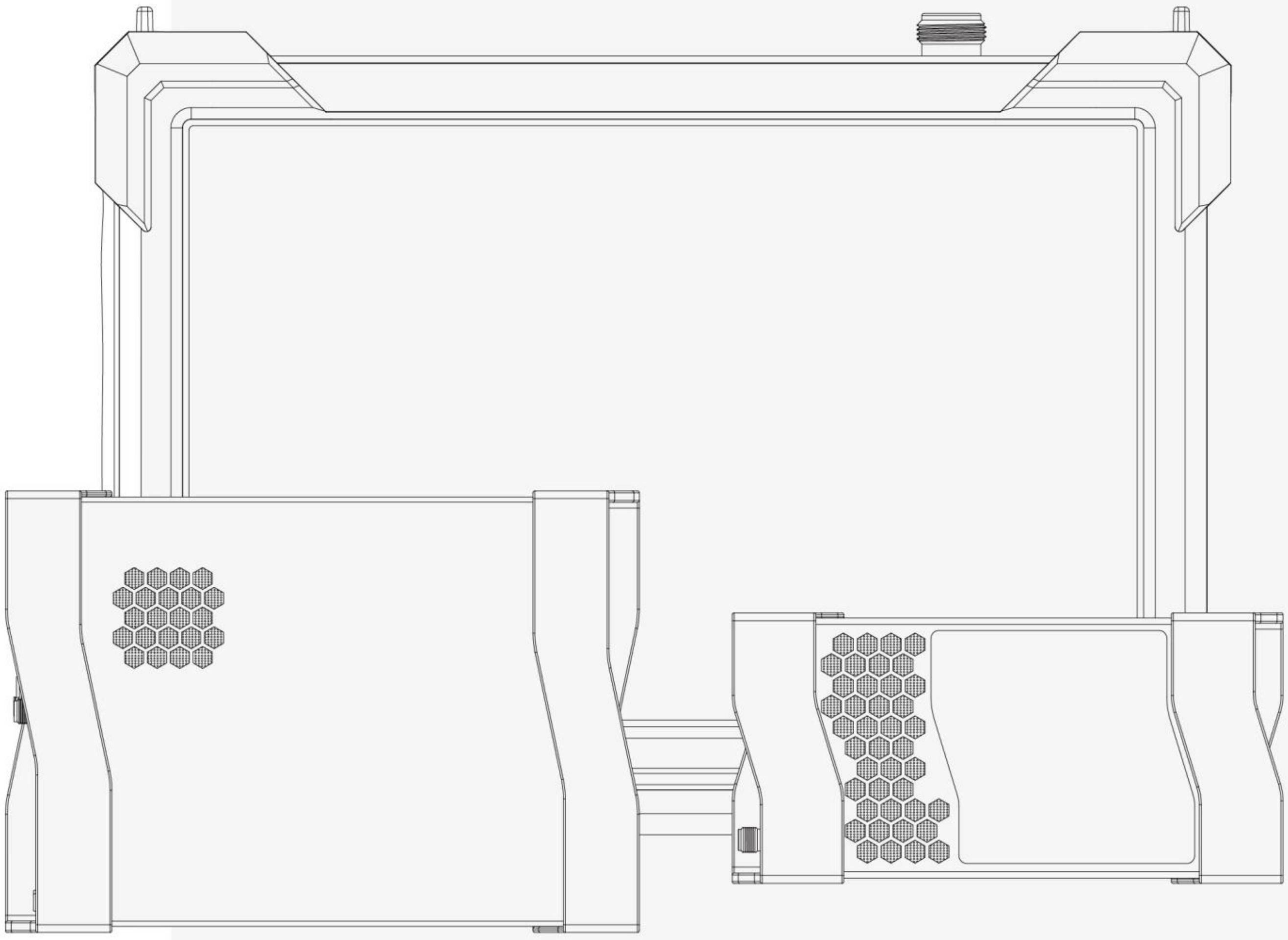
USER
GUIDE

HTRA API Example Usage Guide



V2.0
12/05/2025

HAROGIC

Contents

Version Management

Updated Description Sheet

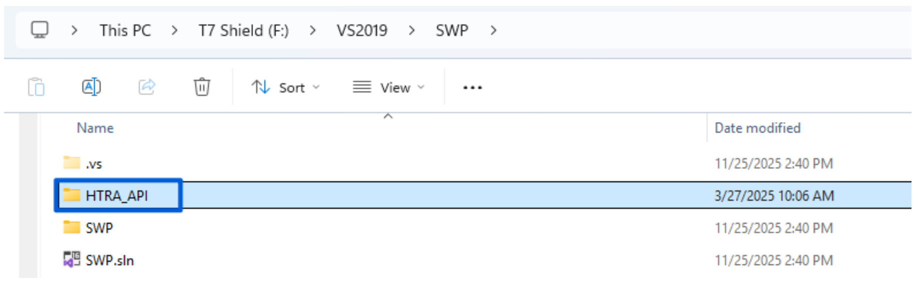
Version	Description	Date
V2.0	Initial Version	12/05/2025

1. C/C++

1.1 Configure Development Environment

The following uses configuring Debug Win32 as an example. For other build types and platforms, please refer to the end of this chapter.

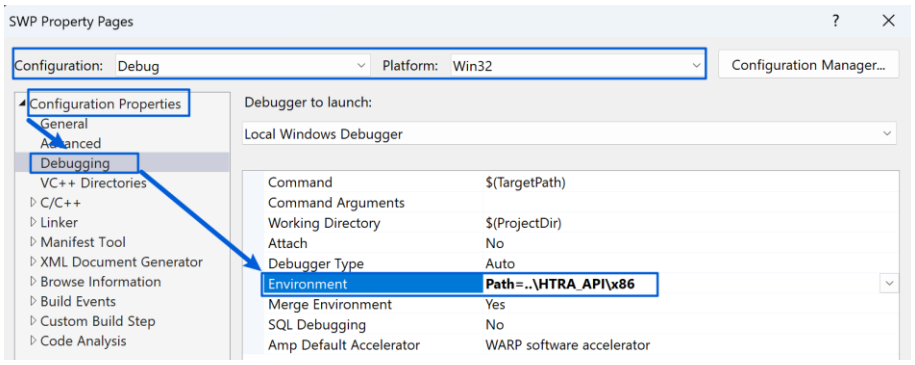
1. Open Visual Studio 2019 and create a new project;
2. After creation is complete, copy the "Windows\HTRA_API" folder from the delivery USB drive to a directory at the same level as the project;



Name	Date modified
.vs	11/25/2025 2:40 PM
HTRA_API	3/27/2025 10:06 AM
SWP	11/25/2025 2:40 PM
SWP.sln	11/25/2025 2:40 PM

Figure 1 Copy the HTRA_API file

3. Double-click to open SWP.sln, right-click on "Source Files" -> "Add" -> "New Item" -> "C++ File (.cpp)" to create a new SWP.cpp file;
4. Click "Project" -> "Properties" in the menu bar, set "Configuration" to "Debug" and "Platform" to "Win32";
5. In Configuration Properties -> Debugging, set the Environment variable to "Path=..\HTRA_API\x86";



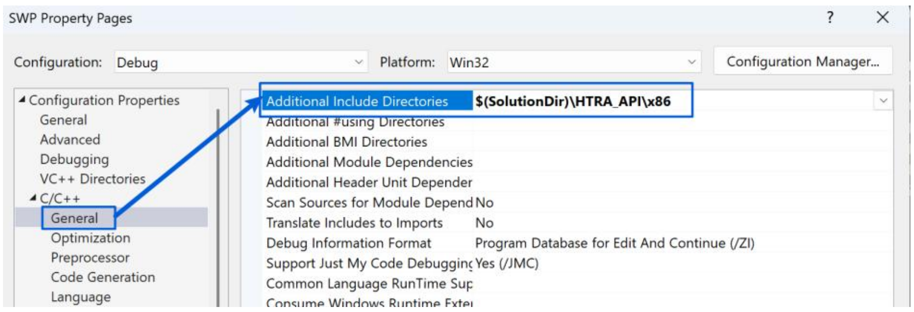
Configuration: Debug
Platform: Win32

Debugger to launch:
Local Windows Debugger

Command	$(TargetPath)
Command Arguments	
Working Directory	$(ProjectDir)
Attach	No
Debugger Type	Auto
Environment	Path=..\HTRA_API\x86
Merge Environment	Yes
SQL Debugging	No
Amp Default Accelerator	WARP software accelerator

Figure 2 Configure environment variables

6. In Configuration Properties -> C/C++ -> General, set the Additional Include Directories to "$(SolutionDir)\HTRA_API\x86";


SWP Property Pages

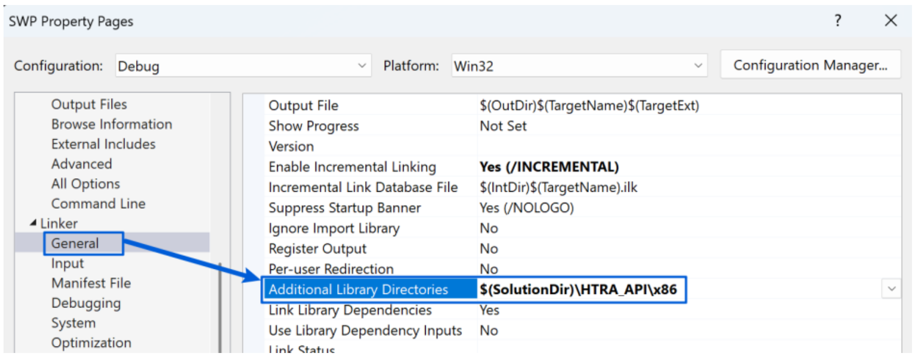
Configuration: Debug　Platform: Win32　Configuration Manager...

- Configuration Properties
  - General
  - Advanced
  - Debugging
  - VC++ Directories
  - C/C++
    - General
    - Optimization
    - Preprocessor
    - Code Generation
    - Language

Additional Include Directories	**$(SolutionDir)\HTRA_API\x86**
Additional #using Directories	
Additional BMI Directories	
Additional Module Dependencies	
Additional Header Unit Depender	
Scan Sources for Module Depend	No
Translate Includes to Imports	No
Debug Information Format	Program Database for Edit And Continue (/ZI)
Support Just My Code Debugging	Yes (/JMC)
Common Language RunTime Sup	
Consume Windows Runtime Exter	

Figure 3 Add additional include directories

7. Set the "Additional Library Directories" under Configuration Properties -> Linker -> General to "$(SolutionDir)\HTRA_API\x86";


SWP Property Pages

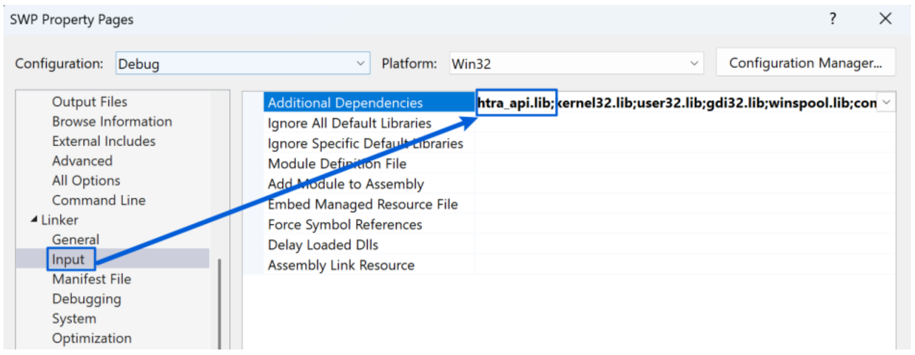
Configuration: Debug　Platform: Win32　Configuration Manager...

- Output Files
- Browse Information
- External Includes
- Advanced
- All Options
- Command Line
- Linker
  - General
  - Input
  - Manifest File
  - Debugging
  - System
  - Optimization

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	**Yes (/INCREMENTAL)**
Incremental Link Database File	$(IntDir)$(TargetName).ilk
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	**$(SolutionDir)\HTRA_API\x86**
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Link Status	

Figure 4 Configure additional library directories

8. Add "htra_api.lib" to "Additional Dependencies" under Configuration Properties -> Linker -> Input, then click "OK".


SWP Property Pages

Configuration: Debug　Platform: Win32　Configuration Manager...

- Output Files
- Browse Information
- External Includes
- Advanced
- All Options
- Command Line
- Linker
  - General
  - Input
  - Manifest File
  - Debugging
  - System
  - Optimization

Additional Dependencies	**htra_api.lib;kernel32.lib;user32.lib;gdi32.lib;winspool.lib;con**
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

Figure 5 Configure additional dependencies

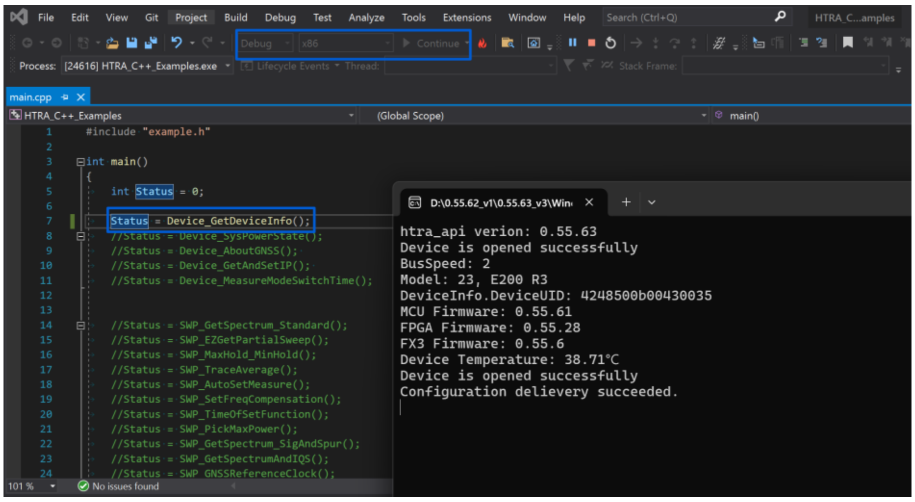
- Configure Release Win32: Set "Configuration" in Step 4 to "Release", and keep all other settings unchanged;
- Configure Debug x64: Set "Configuration" in Step 4 to "Debug" and "Platform" to "x64". Also replace all environment variables or additional directory paths containing "HTRA_API\x86" with "HTRA_API\x64";
- Configure Release x64: Set "Configuration" in Step 4 to "Release" and "Platform" to "x64". Also replace all environment variables or additional directory paths containing "HTRA_API\x86" with "HTRA_API\x64";

1.2 Usage Process for C++ Examples

Note: Only one example can run at a time, and the example cannot run simultaneously with SAStudio4.

1.2.1 Usage of General C++ Example

1. Use Visual Studio to open the solution HTRA_C++_Examples.sln located in the folder Windows\HTRA_API_Example\HTRA_C++_Examples on the provided USB drive;
2. Click the HTRA_C++_Examples project on the right, then open the main.cpp file under the Source Files folder;
3. Each example program is encapsulated in a separate function. To use one, uncomment the corresponding routine, save the file, select the build architecture, and then click Run.



```
#include "example.h"

int main()
{
    int Status = 0;

    Status = Device_GetDeviceInfo();
    //Status = Device_SysPowerState();
    //Status = Device_AboutGNSS();
    //Status = Device_GetAndSetIP();
    //Status = Device_MeasureModeSwitchTime();


    //Status = SWP_GetSpectrum_Standard();
    //Status = SWP_EZGetPartialSweep();
    //Status = SWP_MaxHold_MinHold();
    //Status = SWP_TraceAverage();
    //Status = SWP_AutoSetMeasure();
    //Status = SWP_SetFreqCompensation();
    //Status = SWP_TimeOfSetFunction();
    //Status = SWP_PickMaxPower();
    //Status = SWP_GetSpectrum_SigAndSpur();
    //Status = SWP_GetSpectrumAndIQS();
    //Status = SWP_GNSSReferenceClock();
```

```
htra_api verion: 0.55.63
Device is opened successfully
BusSpeed: 2
Model: 23, E200 R3
DeviceInfo.DeviceUID: 4248500b00430035
MCU Firmware: 0.55.61
FPGA Firmware: 0.55.28
FX3 Firmware: 0.55.6
Device Temperature: 38.71℃
Device is opened successfully
Configuration delievery succeeded.
```

Figure 6 Run the Device_GetDeviceInfo example

1.2.2 Usage of the AM/FM Demodulation Example

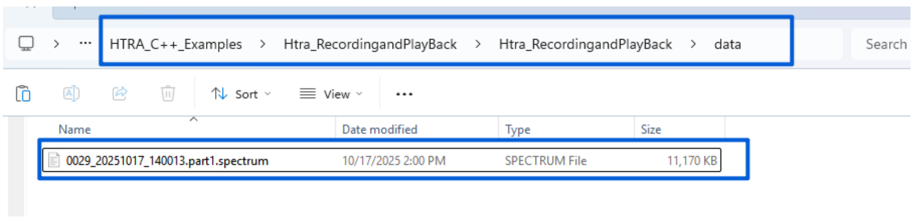
1. Open the solution Htra_Demodulation.sln in Windows\HTRA_C++_Examples\Htra_Demodulation on the supplied USB drive using Visual Studio;
2. Click the Htra_Demodulation project on the right, then open the main.cpp file under the Source Files folder;
3. In the C++ example package, each demo routine is encapsulated in a separate function. To run a demo, uncomment the corresponding routine, save the file, select the build architecture, and click Run.

1.2.3 Usage of the Recording and Playback Example

1. Open the solution Htra_RecordingandPlayBack.sln in Visual Studio, located in the USB drive under Windows\HTRA_API_Example\Htra_RecordingandPlayBack;
2. Double-click the Htra_RecordingandPlayBack project on the right to open the main.cpp file in the source files;
3. Each routine in the recording and playback example is encapsulated in a separate function. To use the example, simply uncomment it.

■ Usage of the Reading Example

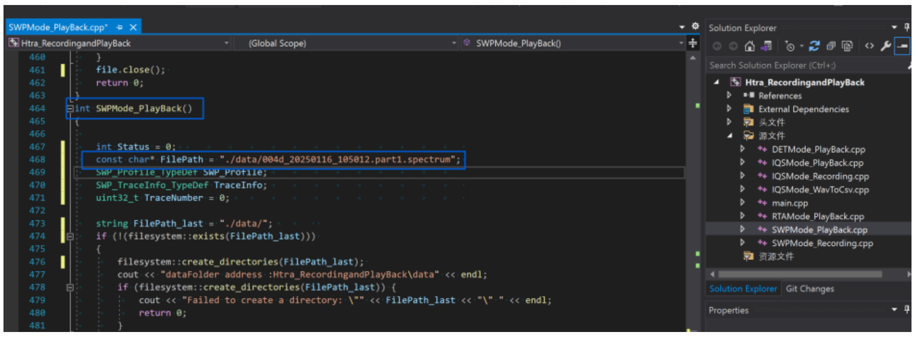
1) For example, when reading SWP mode stream disk data, uncomment the SWPMode_PlayBack function and save;
2) Place the recording file generated in SWP mode into the folder Windows\HTRA_C++_Examples\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data;


HTRA_C++_Examples > Htra_RecordingandPlayBack > Htra_RecordingandPlayBack > data
Search
Sort
View

Name	Date modified	Type	Size
0029_20251017_140013.part1.spectrum	10/17/2025 2:00 PM	SPECTRUM File	11,170 KB

Figure 7 Place the recording file under the standard spectrum analysis mode

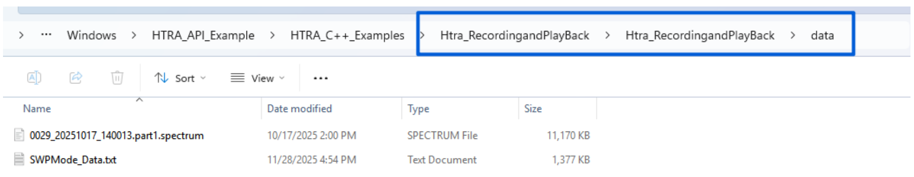
3) Click to enter SWPMode_PlayBack.cpp and modify the name of the recorded file in the SWPMode_PlayBack() function;



```
    }
    file.close();
    return 0;
}
int SWPMode_PlayBack()
{

    int Status = 0;
    const char* FilePath = "./data/004d_20250116_105012.part1.spectrum";
    SWP_Profile_TypeDef SWP_Profile;
    SWP_TraceInfo_TypeDef TraceInfo;
    uint32_t TraceNumber = 0;

    string FilePath_last = "./data/";
    if (!(filesystem::exists(FilePath_last)))
    {
        filesystem::create_directories(FilePath_last);
        cout << "dataFolder address :Htra_RecordingandPlayBack\data" << endl;
        if (filesystem::create_directories(FilePath_last)) {
            cout << "Failed to create a directory: \"" << FilePath_last << "\" " << endl;
            return 0;
        }
```

Figure 8 Modify the recording file name

4) Running the program will generate the parsed data file "SWPMode_Data.txt" in the data folder under the SWP mode.

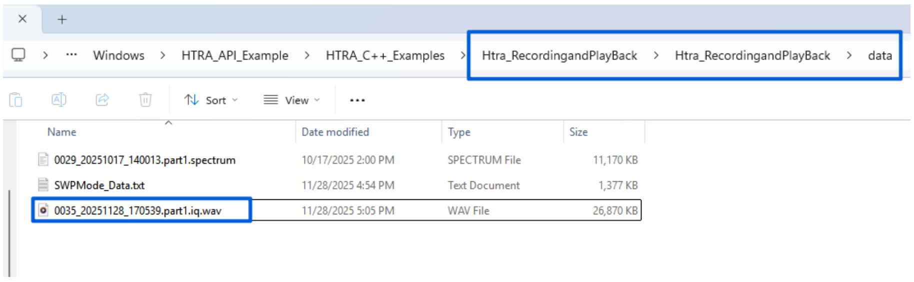

Windows > HTRA_API_Example > HTRA_C++_Examples > Htra_RecordingandPlayBack > Htra_RecordingandPlayBack > data
Sort
View

Name	Date modified	Type	Size
0029_20251017_140013.part1.spectrum	10/17/2025 2:00 PM	SPECTRUM File	11,170 KB
SWPMode_Data.txt	11/28/2025 4:54 PM	Text Document	1,377 KB

Figure 9 View the parsed data in SWP mode

- Usage of the Recording Example

1) For example, when testing the IQSMode_Recording routine, uncomment and save;
2) Click to enter the IQSMode_Recording() function, configure the relevant parameters, and then run the program. During execution, the file will be continuously recorded until manually stopped;
3) The recording file data in IQS mode can be found in the HTRA_C++_Examples\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data folder.



Windows > HTRA_API_Example > HTRA_C++_Examples > Htra_RecordingandPlayBack > Htra_RecordingandPlayBack > data

Name	Date modified	Type	Size
0029_20251017_140013.part1.spectrum	10/17/2025 2:00 PM	SPECTRUM File	11,170 KB
SWPMode_Data.txt	11/28/2025 4:54 PM	Text Document	1,377 KB
0035_20251128_170539.part1.iq.wav	11/28/2025 5:05 PM	WAV File	26,870 KB

Figure 10 View the recorded file data in IQ streaming mode

1.3 Device-related

1.3.1 Get Device Information

Device_GetDeviceInfo.cpp: Get device information, including API version, USB version, device model, device UID, MCU version, FPGA version, and device temperature.

1.3.2 Device Standby

Device_SysPowerState.cpp: An example of setting the device's standby state, which can be configured to normal operating state or RF powered down state (low power).

1.3.3 GNSS-related

Device_AboutGNSS.cpp: Retrieves information such as latitude, longitude, altitude, and time obtained from the GNSS module, acquires GNSS-related latitude and time information from MeasAuxInfo in SWP mode, and obtains latitude and time information from IQStream.DeviceState in IQS mode.

1.3.4 Get and modify the IP address of the NX device

Device_GetAndSetIP.cpp: Retrieve the device's IP address and modify it using the device UID or the device's current IP address.

1.3.5 Mode switching time consumption

Device_MeasureModeSwitchTime.cpp: Get the time required for the current host computer to switch between different modes.

1.4 SWP Mode

1.4.1 10 MHz Reference Clock Using Internal GNSS Module

SWP_GetSpectrum_Standard.cpp: Obtain spectrum data by calling the function interface.

1.4.2 Maximum and minimum hold

SWP_MaxHold_MinHold.cpp: Set the trace mode to MaxHold or MinHold, and use SWP_ResetTraceHold to reset the hold.

1.4.3 Average Trace

SWP_TraceAverage.cpp: Average processing of the acquired trace.

1.4.4 Automatic Configuration Measurement

SWP_AutoSetMeasure.cpp: Automatically configures relevant parameters based on specific SWP applications, completing measurements by issuing automatic configuration parameters.

1.4.5 Frequency Compensation

SWP_SetFreqCompensation.cpp: When an external attenuator is present, compensation can be applied to the corresponding frequency band to ensure that the test results remain accurate.

1.4.6 Function execution time, sweep speed, and throughput

SWP_TimeOfSetFunction.cpp: Obtains the call duration of the SWP_Configuration, SWP_GetPartialSweep, and SWP_GetFullSweep functions, along with the sweep speed and throughput under the current configuration.

1.4.7 Obtain spectrum peak values

SWP_PickMaxPower.cpp: Obtain the maximum power point of the current spectrum and its corresponding frequency point.

1.4.8 Signals and Spurious

SWP_GetSpectrum_SigAndSpur.cpp: This can distinguish between signals and spurious after obtaining spectrum data.

1.4.9 Simultaneous Acquisition of Spectrum and IQ

SWP_GetSpectrumAndIQS.cpp: This allows for the simultaneous acquisition of spectrum data and IQ data.

1.4.10 Reading SWP Stream Disk Data from Application Software

SWPMode_PlayBack.cpp: This can read the recorded file data in SWP mode from Application Software and write the read spectrum data into SWP Mode_Data.txt.

1.4.11 Using GNSS 10MHz Reference Clock

SWP_GNSSReferenceClock.cpp: This uses a high-quality GNSS module's 10MHz reference clock in SWP mode.

1.4.12 External Trigger Mode

SWP_GetSpectrum_Trigger.cpp: This obtains spectrum data when the trigger source is set to external trigger.

1.4.13 Trace Alignment Method

SWP_GetSpectrum_TraceAlign.cpp: Obtain spectrum data when the trace alignment method is set to align to the starting frequency or align to the center frequency.

1.4.14 Number of Spectrum Frames Obtainable within a Certain Time

SWP_Fixedtime_GetFrames.cpp: Loop 50 times to obtain 10 seconds of spectrum data, resulting in the average number of spectrum frames that can be obtained within 10 seconds.

1.4.15 External Trigger Calibration of Internal 10MHz Reference Clock

SWP_GetSpectrum_Calibration.cpp: An example of the device calibrating the clock via GNSS-1PPS or through external trigger.

1.4.16 Phase noise measurement

SWP_Meas_PhaseNoise.cpp: Measures the single-sideband phase noise (unit: dBc/Hz) of the received signal at the user-specified frequency deviation, supports multi-frequency point configuration.

1.4.17 Channel power measurement

SWP_Meas_ChannelPower.cpp: Measures the channel power of the received signal.

1.4.18 Adjacent Channel Power Ratio Measurement

SWP_Meas_ACPR.cpp: Measure the adjacent channel power ratio of the received signal.

1.4.19 Percentage occupied bandwidth Measurement

SWP_Meas_OBW.cpp: Measures the occupied bandwidth of the received signal in percentage.

1.4.20 XdB Occupied Bandwidth Measurement

SWP_Meas_XdBBW.cpp: Measures the occupied bandwidth of the received signal in XdB.

1.4.21 IM3 measurements

SWP_Meas_IM3.cpp: IM3 measurement of the received signal.

1.4.22 Frequency interval matching RBW

SWP_RBW_Spaced_Trace.cpp: Makes the frequency interval between two points of the trace close to the set RBW value.

1.4.23 Using an External 10MHz Reference Clock

SWP_RefCLKSource_External.cpp: Use external 10MHz reference clock. (Take the use of external 10MHz reference clock in SWP mode as an example, the same method is used in other modes).

1.5 IQS Mode

1.5.1 Obtain Fixed Number or Continuous Stream of IQ Data

IQS_GetIQ_Standard.cpp: Obtain a fixed number or continuous stream of IQ data under professional configuration.

1.5.2 IQ data converted to voltage V units

IQS_ScaleIQDataToVolts.cpp: Converts the acquired IQ data into data in volts (V).

1.5.3 Time taken to issue configuration and acquire IQ

IQS_ConfigandGetIQ_Time.cpp: Obtains the call duration of the IQS_Configuration and IQS_GetIQStream_PM1 functions.

1.5.4 IQ to Spectrum Data

DSP_IQSToSpectrum.cpp: Converts the acquired time-domain IQ data into spectrum data using spectral analysis methods.

1.5.5 IQ to Spectrum (using liquid library version)

IQS_ToSpectrumByLiquid.cpp: Uses the liquid library to convert the acquired time-domain IQ data into spectrum data through spectral analysis methods.

1.5.6 FM Demodulation Playback

DSP_FMDemod.cpp: Performs FM demodulation on the IQ data and plays the demodulated audio.

1.5.7 FM Demodulation Data Analysis

IQS_FMDataAnalysis.cpp: Audio analysis of the IQ data after FM demodulation to get the audio voltage, audio frequency, SINAD and total harmonic distortion.

1.5.8 AM Demodulation Playback

DSP_AM_Demod.cpp: Performs AM demodulation on IQ data and plays the demodulated audio.

1.5.9 AM Demodulation Data Analysis

IQS_FMDataAnalysis.cpp: Audio analysis of the IQ data after AM demodulation to get the audio voltage, audio frequency, SINAD and total harmonic distortion.

1.5.10 Digital Downconversion

DSP_DDC.cpp: Resamples the obtained IQ data.

1.5.11 Digital Low-Pass Filter

DSP_LPF.cpp: Perform low-pass filtering on the obtained IQ data.

1.5.12 Audio Analysis

IQS_AudioAnalysis.cpp: Performs audio analysis on the demodulated IQ data to obtain audio voltage, audio frequency, SINAD, and total harmonic distortion.

1.5.13 Read the IQS stream disk data from Application Software

IQSMode_PlayBack.cpp: Parses the recorded file data in IQS mode from Application Software and writes the read spectrum data into the IQSMode_Data.txt file.

1.5.14 Record IQ data in .wav format

IQSMode_Recording.cpp: Stores the acquired IQ data in .wav format.

1.5.15 .wav changed to .csv

IQSMode_WavToCsv.cpp: Parses and extracts I and Q channel data from the .wav recording file data in IQS mode, converting it into a .CSV format file for saving.

1.5.16 Streaming and reading IQ data

IQS_GetIQToTxt.cpp: Writes the obtained IQ data into a .txt file.

1.5.17 Multithreaded acquisition, processing, and streaming of IQ data

IQS_Multithread_GetIQ_FFT_Write: Simultaneously acquires IQ data, performs FFT, and writes IQ data into a .txt file.

1.5.18 GNSS1PPS trigger

IQS_GNSS_1PPS.cpp: Configure the trigger source to be the 1PPS signal provided by the internal GNSS system.

1.5.19 IQS multi-device synchronization

IQS_MultiDevSync_fixed.cpp: Multiple devices simultaneously collect the same signal at the same time.

1.5.20 External Trigger

IQS_ExternalTrigger.cpp: Configure the trigger source as external trigger.

1.5.21 Timer Trigger

IQS_TimerTrigger.cpp: Configure the trigger source as timer trigger.

1.5.22 Level Trigger(pre-trigger)

IQS_LevelTrigger_PreTriggerr.cpp: Configure the trigger source as level trigger and set the pre-trigger time.

1.5.23 Level Trigger (Trigger delay)

IQS_LevelTrigger_TriggerDelay.cpp: Configure the trigger source to be level trigger and set the trigger delay time.

1.5.24 Using GNSS 10MHz Reference Clock

IQS_Enable_GNSS_10MHz.cpp: Use a high-quality GNSS module's 10MHz reference clock in IQS mode.

1.5.25 Fixed Step Multi-Frequency IQ Data Acquisition

IQS_SetFreqScan: Configure the start and end frequencies and frequency points of IQ in advance. After configuring once, the data will be collected sequentially according to the set frequency points.

1.6 DET Mode

1.6.1 Obtain detection data for fixed points or continuous streams

DETMode_Standard.cpp: Obtain detection data for fixed points or continuous streams.

1.6.2 Read the DET stream disk data of Application Software

DETMode_PlayBack.cpp: Read the recorded files in DET mode from Application Software and output the detection data to the DETMode_Data.txt file.

1.6.3 Pulse detection (to be opened later)

1.7 RTA Mode

1.7.1Obtain real-time spectrum data for fixed points or continuous stream

RTAMode_Standard.cpp: Obtain real-time spectrum data for a fixed number of points (duration) or continuous stream.

1.7.2 Read the RTA stream disk data of Application Software

RTAMode_PlayBack.cpp: Read the recorded file data in RTA mode from Application Software, while being able to specify reading a certain packet of data, and write the read spectrum data to the RTAMode_Data.txt file.

1.7.3 Time consumption of each frame of data in RTA mode

RTAMode_Standard_perframe.cpp: Acquire 100 frames of data and calculate the average processing time for each frame.

1.8 ASG Signal Source (optional)

1.8.1 Output single tone/sweep/power scan signals

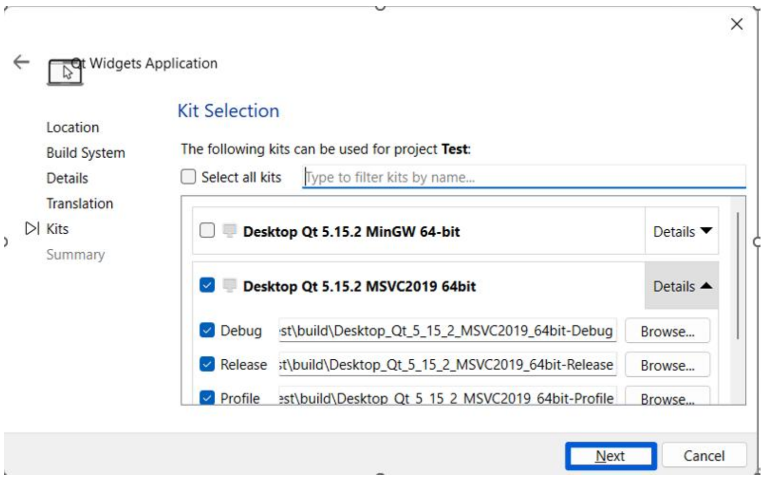
ASG_SignalOutput.cpp: Output single tone/sweep/power scan signals as needed.

2. Qt

2.1 Configure Development Environment

The following uses the configuration of an x64 architecture development environment as an example.

1. Create a new folder (taking "QtTest" as an example) to store the project files;
2. Copy the "Windows\HTRA_API\x64\htra_api" folder from the provided USB drive into the newly created "QtTest" folder;
3. Open Qt Creator, and click "File" -> "New File or Project";
4. In project dialog, click "Application (Qt)", select "Qt Widgets Application", then click "Choose...";
5. Enter the project name (e.g., "test"), click the "Browse" button, select the previously created "QtTest" folder, and then click "Next";
6. Select "qmake", continue by clicking "Next" until the "Kit Selection" screen appears. On this screen, choose a build environment, and then click "Next";



Qt Widgets Application

Kit Selection

The following kits can be used for project **Test**:

☐ Select all kits

☐ **Desktop Qt 5.15.2 MinGW 64-bit**

☑ **Desktop Qt 5.15.2 MSVC2019 64bit**

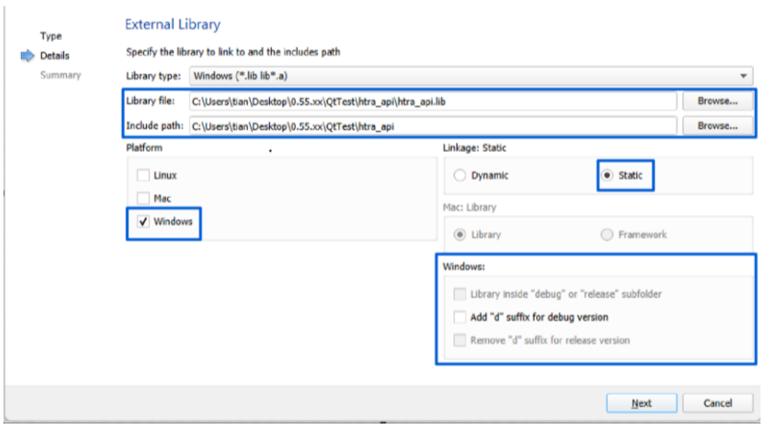
☑ Debug st\build\Desktop_Qt_5_15_2_MSVC2019_64bit-Debug

☑ Release st\build\Desktop_Qt_5_15_2_MSVC2019_64bit-Release

☑ Profile est\build\Desktop_Qt_5_15_2_MSVC2019_64bit-Profile

Figure 11 Select the build environment

7. Click "Finish" to create the project;
8. In the Qt Creator main interface, right-click the "Test" project, select "Add Library..." -> "External Library" -> "Next";
9. Click to browse the library file, select the htra_api.lib library in "QtTest\htra_api", and click "Open";
10. Uncheck all options under Windows, then click "Static Library", finally select the Windows platform, and click "Next";



Type
Details
Summary

External Library

Specify the library to link to and the includes path

Library type: Windows (*.lib lib*.a)

Library file: C:\Users\tian\Desktop\0.55.xx\QtTest\htra_api\htra_api.lib

Include path: C:\Users\tian\Desktop\0.55.xx\QtTest\htra_api

Platform: Linux [ ] Mac [ ] [✓ Windows]

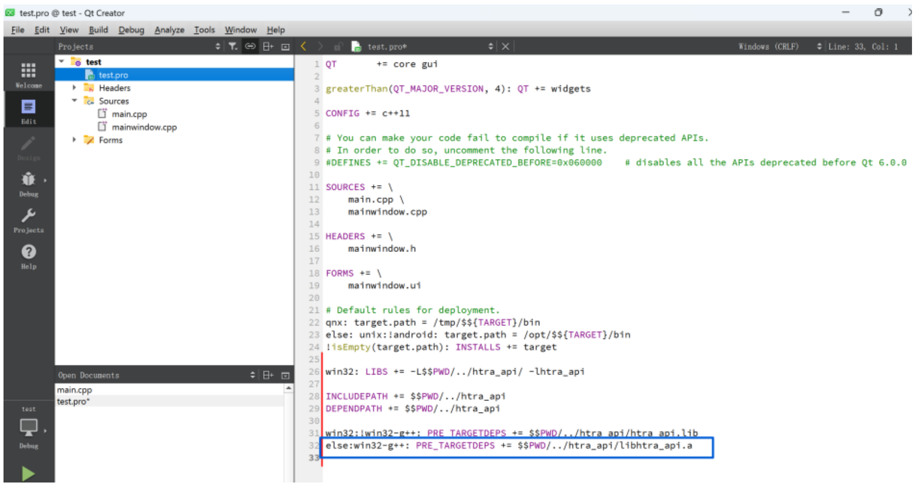
Linkage: Static — Dynamic [✓ Static]

Mac: Library — [✓ Library] Framework

Windows:
[ ] Library inside "debug" or "release" subfolder
[ ] Add "d" suffix for debug version
[ ] Remove "d" suffix for release version

Figure 12 Set the build configuration

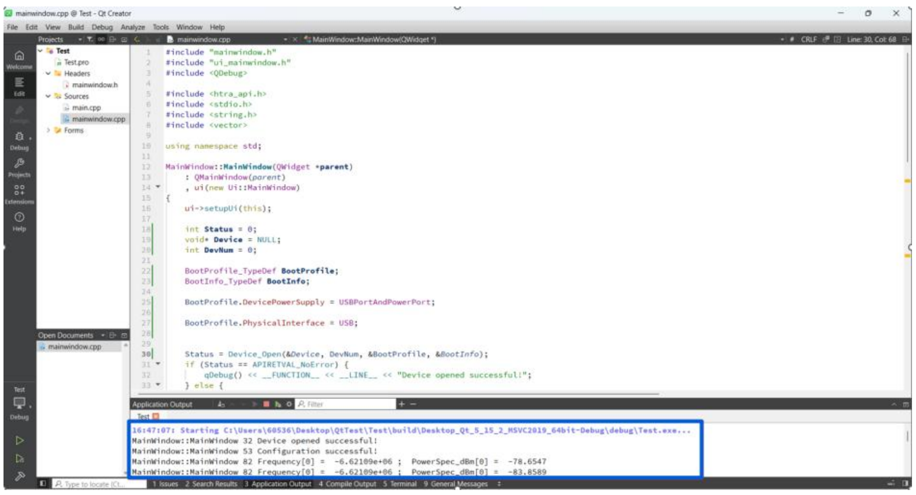
11. Click "Finish" to add the external library; Delete the last line *else:win32-g++:PRE_TARGETDEPS+=$$PWD/../htra_api/libhtra_api.a* in the "test.pro" file. After saving, you can write project code normally;




```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    mainwindow.h

FORMS += \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

win32: LIBS += -L$$PWD/../htra_api/ -lhtra_api

INCLUDEPATH += $$PWD/../htra_api
DEPENDPATH += $$PWD/../htra_api

win32:!win32-g++: PRE_TARGETDEPS += $$PWD/../htra_api/htra_api.lib
else:win32-g++: PRE_TARGETDEPS += $$PWD/../htra_api/libhtra_api.a
```

Figure 13 Modify the test.pro file

12. After writing the code, click "Run". The program running correctly is illustrated as follows.




```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>

#include <htra_api.h>
#include <stdio.h>
#include <string.h>
#include <vector>

using namespace std;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    int Status = 0;
    void* Device = NULL;
    int DevNum = 0;

    BootProfile_TypeDef BootProfile;
    BootInfo_TypeDef BootInfo;

    BootProfile.DevicePowerSupply = USBPortAndPowerPort;

    BootProfile.PhysicalInterface = USB;

    Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
    if (Status == APIRETVAL_NoError) {
        qDebug() << __FUNCTION__ << __LINE__ << "Device opened successful!";
    } else {
```

```
16:47:07: Starting C:\Users\60536\Desktop\QtTest\Test\build\Desktop_Qt_5_15_2_MSVC2019_64bit-Debug\debug\Test.exe...
MainWindow::MainWindow 32 Device opened successful!
MainWindow::MainWindow 53 Configuration successful!
MainWindow::MainWindow 82 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -78.6547
MainWindow::MainWindow 82 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -83.8589
```

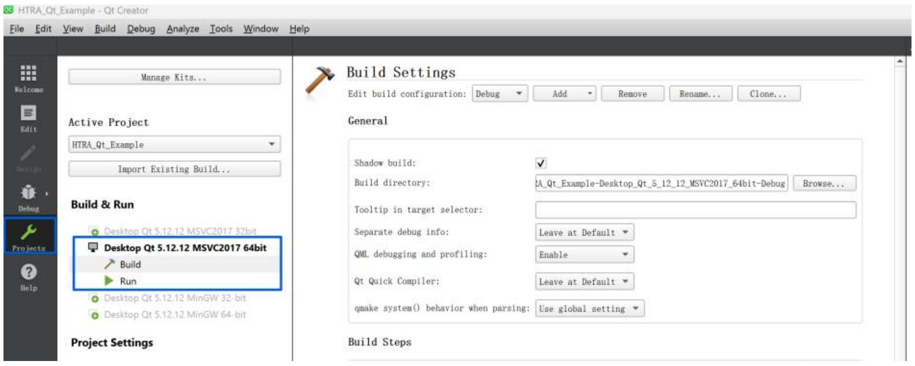
Figure 14 Result

2.2 Qt Example Usage Process

Note: Only one example can run at a time; the example and SAStudio4 cannot run simultaneously.

All Qt examples on the provided USB drive follow the same usage process. The following introduces the usage of the HTRA_Qt_Example project as an example:

1. Use Qt Creator to open the htrademo.pro file located in the Windows\HTRA_API_Example\HTRA_Qt_Examples\htrademo folder on the USB drive;
2. Click "Projects" to configure a 64-bit build environment for the project (the provided example uses 64-bit library files. If you need to use a 32-bit build environment, replace the libraries in the "HTRA_Qt_Example\htra_api" folder with the 32-bit libraries from "\Windows\HTRA_API\x86");

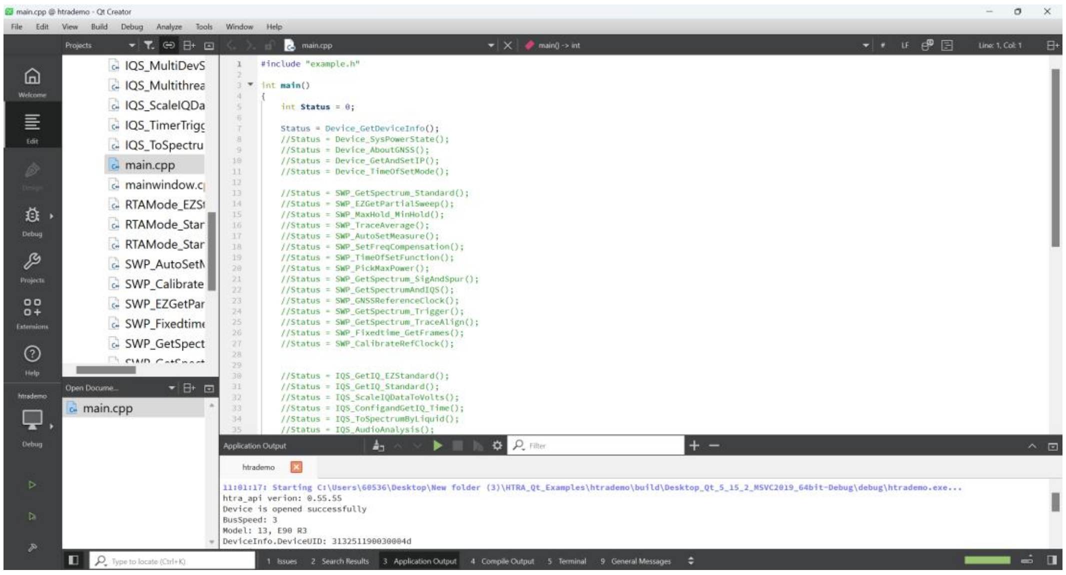

HTRA_Qt_Example - Qt Creator
File Edit View Build Debug Analyze Tools Window Help
Welcome
Edit
Design
Debug
Projects
Help
Manage Kits...
Active Project
HTRA_Qt_Example
Import Existing Build...
Build & Run
Desktop Qt 5.12.12 MSVC2017 32bit
Desktop Qt 5.12.12 MSVC2017 64bit
Build
Run
Desktop Qt 5.12.12 MinGW 32-bit
Desktop Qt 5.12.12 MinGW 64-bit
Project Settings
Build Settings
Add
Remove
Rename...
Clone...
General
Browse...
Build Steps

Edit build configuration: Debug

Shadow build: ✓
Build directory: A_Qt_Example-Desktop_Qt_5_12_12_MSVC2017_64bit-Debug
Tooltip in target selector:
Separate debug info: Leave at Default
QML debugging and profiling: Enable
Qt Quick Compiler: Leave at Default
qmake system() behavior when parsing: Use global setting

Figure 15 Configure a 64-bit build environment

3. After configuring the environment, click "Edit" to open main.cpp in the Sources folder of the "HTRA_Qt_Example" project. Uncomment the relevant functions, save the file, and click "Run" to execute different examples.



```
#include "example.h"

int main()
{
    int Status = 0;

    Status = Device_GetDeviceInfo();
    //Status = Device_SysPowerState();
    //Status = Device_AboutGNSS();
    //Status = Device_GetAndSetIP();
    //Status = Device_TimeOfSetMode();

    //Status = SWP_GetSpectrum_Standard();
    //Status = SWP_EZGetPartialSweep();
    //Status = SWP_MaxHold_MinHold();
    //Status = SWP_TraceAverage();
    //Status = SWP_AutoSetMeasure();
    //Status = SWP_SetFreqCompensation();
    //Status = SWP_TimeOfSetFunction();
    //Status = SWP_PickMaxPower();
    //Status = SWP_GetSpectrum_SigAndSpur();
    //Status = SWP_GetSpectrumAndIQS();
    //Status = SWP_GNSSReferenceClock();
    //Status = SWP_GetSpectrum_Trigger();
    //Status = SWP_GetSpectrum_TraceAlign();
    //Status = SWP_Fixedtime_GetFrames();
    //Status = SWP_CalibrateRefClock();


    //Status = IQS_GetIQ_EZStandard();
    //Status = IQS_GetIQ_Standard();
    //Status = IQS_ScaleIQDataToVolts();
    //Status = IQS_ConfigandGetIQ_Time();
    //Status = IQS_ToSpectrumByLiquid();
    //Status = IQS_AudioAnalysis();
```

```
11:01:17: Starting C:\Users\60536\Desktop\New folder (3)\HTRA_Qt_Examples\htrademo\build\Desktop_Qt_5_15_2_MSVC2019_64bit-Debug\debug\htrademo.exe...
htra_api verion: 0.55.55
Device is opened successfully
BusSpeed: 3
Model: 13, E90 R3
DeviceInfo.DeviceUID: 313251190030004d
```

Figure 16 Running the QT Example

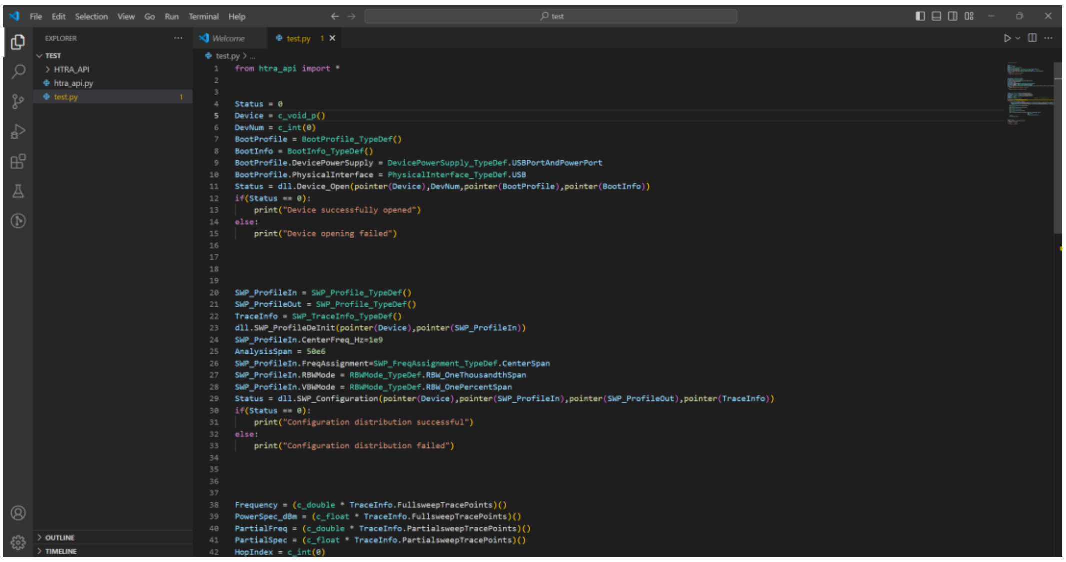
2.3 Qt Example Description

The Qt examples on the USB stick include a general example (HTRA_Qt_Example), an FM and AM demodulation example (HTRA_Demodulation), and an IQ to Spectrum example (IQS_ToSpectrumBuLiquid) using the Liquid library, refer to the C/C++ examples chapter for specific examples.

3. Python

3.1 Configure Development Environment

1. Create a new folder on the desktop (e.g., TEST);
2. Open the provided USB drive, go to "\Windows\HTRA_API_Example\HTRA_Python_Examples", and copy the "HTRA_API" folder and "htra_api.py" file into the newly created test folder;
3. Open Visual Studio Code, click "File" → "Open Folder", and select the newly created test folder;
4. In the left Explorer panel, click the "New File" icon to create a new Python file (e.g., test.py), and you can start writing code normally.

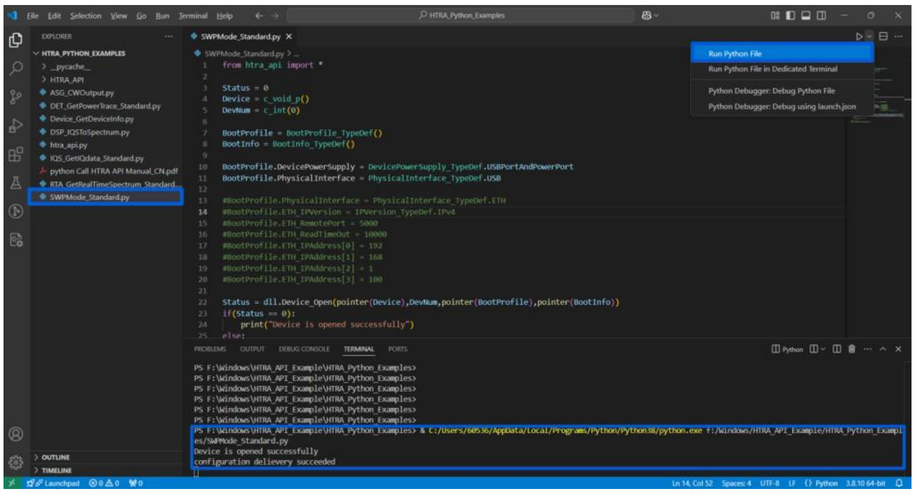


```python
from htra_api import *


Status = 0
Device = c_void_p()
DevNum = c_int(0)
BootProfile = BootProfile_TypeDef()
BootInfo = BootInfo_TypeDef()
BootProfile.DevicePowerSupply = DevicePowerSupply_TypeDef.USBPortAndPowerPort
BootProfile.PhysicalInterface = PhysicalInterface_TypeDef.USB
Status = dll.Device_Open(pointer(Device),DevNum,pointer(BootProfile),pointer(BootInfo))
if(Status == 0):
    print("Device successfully opened")
else:
    print("Device opening failed")




SWP_ProfileIn = SWP_Profile_TypeDef()
SWP_ProfileOut = SWP_Profile_TypeDef()
TraceInfo = SWP_TraceInfo_TypeDef()
dll.SWP_ProfileDeInit(pointer(Device),pointer(SWP_ProfileIn))
SWP_ProfileIn.CenterFreq_Hz=1e9
AnalysisSpan = 50e6
SWP_ProfileIn.FreqAssignment=SWP_FreqAssignment_TypeDef.CenterSpan
SWP_ProfileIn.RBWMode = RBWMode_TypeDef.RBW_OneThousandthSpan
SWP_ProfileIn.VBWMode = RBWMode_TypeDef.RBW_OnePercentSpan
Status = dll.SWP_Configuration(pointer(Device),pointer(SWP_ProfileIn),pointer(SWP_ProfileOut),pointer(TraceInfo))
if(Status == 0):
    print("Configuration distribution successful")
else:
    print("Configuration distribution failed")




Frequency = (c_double * TraceInfo.FullsweepTracePoints)()
PowerSpec_dBm = (c_float * TraceInfo.FullsweepTracePoints)()
PartialFreq = (c_double * TraceInfo.PartialsweepTracePoints)()
PartialSpec = (c_float * TraceInfo.PartialsweepTracePoints)()
HopIndex = c_int(0)
```

Figure 17 Creating a New Python Project

3.2 Python Example Usage Process

1. Open the "Windows\HTRA_API_Example\HTRA_Python_Examples" project folder on the provided USB drive using Visual Studio Code or another IDE. The "htra_api.py" file is the Python mapping for the dynamic library, and the other files are example programs;
2. After properly configuring the Python environment, select any example program (e.g., SWPMode_Standard.py) and run the file directly. The program running successfully is illustrated as follows:



```python
from htra_api import *

Status = 0
Device = c_void_p()
DevNum = c_int(0)

BootProfile = BootProfile_TypeDef()
BootInfo = BootInfo_TypeDef()

BootProfile.DevicePowerSupply = DevicePowerSupply_TypeDef.USBPortAndPowerPort
BootProfile.PhysicalInterface = PhysicalInterface_TypeDef.USB

#BootProfile.PhysicalInterface = PhysicalInterface_TypeDef.ETH
#BootProfile.ETH_IPVersion = IPVersion_TypeDef.IPv4
#BootProfile.ETH_RemotePort = 5000
#BootProfile.ETH_ReadTimeOut = 10000
#BootProfile.ETH_IPAddress[0] = 192
#BootProfile.ETH_IPAddress[1] = 168
#BootProfile.ETH_IPAddress[2] = 1
#BootProfile.ETH_IPAddress[3] = 100

Status = dll.Device_Open(pointer(Device),DevNum,pointer(BootProfile),pointer(BootInfo))
if(Status == 0):
    print("Device is opened successfully")
else:
```

```
PS F:\Windows\HTRA_API_Example\HTRA_Python_Examples>
PS F:\Windows\HTRA_API_Example\HTRA_Python_Examples>
PS F:\Windows\HTRA_API_Example\HTRA_Python_Examples>
PS F:\Windows\HTRA_API_Example\HTRA_Python_Examples>
PS F:\Windows\HTRA_API_Example\HTRA_Python_Examples>
PS F:\Windows\HTRA_API_Example\HTRA_Python_Examples>
PS F:\Windows\HTRA_API_Example\HTRA_Python_Examples> & C:/Users/60536/AppData/Local/Programs/Python/Python38/python.exe f:/Windows/HTRA_API_Example/HTRA_Python_Exampl
es/SWPMode_Standard.py
Device is opened successfully
configuration delievery succeeded
```

Figure 18 Running SWPMode_Standard.py

3.3 Python Example Description

3.3.1 Get device information

Device_GetDeviceInfo.py: Retrieves various device information, including API version, USB version, device model, device UID, MCU version, FPGA version, and device temperature.

3.3.2 Obtain Standard Spectrum Data

SWP_GetSpectrum_Standard.py: Obtains complete spectrum data within a specified frequency band(Plotting the spectrum if the host computer includes the matplotlib library).

3.3.3 Obtain IQ Data for a Fixed Number of Points or Duration

IQS_GetIQdata_Standard.py: Obtains IQ data under different trigger modes in IQS mode.

3.3.4 Obtain Power Detection Data for a Fixed Number of Points or Duration

DET_GetPowerTrace_Standard.py: Obtains power detection data under different trigger modes in DET mode(Plotting the Time-power graph if the host computer includes the matplotlib library).

3.3.5 Obtain real-time spectrum data for a fixed number of points or duration

RTA_GetRealTimeSpectrum_Standard.py: Obtain real-time spectrum data under different trigger modes in RTA mode.

3.3.6 IQ to Spectrum Data

DSP_IQSToSpectrum.py: Convert the IQ data obtained in IQS mode into spectrum data(If the host computer contains the matplotlib library then plot the spectrum and IQ time-domain graphs).

3.3.7 GNSS Related

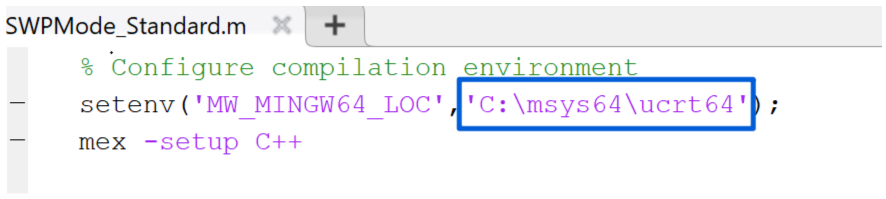
Device_AboutGNSS.py: Obtain information such as longitude, latitude, altitude, and time acquired by the GNSS module; obtain the longitude, latitude, and time information from the MeasAuxInfo structure in IQS mode.

4. Matlab

The following uses Matlab 2016a as an example to demonstrate how to call the 64-bit htra_api dynamic link library.

4.1 Running the Matlab Example

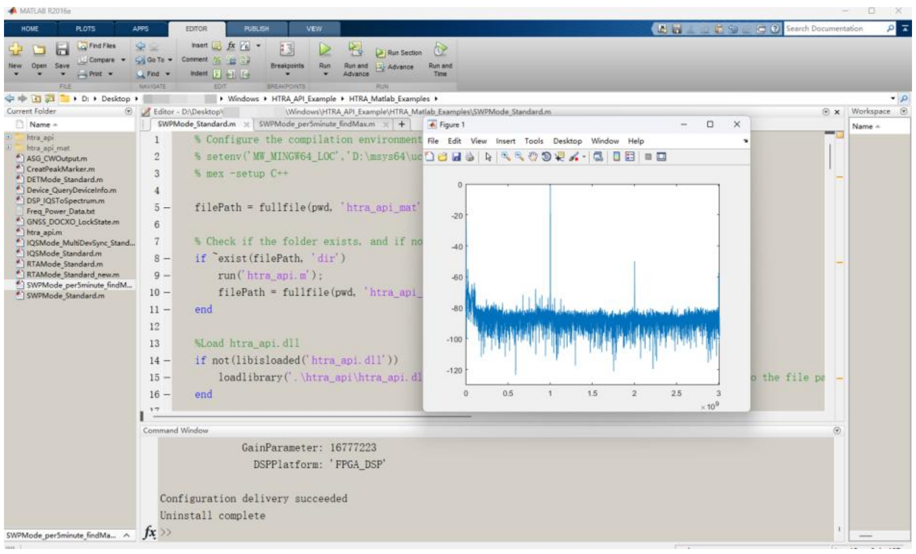
1. Download and install a C++ compiler;
2. After properly connecting the device, open Matlab. Copy the current path of the "Windows\HTRA_API_Example\HTRA_Matlab_Examples" folder from the provided USB drive into Matlab's file path box and press Enter;
3. Click the SWPMode_Standard.m file on the left, and confirm whether the compiler address in the first line of the example is consistent with the compiler path installed on the local system;



```
SWPMode_Standard.m
% Configure compilation environment
setenv('MW_MINGW64_LOC','C:\msys64\ucrt64');
mex -setup C++
```

Figure 19 Confirm the compilatopm address

4. Click "Run". The appearance of the Figure 1 window indicates successful execution of the example. For functional descriptions of each example, please refer to the Introduction to Accompanying Examples section in the accompanying materials.



```
% Configure the compilation environment
% setenv('MW_MINGW64_LOC','D:\msys64\uc
% mex -setup C++

filePath = fullfile(pwd, 'htra_api_mat'

% Check if the folder exists, and if no
if ~exist(filePath, 'dir')
    run('htra_api.m');
    filePath = fullfile(pwd, 'htra_api_
end

%Load htra_api.dll
if not(libisloaded('htra_api.dll'))
    loadlibrary('.\htra_api\htra_api.dl
end
```

```
              GainParameter: 16777223
                DSPPlatform: 'FPGA_DSP'

Configuration delivery succeeded
Uninstall complete
```

Figure 20 Run the SWPMode_Standard.m File

Note: Since calling the dynamic link library requires a C++ compiler, it is recommended to use MSYS2 to install the g++ compiler. Refer to https://www.msys2.org/ for download and installation instructions.

4.2 Introduction to Accompanying Examples

4.2.1 Get device information

Device_QueryDeviceInfo.m: Retrieve device information, including API version, USB version, device model, device UID, MCU version, FPGA version, and device temperature.

4.2.2 Obtain Standard Spectrum Data

SWPMode_Standard.m: Obtain complete spectrum data within the specified frequency band.

4.2.3 Create multiple markers to display frequency and power of the markers

CreatPeakMarker.m: Obtain spectrum data within the specified frequency band, create markers, and perform peak searching.

4.2.4 Collect the peak spectrum every five minutes

SWPMode_per5minute_findMax.m: Obtain spectrum data within the specified frequency band and search for the peak globally every five minutes.

4.2.5 Obtain continuous stream or fixed number of IQ data

IQSMode_Standard.m: Acquire IQ data under different trigger modes in IQS mode.

4.2.6 The acquired IQ data is converted into spectrum data

DSP_IQSToSpectrum.m: After acquiring IQ data, the obtained IQ data is converted into spectrum data.

4.2.7 Acquire continuous stream or fixed number of power detection data

DET_GetPowerTrace_FixedPoints.m: Acquire power detection data under different triggering modes in DET mode.

4.2.8 Acquire continuous stream or fixed duration real-time spectrum data

RTAMode_FixedPoints.m: Acquire real-time spectrum data under different triggering modes in RTA mode.

4.2.9 Internal signal source output signal

ASG_CWOutput.m: Output single-tone signals, frequency sweep signals, and power sweep signals. Applicable only to devices with signal source options.

4.2.10 Lock GNSS antenna and DOCXO oscillator

GNSS_DOCXO_LockState.m: Call the API interface to lock the GNSS antenna and DOCXO oscillator, applicable only to devices with IO expansion board options.

4.2.11 Multi-machine synchronization

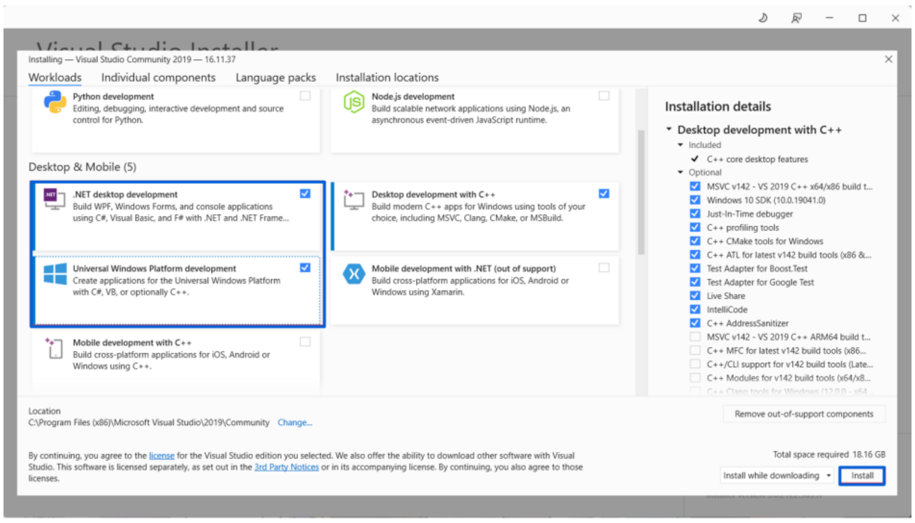
IQSMode_MultiDevSync_Standard.m: When using the same reference clock source input and the same trigger source input, two devices simultaneously acquire IQ data, allowing for the observation of the synchronization of their collected data.

5. C#

5.1 Configure Development Environment

5.1.1 Development Environment Confirmation

Open the Visual Studio Installer, select the .NET Desktop Development and Universal Windows Platform Development components, then click Modify to ensure that Visual Studio 2019 includes the C# development environment.



Installing — Visual Studio Community 2019 — 16.11.37

Python development ☐
Editing, debugging, interactive development and source control for Python.

Node.js development ☐
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

.NET desktop development ☑
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...

Desktop development with C++ ☑
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.

Universal Windows Platform development ☑
Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

Mobile development with .NET (out of support) ☐
Build cross-platform applications for iOS, Android or Windows using Xamarin.

Mobile development with C++ ☐
Build cross-platform applications for iOS, Android or Windows using C++.

Installation details

Desktop development with C++
- Included
  - ✓ C++ core desktop features
- Optional
  - ☑ MSVC v142 - VS 2019 C++ x64/x86 build t...
  - ☑ Windows 10 SDK (10.0.19041.0)
  - ☑ Just-In-Time debugger
  - ☑ C++ profiling tools
  - ☑ C++ CMake tools for Windows
  - ☑ C++ ATL for latest v142 build tools (x86 &...
  - ☑ Test Adapter for Boost.Test
  - ☑ Test Adapter for Google Test
  - ☑ Live Share
  - ☑ IntelliCode
  - ☑ C++ AddressSanitizer
  - ☐ MSVC v142 - VS 2019 C++ ARM64 build t...
  - ☐ C++ MFC for latest v142 build tools (x86...
  - ☐ C++/CLI support for v142 build tools (Late...
  - ☐ C++ Modules for v142 build tools (x64/x8...
  - ☐ C++ Clang tools for Windows (12.0.0 - x64...

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

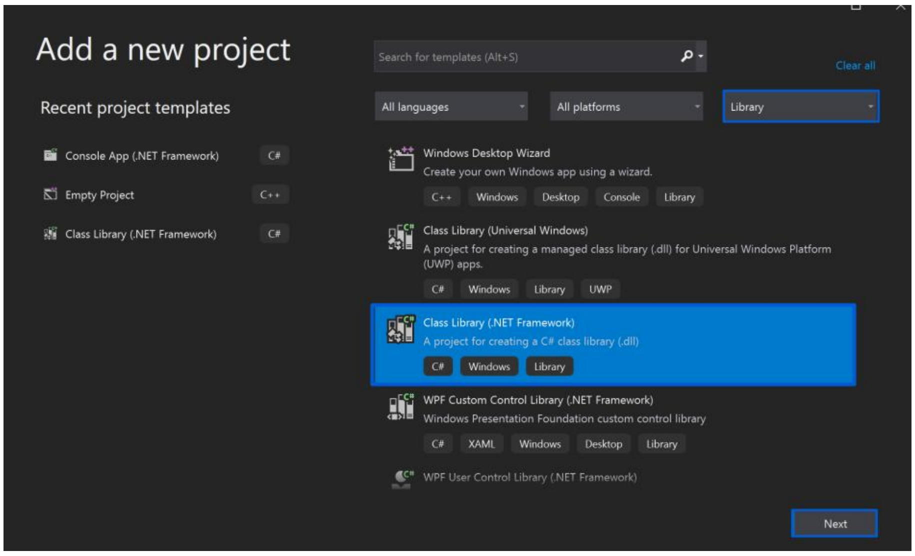
By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 18.16 GB

Figure 21 Configuring the C# Development Environment in Visual Studio

5.1.2 Project Setup

1. Open Visual Studio 2019 and click "Create a new project";
2. Select "C# Console Application" and click "Next";
3. Enter the project name and location, uncheck "Place solution and project in the same directory." Select ".NET Framework 4.5" as the framework, and then click "Create";
4. After the project is created, open the project, right-click the solution, and select "Add New Project";
5. In the "Add New Project" dialog, select "Library" as the project type, then choose "Class Library (Universal Windows)" under C#, and click "Next";


Add a new project

Recent project templates

Console App (.NET Framework) C#

Empty Project C++

Class Library (.NET Framework) C#

Search for templates (Alt+S)

All languages | All platforms | Library

Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library

Class Library (Universal Windows)
A project for creating a managed class library (.dll) for Universal Windows Platform (UWP) apps.
C# Windows Library UWP

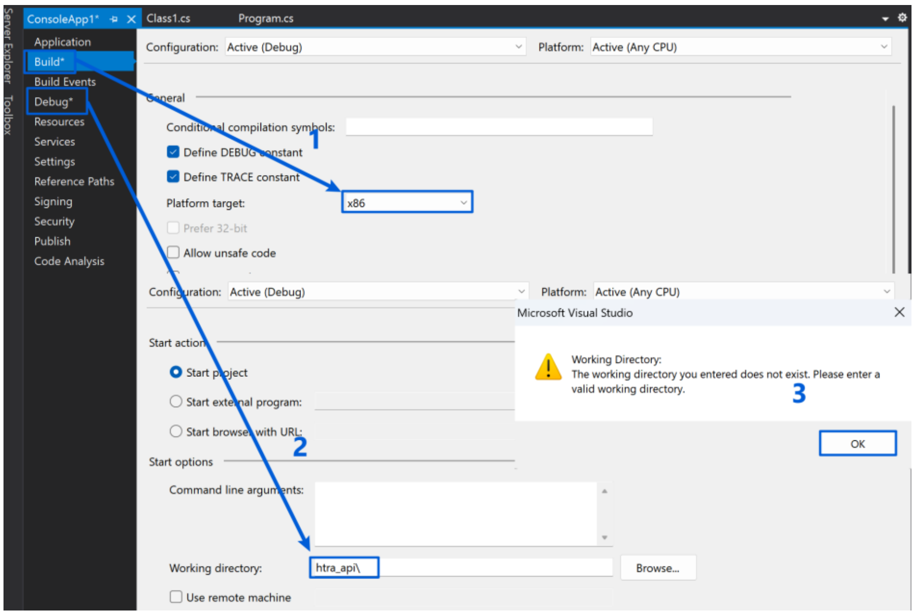
Class Library (.NET Framework)
A project for creating a C# class library (.dll)
C# Windows Library

WPF Custom Control Library (.NET Framework)
Windows Presentation Foundation custom control library
C# XAML Windows Desktop Library

WPF User Control Library (.NET Framework)

Figure 22 Add New Project

6. Modify the project name as needed (e.g., HtraApi), keep the default location, select ".NET Framework 4.5" as the framework, and click "Create";
7. Right-click ConsoleApp1 and select "Properties". In the project properties window, click the "Build" tab and change the "Platform target" to "x86". Then click the "Debug" tab, enter "htra_api" in the "Working directory" field, click "OK" in the popup, and finally save and close the properties window;


Configuration: Active (Debug)   Platform: Active (Any CPU)

General

Conditional compilation symbols:

☑ Define DEBUG constant

☑ Define TRACE constant

Platform target: x86

☐ Prefer 32-bit

☐ Allow unsafe code

Start action

● Start project

○ Start external program:

○ Start browser with URL:

Start options

Command line arguments:

Working directory: htra_api\

☐ Use remote machine

Microsoft Visual Studio

Working Directory:
The working directory you entered does not exist. Please enter a valid working directory.

Figure 23 Configure Project Properties

8. Right-click the library HtraApi and select "Properties". In the library properties window, click the "Build" tab, change the "Platform target" to "x86," and save;
9. Copy the contents of the HtraApi.cs file from the "\Windows\HTRA_API_Example\HTRA_CSharp_Examples" folder on the provided USB drive into the Class1.cs file in the project library, replace the code, and save;

10. Select the ConsoleApp1 project, right-click "References," choose "Add Reference," check the HtraApi library in the project, click "OK," and confirm that the library reference has been added;

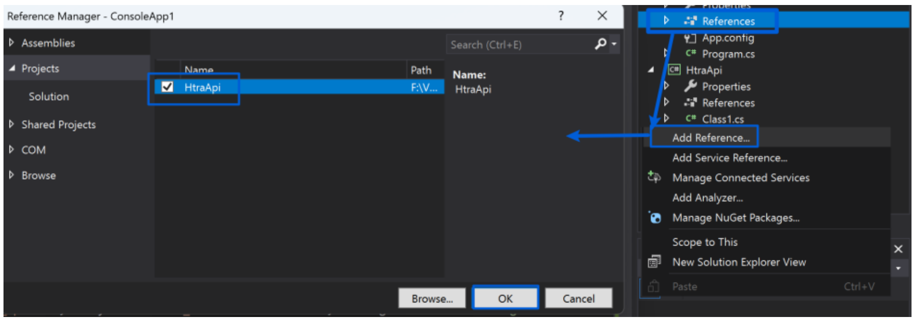


Figure 24 Add Library Reference

11. Copy the htra_api folder from the "\Windows\HTRA_API\x86" directory on the provided USB drive to the "bin\Debug" folder of the project, and ensure that the "Htra_api\CalFile" folder contains the calibration files for the instruments being used;

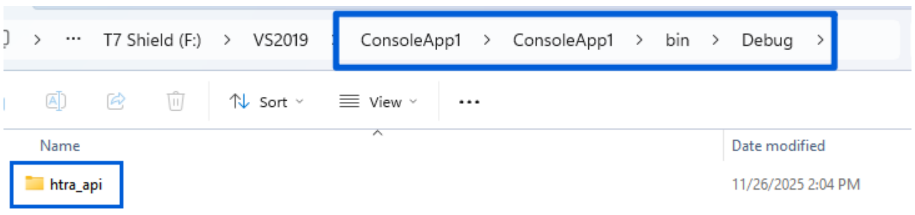
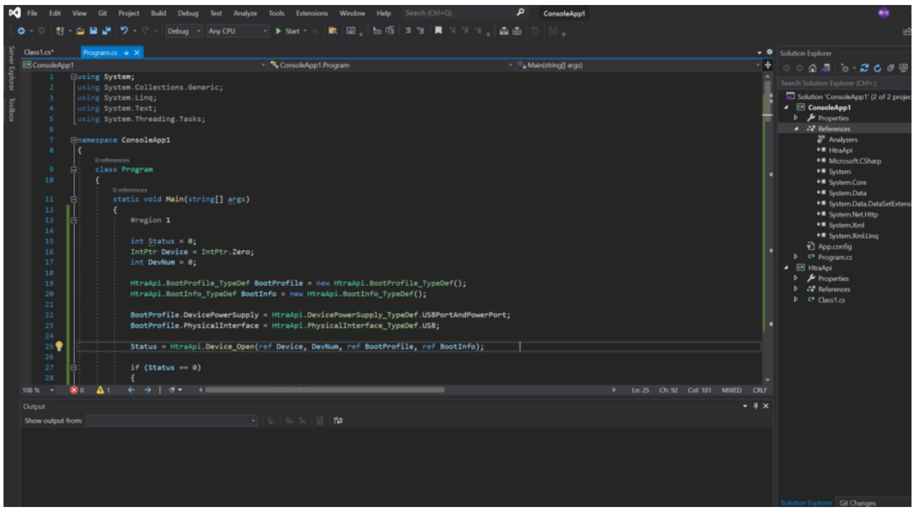


Figure 25 Copy the htra_api file

12. You can refer to the C# examples on the provided USB drive and write code as needed;

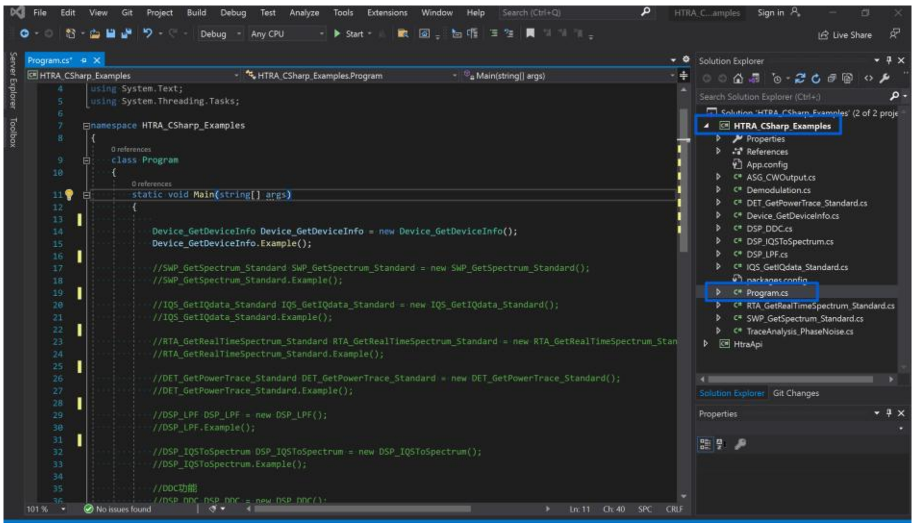


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            #region 1

            int Status = 0;
            IntPtr Device = IntPtr.Zero;
            int DevNum = 0;

            HtraApi.BootProfile_TypeDef BootProfile = new HtraApi.BootProfile_TypeDef();
            HtraApi.BootInfo_TypeDef BootInfo = new HtraApi.BootInfo_TypeDef();

            BootProfile.DevicePowerSupply = HtraApi.DevicePowerSupply_TypeDef.USBPortAndPowerPort;
            BootProfile.PhysicalInterface = HtraApi.PhysicalInterface_TypeDef.USB;

            Status = HtraApi.Device_Open(ref Device, DevNum, ref BootProfile, ref BootInfo);

            if (Status == 0)
            {
```

Figure 26 Write C# example

5.2 C# Example Usage Process

1. Open the solution HTRA_CSharp_Examples.sln in Visual Studio from the "\Windows\HTRA_API_Example\HTRA_CSharp_Examples" folder on the provided USB drive;
2. Click the HTRA_CSharp_Examples project on the right, then open the Program.cs file within it;

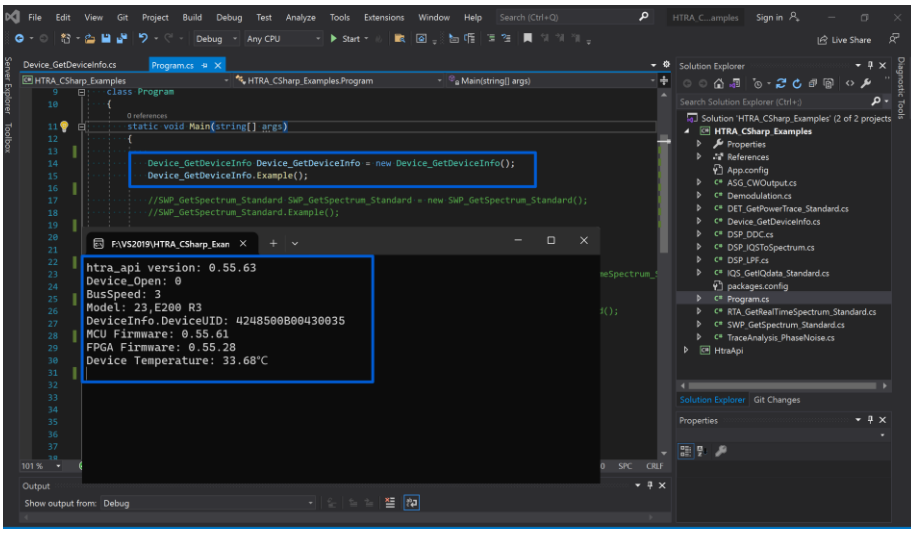


```
using System.Text;
using System.Threading.Tasks;

namespace HTRA_CSharp_Examples
{
    class Program
    {
        static void Main(string[] args)
        {

            Device_GetDeviceInfo Device_GetDeviceInfo = new Device_GetDeviceInfo();
            Device_GetDeviceInfo.Example();

            //SWP_GetSpectrum_Standard SWP_GetSpectrum_Standard = new SWP_GetSpectrum_Standard();
            //SWP_GetSpectrum_Standard.Example();

            //IQS_GetIQdata_Standard IQS_GetIQdata_Standard = new IQS_GetIQdata_Standard();
            //IQS_GetIQdata_Standard.Example();

            //RTA_GetRealTimeSpectrum_Standard RTA_GetRealTimeSpectrum_Standard = new RTA_GetRealTimeSpectrum_Stan
            //RTA_GetRealTimeSpectrum_Standard.Example();

            //DET_GetPowerTrace_Standard DET_GetPowerTrace_Standard = new DET_GetPowerTrace_Standard();
            //DET_GetPowerTrace_Standard.Example();

            //DSP_LPF DSP_LPF = new DSP_LPF();
            //DSP_LPF.Example();

            //DSP_IQSToSpectrum DSP_IQSToSpectrum = new DSP_IQSToSpectrum();
            //DSP_IQSToSpectrum.Example();

            //DDC功能
```

Figure 27 Open the Programe.cs file

3. Each routine in the provided C# examples is encapsulated in a separate class. To use a routine, simply uncomment it (multiple examples cannot be used simultaneously).

   For example, to use the Device_GetDeviceInfo routine, uncomment it and run the program. A successful run is illustrated as follows;



```
            Device_GetDeviceInfo Device_GetDeviceInfo = new Device_GetDeviceInfo();
            Device_GetDeviceInfo.Example();

            //SWP_GetSpectrum_Standard SWP_GetSpectrum_Standard = new SWP_GetSpectrum_Standard();
            //SWP_GetSpectrum_Standard.Example();
```

```
htra_api version: 0.55.63
Device_Open: 0
BusSpeed: 3
Model: 23,E200 R3
DeviceInfo.DeviceUID: 4248500B00430035
MCU Firmware: 0.55.61
FPGA Firmware: 0.55.28
Device Temperature: 33.68°C
```

Figure 28 Run Device_GetDeviceInfo.cs

5.3 C# Example Descriptions

5.3.1 Get device information

Device_GetDeviceInfo.cs: Retrieves device information including API version, USB version, device model, device UID, MCU version, FPGA version, and device temperature.

5.3.2 Obtain Standard Spectrum Data

SWP_GetSpectrum_Standard.cs: Obtains complete spectrum data within a specified frequency band.

5.3.3 Obtain IQ Data for a Fixed Number of Points or Duration

IQS_GetIQdata_Standard.cs: Acquires IQ data under different trigger modes in IQS mode.

5.3.4 Obtain Power Detection Data for a Fixed Number of Points or Duration

DET_GetPowerTrace_Standard.cs: Obtains power detection data under different trigger modes in DET mode.

5.3.5 Obtain real-time spectrum data for a fixed number of points or duration

RTA_GetRealTimeSpectrum_Standard.cs: Retrieves real-time spectrum data under different trigger modes in RTA mode.

5.3.6 Output single-tone signal

ASG_CWOutput.cs: Devices with signal source functionality options output single-tone signals, frequency sweep signals, or power sweep signals through ASG functionality.

5.3.7 AM/FM Demodulation

Demodulation.cs: DSP_FMDemod performs FM demodulation and playback of the acquired IQ data. DSP_AMDemod performs AM demodulation and playback of the acquired IQ data.

5.3.8 IQ to Spectrum Data

DSP_IQSToSpectrum.cs: Converts the IQ data obtained in IQS mode into spectrum data.

5.3.9 Low-pass filtering

DSP_LPF.cs: Applies low-pass filtering to the acquired IQ data and converts it to spectrum.

5.3.10 Digital Downconversion

DSP_DDC.cs: Performs digital down-conversion on the acquired IQ data and converts it to spectrum.

5.3.11 Phase noise test

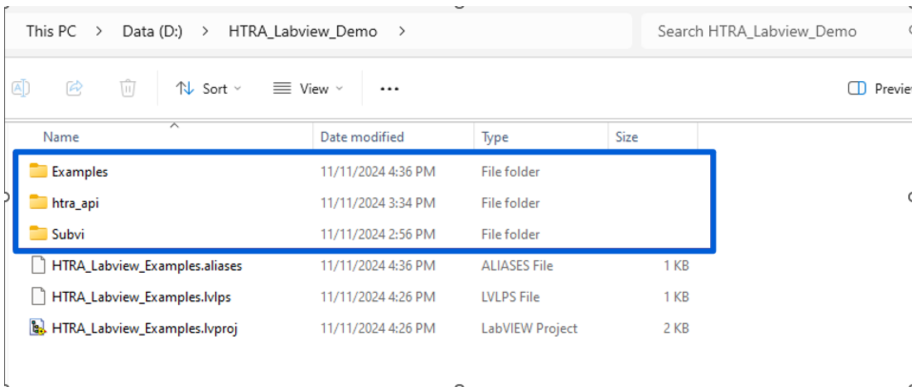
DSP_TraceAnalysis_PhaseNoise.cs: Demonstration of Phase Noise Test Function.

6. Labview

6.1 Configure Development Environment

6.1.1 Using the API in the LabVIEW Environment

1. Open LabVIEW, click "Create Project", select "Project", and click "Finish". An untitled empty project will appear. Press "Ctrl+S" to save the project, choose the save path, enter a project name (e.g., HTRA_Labview_Examples), and click "OK";
2. Copy the "htra_api" folder from the "\Windows\HTRA_API_Example\HTRA_Labview_Examples" directory on the USB drive to the same directory level as the project. Additionally, you may create an "Examples" folder to store example VIs. If needed, you can also create another folder such as "Subvi" to store sub-VIs;


This PC > Data (D:) > HTRA_Labview_Demo >

Name	Date modified	Type	Size
Examples	11/11/2024 4:36 PM	File folder	
htra_api	11/11/2024 3:34 PM	File folder	
Subvi	11/11/2024 2:56 PM	File folder	
HTRA_Labview_Examples.aliases	11/11/2024 4:36 PM	ALIASES File	1 KB
HTRA_Labview_Examples.lvlps	11/11/2024 4:26 PM	LVLPS File	1 KB
HTRA_Labview_Examples.lvproj	11/11/2024 4:26 PM	LabVIEW Project	2 KB

Figure 29 Copy htra_api

3. In the LabVIEW project, add the newly created Examples folder and the htra_api folder to the project, then save it;

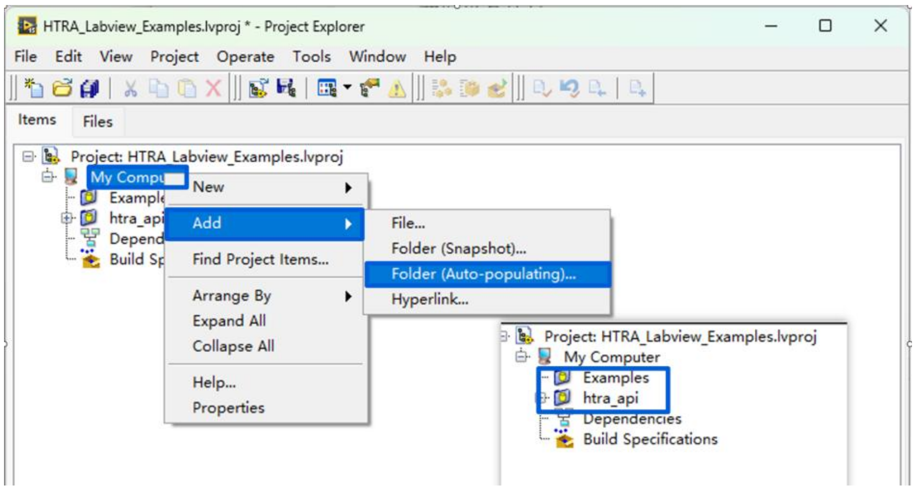

HTRA_Labview_Examples.lvproj * - Project Explorer
File Edit View Project Operate Tools Window Help
Items Files
Project: HTRA_Labview_Examples.lvproj
My Compu
New
Add
Find Project Items...
Arrange By
Expand All
Collapse All
Help...
Properties
File...
Folder (Snapshot)...
Folder (Auto-populating)...
Hyperlink...
Project: HTRA_Labview_Examples.lvproj
My Computer
Examples
htra_api
Dependencies
Build Specifications

Figure 30 Add to project

4. Create a new VI in the Examples folder. In the block diagram, you can call the exported

LabVIEW API functions. The calling process is the same as in the C environment;

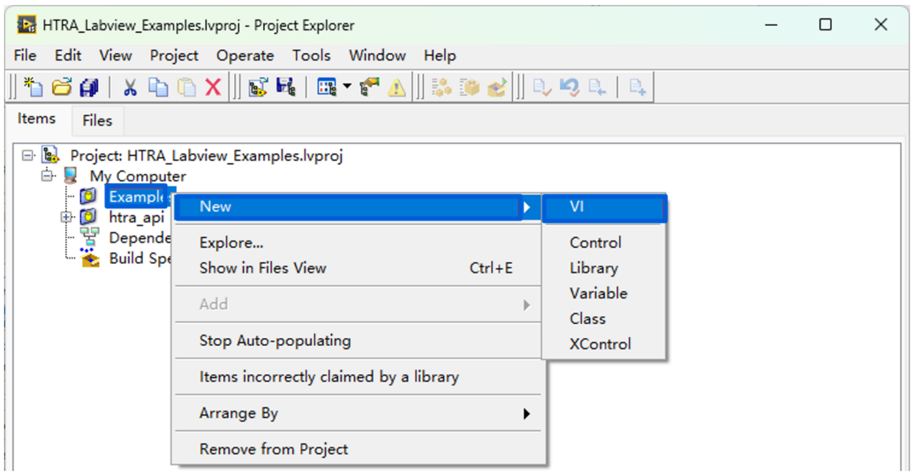

HTRA_Labview_Examples.lvproj - Project Explorer
File Edit View Project Operate Tools Window Help
Items Files
Project: HTRA_Labview_Examples.lvproj
My Computer
Exampl
htra_api
Depende
Build Spe
New
Explore...
Show in Files View Ctrl+E
Add
Stop Auto-populating
Items incorrectly claimed by a library
Arrange By
Remove from Project
VI
Control
Library
Variable
Class
XControl

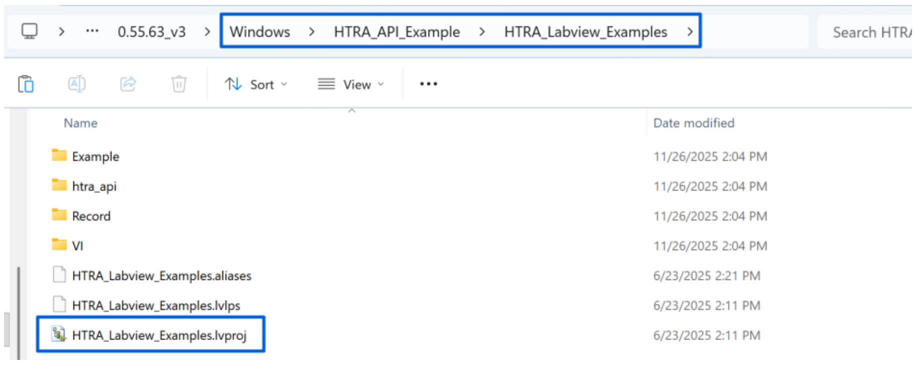
Figure 31 Create VI

5. Finally, save the VI to the Examples folder and rename it as needed.

6.2 Labview Example Usage Process

The usage process of Labview examples included in the USB drive is as follows:

1. Use LabVIEW to open the project "HTRA_Labview_Examples.lvproj" located in the USB drive under Windows\HTRA_API_Example\HTRA_Labview_Examples;



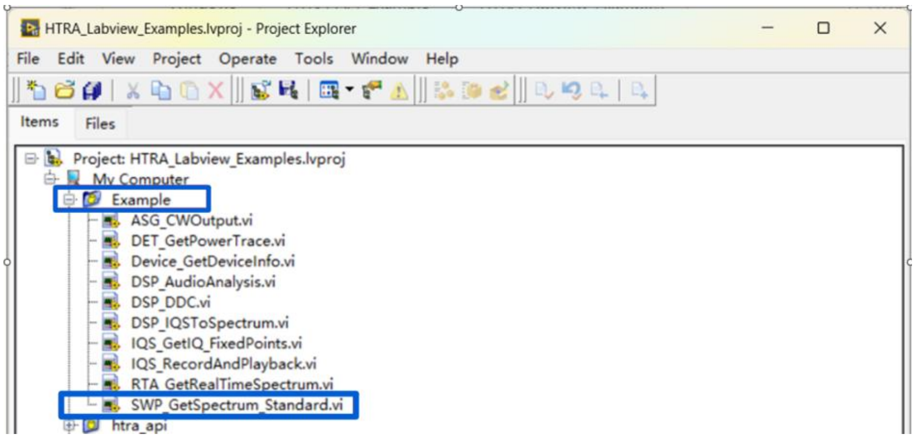
0.55.63_v3 > Windows > HTRA_API_Example > HTRA_Labview_Examples > Search HTR

Sort View

Name	Date modified
Example	11/26/2025 2:04 PM
htra_api	11/26/2025 2:04 PM
Record	11/26/2025 2:04 PM
VI	11/26/2025 2:04 PM
HTRA_Labview_Examples.aliases	6/23/2025 2:21 PM
HTRA_Labview_Examples.lvlps	6/23/2025 2:11 PM
HTRA_Labview_Examples.lvproj	6/23/2025 2:11 PM

Figure 32 Open Labview Project

2. Double-click any example VI in the Example folder. Here, SWP_GetSpectrum_Standard.vi is used as an example;



HTRA_Labview_Examples.lvproj - Project Explorer

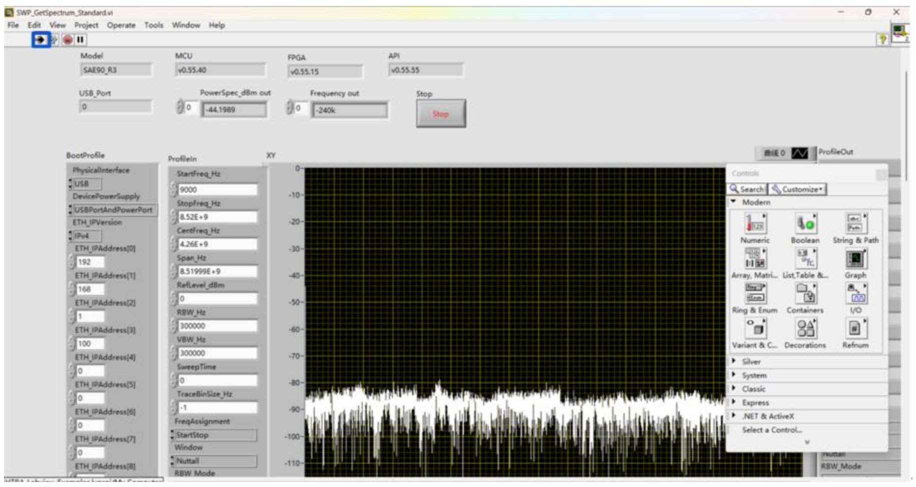
File Edit View Project Operate Tools Window Help

Items Files

```
Project: HTRA_Labview_Examples.lvproj
  My Computer
    Example
      ASG_CWOutput.vi
      DET_GetPowerTrace.vi
      Device_GetDeviceInfo.vi
      DSP_AudioAnalysis.vi
      DSP_DDC.vi
      DSP_IQSToSpectrum.vi
      IQS_GetIQ_FixedPoints.vi
      IQS_RecordAndPlayback.vi
      RTA_GetRealTimeSpectrum.vi
      SWP_GetSpectrum_Standard.vi
    htra_api
```

Figure 33 Open example

3. After opening the VI, click the Run button in the upper-left corner to execute the program. The running interface is shown below;



Model: SAE90_R3
MCU: v0.55.40
FPGA: v0.55.15
API: v0.55.55
USB_Port: 0
PowerSpec_dBm out: 0 -44.1989
Frequency out: 0 -240k

BootProfile
PhysicalInterface: USB
DevicePowerSupply: USBPortAndPowerPort
ETH_IPVersion: IPv4
ETH_IPAddress[0]: 192
ETH_IPAddress[1]: 168
ETH_IPAddress[2]: 1
ETH_IPAddress[3]: 100
ETH_IPAddress[4]: 0
ETH_IPAddress[5]: 0
ETH_IPAddress[6]: 0
ETH_IPAddress[7]: 0
ETH_IPAddress[8]

ProfileIn
StartFreq_Hz: 9000
StopFreq_Hz: 8.52E+9
CentFreq_Hz: 4.26E+9
Span_Hz: 8.51999E+9
RefLevel_dBm: 0
RBW_Hz: 300000
VBW_Hz: 300000
SweepTime: 0
TraceBinSize_Hz: -1
FreqAssignment: StartStop
Window: Nuttall
RBW Mode

Figure 34 Successfully Run the SWP_GetSpectrum_Standard Example

6.3 Labview Example Description

6.3.1 Get device information

Device_GetDeviceInfo.vi: An example for obtaining various device information, including: API version, device model, device UID, MCU version, FPGA version, and device temperature.

6.3.2 Standard spectrum acquisition

SWP_GetSpectrum_Standard.vi: Obtains standard spectrum data within a specified frequency band and displays the image.

6.3.3 Obtain IQ Data for a Fixed Number of Points or Duration

IQS_GetIQ_FixedPoints.vi: Obtains IQ data with a fixed number of points. When the device receives a Bus trigger signal, it returns IQ data with a fixed number of points.

6.3.4 Streaming and reading IQ data

IQS_RecordAndPlayback.vi: Obtains IQ data, records the IQ data as a txt file, and plays back the recorded IQ data.

6.3.5 IQ to Spectrum Data

DSP_IQSToSpectrum.vi: Obtains IQ data and converts the acquired IQ data into spectrum data.

6.3.6 Digital Downconversion

DSP_DDC.vi: Performs digital downconversion on the IQ data stream and resamples to generate a sub-IQ stream for further spectrum analysis.

6.3.7 Audio Analysis

DSP_AudioAnalysis.vi: Analyzes audio voltage (V), audio frequency (Hz), signal-to-noise ratio (dB), and total harmonic distortion (%).

6.3.8 Obtain Power Detection Data for a Fixed Number of Points or Duration

DET_GetPowerTrace.vi: Obtain a fixed number of DET data points. When the device receives a Bus trigger signal, it returns a fixed number of DET data points.

6.3.9 Obtain real-time spectrum data for a fixed number of points or duration

RTA_GetRealTimeSpectrum.vi: Obtain a fixed number of RTA data points. When the device receives a Bus trigger signal, it returns a fixed number of RTA data points.

6.3.10 ASG Signal Source Output Signal

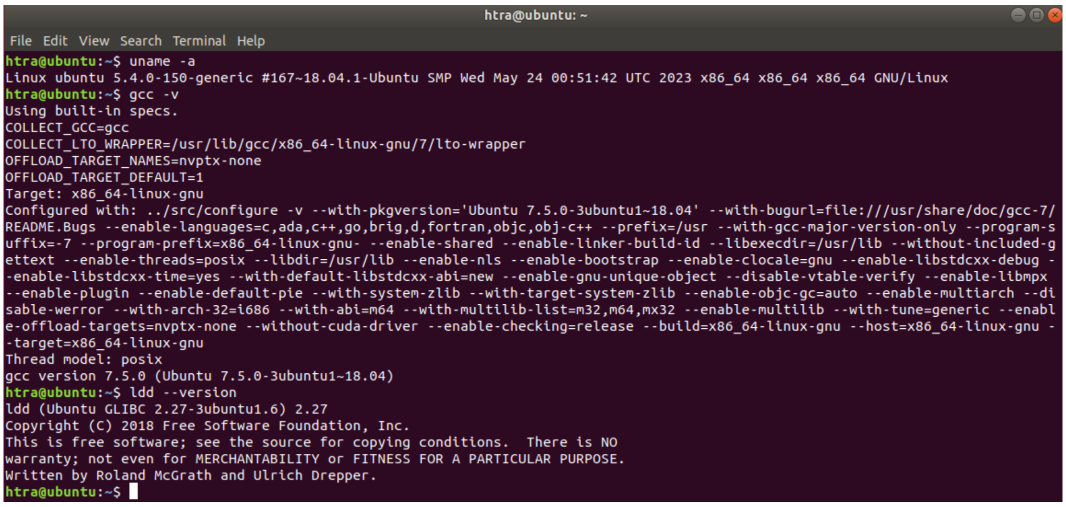
ASG_CWOutput.vi: Control the internal signal generator of the device to output single-tone signals, sweep signals, and power scan signals.

7. Linux

7.1 Environment Version Compatibility Self-Check

When using the device in a Linux system, you first need to confirm whether the current Linux environment's system architecture, gcc version, and GLIBC version are supported according to the following process:

1. Open the terminal and enter *uname -a* to check the Linux system architecture, for example, here the Linux system architecture is x86_64;
2. In the terminal, enter *gcc -v* to check the system gcc version, for example, here the gcc version is 7.5.0;
3. In the terminal, enter *ldd --version* to check the system GLIBC version, for example, here the GLIBC version is 2.27;



```
htra@ubuntu:~$ uname -a
Linux ubuntu 5.4.0-150-generic #167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
htra@ubuntu:~$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.5.0-3ubuntu1~18.04' --with-bugurl=file:///usr/share/doc/gcc-7/
README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only --program-s
uffix=-7 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-g
ettext --enable-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-clocale=gnu --enable-libstdcxx-debug -
-enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx
--enable-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib --enable-objc-gc=auto --enable-multiarch --di
sable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enabl
e-offload-targets=nvptx-none --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu -
-target=x86_64-linux-gnu
Thread model: posix
gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)
htra@ubuntu:~$ ldd --version
ldd (Ubuntu GLIBC 2.27-3ubuntu1.6) 2.27
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
htra@ubuntu:~$ 
```

Figure 35 Check the Linux System Architecture and Version

4. Compare the terminal information with the table to confirm whether the current environment is supported. If it is not supported, please contact technical support.

Table 1 Terminal Information Comparison Table

x86 processor	Support Intel and AMD processors
ARM processor	aarch64 (armv8), armv7 processors, such as: Raspberry Pi 4b, RK3399, RK3568, RK3588, T507, NVIDIA Jetson TX2
Compilation environment	gcc4.8, glib2.17 and above
Distribution	Customized system for Raspberry Pi 4b, Ubuntu 18.04, etc.

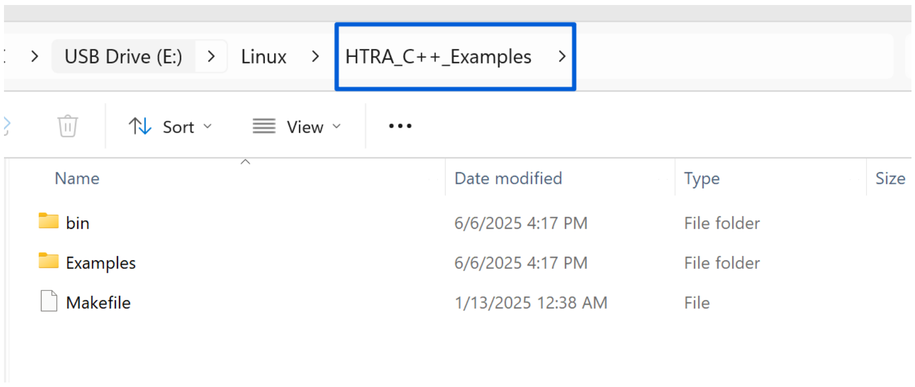
7.2 Accompanying documentation

Currently, the Linux section of the accompanying USB drive contains the following materials:

7.2.1 HTRA_C++_Examples

The HTRA_C++_Examples folder contains:

1. Examples folder: C++ example programs (see the C++ Example Usage section for instructions);
2. Makefile: Build script used to compile the example programs into executables;
3. bin folder: Stores device calibration files and the executables generated from building the example programs.



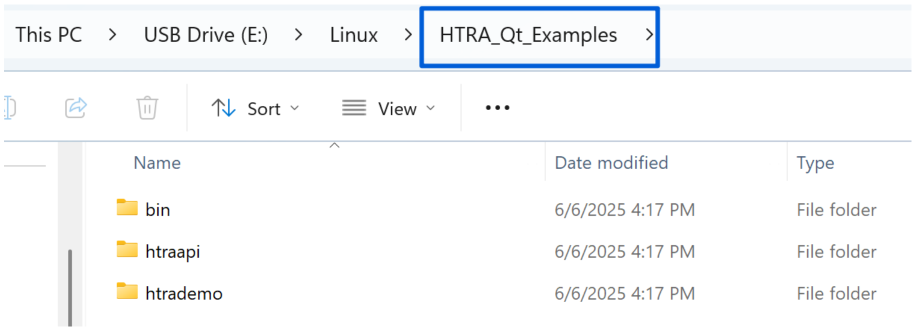
Name	Date modified	Type	Size
bin	6/6/2025 4:17 PM	File folder	
Examples	6/6/2025 4:17 PM	File folder	
Makefile	1/13/2025 12:38 AM	File	

Figure 36 Contents of the HTRA_C++_Examples Folder on Linux

7.2.2 HTRA_Qt_Examples

The HTRA_Qt_Examples folder contains:

1. htrademo folder: Qt examples and .pro files (see the Qt Example Usage section for instructions);
2. bin folder: Stores device calibration files and executables generated from compiling the example programs;
3. htraapi folder: Stores the dynamic link libraries.



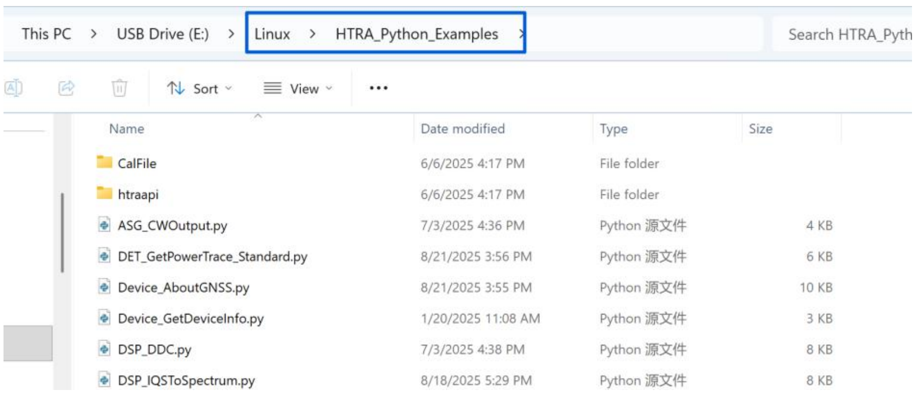
Name	Date modified	Type
bin	6/6/2025 4:17 PM	File folder
htraapi	6/6/2025 4:17 PM	File folder
htrademo	6/6/2025 4:17 PM	File folder

Figure 37 Contents of the HTRA_Qt_Examples Folder on Linux

7.2.3 HTRA_Python_Examples

The HTRA_Python_Examples folder specifically contains:

1. Python example programs (see the Python Example Usage section for instructions);
2. CalFile folder: Stores device calibration files;
3. Htraapi folder: Stores the dynamic link libraries.



Name	Date modified	Type	Size
CalFile	6/6/2025 4:17 PM	File folder	
htraapi	6/6/2025 4:17 PM	File folder	
ASG_CWOutput.py	7/3/2025 4:36 PM	Python 源文件	4 KB
DET_GetPowerTrace_Standard.py	8/21/2025 3:56 PM	Python 源文件	6 KB
Device_AboutGNSS.py	8/21/2025 3:55 PM	Python 源文件	10 KB
Device_GetDeviceInfo.py	1/20/2025 11:08 AM	Python 源文件	3 KB
DSP_DDC.py	7/3/2025 4:38 PM	Python 源文件	8 KB
DSP_IQSToSpectrum.py	8/18/2025 5:29 PM	Python 源文件	8 KB

Figure 38 Contents of the HTRA_Python_Examples folder in Linux

7.2.4 Install_HTRA_SDK

1. The Install_HTRA_SDK folder contains:
   - install_htraapi_lib.sh: Driver configuration script
   - htraapi folder: Stores drivers, header files, and libraries
2. The Install_HTRA_SDK/htraapi folder contains:
   - configs folder: Driver configuration files
   - inc folder: Header files
   - lib folder: Dynamic link libraries
3. The Install_HTRA_SDK/htraapi/lib folder contains:
   - arrch64 folder: ynamic link libraries for the arrch64 architecture
   - arrch64_gcc7.5 folder: Dynamic link libraries for arrch64 with more efficient FFT (requires GCC version above 7.5)
   - x86_64 folder: Dynamic link libraries for the x86_64 architecture
   - x86_64_gcc5.4 folder: Dynamic link libraries for x86_64 with more efficient FFT (requires GCC version above 5.4)
   - armv7 folder: Dynamic link libraries for the armv7 architecture

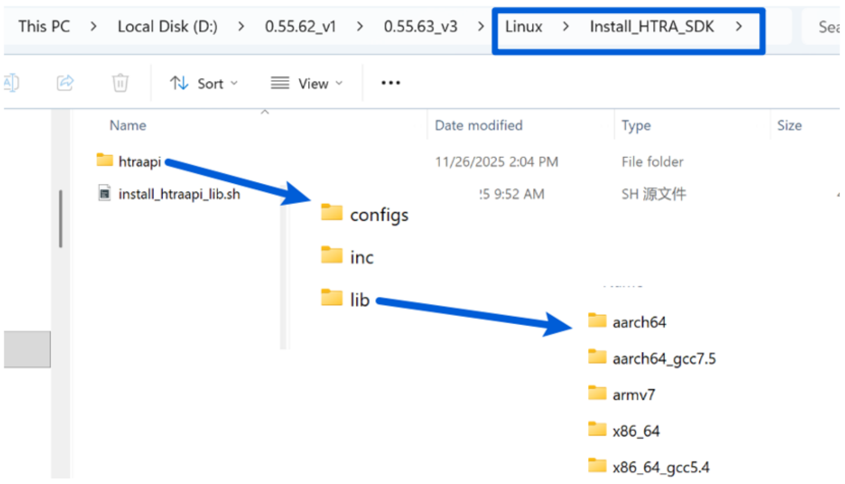


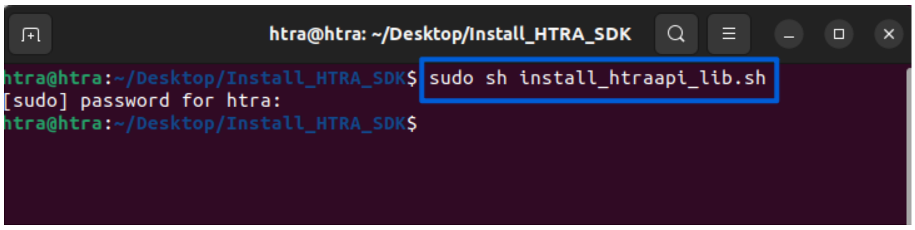
Figure 39 Contents of the Install_HTRA_SDK Folder on Linux

7.3 Driver File Configuration

To use the device in Linux, the driver files must be configured first. The specific procedure is as follows:

1. Configure the driver files: First, copy the Install_HTRA_SDK folder to the Linux host. Then, open a terminal in the Install_HTRA_SDK folder and enter *sudo sh install_htraapi_lib.sh* to configure the driver files.

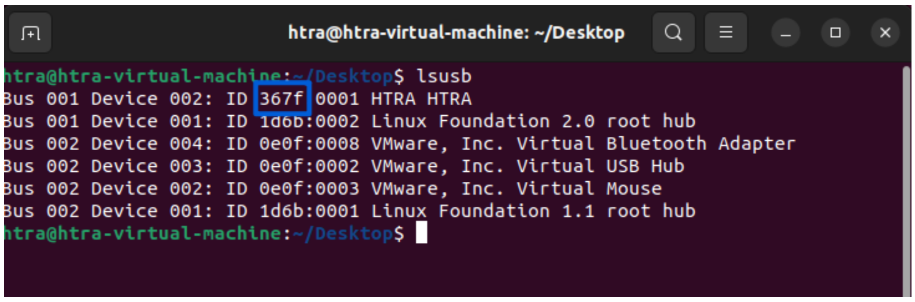
   Note: If the development board does not have the sudo command, simply enter *sh install_htraapi_lib.sh*.



```
htra@htra:~/Desktop/Install_HTRA_SDK$ sudo sh install_htraapi_lib.sh
[sudo] password for htra:
htra@htra:~/Desktop/Install_HTRA_SDK$
```

Figure 40 Configure driver files on Linux

2. Configuration check: Ensure the device is correctly connected to the host computer (if the host computer is a virtual machine, ensure the device is connected to the virtual machine and that USB compatibility is 3.1) and provide normal power to the device. At this point, as shown in the figure, enter *lsusb* in the terminal to view the list of USB devices on the machine, where "ID: 6430 (or ID: 3675 or ID: 04b5)" indicates successful device connection.



```
htra@htra-virtual-machine:~/Desktop$ lsusb
Bus 001 Device 002: ID 367f:0001 HTRA HTRA
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 004: ID 0e0f:0008 VMware, Inc. Virtual Bluetooth Adapter
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
htra@htra-virtual-machine:~/Desktop$
```

Figure 41 Check device connection status on Linux

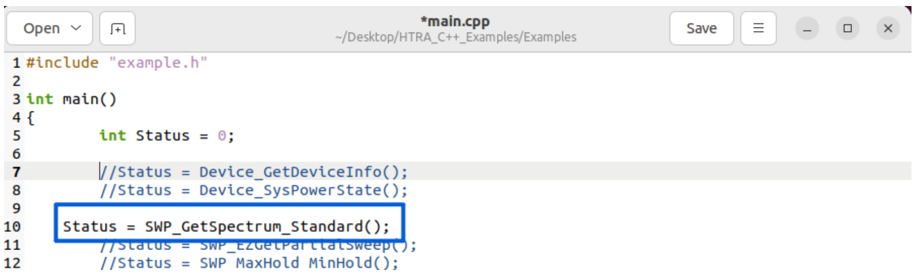
7.4 C++ Example Usage and Project Creation

7.4.1 C++ Example Usage

Prerequisite: Ensure the device is properly connected and the driver files have been correctly configured as described in the Driver File Configuration section.

You can follow the process below and use the C++ examples provided on the USB drive (the specific functions of the examples can be found in the corresponding C/C++ sections).

1. Select the program to compile: First, copy the "Linux\HTRA_C++_Examples" folder from the provided USB drive to the host computer. Then, open main.cpp in the Examples folder and uncomment the examples you want to test or use;



```
#include "example.h"

int main()
{
        int Status = 0;

        //Status = Device_GetDeviceInfo();
        //Status = Device_SysPowerState();

        Status = SWP_GetSpectrum_Standard();
        //Status = SWP_EZGetPartialSweep();
        //Status = SWP_MaxHold_MinHold();
```

Figure 42 Uncomment the SWP_GetSpectrum_Standard Example

2. Compile the selected example: First, check the system architecture according to the Environment Version Compatibility Self-Check section. Open a terminal in the HTRA_C++_Examples folder and enter the corresponding command based on the host system architecture (the following example shows how to compile the SWP_GetSpectrum_Standard routine).

   - For x86_64 system:

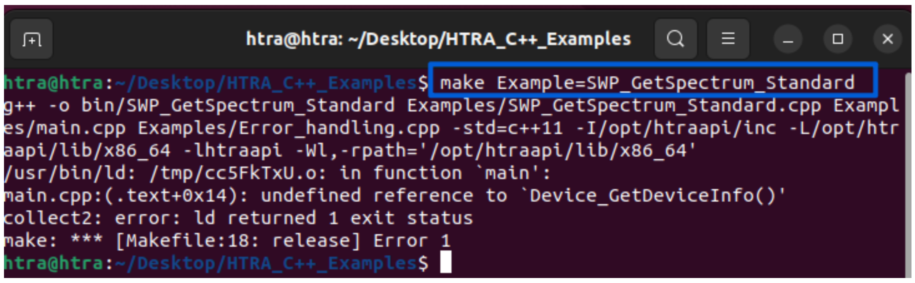
     make Example=SWP_GetSpectrum_Standard

   - For aarch64 system:

     make TARG=aarch64 Example=SWP_GetSpectrum_Standard

   - For armv7 system:

     make TARG=armv7 Example=SWP_GetSpectrum_Standard



```
htra@htra:~/Desktop/HTRA_C++_Examples$ make Example=SWP_GetSpectrum_Standard
g++ -o bin/SWP_GetSpectrum_Standard Examples/SWP_GetSpectrum_Standard.cpp Exampl
es/main.cpp Examples/Error_handling.cpp -std=c++11 -I/opt/htraapi/inc -L/opt/htr
aapi/lib/x86_64 -lhtraapi -Wl,-rpath='/opt/htraapi/lib/x86_64'
/usr/bin/ld: /tmp/cc5FkTxU.o: in function `main':
main.cpp:(.text+0x14): undefined reference to `Device_GetDeviceInfo()'
collect2: error: ld returned 1 exit status
make: *** [Makefile:18: release] Error 1
htra@htra:~/Desktop/HTRA_C++_Examples$
```

Figure 43 Compile SWP_GetSpectrum_Standard on x86_64 System

3. Verify the calibration files: Go to the "HTRA_C++_Examples\bin\CalFile" folder and ensure it contains the calibration files for the devices being used;

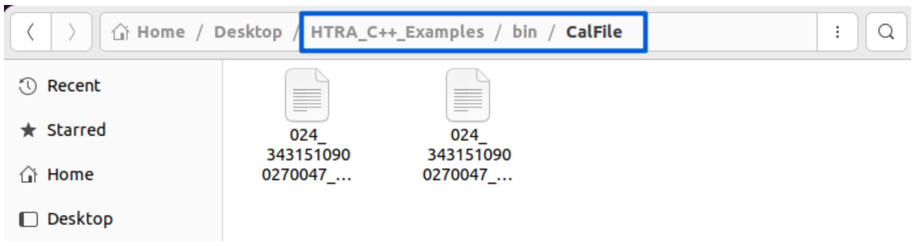
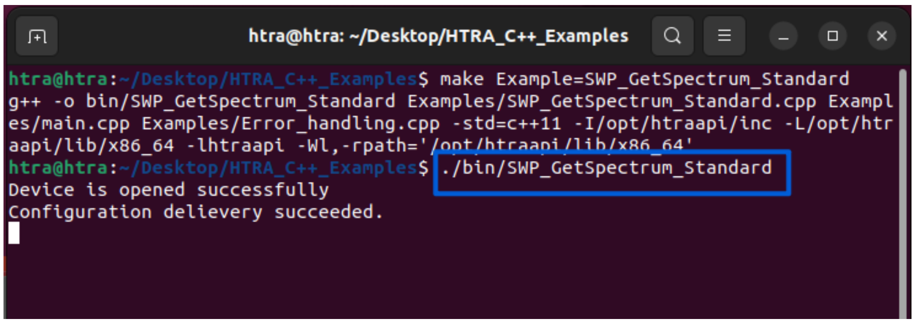


Figure 44 Verify the calibration files

4. Run the program: After successfully compiling the program and verifying the calibration files, open a terminal and enter:

   ./bin/SWP_GetSpectrum_Standard to run the selected example.



```
htra@htra:~/Desktop/HTRA_C++_Examples$ make Example=SWP_GetSpectrum_Standard
g++ -o bin/SWP_GetSpectrum_Standard Examples/SWP_GetSpectrum_Standard.cpp Exampl
es/main.cpp Examples/Error_handling.cpp -std=c++11 -I/opt/htraapi/inc -L/opt/htr
aapi/lib/x86_64 -lhtraapi -Wl,-rpath='/opt/htraapi/lib/x86_64'
htra@htra:~/Desktop/HTRA_C++_Examples$ ./bin/SWP_GetSpectrum_Standard
Device is opened successfully
Configuration delievery succeeded.
```

Figure 45 Run the Compiled SWP_GetSpectrum_Standard Example

7.4.2 C++ Project Creation and Compilation

Prerequisite: Correctly configure the driver files as described in the Driver File Configuration section.

1 . Write the code:

   The Linux dynamic libraries provided on the USB drive are identical to those on Windows. The code you write only needs to comply with the HAROGIC HTRA API Programming Guide.

2 . Compile and run:

1). Create a new folder to store the entire project (e.g., C++_Test). Inside this folder, create a CalFile folder to store calibration files and an htraapi folder to store header files and dynamic link libraries;

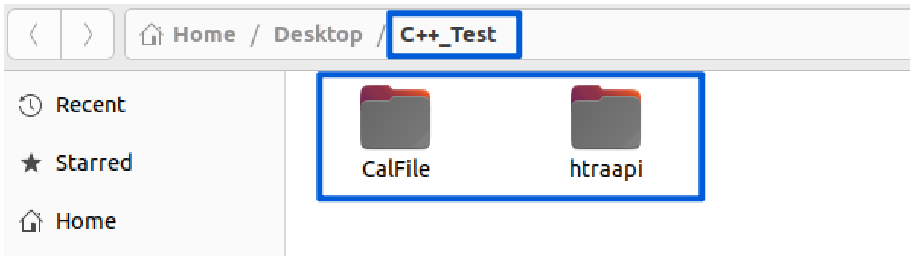


Figure 46 Create Project Folder

2). Inside the htraapi folder, create an inc folder to store header files and a lib folder to store dynamic link libraries;

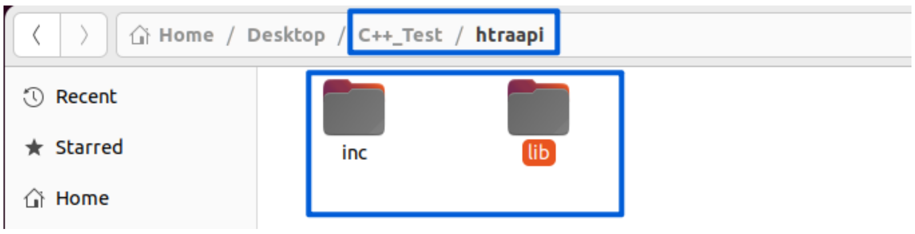
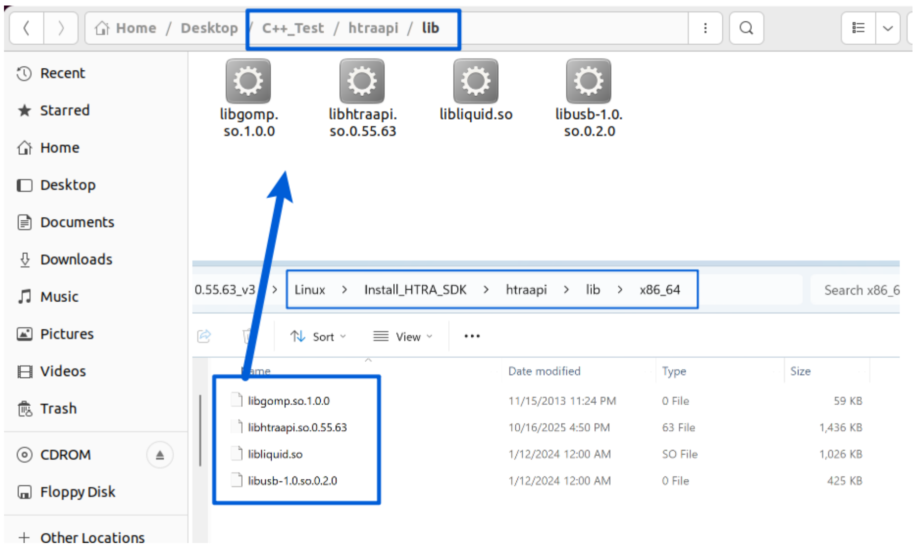


Figure 47 Create inc and lib Folders under htraapi

3). Copy the files from the CalFile folder on the USB drive into the newly created "C++_Test\CalFile" folder;

4). Copy the header files from "Linux\Install_HTRA_SDK\htraapi\inc" on the USB drive into the newly created "C++_Test\htraapi\inc" folder;

5). heck the system architecture according to the Environment Version Compatibility Self-Check section. Then, following the description of the lib folder, copy the files from the folder corresponding to your architecture into the "C++_Test\htraapi\lib" folder. (The illustration shows an x86_64 architecture host.);

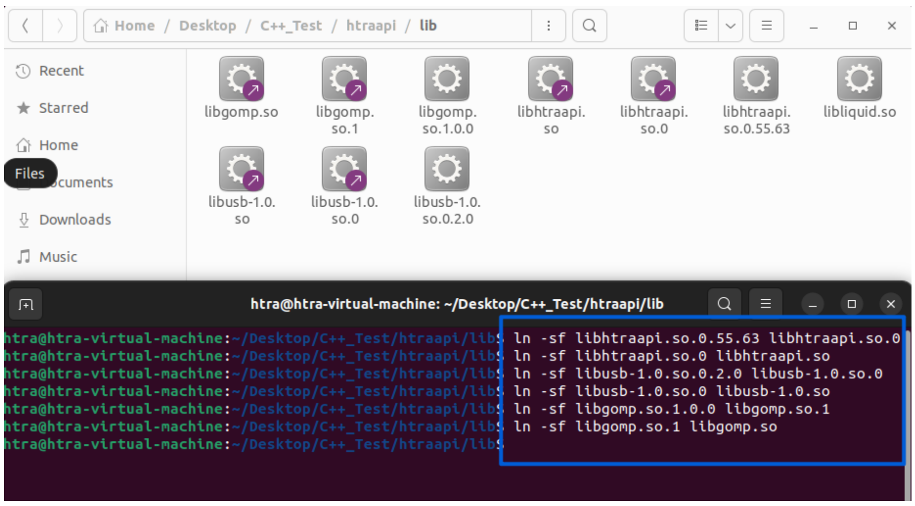


Name	Date modified	Type	Size
libgomp.so.1.0.0	11/15/2013 11:24 PM	0 File	59 KB
libhtraapi.so.0.55.63	10/16/2025 4:50 PM	63 File	1,436 KB
libliquid.so	1/12/2024 12:00 AM	SO File	1,026 KB
libusb-1.0.so.0.2.0	1/12/2024 12:00 AM	0 File	425 KB

Figure 48 Copy the Dynamic Link Libraries for the Corresponding Architecture into the lib Folder

6). Open a terminal in the lib folder and enter the following command to create symbolic links for the copied dynamic link libraries (the commands are the same for all three architectures): In the following, the libhtraapi.so library uses version 0.55.63 as an example; for other versions, simply adjust the version number accordingly:

- *In -sf libhtraapi.so.0.55.63 libhtraapi.so.0*
- *In -sf libhtraapi.so.0 libhtraapi.so*
- *In -sf libusb-1.0.so.0.2.0 libusb-1.0.so.0*
- *In -sf libusb-1.0.so.0 libusb-1.0.so*
- *In -sf libgomp.so.1.0.0 libgomp.so.1*
- *In -sf libgomp.so.1 libgomp.so*



```
htra@htra-virtual-machine:~/Desktop/C++_Test/htraapi/lib$ ln -sf libhtraapi.so.0.55.63 libhtraapi.so.0
htra@htra-virtual-machine:~/Desktop/C++_Test/htraapi/lib$ ln -sf libhtraapi.so.0 libhtraapi.so
htra@htra-virtual-machine:~/Desktop/C++_Test/htraapi/lib$ ln -sf libusb-1.0.so.0.2.0 libusb-1.0.so.0
htra@htra-virtual-machine:~/Desktop/C++_Test/htraapi/lib$ ln -sf libusb-1.0.so.0 libusb-1.0.so
htra@htra-virtual-machine:~/Desktop/C++_Test/htraapi/lib$ ln -sf libgomp.so.1.0.0 libgomp.so.1
htra@htra-virtual-machine:~/Desktop/C++_Test/htraapi/lib$ ln -sf libgomp.so.1 libgomp.so
htra@htra-virtual-machine:~/Desktop/C++_Test/htraapi/lib$
```

Figure 49 Create Symbolic Links for Dynamic Link Libraries

7). Place the written code files in the top-level folder of C++_Test;

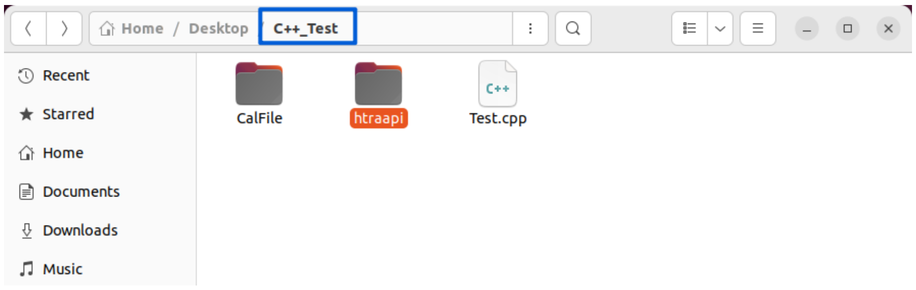


Figure 50 Place the Code Files

8). Compile to generate the executable: Based on the system architecture confirmed in the Environment Version Compatibility Self-Check section, open a terminal in the C++_Test folder and enter the command corresponding to your architecture;

The following example uses Test.cpp. Compile it on different operating systems to generate an executable named Test:

- For x86_64 system:

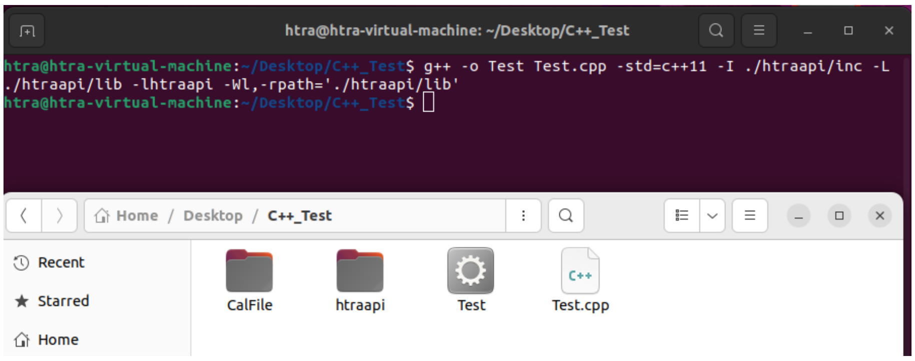
g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L ./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'

- For arrch64 system:

aarch64-linux-gnu-g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L ./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'

- For armv7 system:

arm-linux-gnueabihf-g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L ./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'

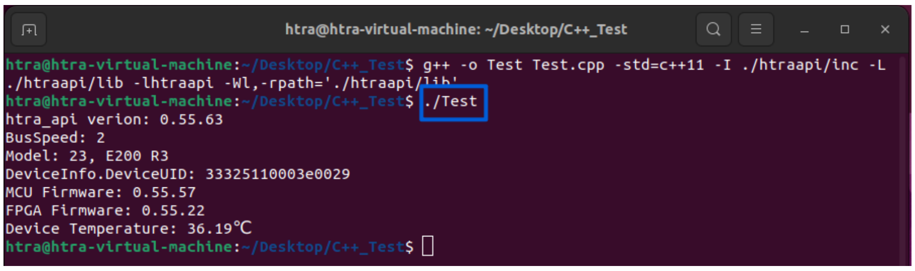


```
htra@htra-virtual-machine:~/Desktop/C++_Test$ g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L
./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'
htra@htra-virtual-machine:~/Desktop/C++_Test$ 
```

htra@htra-virtual-machine: ~/Desktop/C++_Test
Home / Desktop / C++_Test
Recent
Starred
Home
CalFile
htraapi
Test
Test.cpp

Figure 51 Compile to Generate Executable

9). Run the program: Enter *./Test* to start the executable file.



```
htra@htra-virtual-machine:~/Desktop/C++_Test$ g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L
./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'
htra@htra-virtual-machine:~/Desktop/C++_Test$ ./Test
htra_api verion: 0.55.63
BusSpeed: 2
Model: 23, E200 R3
DeviceInfo.DeviceUID: 33325110003e0029
MCU Firmware: 0.55.57
FPGA Firmware: 0.55.22
Device Temperature: 36.19℃
htra@htra-virtual-machine:~/Desktop/C++_Test$ 
```

Figure 52 Run the Test Executable

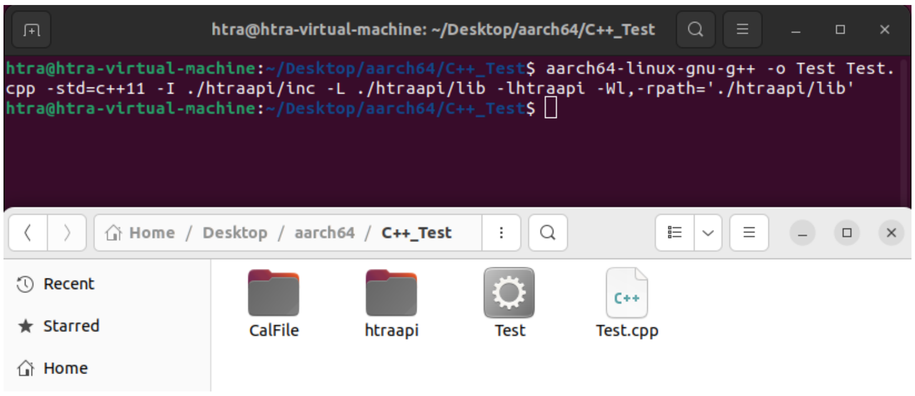
7.4.3 C++ Project Cross Compilation

If the host computer has a cross-compilation toolchain and you want to cross-compile for the device, follow the process below.

The following example shows how to cross-compile an aarch64 executable on an x86_64 host:

1. Generate the executable for the target architecture:

1). Follow steps 1 to 8 in the "C++ Project Creation and Compilation" section to create the project, place the calibration files, and add the aarch64 architecture header and library files. Then, create symbolic links, place the program code, and compile the executable using the target architecture compilation command. As shown below, when compiling an aarch64 executable, use the compilation command for an aarch64 system.

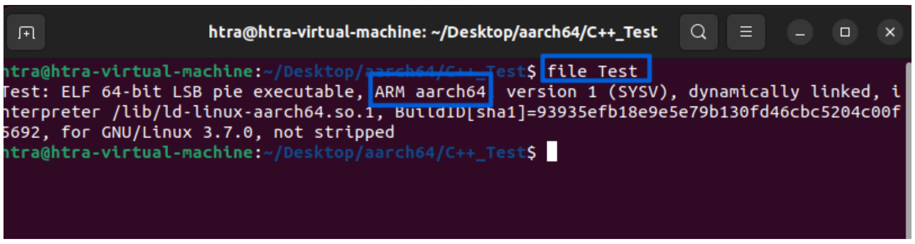


```
htra@htra-virtual-machine:~/Desktop/aarch64/C++_Test$ aarch64-linux-gnu-g++ -o Test Test.
cpp -std=c++11 -I ./htraapi/inc -L ./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'
htra@htra-virtual-machine:~/Desktop/aarch64/C++_Test$ 
```

htra@htra-virtual-machine: ~/Desktop/aarch64/C++_Test
Home / Desktop / aarch64 / C++_Test
Recent
Starred
Home
CalFile
htraapi
Test
Test.cpp

Figure 53 Cross-Compile an aarch64 Executable

2). After generating the executable, enter *file Test* to check the architecture of the executable. You should see that the current executable is for the aarch64 architecture.

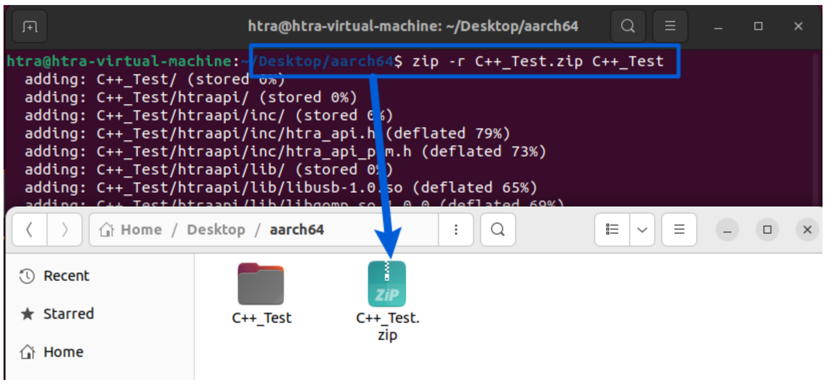


```
htra@htra-virtual-machine: ~/Desktop/aarch64/C++_Test
ntra@htra-virtual-machine:~/Desktop/aarch64/C++_Test$ file Test
Test: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically linked, i
nterpreter /lib/ld-linux-aarch64.so.1, BuildID[sha1]=93935efb18e9e5e79b130fd46cbc5204c00f
5692, for GNU/Linux 3.7.0, not stripped
ntra@htra-virtual-machine:~/Desktop/aarch64/C++_Test$
```

Figure 54 Check Executable Architecture

2. Run the executable on an aarch64 host:

1). Navigate to the project directory and execute *zip -r C++_Test.zip C++_Test* to compress the entire C++_Test folder into a ZIP archive. Then, copy the generated ZIP file to the aarch64 host;



```
htra@htra-virtual-machine: ~/Desktop/aarch64
htra@htra-virtual-machine:~/Desktop/aarch64$ zip -r C++_Test.zip C++_Test
  adding: C++_Test/ (stored 0%)
  adding: C++_Test/htraapi/ (stored 0%)
  adding: C++_Test/htraapi/inc/ (stored 0%)
  adding: C++_Test/htraapi/inc/htra_api.h (deflated 79%)
  adding: C++_Test/htraapi/inc/htra_api_pam.h (deflated 73%)
  adding: C++_Test/htraapi/lib/ (stored 0%)
  adding: C++_Test/htraapi/lib/libusb-1.0.so (deflated 65%)
  adding: C++_Test/htraapi/lib/libgomp.so.1.0.0 (deflated 69%)
```

Figure 55 Compress the Created Project

2). On the aarch64 host, enter *unzip C++_Test.zip* to extract the project;

3). Following the Driver File Configuration section, configure the driver files on the aarch64 host;

4). After configuring the driver files, go to the C++_Test folder and run the program by entering ./Test in the terminal.

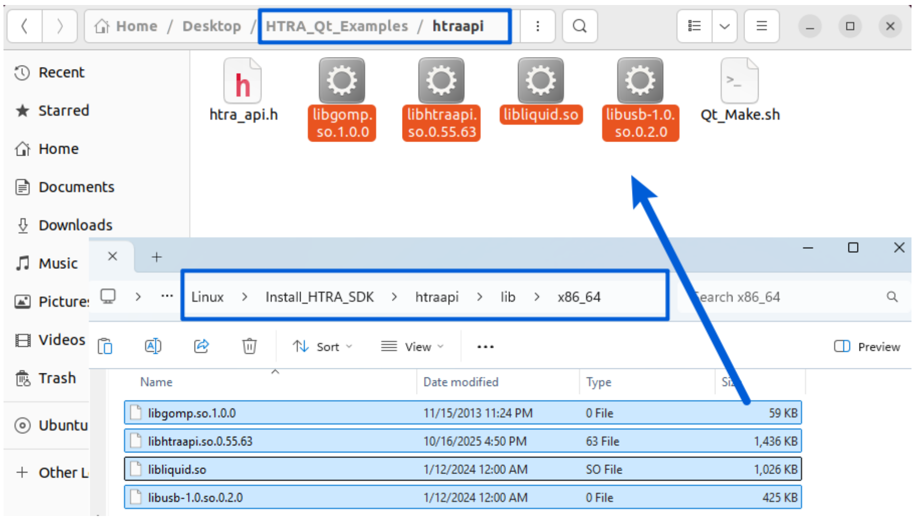
7.5 Using QT Examples and Project Creation

7.5.1 Qt Example Usage Process

Usage prerequisite: Ensure the device is properly connected and the driver files have been correctly configured according to the Driver File Configuration chapter.

The following example demonstrates how to run the Qt sample on Ubuntu 22.04, using an x86_64 architecture.

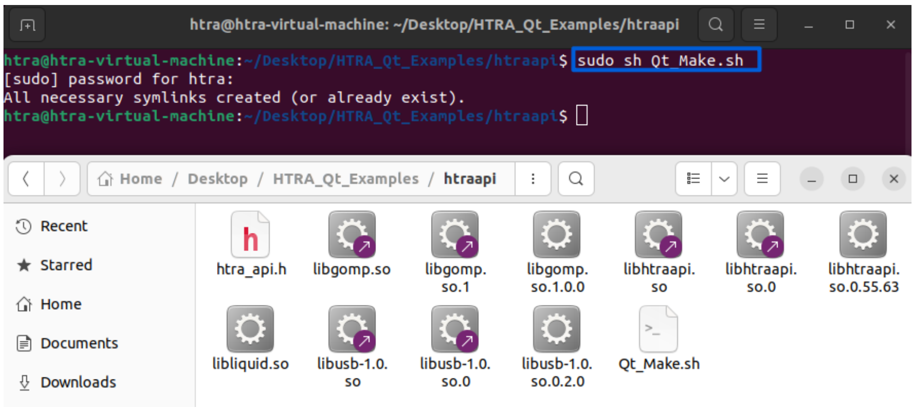
1. Copy the "Linux\HTRA_Qt_Examples" folder from the included USB drive to the host PC;
2. Refer to the Environment Version Compatibility Self-Check chapter to confirm the system architecture, then copy the contents from the corresponding architecture folder under "Linux\Install_HTRA_SDK\htraapi\lib" on the USB drive into the host PC's "HTRA_Qt_Examples\htraapi" folder;



Name	Date modified	Type	Size
libgomp.so.1.0.0	11/15/2013 11:24 PM	0 File	59 KB
libhtraapi.so.0.55.63	10/16/2025 4:50 PM	63 File	1,436 KB
libliquid.so	1/12/2024 12:00 AM	SO File	1,026 KB
libusb-1.0.so.0.2.0	1/12/2024 12:00 AM	0 File	425 KB

Figure 56 Copy the dynamic libraries for the corresponding architecture

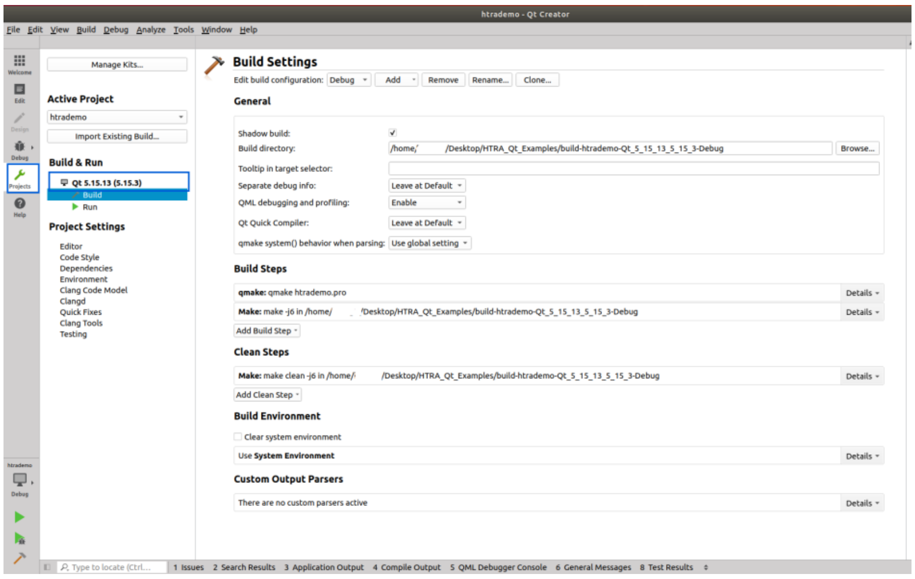
3. Open a terminal in the htraapi folder and enter *sudo sh Qt_Make.sh*. When prompted, enter the password to authorize the creation of symbolic link;



```
htra@htra-virtual-machine:~/Desktop/HTRA_Qt_Examples/htraapi$ sudo sh Qt_Make.sh
[sudo] password for htra:
All necessary symlinks created (or already exist).
htra@htra-virtual-machine:~/Desktop/HTRA_Qt_Examples/htraapi$
```

Figure 57 Create symbolic links for the dynamic libraries

4. Enter the "HTRA_Qt_Examples\bin\CalFile" folder and ensure that it contains the calibration files for the device in use;
5. Use Qt Creator to open the htrademo.pro file located in "HTRA_Qt_Examples\htrademo", and configure the project's build environment;



Build Settings

Edit build configuration: Debug

General

```
Shadow build:                               ✓
Build directory:                            /home/        /Desktop/HTRA_Qt_Examples/build-htrademo-Qt_5_15_13_5_15_3-Debug
Tooltip in target selector:
Separate debug info:                        Leave at Default
QML debugging and profiling:                Enable
Qt Quick Compiler:                          Leave at Default
qmake system() behavior when parsing:       Use global setting
```

Build Steps

```
qmake: qmake htrademo.pro
Make: make -j6 in /home/        /Desktop/HTRA_Qt_Examples/build-htrademo-Qt_5_15_13_5_15_3-Debug
```

Clean Steps

```
Make: make clean -j6 in /home/        /Desktop/HTRA_Qt_Examples/build-htrademo-Qt_5_15_13_5_15_3-Debug
```

Build Environment

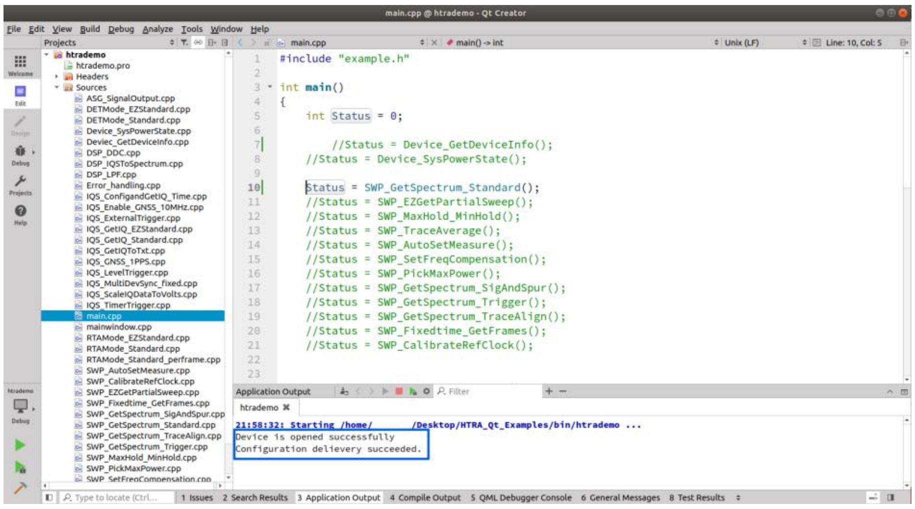
```
☐ Clear system environment
Use System Environment
```

Custom Output Parsers

```
There are no custom parsers active
```

Figure 58 Build environment

6. Click "Edit", open main.cpp under the "Sources" folder of the "htrademo" project, uncomment the relevant functions, save the file, and click Run to execute different examples;



```cpp
#include "example.h"

int main()
{
    int Status = 0;

        //Status = Device_GetDeviceInfo();
    //Status = Device_SysPowerState();

    Status = SWP_GetSpectrum_Standard();
    //Status = SWP_EZGetPartialSweep();
    //Status = SWP_MaxHold_MinHold();
    //Status = SWP_TraceAverage();
    //Status = SWP_AutoSetMeasure();
    //Status = SWP_SetFreqCompensation();
    //Status = SWP_PickMaxPower();
    //Status = SWP_GetSpectrum_SigAndSpur();
    //Status = SWP_GetSpectrum_Trigger();
    //Status = SWP_GetSpectrum_TraceAlign();
    //Status = SWP_Fixedtime_GetFrames();
    //Status = SWP_CalibrateRefClock();
```

```
21:58:32: Starting /home/        /Desktop/HTRA_Qt_Examples/bin/htrademo ...
Device is opened successfully
Configuration delievery succeeded.
```

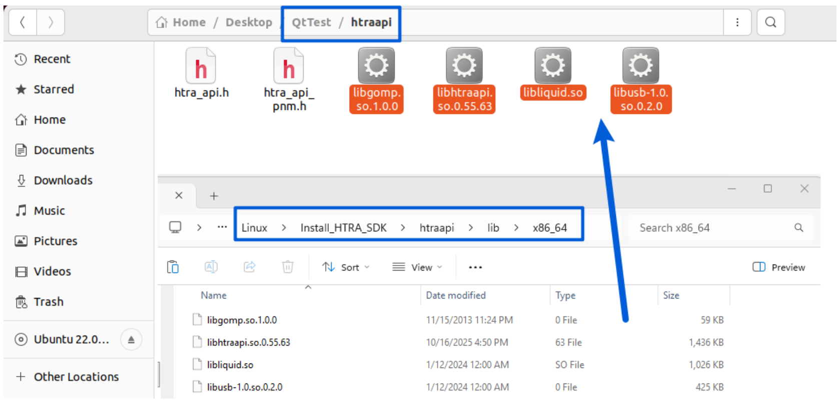
Figure 59 Run the SWP_GetSpectrum_Standard example

7.5.2 Qt Project Creation and Compilation

Provided that the driver files have been correctly configured as described in the Driver File Configuration section, if you want to create and compile a Qt project, follow the process below:

The code should follow the API Programming Guide, and the steps for creating a GUI application are as follows:

1. Create a new folder to store the entire project (e.g., QtTest), including bin (for calibration files and generated executables) and htraapi (for header files and dynamic link libraries) subfolders;
2. Copy the instrument-specific CalFile folder from the USB drive into the QtTest\bin folder;
3. Copy the header files from Linux\Install_HTRA_SDK\htraapi\inc on the USB drive into the QtTest\htraapi folder;
4. Copy the dynamic link libraries from the folder corresponding to your system architecture under Linux\Install_HTRA_SDK\htraapi\lib on the USB drive into the QtTest\htraapi folder (here, an x86_64 host is used as an example);



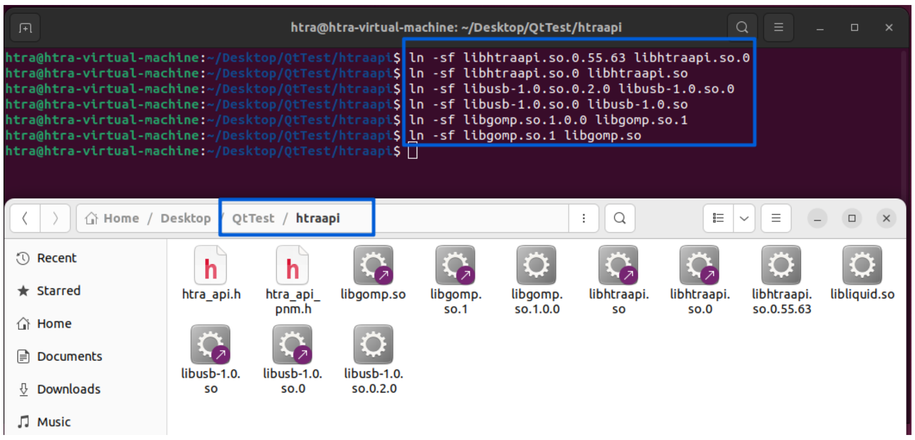
Name	Date modified	Type	Size
libgomp.so.1.0.0	11/15/2013 11:24 PM	0 File	59 KB
libhtraapi.so.0.55.63	10/16/2025 4:50 PM	63 File	1,436 KB
libliquid.so	1/12/2024 12:00 AM	SO File	1,026 KB
libusb-1.0.so.0.2.0	1/12/2024 12:00 AM	0 File	425 KB

Figure 60 Copy the dynamic link libraries for the corresponding architecture

5. Open a terminal in the htraapi folder and enter the following commands in sequence to create symbolic links for the copied dynamic link libraries (the commands are the same for all architectures):

   In the following, the libhtraapi.so library uses version 0.55.63 as an example; for other versions, simply adjust the version number accordingly:

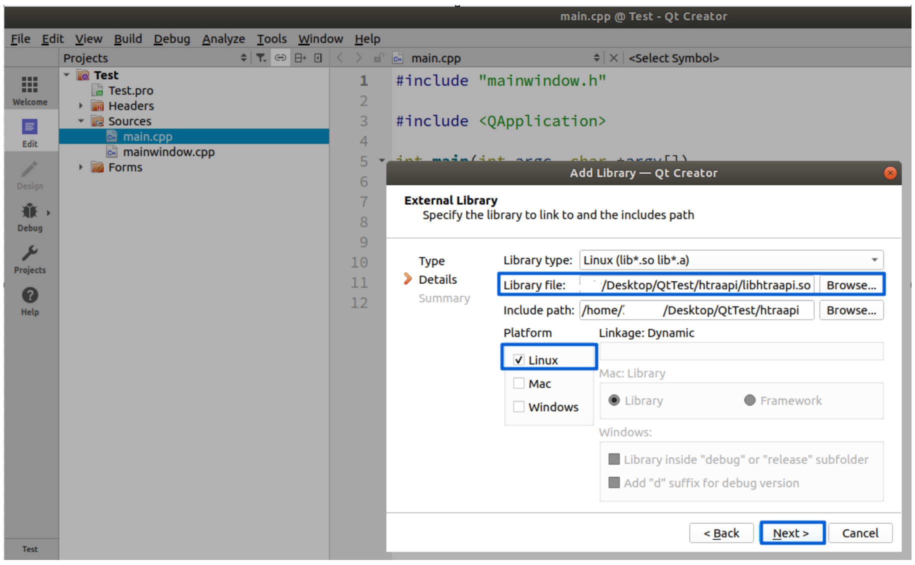
   - *ln -sf libhtraapi.so.0.55.63 libhtraapi.so.0*
   - *ln -sf libhtraapi.so.0 libhtraapi.so*
   - *ln -sf libusb-1.0.so.0.2.0 libusb-1.0.so.0*
   - *ln -sf libusb-1.0.so.0 libusb-1.0.so*
   - *ln -sf libgomp.so.1.0.0 libgomp.so.1*
   - *ln -sf libgomp.so.1 libgomp.so*



```
htra@htra-virtual-machine:~/Desktop/QtTest/htraapi$ ln -sf libhtraapi.so.0.55.63 libhtraapi.so.0
htra@htra-virtual-machine:~/Desktop/QtTest/htraapi$ ln -sf libhtraapi.so.0 libhtraapi.so
htra@htra-virtual-machine:~/Desktop/QtTest/htraapi$ ln -sf libusb-1.0.so.0.2.0 libusb-1.0.so.0
htra@htra-virtual-machine:~/Desktop/QtTest/htraapi$ ln -sf libusb-1.0.so.0 libusb-1.0.so
htra@htra-virtual-machine:~/Desktop/QtTest/htraapi$ ln -sf libgomp.so.1.0.0 libgomp.so.1
htra@htra-virtual-machine:~/Desktop/QtTest/htraapi$ ln -sf libgomp.so.1 libgomp.so
htra@htra-virtual-machine:~/Desktop/QtTest/htraapi$
```

Figure 61 Create symbolic links for the dynamic libraries in htraapi

6. Open Qt Creator and click "File" -> "New File or Project";
7. In the project options, click "Application", select "Qt Widgets Application", and click "Choose...";
8. Enter the project name (e.g., "test"), click "Browse", select the previously created "QtTest" folder, and then click "Next";
9. Select "qmake", click "Next" until the "Kit Selection" screen appears. On this screen, choose a build environment and click "Next";
10. Click "Finish" to create the project;
11. In the Qt Creator main interface, right-click the "Test" project, select "Add Library..." -> "External Library" -> "Next";
12. Click "Browse" for the library file, select libhtraapi.so in "QtTest\htraapi", click "Open", choose the "Linux" platform, and click "Next".



```
#include "mainwindow.h"

#include <QApplication>
```

Add Library — Qt Creator

External Library
Specify the library to link to and the includes path

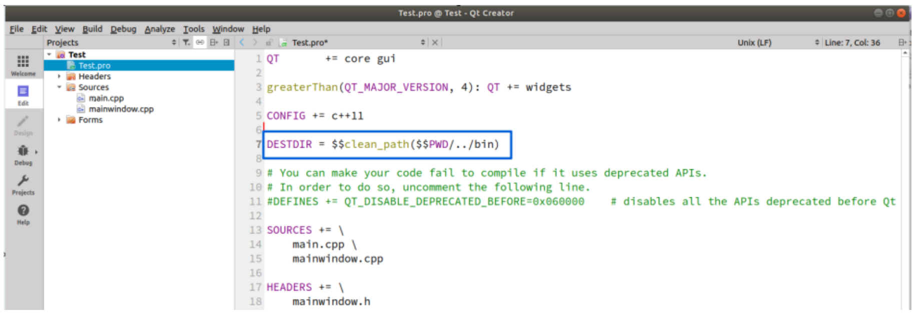
Type
Details
Summary

Library type: Linux (lib*.so lib*.a)
Library file: /Desktop/QtTest/htraapi/libhtraapi.so
Include path: /home/ /Desktop/QtTest/htraapi
Platform
[✓ Linux]
Mac
Windows
Linkage: Dynamic
Mac: Library
● Library ○ Framework
Windows:
Library inside "debug" or "release" subfolder
Add "d" suffix for debug version

Figure 62 Select the Linux Platform

13. Click "Finish" to add the external library;

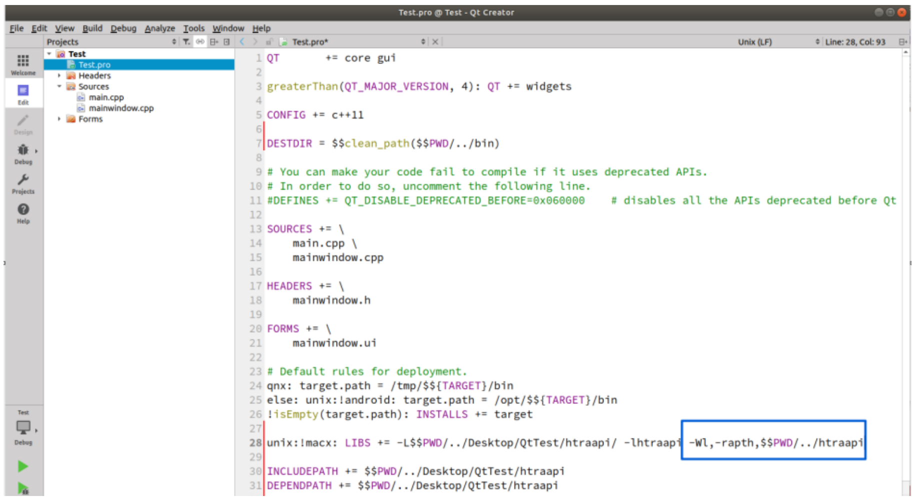
14. In Test.pro, after CONFIG += c++11, add *DESTDIR = $$clean_path($$PWD/../bin)* to specify the location for the generated executable;



```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

DESTDIR = $$clean_path($$PWD/../bin)

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    mainwindow.h
```

Figure 63 Specify the Executable Output Location

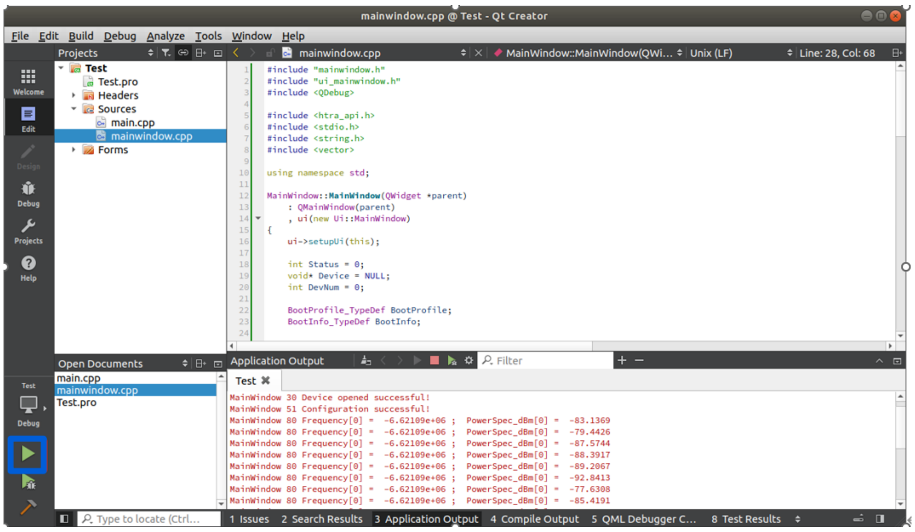
15. After the library file, add *-Wl,-rpath,$$PWD/../htraapi* to specify the runtime library path for the executable;



```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

DESTDIR = $$clean_path($$PWD/../bin)

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    mainwindow.h

FORMS += \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

unix:!macx: LIBS += -L$$PWD/../Desktop/QtTest/htraapi/ -lhtraapi -Wl,-rapth,$$PWD/../htraapi

INCLUDEPATH += $$PWD/../Desktop/QtTest/htraapi
DEPENDPATH += $$PWD/../Desktop/QtTest/htraapi
```

Figure 64 Specify the Runtime Library Path for the Executable

16. Save the Test.pro file, then write the code in mainwindow.cpp and click Run. The normal operating interface is shown below;

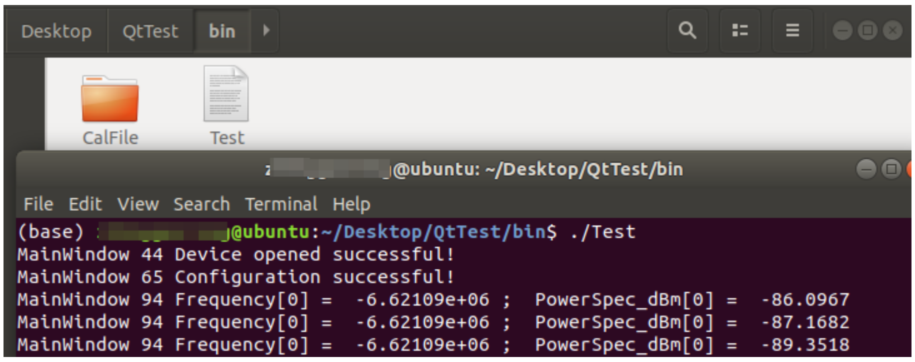


```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>

#include <htra_api.h>
#include <stdio.h>
#include <string.h>
#include <vector>

using namespace std;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    int Status = 0;
    void* Device = NULL;
    int DevNum = 0;

    BootProfile_TypeDef BootProfile;
    BootInfo_TypeDef BootInfo;

```

```
MainWindow 30 Device opened successful!
MainWindow 51 Configuration successful!
MainWindow 80 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -83.1369
MainWindow 80 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -79.4426
MainWindow 80 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -87.5744
MainWindow 80 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -88.3917
MainWindow 80 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -89.2067
MainWindow 80 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -92.8413
MainWindow 80 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -77.6308
MainWindow 80 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -85.4191
```

Figure 65 Run the Code

17. Close Qt Creator, navigate to the QtTest\bin folder, open a terminal, and enter *./Test* to run the executable;


Desktop
QtTest
bin
CalFile
Test
[redacted]@ubuntu: ~/Desktop/QtTest/bin
File Edit View Search Terminal Help

```
(base) [redacted]@ubuntu:~/Desktop/QtTest/bin$ ./Test
MainWindow 44 Device opened successful!
MainWindow 65 Configuration successful!
MainWindow 94 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -86.0967
MainWindow 94 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -87.1682
MainWindow 94 Frequency[0] =  -6.62109e+06 ;  PowerSpec_dBm[0] =  -89.3518
```

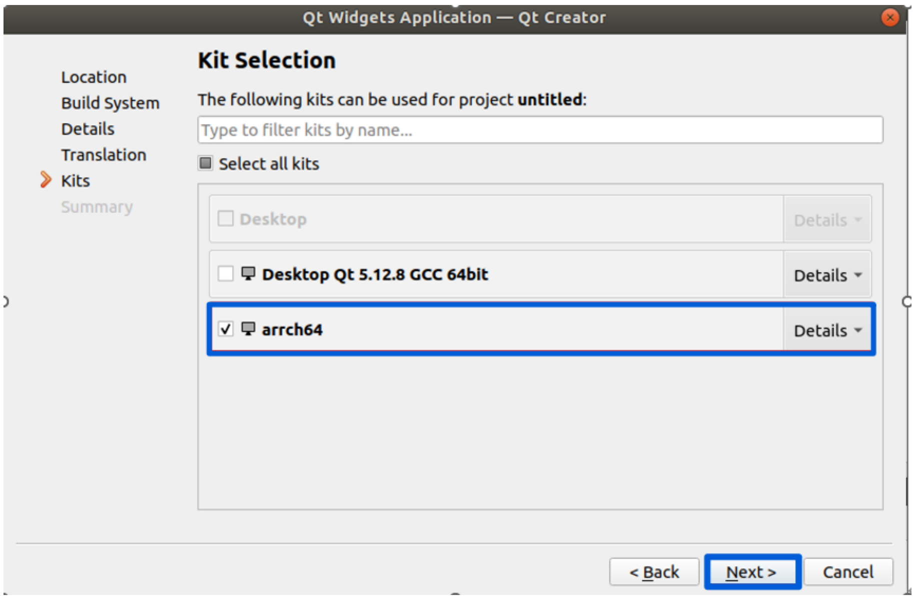
Figure 66 Run the Executable

7.5.3 Qt Project Cross-Compilation

If the host computer has a cross-compilation toolchain and you want to cross-compile for the device, follow the procedure below (here, an example of cross-compiling an aarch64 executable on an x86_64 host is given):

1. First, generate the executable for the target architecture:

1). Following steps 1 to 9 of the Qt Project Creation and Compilation process for a windowed application, create the project, place the calibration and header files, place the library files for the cross-compilation target architecture (aarch64), create a soft link (symlink) for the dynamic link libraries, create the program in Qt Creator, and select the aarch64 Build Kit



Kit Selection

The following kits can be used for project **untitled**:

Type to filter kits by name...

☐ Select all kits

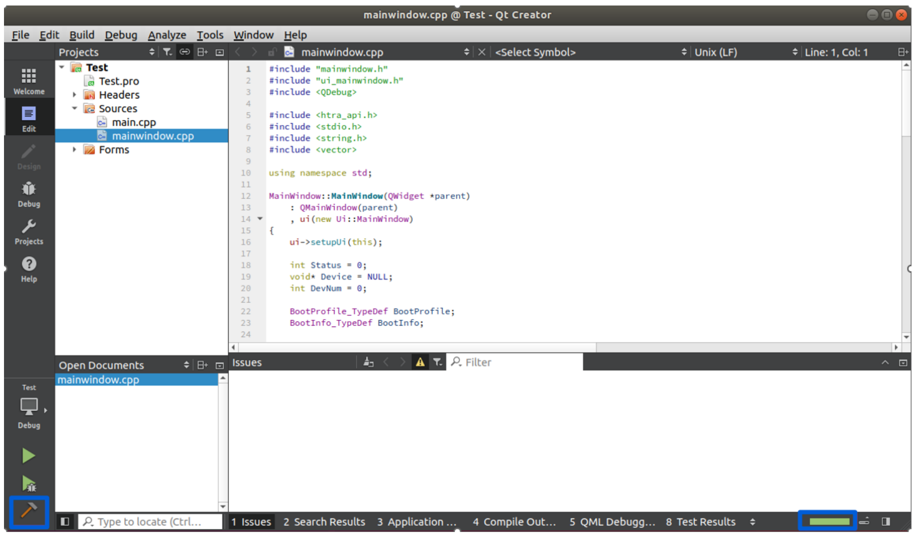
☐ Desktop

☐ **Desktop Qt 5.12.8 GCC 64bit**

✓ **arrch64**

Figure 67 Select the target kit for the cross-compilation target architecture

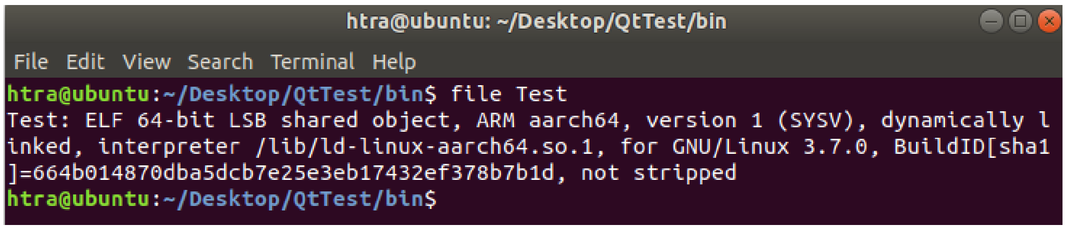
2). Follow steps 10 to 16 of the windowed application creation process to perform project creation, library referencing, modification of the executable program generation location, project coding, and running;


mainwindow.cpp @ Test - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Projects
Test
Test.pro
Headers
Sources
main.cpp
mainwindow.cpp
Forms
Welcome
Edit
Design
Debug
Projects
Help
<Select Symbol>
Unix (LF)
Line: 1, Col: 1
Open Documents
Issues
Filter
1 Issues 2 Search Results 3 Application ... 4 Compile Out... 5 QML Debugg... 8 Test Results

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>

#include <htra_api.h>
#include <stdio.h>
#include <string.h>
#include <vector>

using namespace std;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    int Status = 0;
    void* Device = NULL;
    int DevNum = 0;

    BootProfile_TypeDef BootProfile;
    BootInfo_TypeDef BootInfo;

```

Figure 68 Build the executable program for the aarch64 architecture

3). After building the executable program, open the terminal in the QtTest\bin folder and enter file Test to view the executable program's architecture (the executable program name is Test, so the input command is file Test);

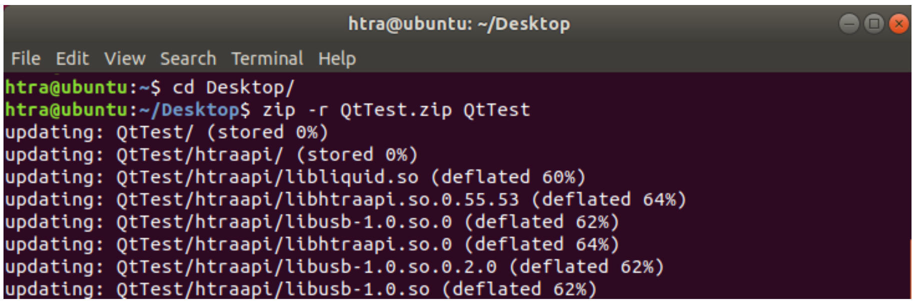

htra@ubuntu: ~/Desktop/QtTest/bin
File Edit View Search Terminal Help

```
htra@ubuntu:~/Desktop/QtTest/bin$ file Test
Test: ELF 64-bit LSB shared object, ARM aarch64, version 1 (SYSV), dynamically l
inked, interpreter /lib/ld-linux-aarch64.so.1, for GNU/Linux 3.7.0, BuildID[sha1
]=664b014870dba5dcb7e25e3eb17432ef378b7b1d, not stripped
htra@ubuntu:~/Desktop/QtTest/bin$
```

Figure 69 View the executable program's architecture

2. Run the executable program on the aarch64 architecture host machine:

1). Enter the project directory (The example project is located on the Desktop, so enter cd Desktop/). Execute *zip -r QtTest.zip QtTest* to package the entire QtTest folder into a ZIP file. Then, copy the generated compressed package to the aarch64 host machine;



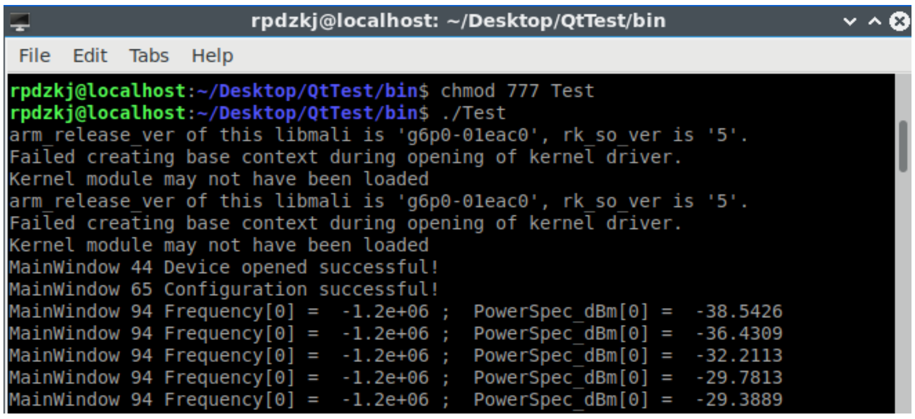
```
htra@ubuntu:~$ cd Desktop/
htra@ubuntu:~/Desktop$ zip -r QtTest.zip QtTest
updating: QtTest/ (stored 0%)
updating: QtTest/htraapi/ (stored 0%)
updating: QtTest/htraapi/libliquid.so (deflated 60%)
updating: QtTest/htraapi/libhtraapi.so.0.55.53 (deflated 64%)
updating: QtTest/htraapi/libusb-1.0.so.0 (deflated 62%)
updating: QtTest/htraapi/libhtraapi.so.0 (deflated 64%)
updating: QtTest/htraapi/libusb-1.0.so.0.2.0 (deflated 62%)
updating: QtTest/htraapi/libusb-1.0.so (deflated 62%)
```

Figure 70 Compress the created project

2). On the aarch64 host machine, enter *unzip QtTest.zip* to unzip (or extract) the project;

3). Refer to the section on Driver File Configuration to configure the driver files on the aarch64 host machine;

4). After configuring the driver files, enter the QtTest folder. In the terminal, enter *chmod 777 Test* to grant permissions to the executable program. Subsequently, you can run the program by entering ./Test.



```
rpdzkj@localhost:~/Desktop/QtTest/bin$ chmod 777 Test
rpdzkj@localhost:~/Desktop/QtTest/bin$ ./Test
arm_release_ver of this libmali is 'g6p0-01eac0', rk_so_ver is '5'.
Failed creating base context during opening of kernel driver.
Kernel module may not have been loaded
arm_release_ver of this libmali is 'g6p0-01eac0', rk_so_ver is '5'.
Failed creating base context during opening of kernel driver.
Kernel module may not have been loaded
MainWindow 44 Device opened successful!
MainWindow 65 Configuration successful!
MainWindow 94 Frequency[0] =  -1.2e+06 ;  PowerSpec_dBm[0] =  -38.5426
MainWindow 94 Frequency[0] =  -1.2e+06 ;  PowerSpec_dBm[0] =  -36.4309
MainWindow 94 Frequency[0] =  -1.2e+06 ;  PowerSpec_dBm[0] =  -32.2113
MainWindow 94 Frequency[0] =  -1.2e+06 ;  PowerSpec_dBm[0] =  -29.7813
MainWindow 94 Frequency[0] =  -1.2e+06 ;  PowerSpec_dBm[0] =  -29.3889
```

Figure 71 Grant permissions and run the executable program

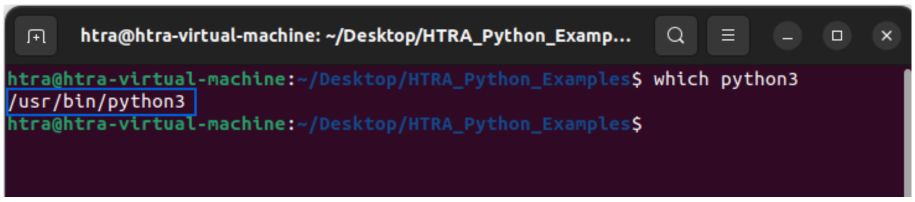
7.6 Usage of Python Examples and Project Creation

7.6.1 Usage of Python Examples

Prerequisite: Ensure the device is properly connected and the driver files have been correctly configured according to the Diver File Configuration section.

The functionality of the Python examples can be found in the Python Example Description chapter.

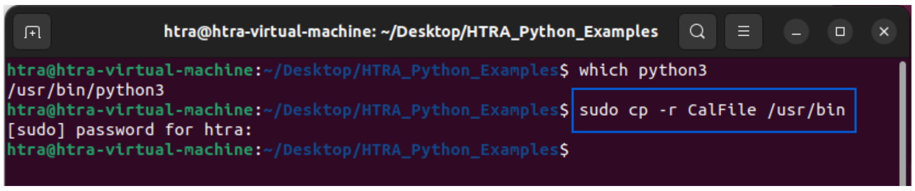
1. Copy the "Linux\HTRA_Python_Examples" folder from the included USB drive to the host PC. In this folder, open a terminal and enter *which python3* to check the location of the Python interpreter (for example, here it is /usr/bin);



```
htra@htra-virtual-machine:~/Desktop/HTRA_Python_Examples$ which python3
/usr/bin/python3
htra@htra-virtual-machine:~/Desktop/HTRA_Python_Examples$
```

Figure 72 Check the Python3 Interpreter Location

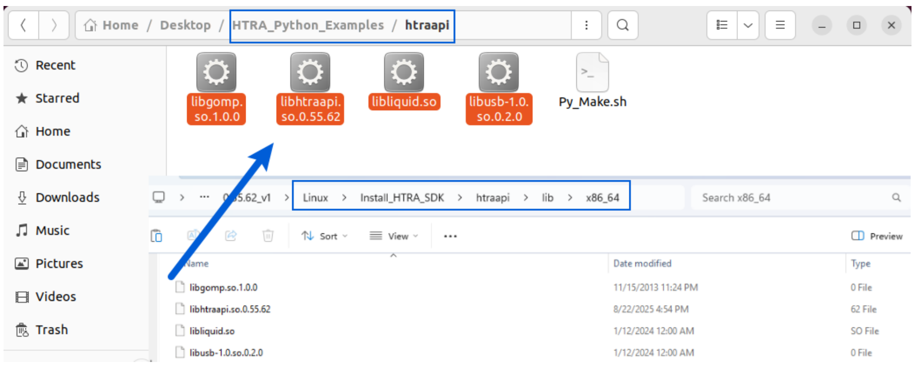
2. Based on the obtained interpreter path, enter *sudo cp -r CalFile /usr/bin* to copy the device calibration files to the same directory as the Python interpreter. (If the interpreter is not located in /usr/bin, replace the path with the actual location.).



```
htra@htra-virtual-machine:~/Desktop/HTRA_Python_Examples$ which python3
/usr/bin/python3
htra@htra-virtual-machine:~/Desktop/HTRA_Python_Examples$ sudo cp -r CalFile /usr/bin
[sudo] password for htra:
htra@htra-virtual-machine:~/Desktop/HTRA_Python_Examples$
```

Figure 73 Copy the Calibration Files to the Same Directory as the Interpreter

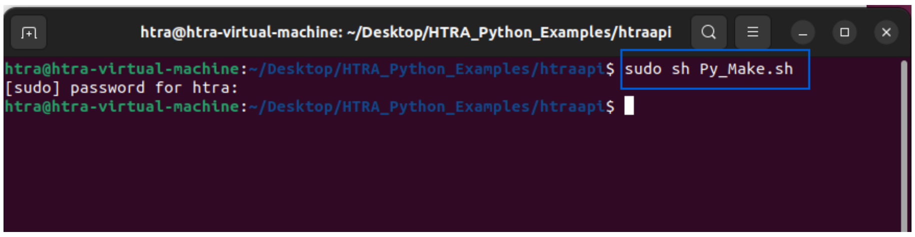
3. Refer to the Environment Version Compatibility Self-Check chapter to confirm the system architecture, then copy the contents from the corresponding architecture folder under "Linux\Install_HTRA_SDK\htraapi\lib" on the USB drive into the host PC's "HTRA_Python_Examples\htraapi" folder;



Name	Date modified	Type
libgomp.so.1.0.0	11/15/2013 11:24 PM	0 File
libhtraapi.so.0.55.62	8/22/2025 4:54 PM	62 File
libliquid.so	1/12/2024 12:00 AM	SO File
libusb-1.0.so.0.2.0	1/12/2024 12:00 AM	0 File

Figure 74 Copy the Dynamic Link Libraries

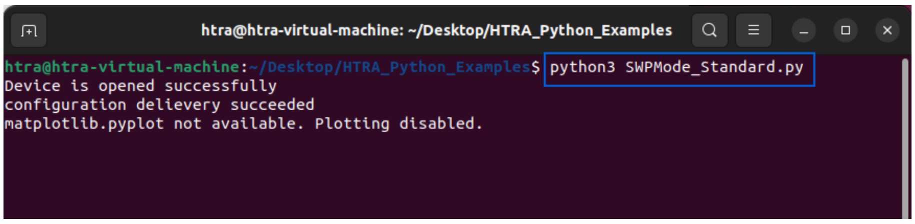
4. Open a terminal in the htraapi folder, enter *sudo sh Py_Make.sh*, and provide the password when prompted to grant permission for creating symbolic links for the libraries;



```
htra@htra-virtual-machine: ~/Desktop/HTRA_Python_Examples/htraapi
htra@htra-virtual-machine:~/Desktop/HTRA_Python_Examples/htraapi$ sudo sh Py_Make.sh
[sudo] password for htra:
htra@htra-virtual-machine:~/Desktop/HTRA_Python_Examples/htraapi$ 
```

Figure 75 Create symbolic links for the libraries in htraapi

5. In the HTRA_Python_Examples folder, open a terminal and enter *python3 SWPMode_Standard.py* to run the provided SWPMode_Standard.py example.



```
htra@htra-virtual-machine: ~/Desktop/HTRA_Python_Examples
htra@htra-virtual-machine:~/Desktop/HTRA_Python_Examples$ python3 SWPMode_Standard.py
Device is opened successfully
configuration delievery succeeded
matplotlib.pyplot not available. Plotting disabled.
```

Figure 76 Run the Python Example

Note: If running the Python example on a different device, first copy the calibration files to the "HTRA_Python_Examples\CalFile" folder. Then, in a terminal within the HTRA_Python_Examples folder, enter *sudo cp -r CalFile /usr/bin* to update the calibration files to the same directory as the Python interpreter.

7.6.2 Python Project Creation

Provided that the driver files have been correctly configured as described in the Diver File Configuration section, if you want to create a Python project, follow the procedure below:

1. The code should follow the HAROGIC HTRA API Programming Guide;
2. When running the program, place it in the example folder (HTRA_Python_Examples) and follow the procedure in the Usage of Python Examples chapter.

www.harogic.com

info@harogic.com

+65-8299 8857