API Example Usage Guide

API Version 0.55.55

2025-1-20

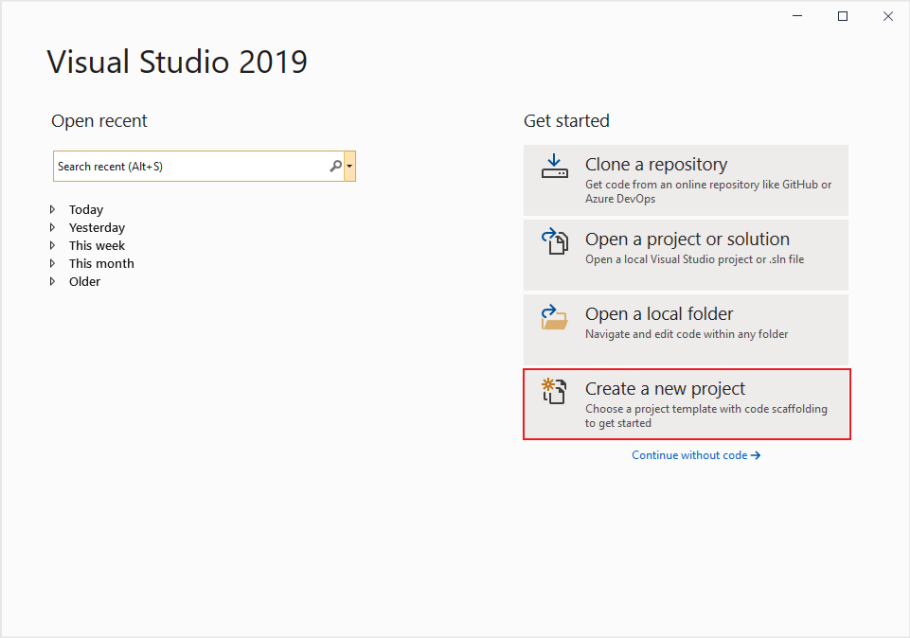
HAROGIC

Table of Contents

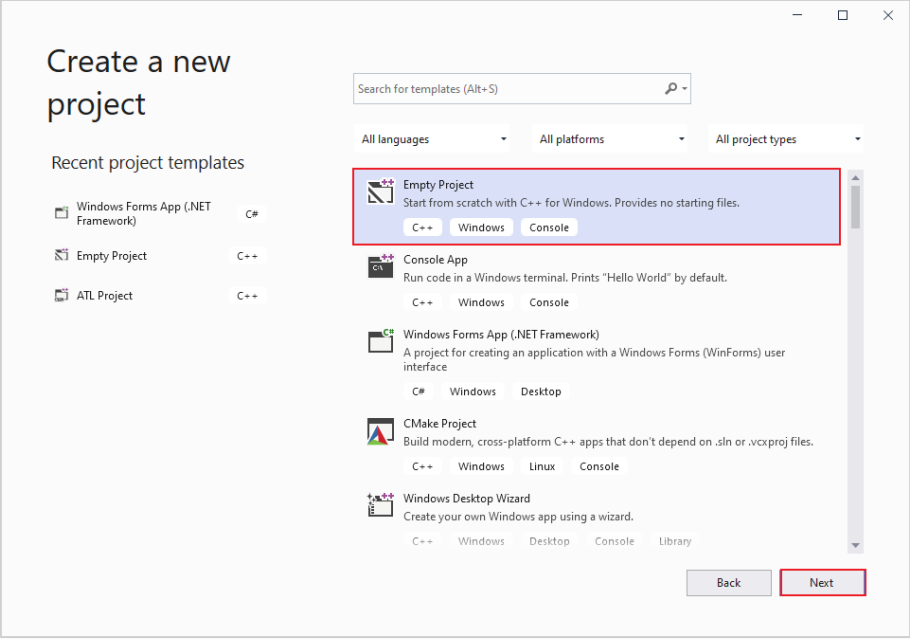
1. C/C++

1.1 Configure Development Environment

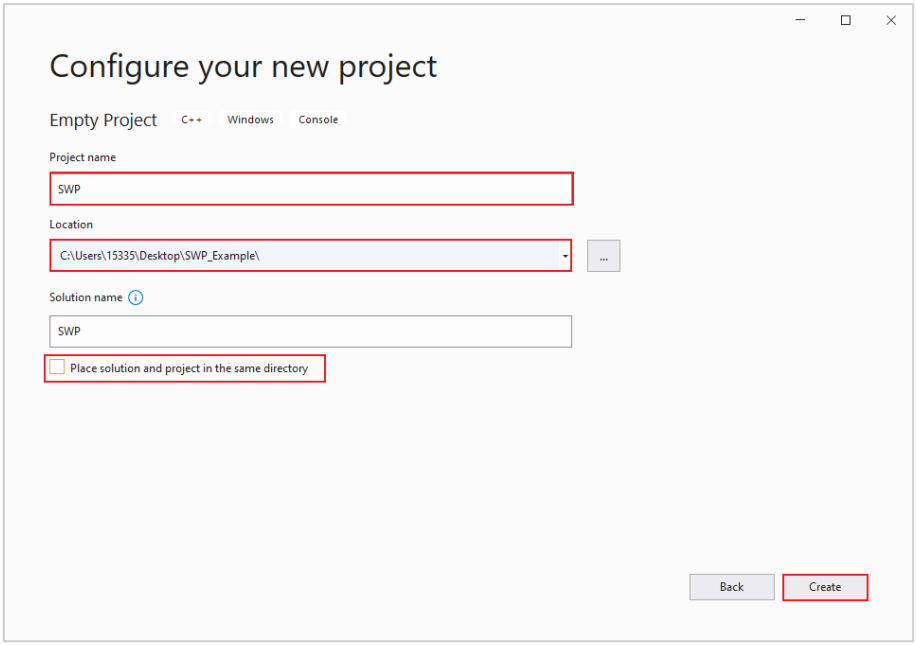
1. Open VS Studio 2019 and create a new project.



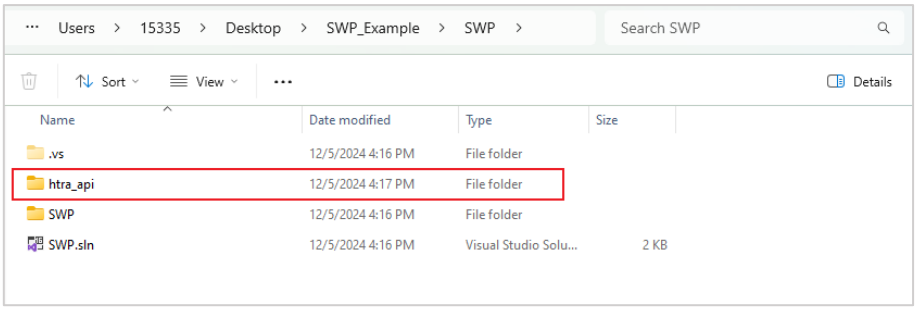
2. Select Empty Project and click Next.



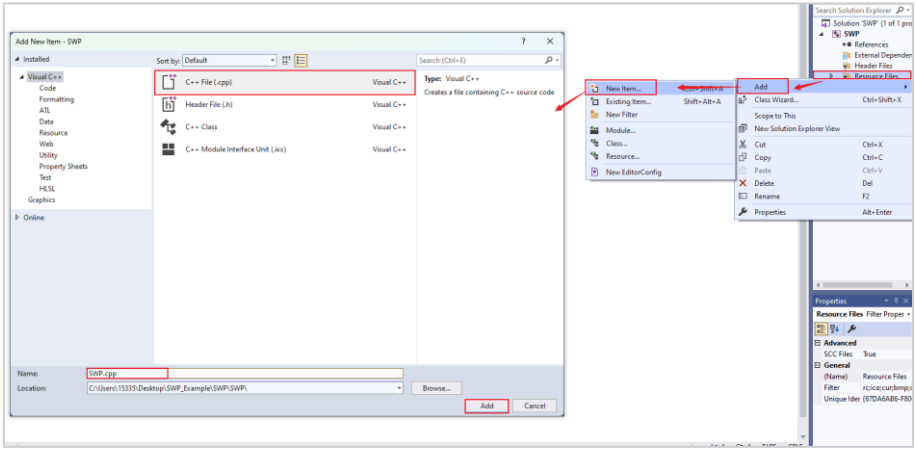
3. Fill in the project name and storage location, uncheck "Place solution and project in the same directory," and then click Create.



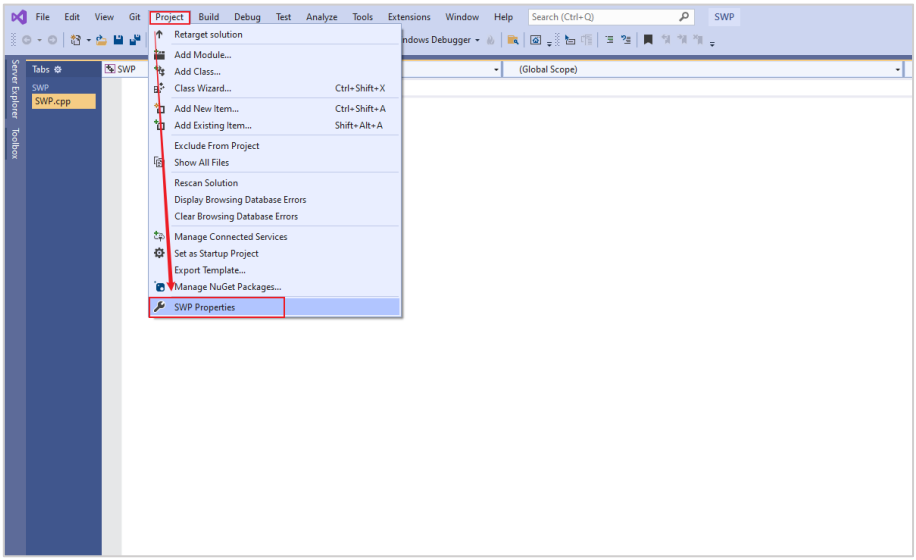
4. Once created, copy the htra_api folder from the delivery USB drive Windows\HTRA_API\x86 to the same level directory of the project (this example is for configuring an x86 architecture project; if you want to configure an x64 architecture project, copy the Windows\HTRA_API\x64\htra_api folder).



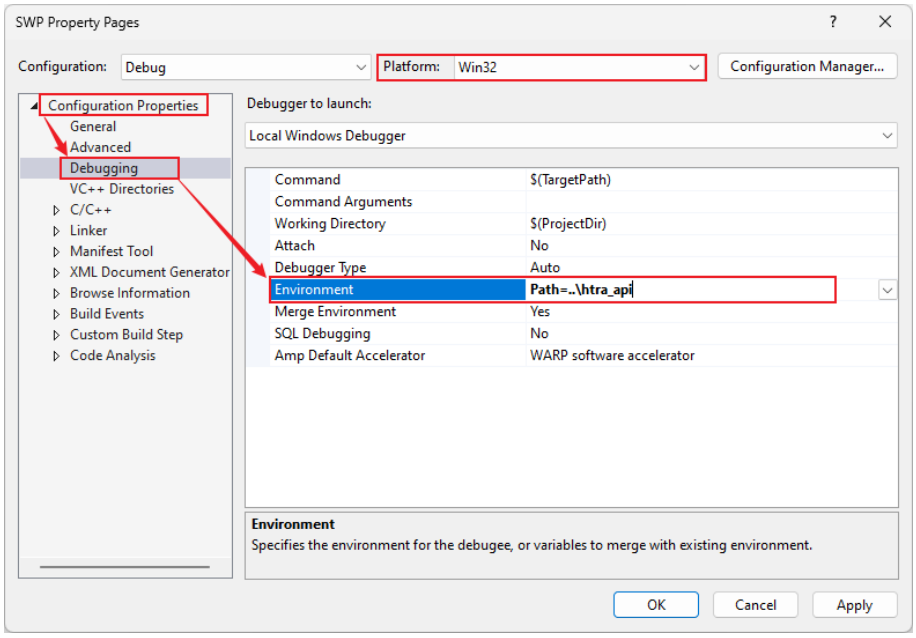
5. Double-click to open SWP.sln, and create a new SWP.cpp file in the source files.



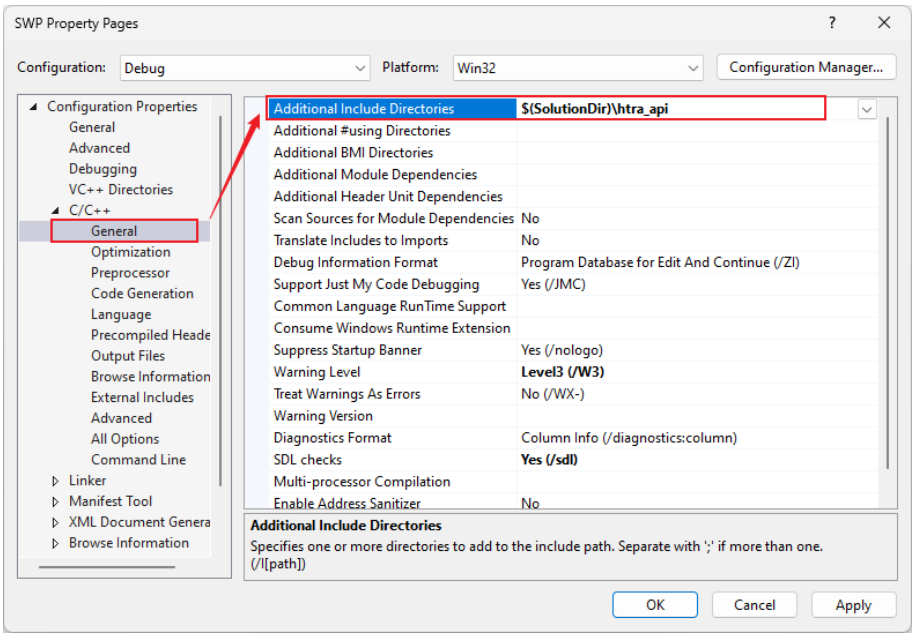
6. Click on "Project" in the menu bar and select "Properties."



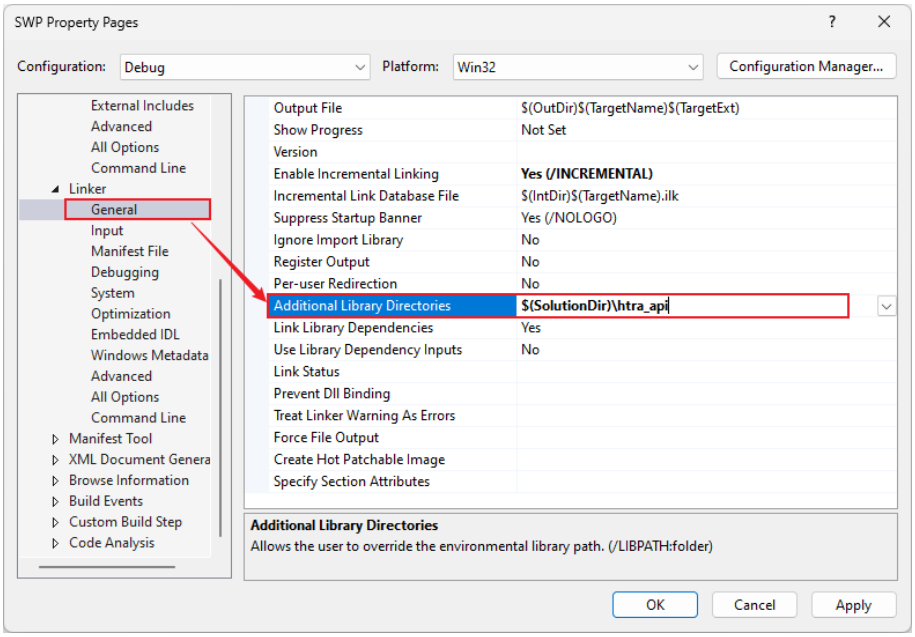
7. Select "Win32" for the configuration platform, and set the environment variable in Configuration Properties -> Debug to Path=..\htra_api (when configuring for the x64 architecture, select "x64" for the configuration platform; otherwise, steps 7-10 of the configuration process are the same as for the x86 architecture (Win32)).



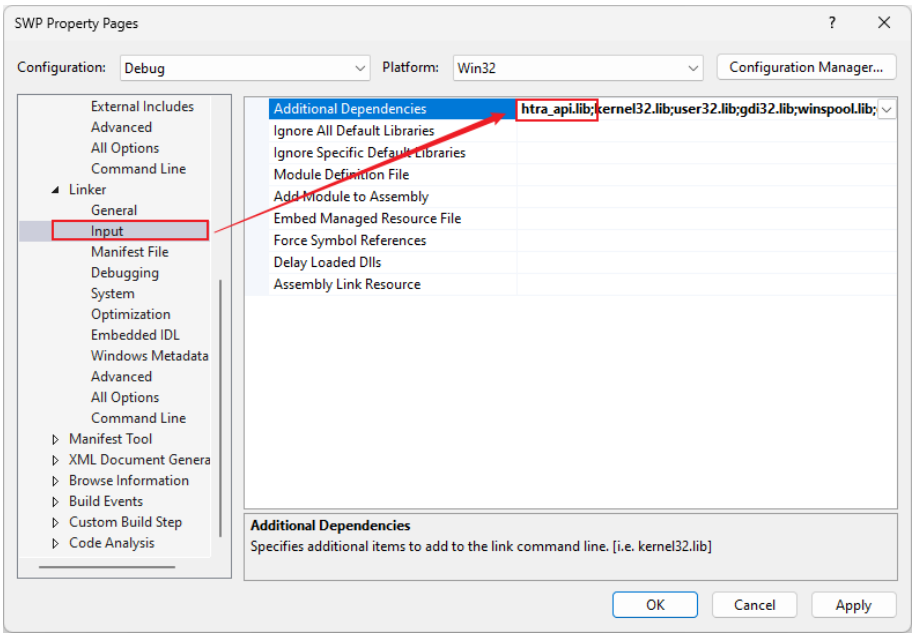
8. Set the Additional Include Directories in Configuration Properties -> C/C++ -> General to $(SolutionDir)\htra_api.



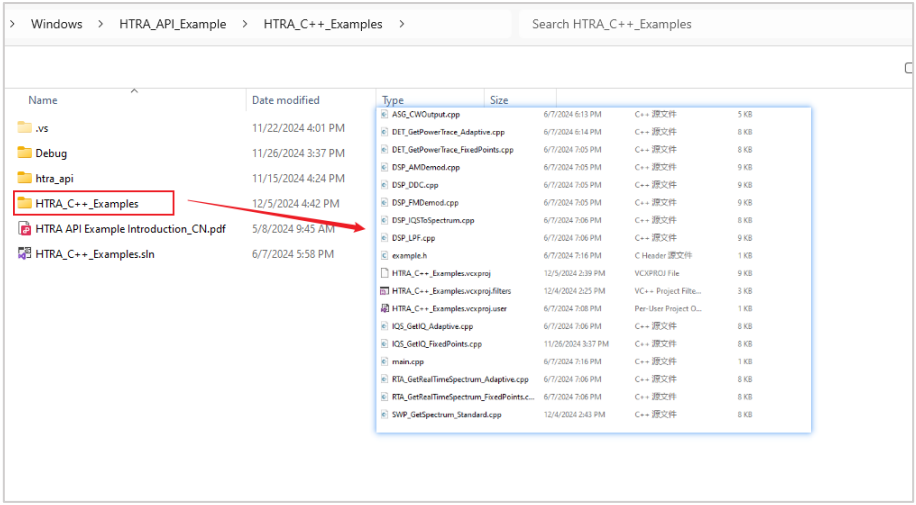
9. Set the Additional Library Directories in Configuration Properties -> Linker -> General to $(SolutionDir)\htra_api.



10. Add htra_api.lib to the Additional Dependencies in Configuration Properties -> Linker -> Input.



11. At this point, the environment configuration is complete, and programming development can begin. You can refer to the C/C++ examples included on the USB drive, specifically in Windows\HTRA_API_Example\HTRA_C++_Examples\HTRA_C++_Examples.

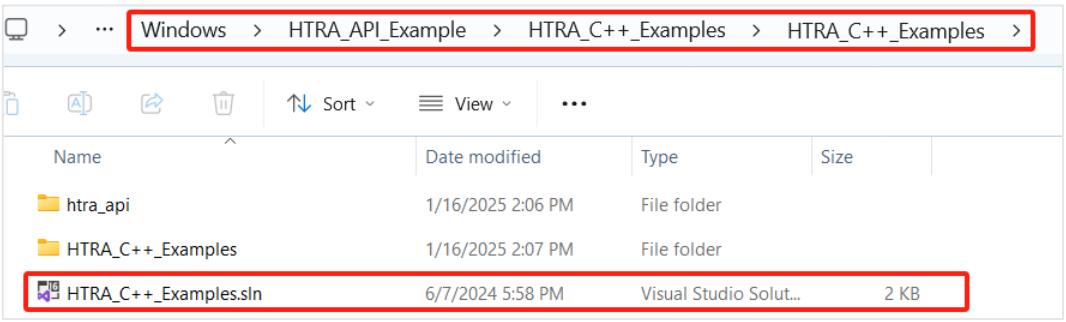


1.2 Usage Process for C++ Examples

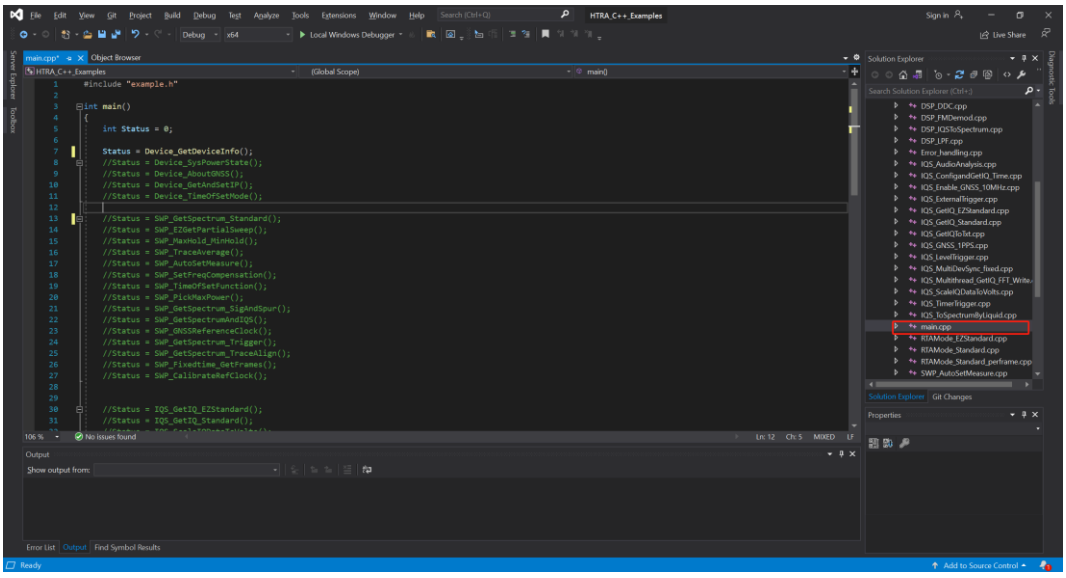
1.2.1 Usage of General C++ Examples

The usage process for general C++ examples included on the USB drive is as follows:

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.

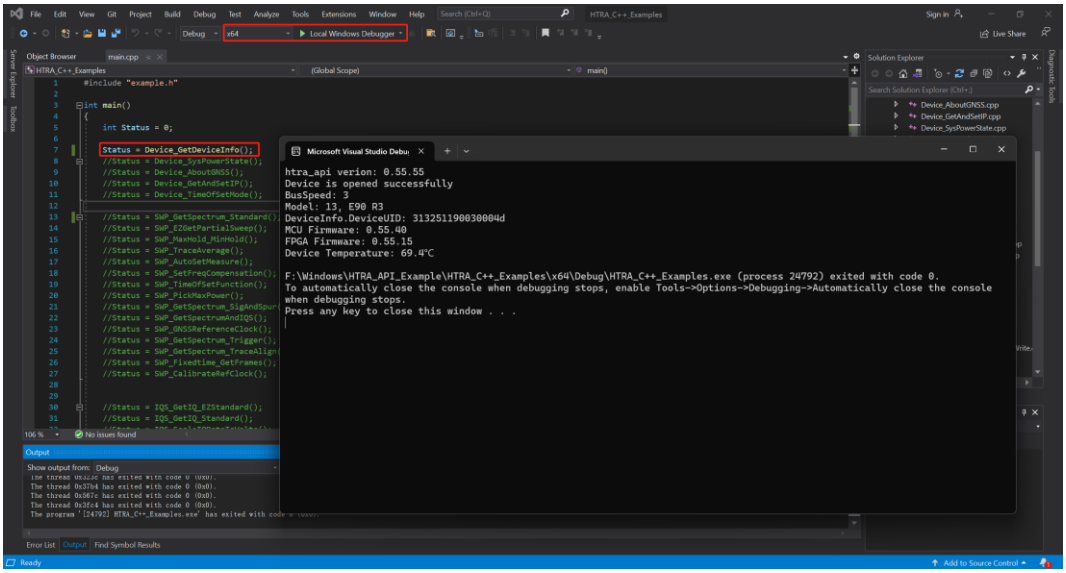


2. Click on the right side to access the HTRA_C++_Examples project and click on the main.cpp file within it.



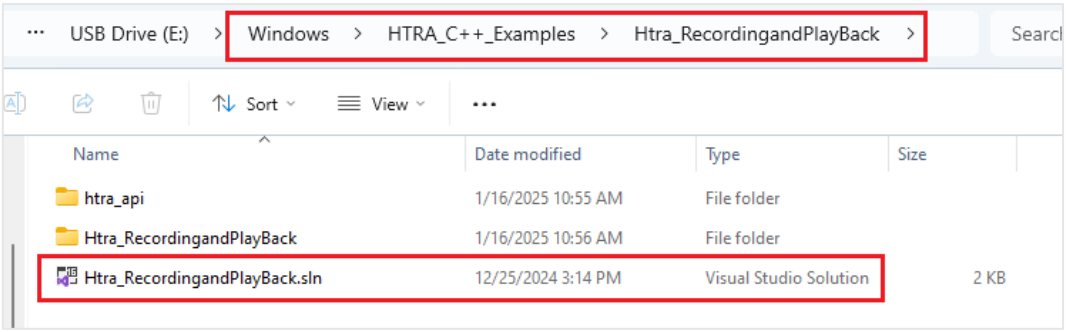
3. Each routine in the C++ example is encapsulated in a separate function. To use the example, simply uncomment it (multiple examples cannot be used simultaneously). For instance, when testing the Device_GetDeviceInfo routin

e, uncomment it, save, select the expected compilation architecture (both x86 and x64 are acceptable), and click run. The image shown indicates that the device is running normally.



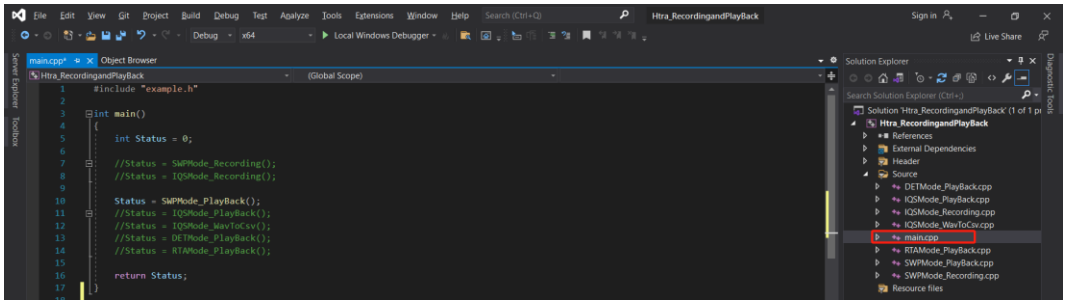
1.2.2 Usage of the recording and playback example

1. Open the solution Htra_RecordingandPlayBack.sln located in the folder Windows\HTRA_API_Example\Htra_RecordingandPlayBack on the provided USB drive using Visual Studio.



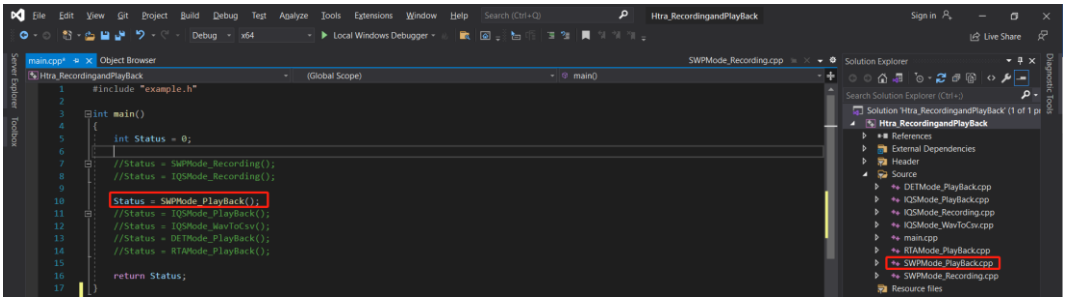
2. Click on the right side to access the Htra_RecordingandPlayBack projec t and click on the main.cpp file within it.

3. Each routine in the recording and playback example is encapsulated i n a separate function. To use the example, simply uncomment it (multiple ex amples cannot be used simultaneously).

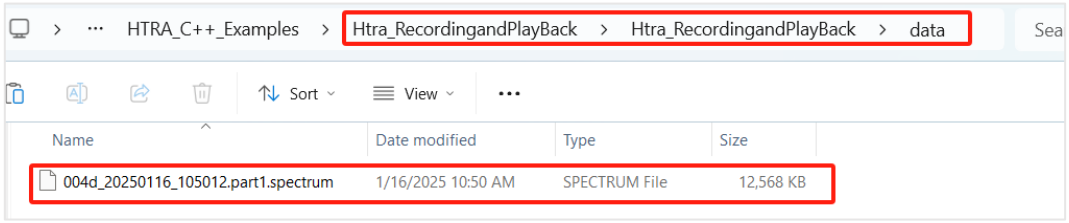


(1) Usage of the reading example

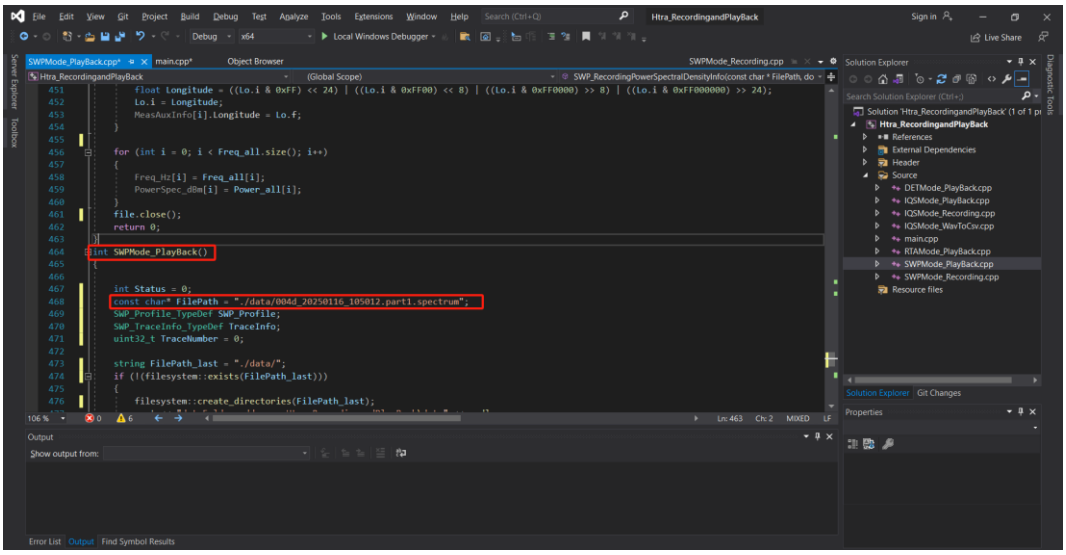
1) For example, when reading SWP mode stream disk data, uncomment the SWPMode_PlayBack function and save.



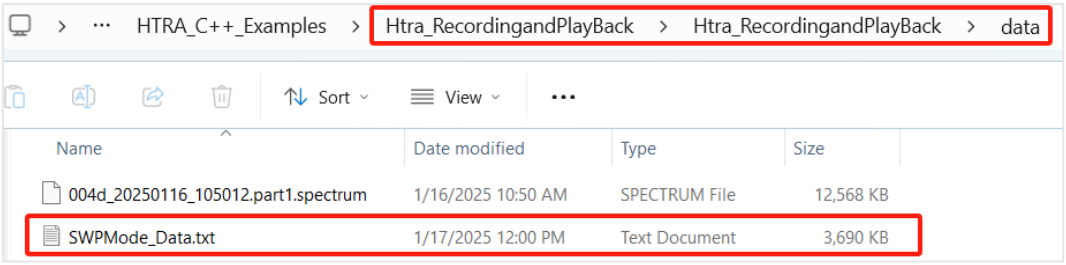
2) Place the recorded file data from SWP mode into the folder "Windows\HTRA_C++_Examples\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data".



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

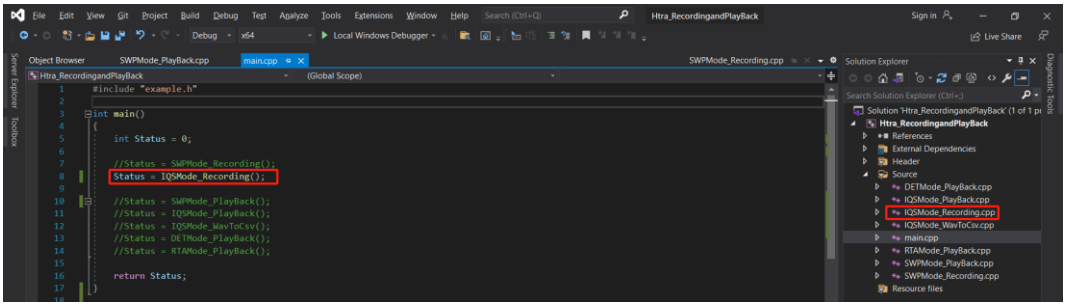


4) Running the program will generate the parsed data file "SWPMode_Data.txt" in the data folder under the SWP mode.

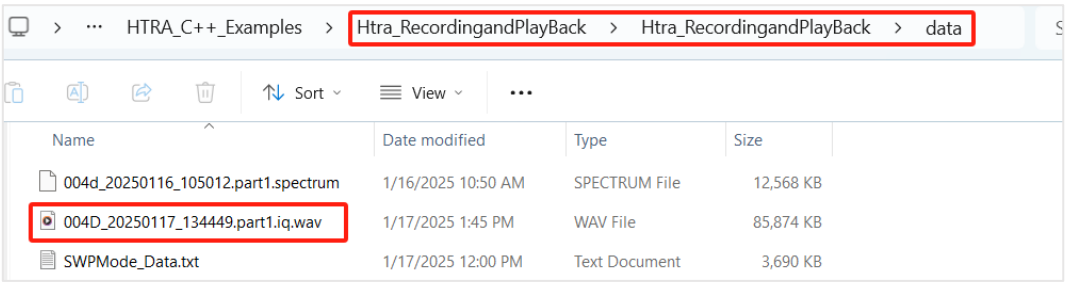


(2) Usage of the recording example

1) For example, when testing the IQSMode_Recording routine, uncomment and save.



2) Click to enter the IQSMode_Recording() function, configure the parameters, and run the program. You can find the recorded file data in the "Windows\HTRA_C++_Examples\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data" folder under the IQS mode.


HTRA_C++_Examples
Htra_RecordingandPlayBack
Htra_RecordingandPlayBack
data
Sort
View
Name
Date modified
Type
Size
004d_20250116_105012.part1.spectrum
1/16/2025 10:50 AM
SPECTRUM File
12,568 KB
004D_20250117_134449.part1.iq.wav
1/17/2025 1:45 PM
WAV File
85,874 KB
SWPMode_Data.txt
1/17/2025 12:00 PM
Text Document
3,690 KB

1.3 Device-related

1.3.1 Get device information

Device_GetDeviceInfo.cpp: Retrieves 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 stat e, which can be configured to normal operating state or RF powered down st ate (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_TimeOfSetMode.cpp: Get the time required for the current host computer to switch between different modes.

1.4 SWP mode

1.4.1 Standard spectrum acquisition

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

1.4.2 Simplified configuration mode

SWP_EZGetPartialSweep.cpp: Quickly acquire spectrum data using a simplified configuration.

1.4.3 Maximum and minimum hold

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

1.4.4 Average Trace

SWP_TraceAverage.cpp: Average processing of the acquired trace.

1.4.5 Automatic Configuration Measurement

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

1.4.6 Frequency Compensation

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

1.4.7 Function execution time, scanning speed, and throughput

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

1.4.8 Obtain spectrum peak values

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

1.4.9 Signals and Spurious

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

1.4.10 Simultaneous Acquisition of Spectrum and IQ

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

1.4.11 Reading SWP Stream Disk Data from SAStudio4

SWPMode_PlayBack.cpp: This can read the recorded file data in SWP mo de from SAStudio4 and write the read spectrum data into SWP Mode_Data.tx t.

1.4.12 Using GNSS 10MHz Reference Clock

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

1.4.13 External Trigger Mode

SWP_GetSpectrum_Trigger.cpp: This obtains spectrum data when the trigg

er source is set to external trigger.

1.4.14 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.

1.4.15 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.

1.4.16 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.

1.5 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 Simplified configuration mode

IQS_GetIQ_EZStandard.cpp: Quickly obtain IQ data using a simplified configuration.

1.5.3 IQ data converted to voltage V units

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

1.5.4 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.

1.5.5 IQ to Spectrum Data

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

1.5.6 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.

1.5.7 FM Demodulation

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

1.5.8 AM Demodulation

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

1.5.9 Digital Downconversion

DSP_DDC.cpp: Resamples the obtained IQ data.

1.5.10 Digital Low-Pass Filter

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

1.5.11 Audio Analysis

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

1.5.12 Read the IQS stream disk data from SAStudio4

IQSMode_PlayBack.cpp: Parses the recorded file data in IQS mode from SAStudio4 and writes the read spectrum data into the IQS Mode_Data.txt file.

1.5.13 Record IQ data in .wav format

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

1.5.14 .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.

1.5.15 Streaming and reading IQ data

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

1.5.16 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.

1.5.17 GNSS 1PPS trigger

IQS_GNSS_1PPS.cpp: Configures the trigger source to be the 1PPS signal provided

by the internal GNSS system.

1.5.18 IQS multi-device synchronization

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

1.5.19 External Trigger

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

1.5.20 Timer Trigger

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

1.5.21 Level Trigger

IQS_LevelTrigger.cpp: Configure the trigger source as level trigger.

1.5.22 Using GNSS 10MHz Reference Clock

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

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 Simplified configuration mode

DETMode_EZStandard.cpp: Quickly obtain detection data through a simplified configuration.

1.6.3 Read the DET stream disk data of SAStudio4

DETMode_PlayBack.cpp: Read the recorded files in DET mode from SAStudio4 and output the detection data to the DETMode_Data.txt file.

1.6.4 Pulse detection (to be opened later)

1.7 RTA mode

1.7.1 Obtain real-time spectrum data for a fixed number of 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 Simplified configuration mode

RTAMode_EZStandard.cpp: Quickly obtain real-time spectrum data through simple configuration.

1.7.3 Read the RTA stream disk data of SAStudio4

RTA_Mode_PlayBack.cpp: Read the recorded file data in RTA mode from SAStudio4, while being able to specify reading a certain packet of data, and write the read spectrum data to the RTAMode_Data.txt file.

1.7.4 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

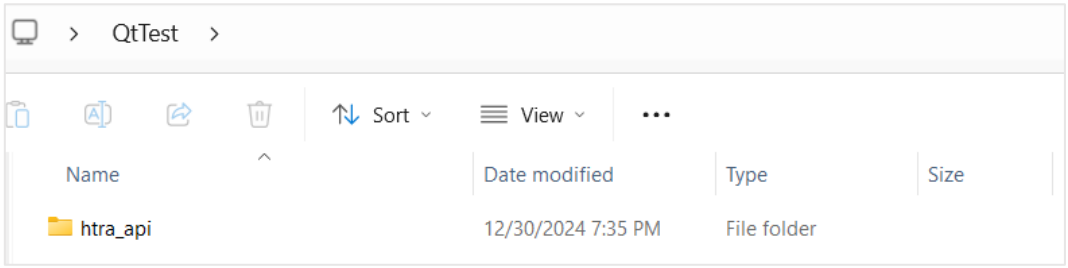
ASG_SignalOutput.cpp: Output single tone/sweep/power scan signals as ne eded.

2. Digital Demodulation (optional)

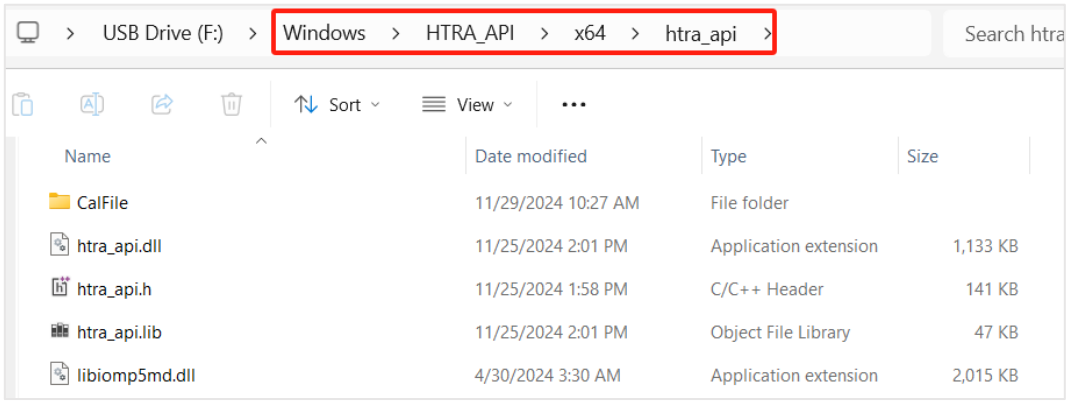
3. Qt

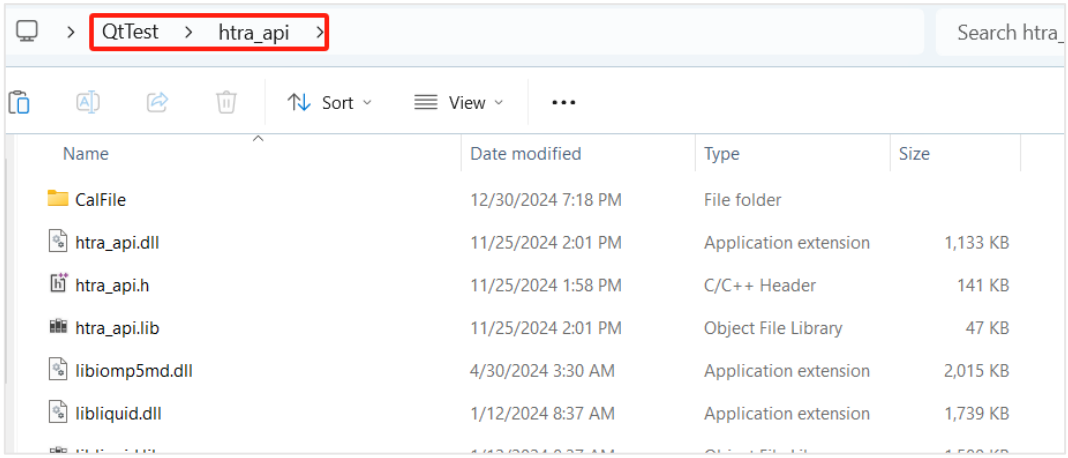
3.1 Configure Development Environment

1. As shown in the figure, first create a new folder to store the entire project (taking QtTest as an example, it is recommended not to use a Chinese path), and then create a htra_api folder within the folder to store the dynamic link libraries and calibration files.

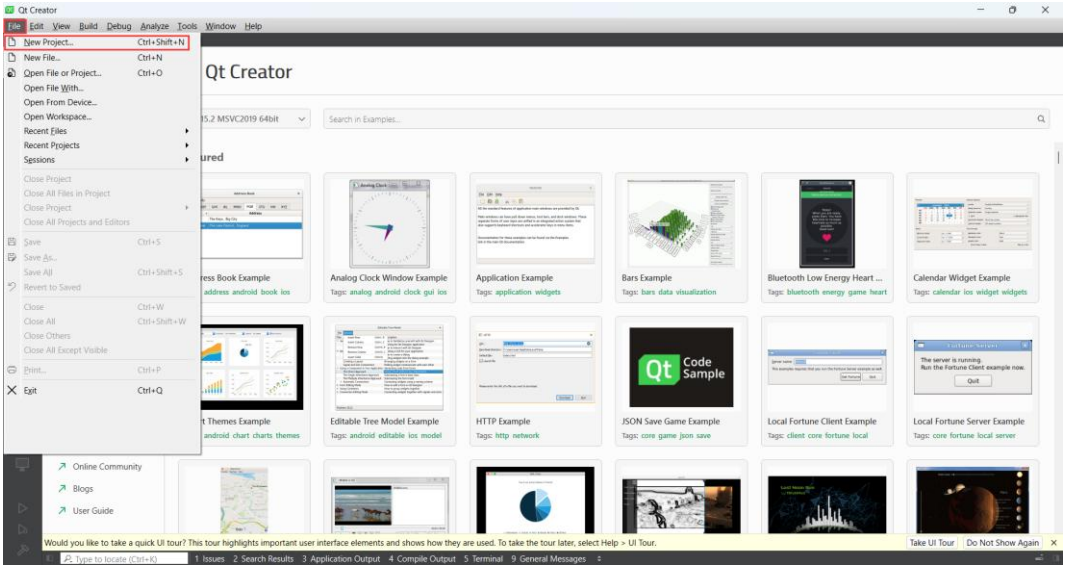


2. Copy all files from the Windows\HTRA_API\x64\htra_api folder on the USB drive to the newly created QtTest\htra_api folder (taking the x64 architecture program as an example; for the x86 architecture program, simply copy the corresponding architecture's libraries).

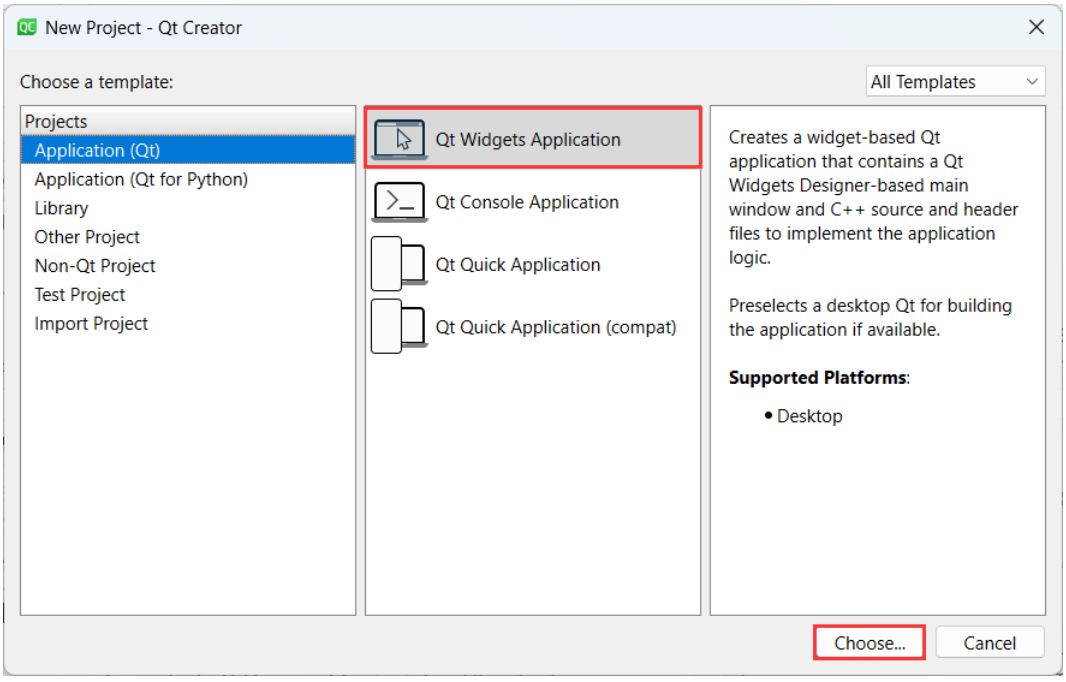




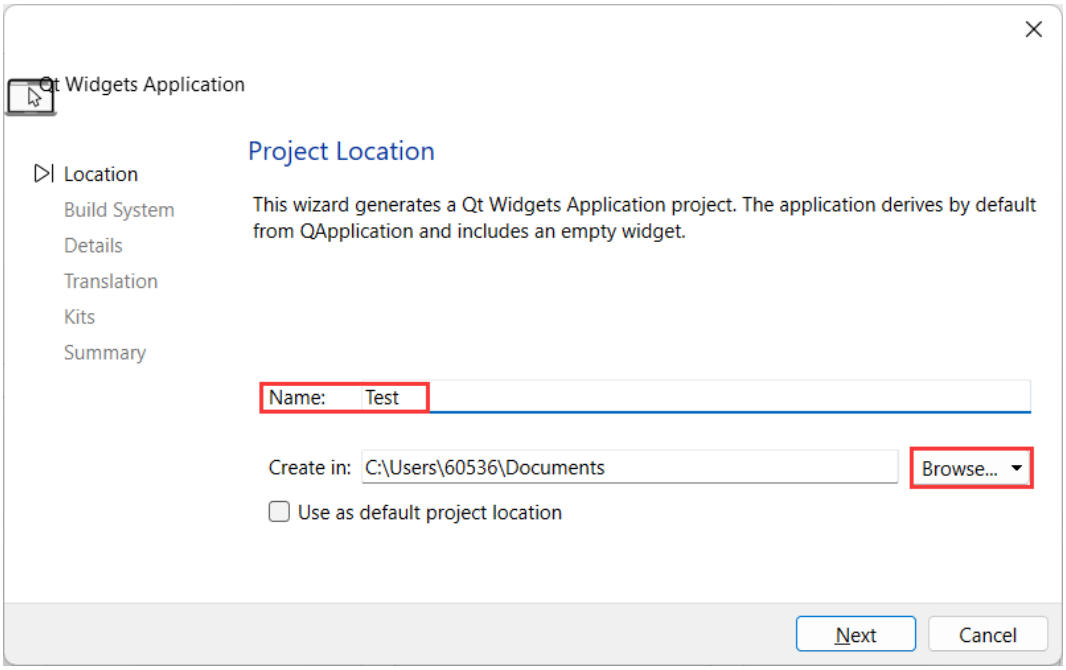
3. Open Qt Creator, click on File, and select New File or Project.



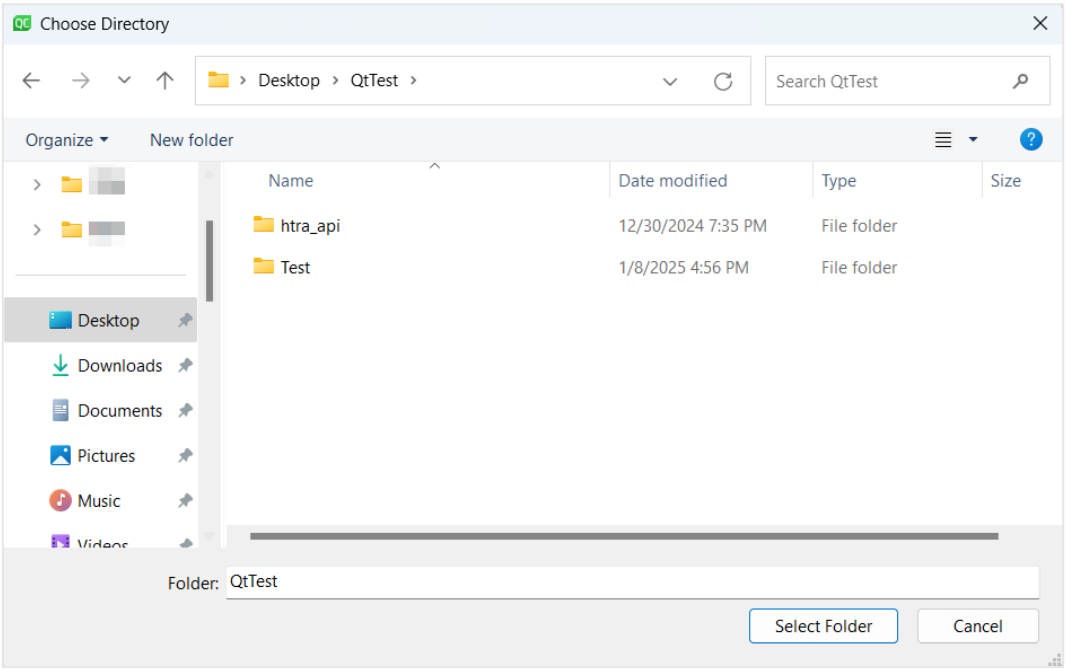
4. Select Create Form Application.



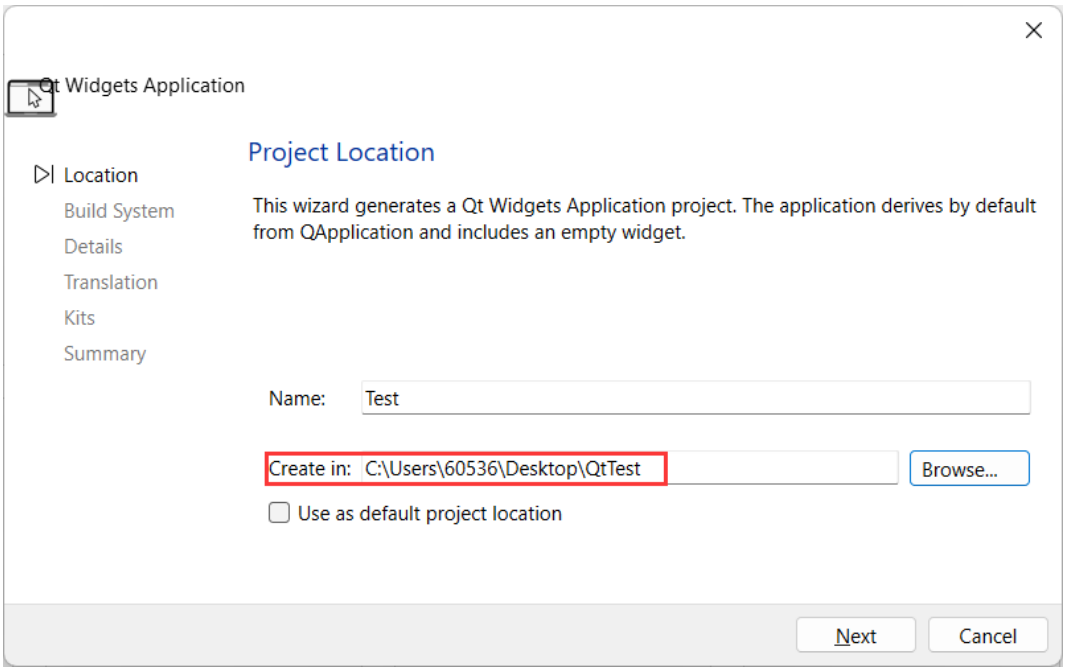
5. After filling in the project name, click Browse to change the project path.



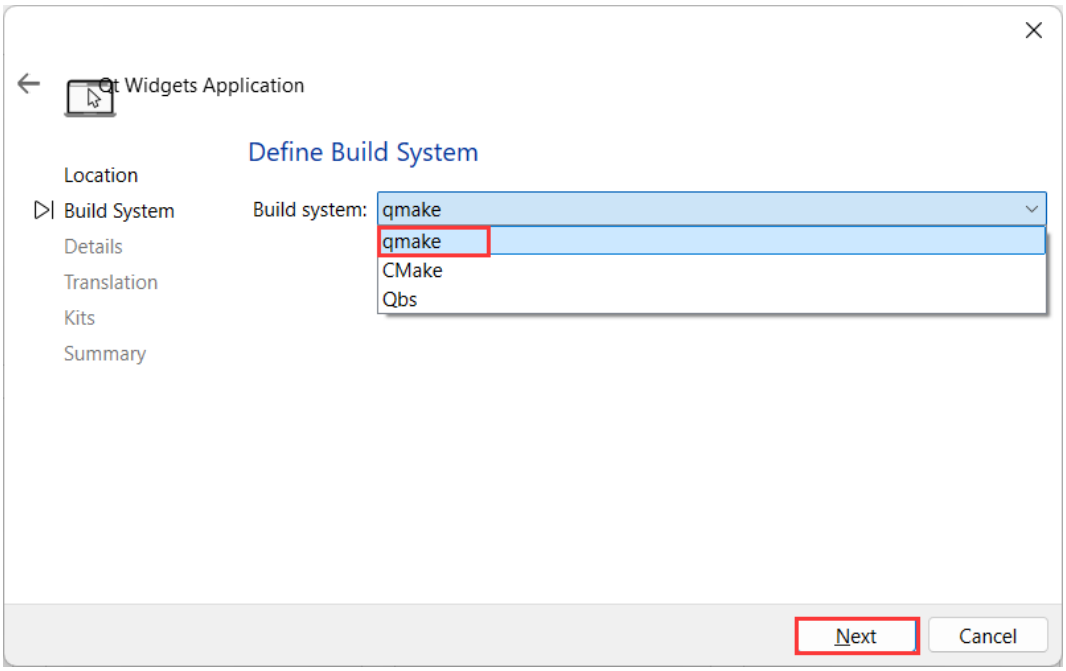
6. Select the directory as the QtTest address created in the first step and click Open.



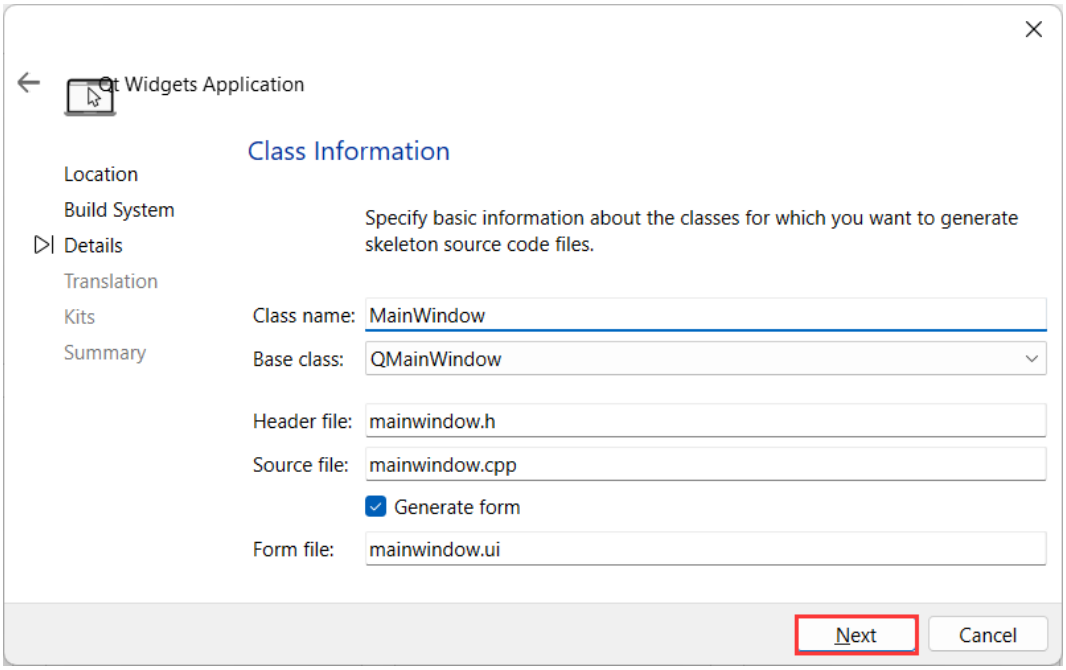
7. After selecting the path, click Next.



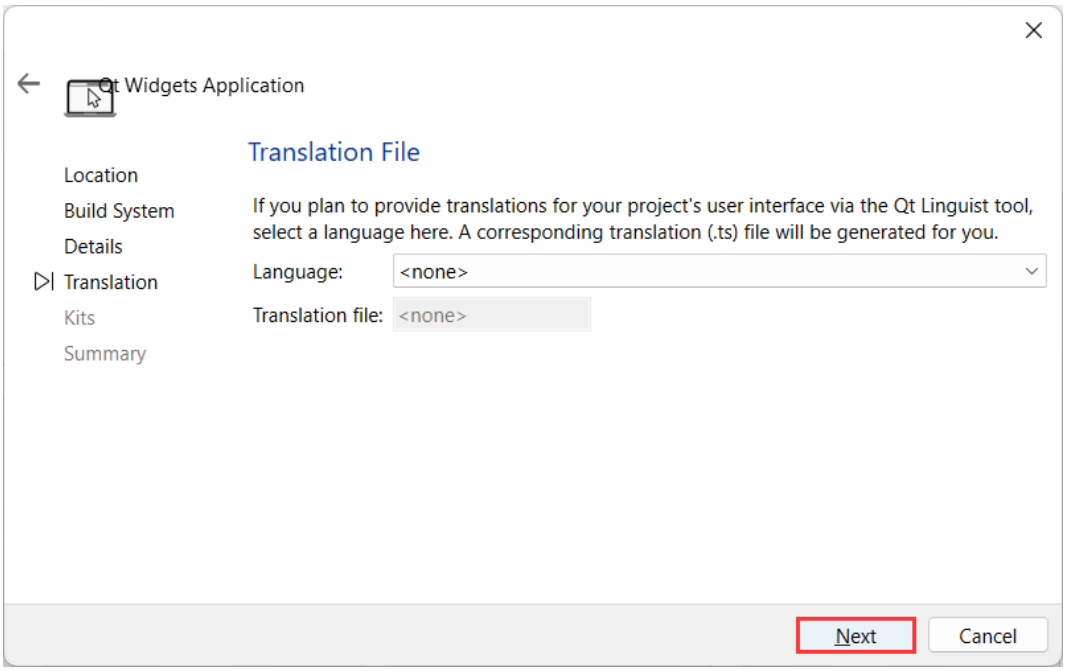
8. Select qmake and click Next to continue.



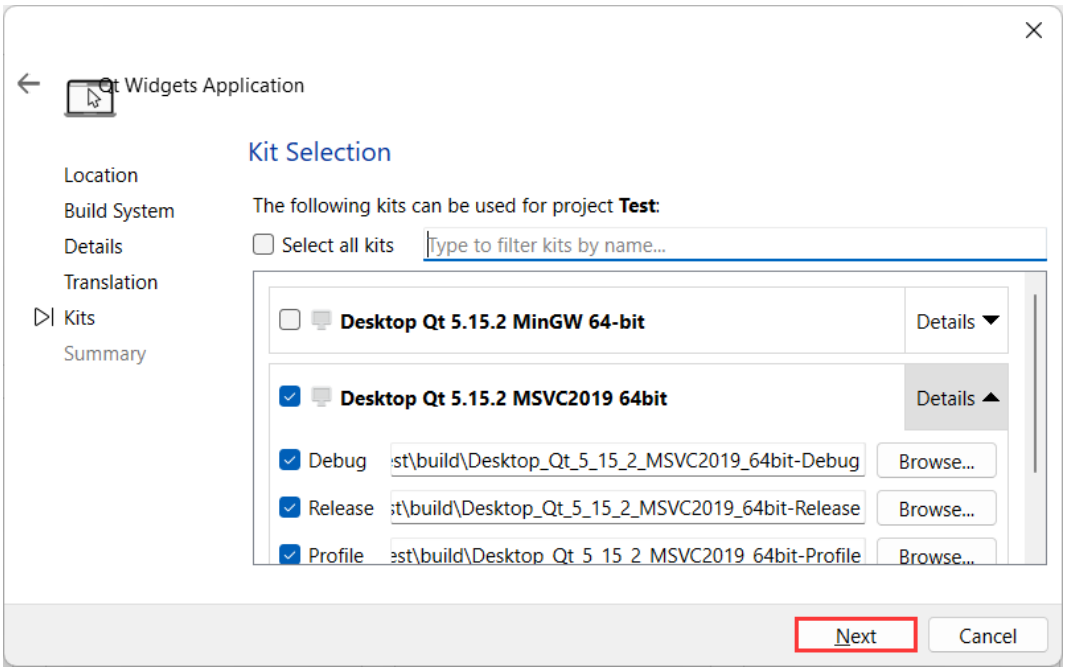
9. Click Next to continue.



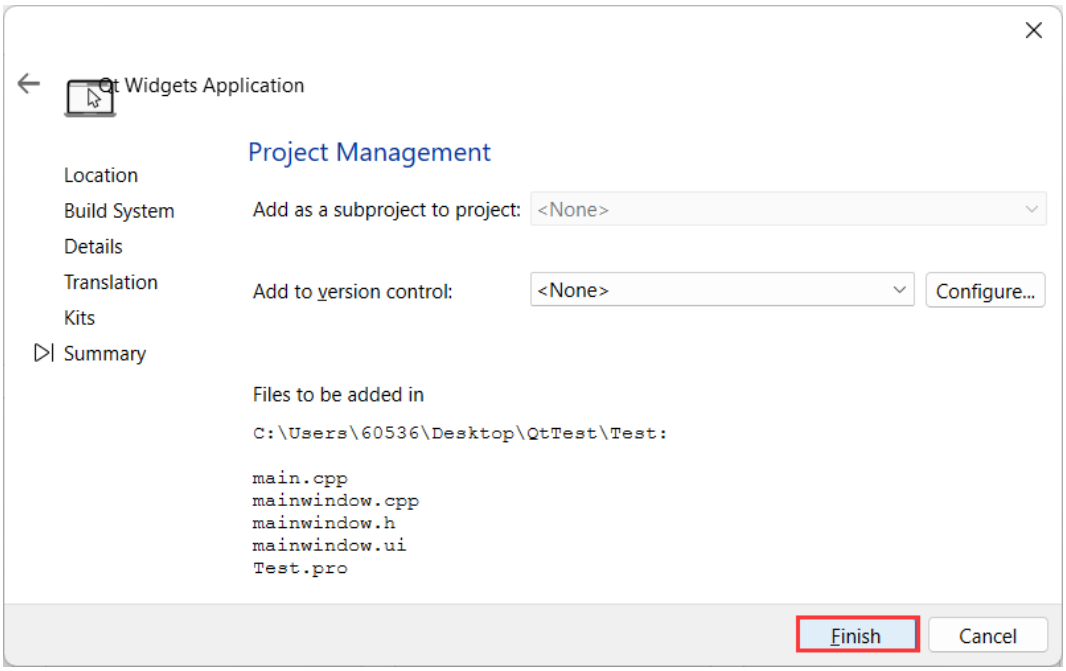
10. Click Next to continue.



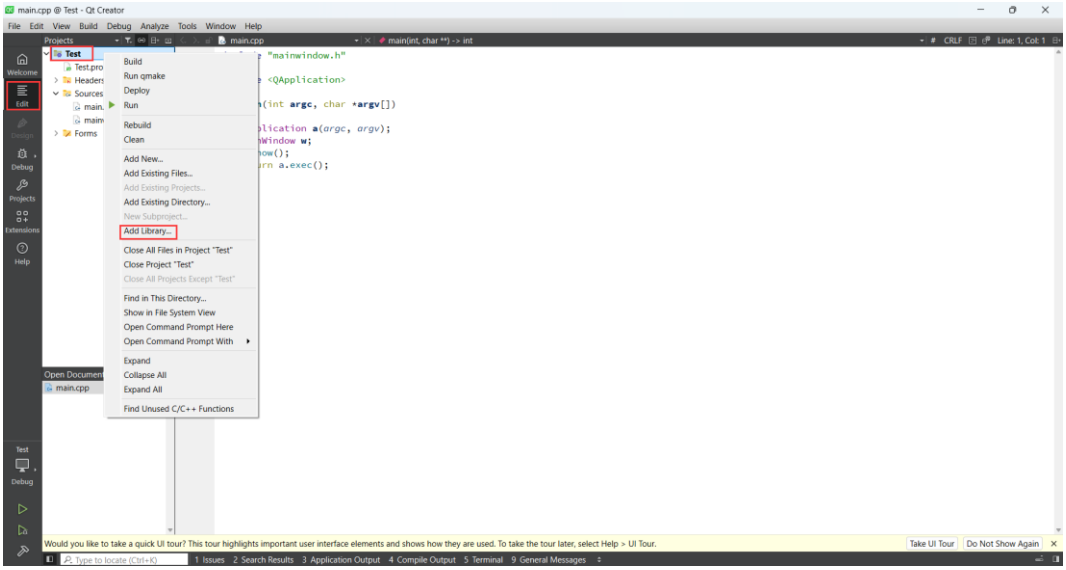
11. Select a build environment for the project and click Next to continue.



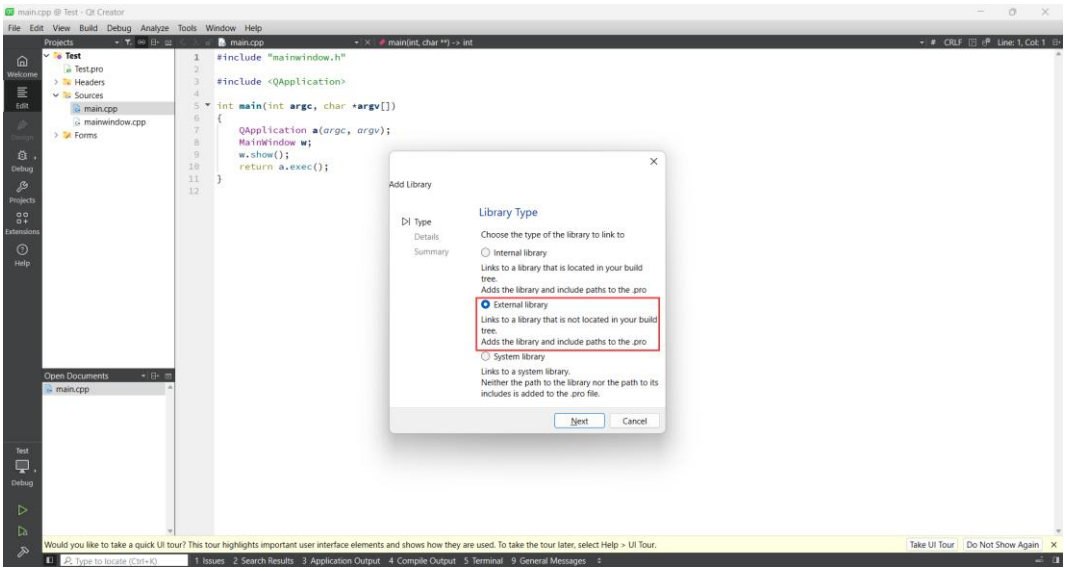
12. Click Finish to create the project.



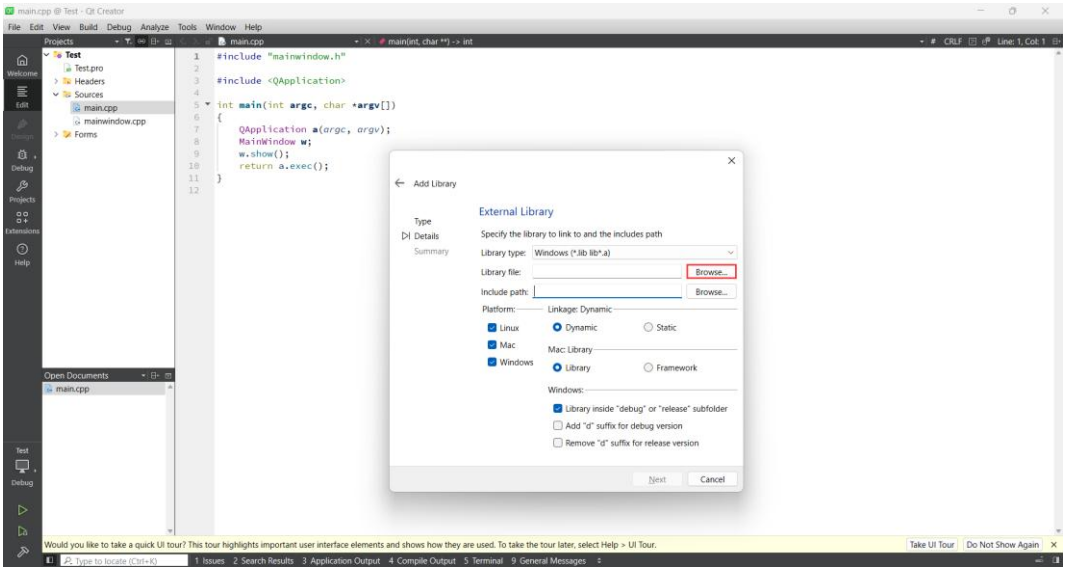
13. Click Edit, right-click the Test project, and click Add Library.



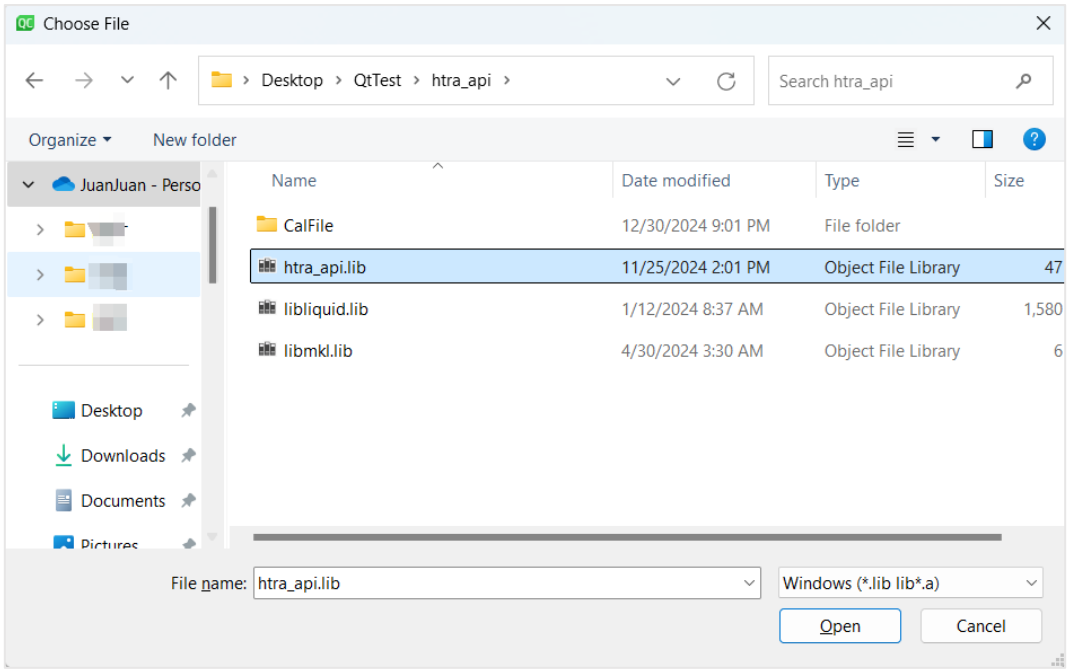
14. Select External Library and click Next.



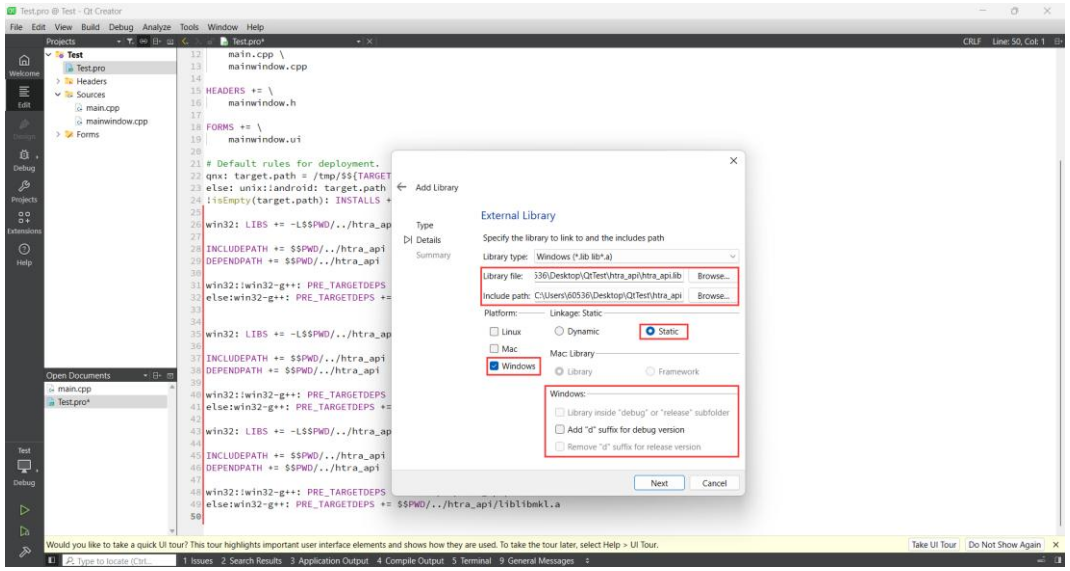
15. Click Browse Library File.



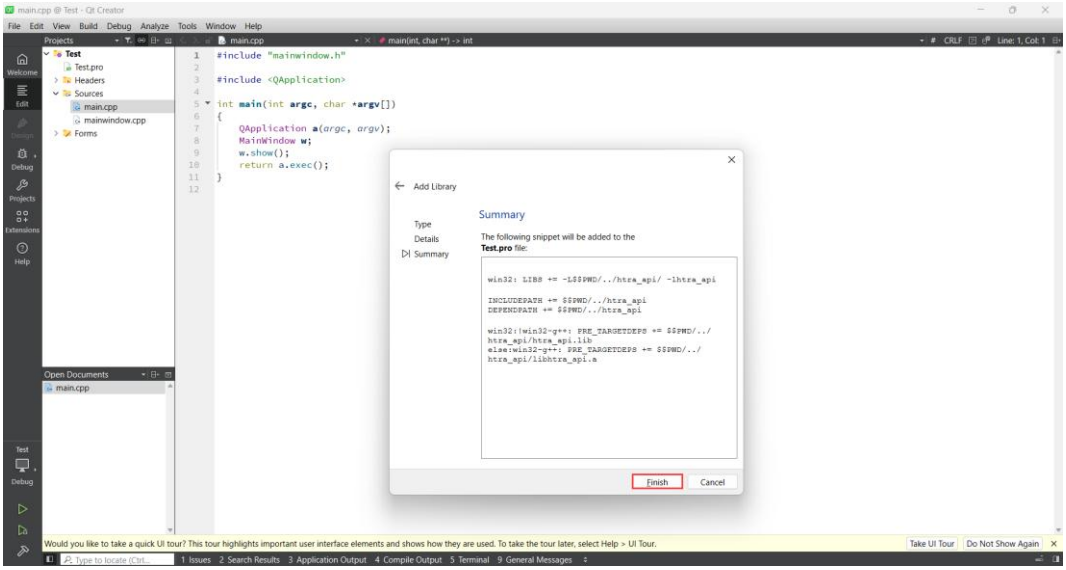
16. Select the htra_api.lib library in QTest\htra_api and click Open.



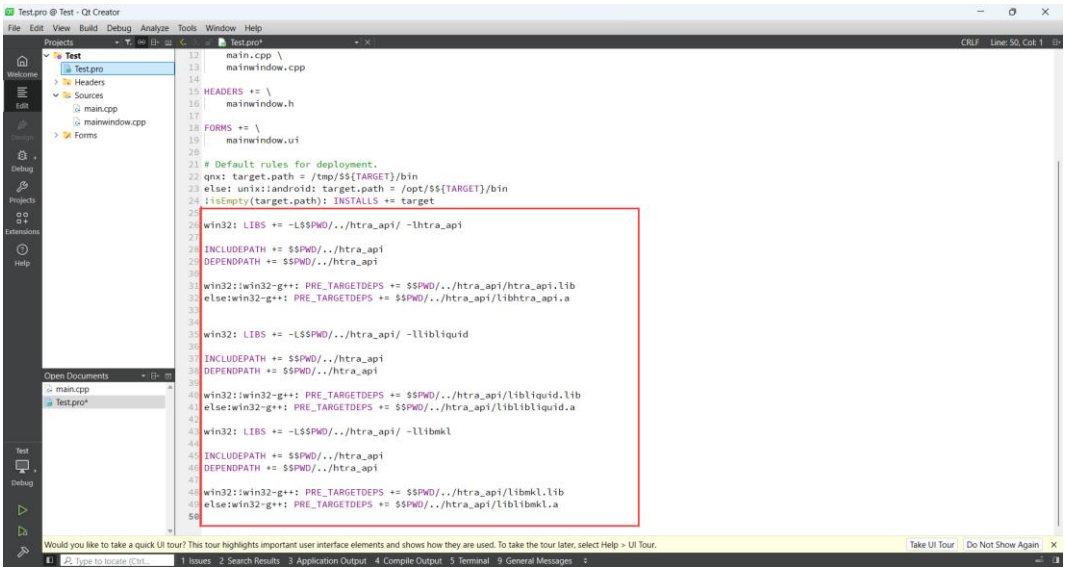
17. Uncheck all options in Windows, click on Static Library, select the Windows platform, and click Next.



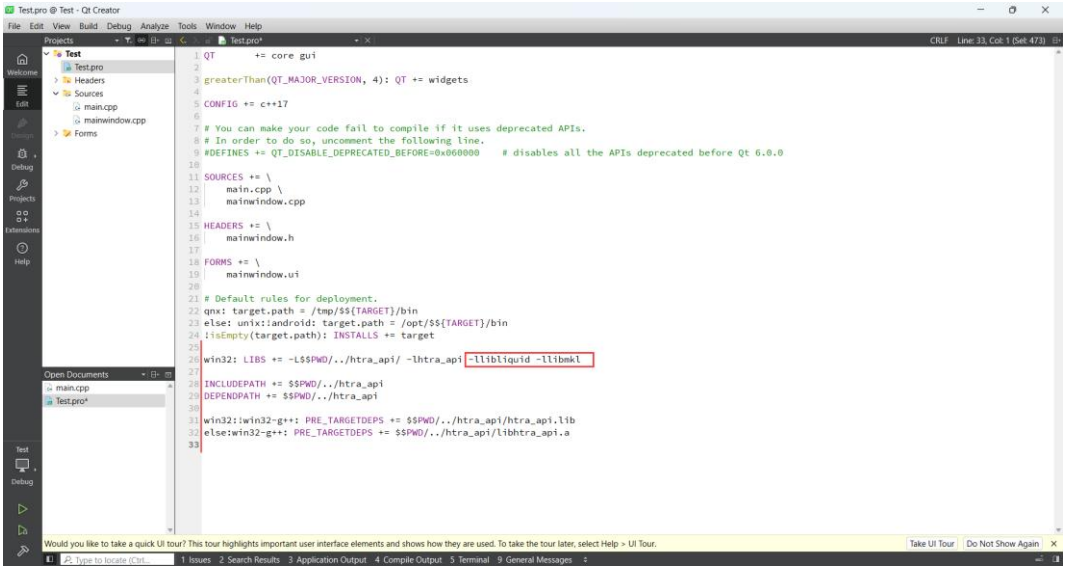
18. Click Finish to add the external library.



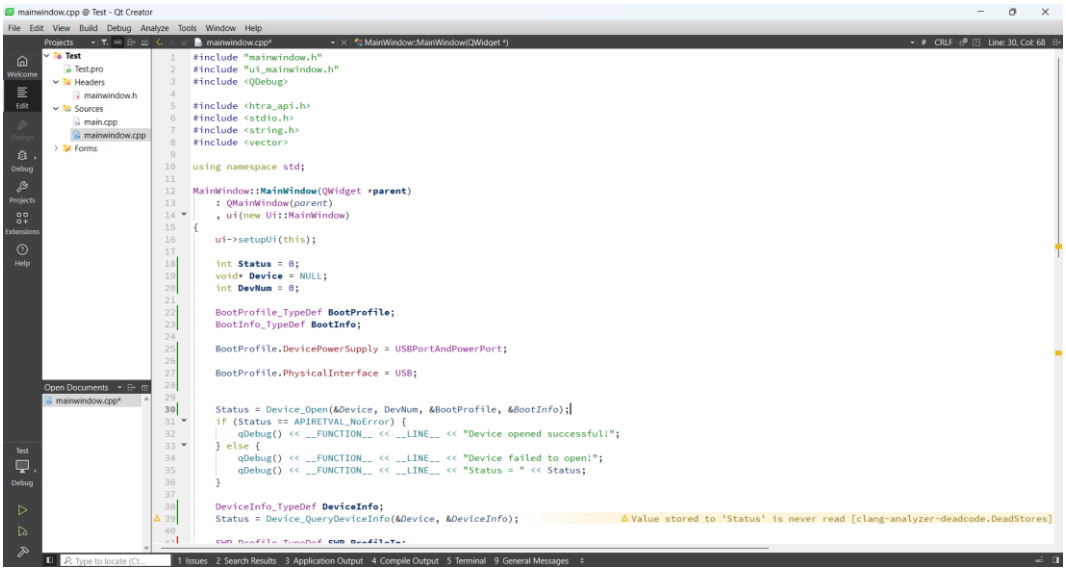
19. Continue to add the libliquid.lib and libmkl.lib libraries following the same steps as before.



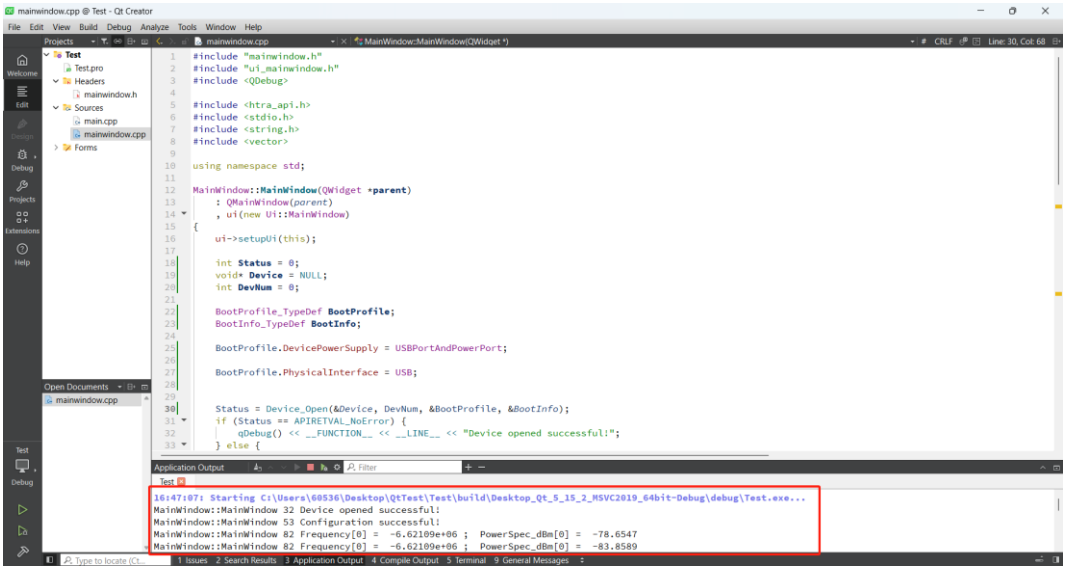
You can also directly add -llibliquid and -llibmkl after -llibhtra_api as shown below.



20. Save the Test.pro file, and then you can write the code normally.



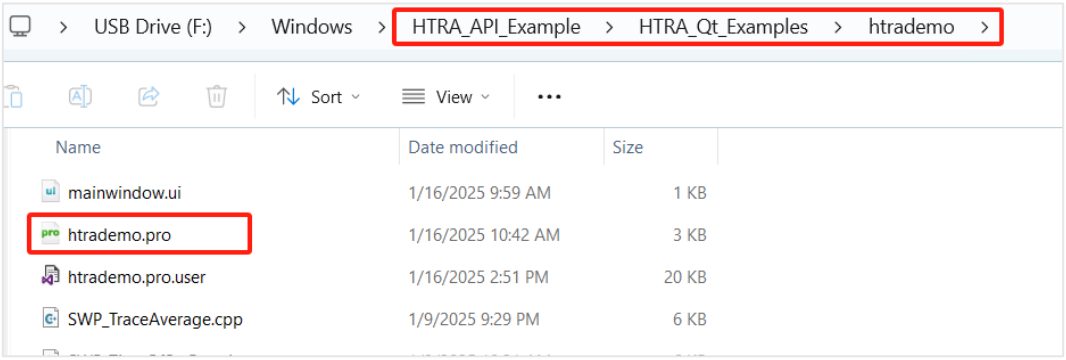
21. After writing the code, click Run. The device should function normally as shown in the figure.


mainwindow.cpp @ Test - Qt Creator
File Edit View Build Debug Analyze Tools Window Help
Projects
Test
Test.pro
Headers
mainwindow.h
Sources
main.cpp
mainwindow.cpp
Forms
Open Documents
mainwindow.cpp
#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 {
Application Output
Test
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
1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 Terminal 9 General Messages

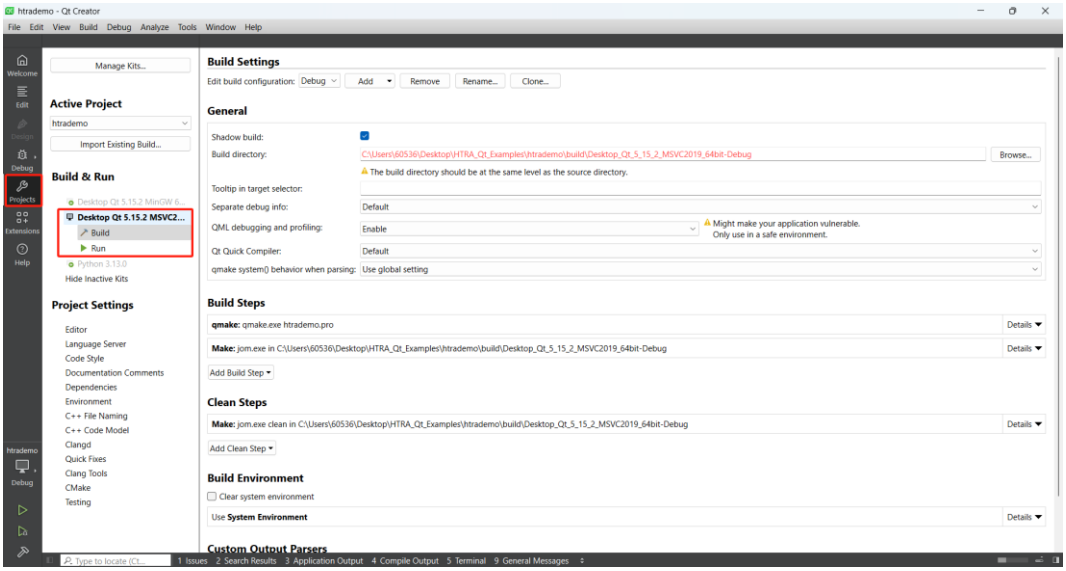
3.2 Qt Example Usage Process

The usage process for the Qt examples included in the USB drive is as follows:

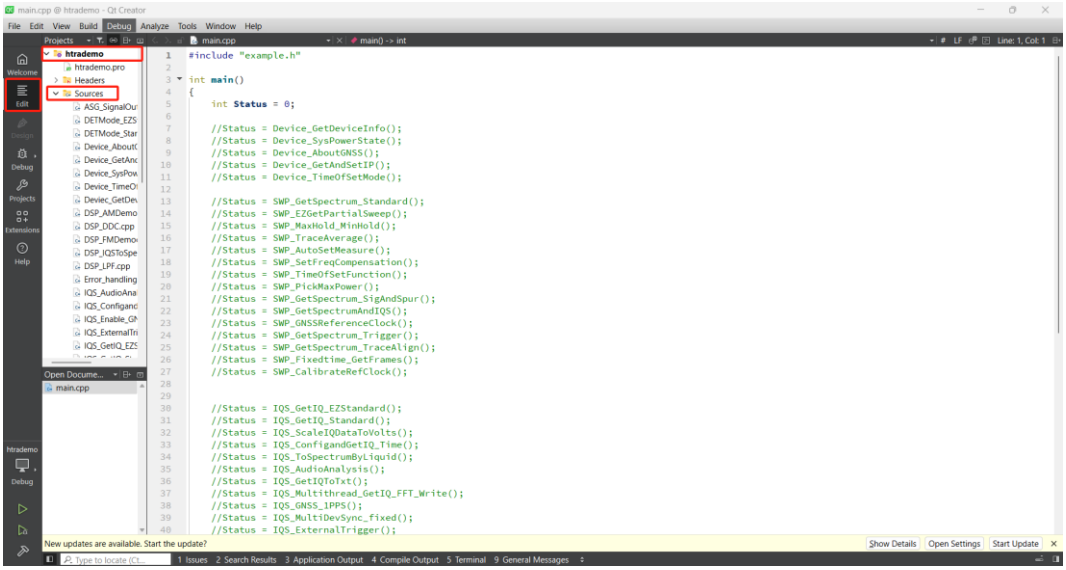
1. As shown in the figure, use Qt Creator to open the htrademo.pro file located in the Windows\HTRA_API_Example\HTRA_Qt_Examples\htrademo folder on the USB drive (please ensure the project path does not contain Chinese characters).



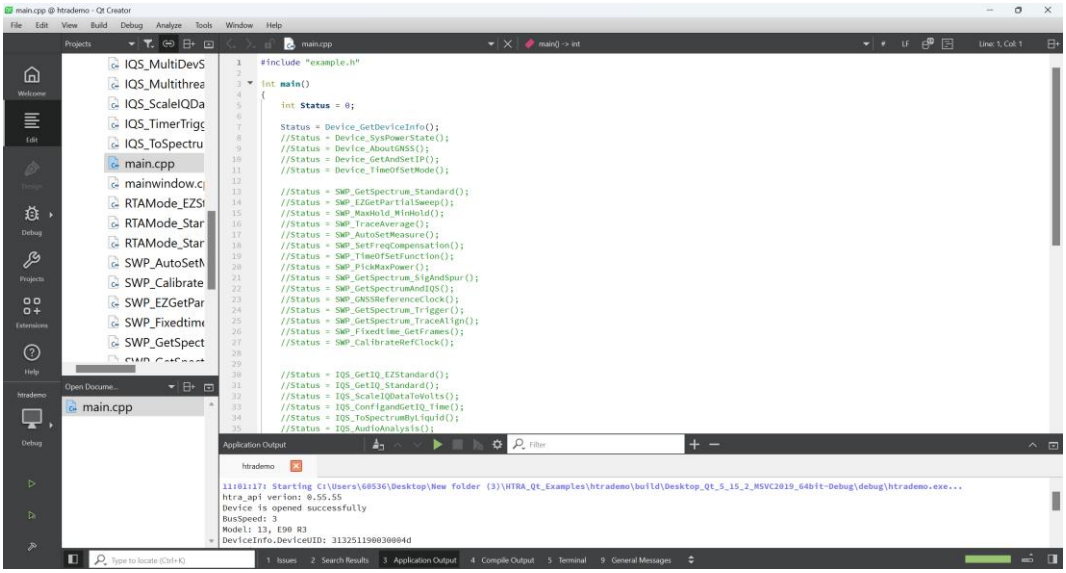
2. Click on the project to configure a build environment for it.



3. After configuring the build environment, click on Edit, and then click on main.cpp in the Sources folder of the htrademo project.



4. Since each example in the Qt provided examples is encapsulated in a separate function, you can simply uncomment the desired example when using it (multiple examples cannot be used simultaneously). For instance, when testing the Device_GetDeviceInfo example, uncomment it, save, and click run. The image shown indicates that the device is operating normally.



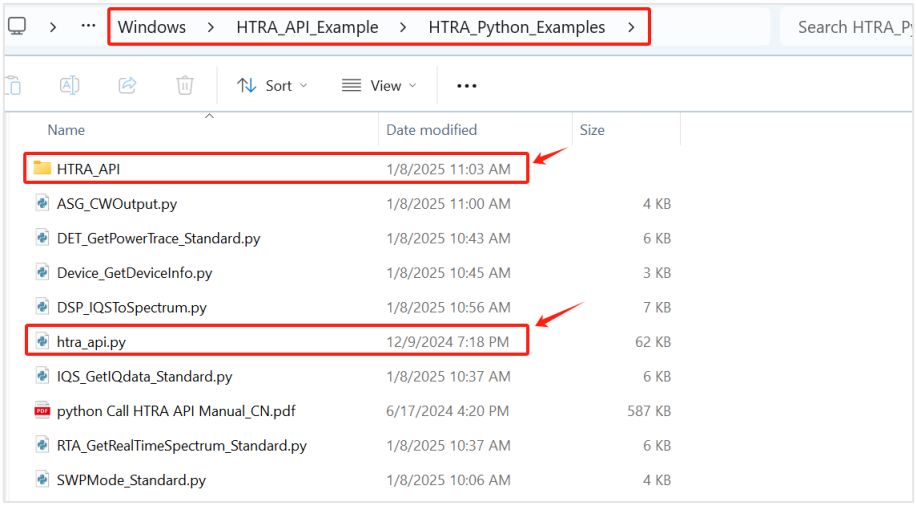
3.3 Qt Example Description

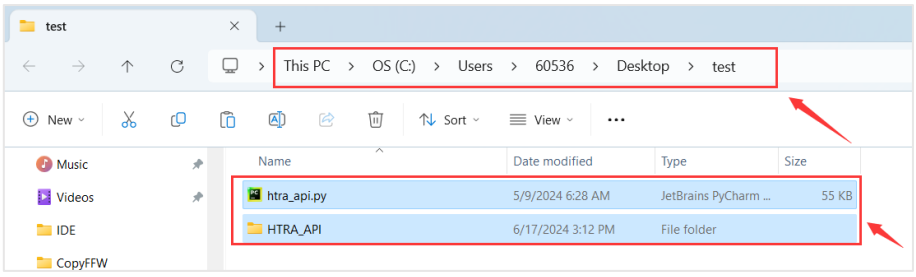
The purpose of the Qt examples can be directly referenced in the C/C++ examples section.

4. Python

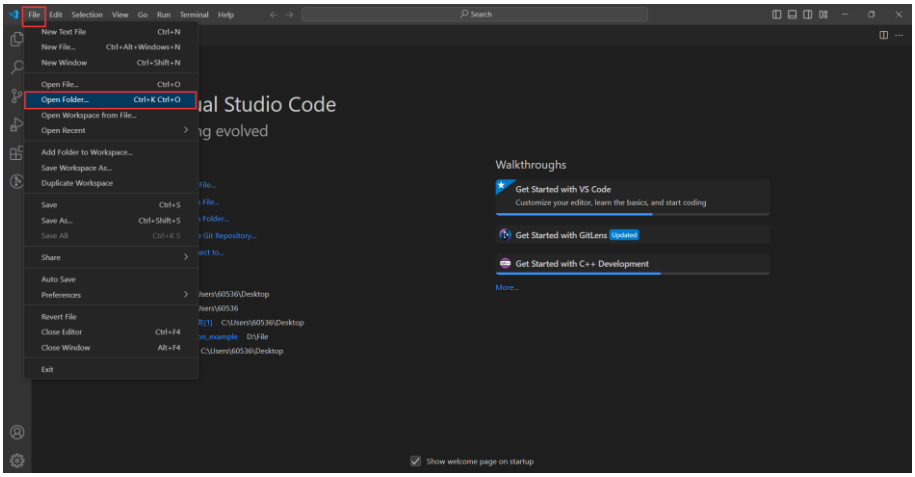
4.1 Configure Development Environment

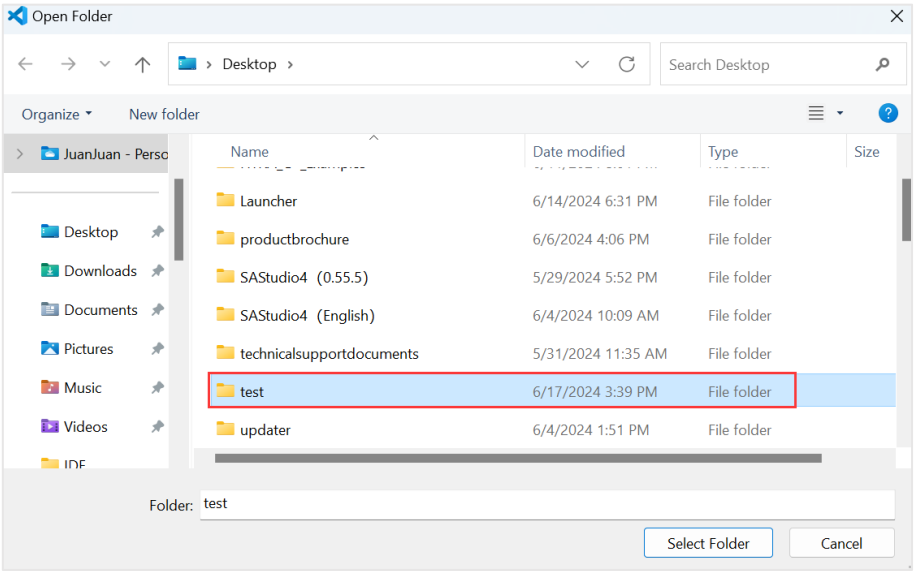
1. Create a folder on the desktop and name it, for example, test. Open the USB drive and copy the "HTRA_API" folder and "htra_api.py" file from "\Windows\HTRA_API_Example\HTRA_Python_Examples" on the USB drive to the newly created folder.



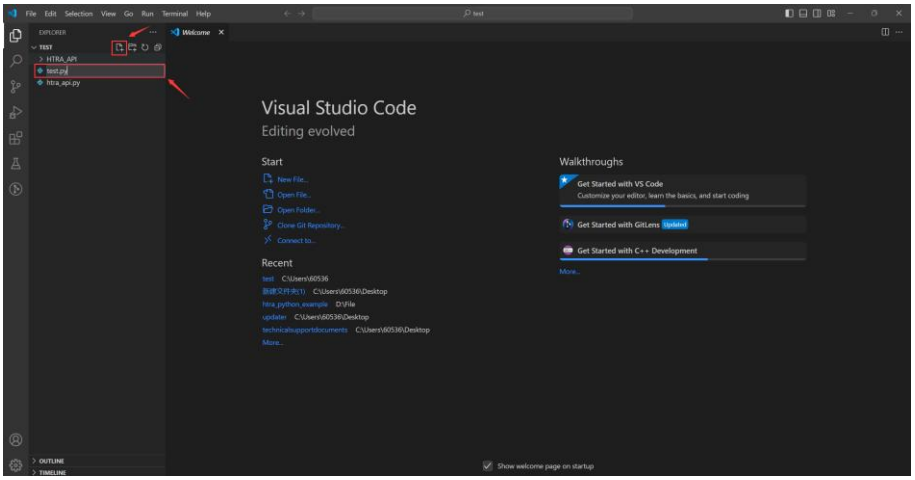


2. Open Visual Studio Code, click on File and then Open Folder, and open the folder you just created.

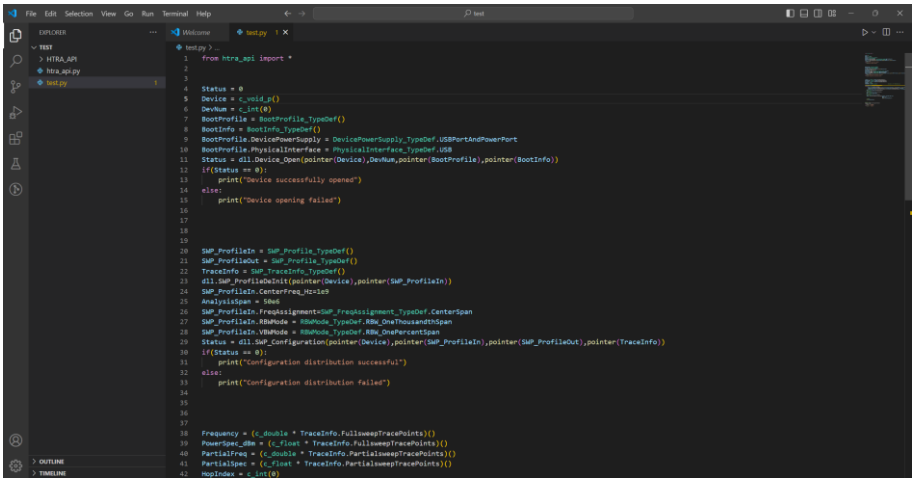




3. Create a new Python file.



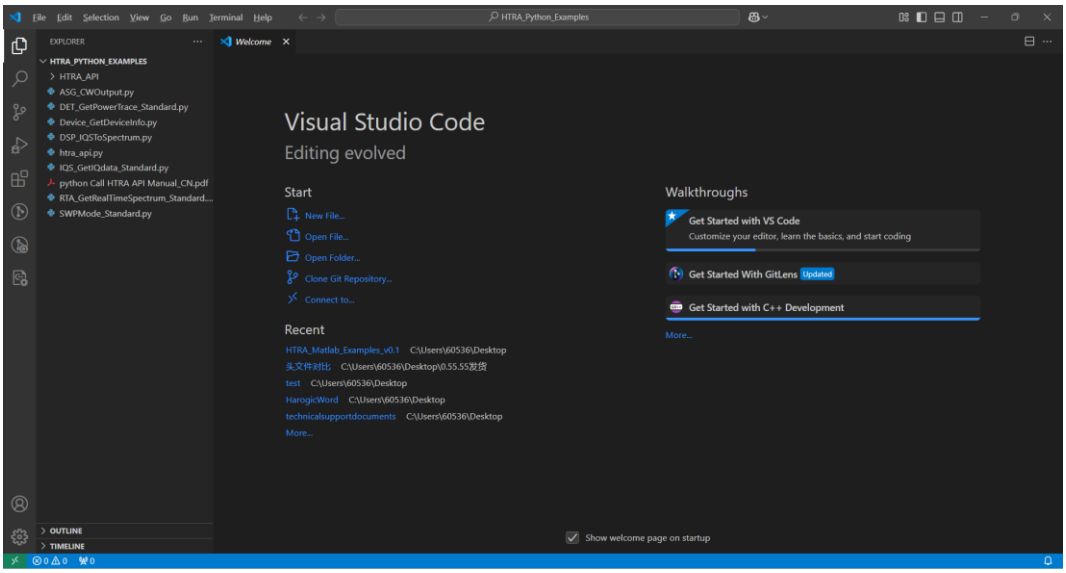
4. Write code normally. You can refer to the Python examples included in the USB drive, specifically in the folder "\Windows\HTRA_API_Example\HTRA_Python_Examples" for the project.


File Edit Selection View Go Run Terminal Help
EXPLORER
TEST
HTRA_API
htra_api.py
test.py
Welcome
test.py
OUTLINE
TIMELINE
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 = 80e6
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)

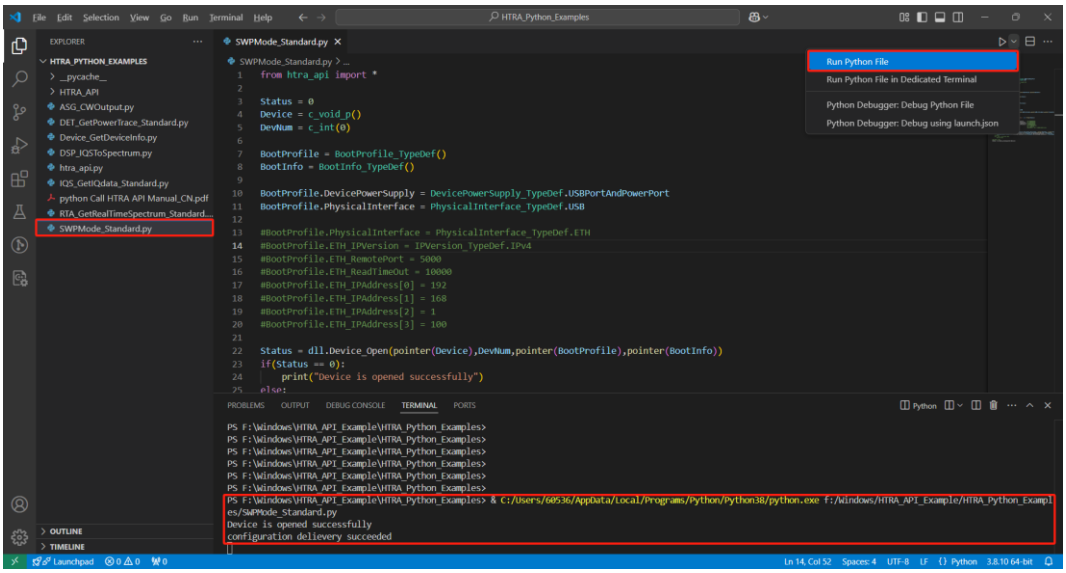
4.2 Python Example Usage Process

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

1. As shown in the figure, open the entire project using vscode or another compiler from the USB drive provided: Windows\HTRA_API_Example\HTRA_Python_Examples. The htra_api.py file in the project is the mapping file for the dynamic link library in Python, while the other files are example programs (the role of the examples will be described in subsequent chapters).



2. Select any example program, configure the Python environment for it, and run the program directly. For instance, when using the SWPMode_Standard.py example, the device is shown to be operating normally as illustrated.



4.3 Python Example Description

4.3.1 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.

4.3.2 Obtain Standard Spectrum Data

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

4.3.3 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.

4.3.4 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.

4.3.5 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.

4.3.6 IQ to Spectrum Data

DSP_IQSToSpectrum.py: Convert the IQ data obtained in IQS mode into spectrum

data.

5. Matlab

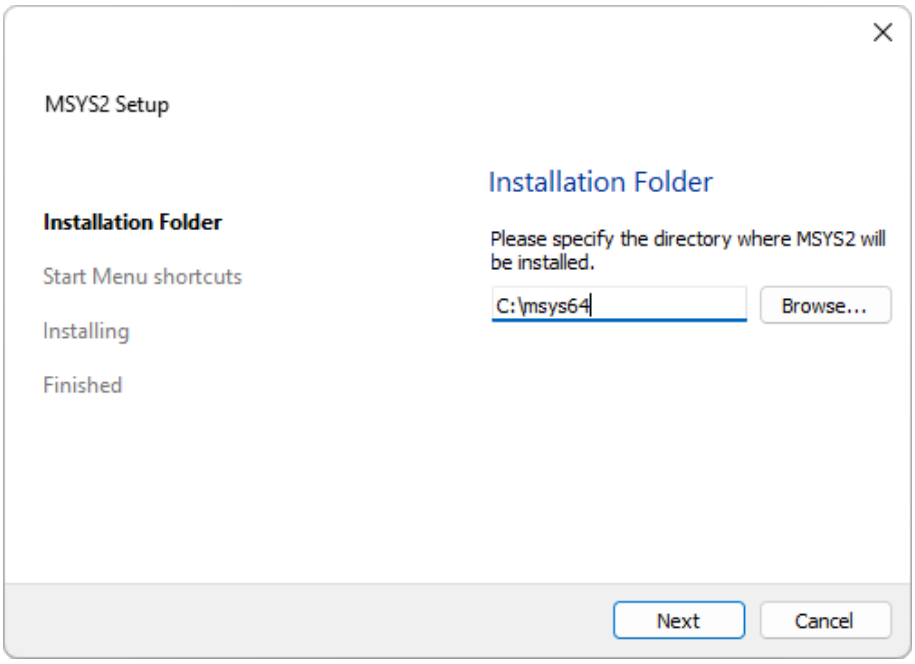
5.1 Configure Development Environment

The method of calling htra_api in 32-bit is basically the same as in 64-bit, so the following tutorial uses Matlab 2016a as an example to illustrate how to call the 64-bit htra_api.

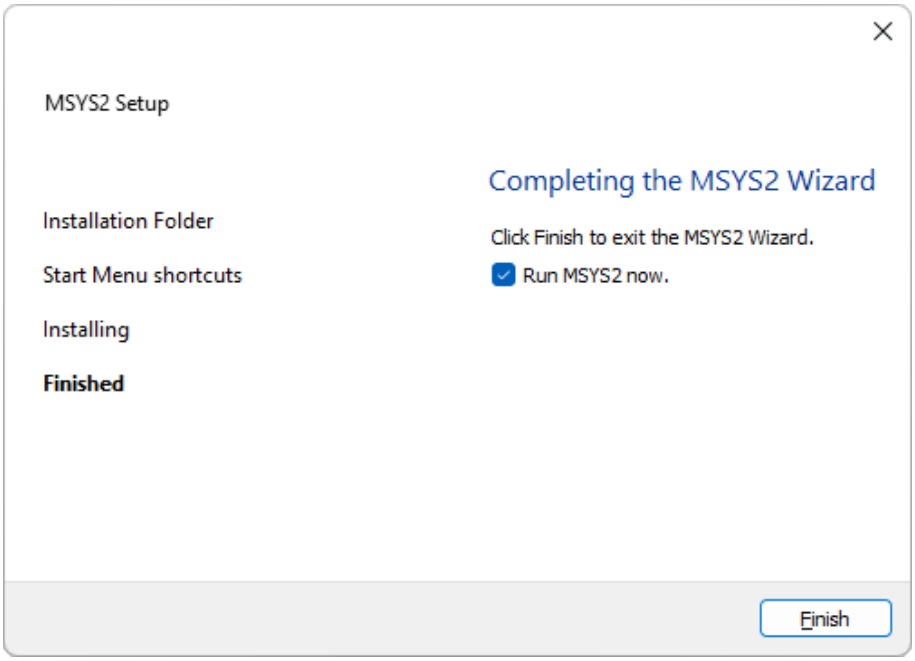
5.1.1 Install MSYS2

Download and installation link: https://www.msys2.org/https://www.msys2.org/

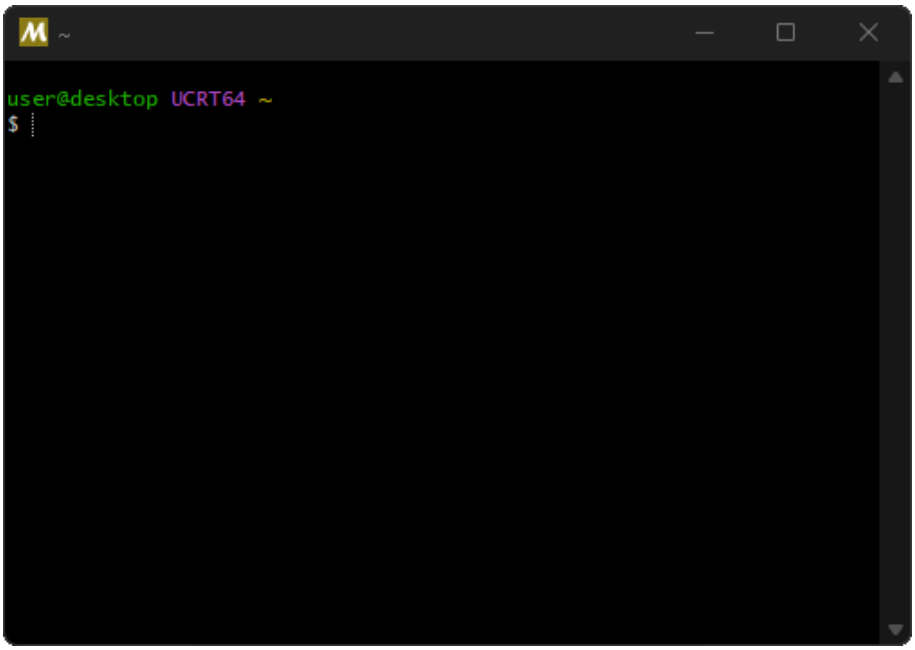
1. Download the installer MSYS2-x86_64-20231026.exe
2. Run the installer. MSYS2 requires 64-bit Windows 8.1 or higher.
3. The default installation path is C:\msys64, but you can choose a different path as needed.



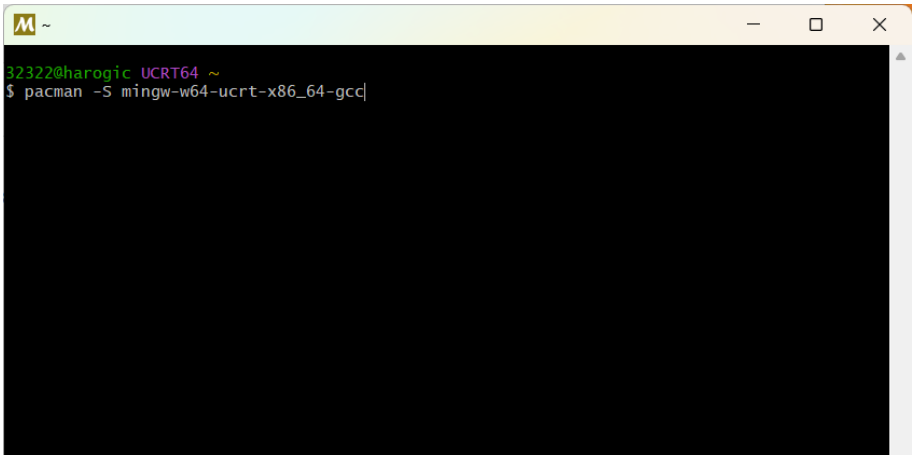
4. Once completed, click Finish.



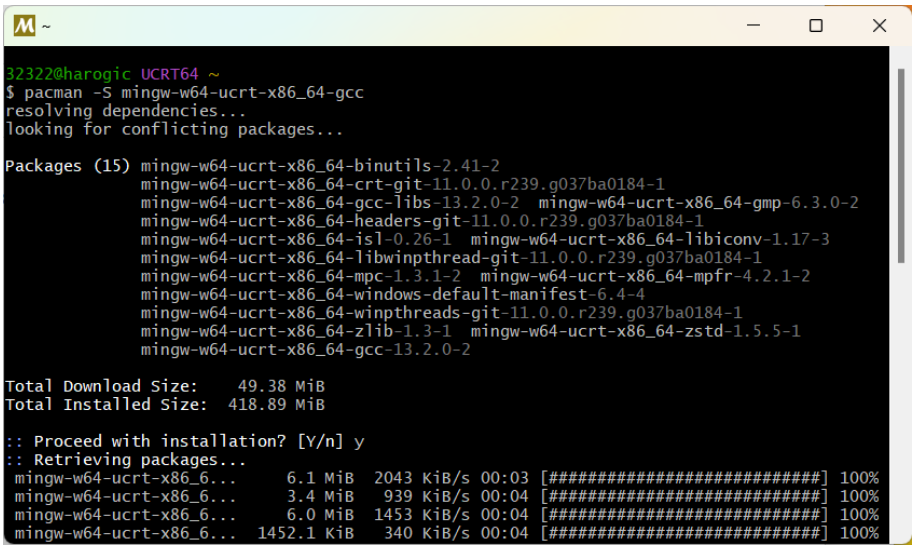
5. Now, MSYS2 is ready, and the terminal for the UCRT64 environment has started.


~
user@desktop UCRT64 ~
$

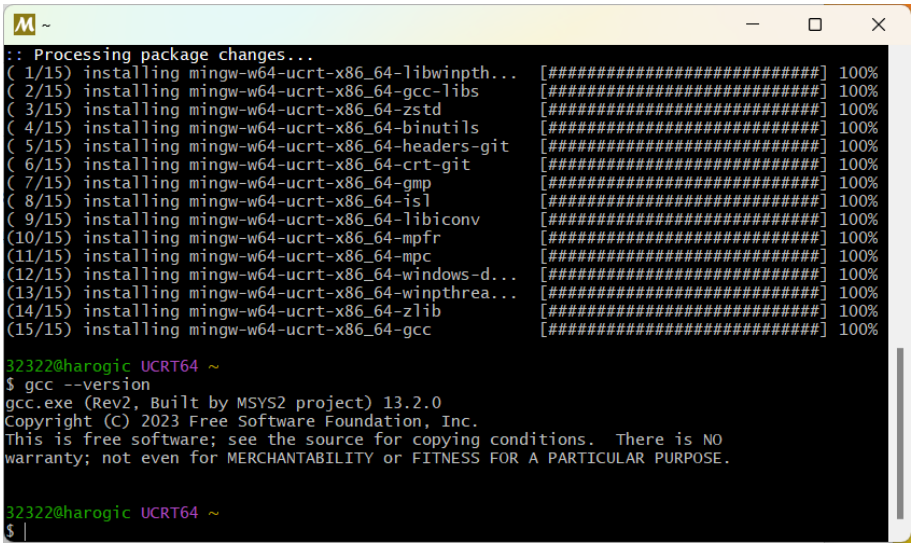
6. To install the GCC tools, enter the command: pacman -S mingw-w64-ucrt-x86_64-gcc



7. The terminal window will display the following output. Press "Enter" to continue.



8. Enter the command gcc --version to check the version information of GCC.



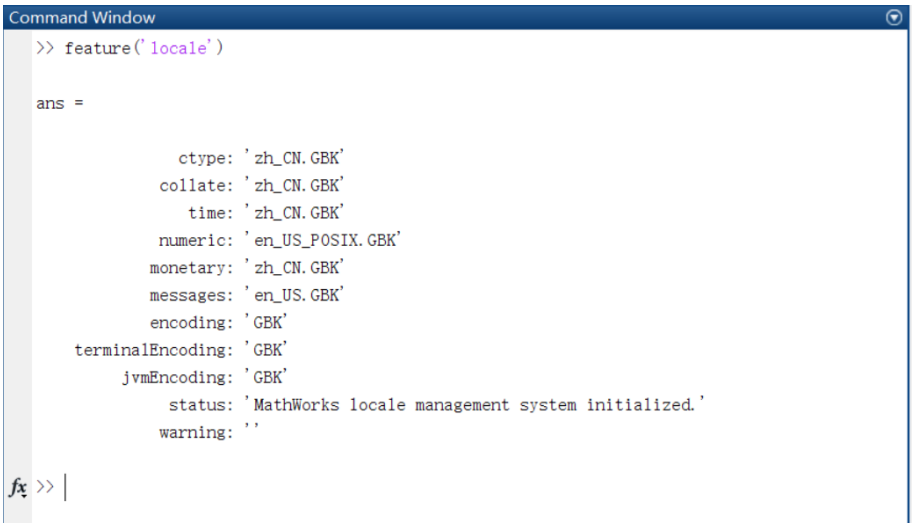
5.1.2 Configure Matlab

1. Resolve the issue of Chinese characters appearing garbled when opening .m files in Matlab 2016a.

Note: If you are using a version of Matlab higher than 2019a, please ignore this step.

（1） Check the current encoding format:

In the Matlab command line, enter: feature('locale')



From the figure, it can be seen that the encoding format is GBK.

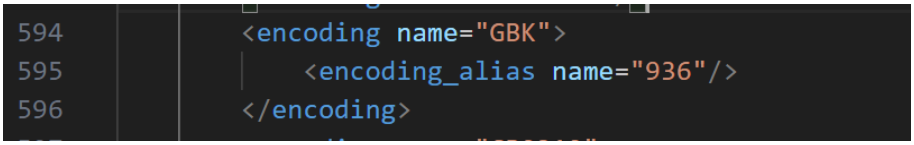
（2） Right-click the shortcut of Matlab2016a and select "Open file location"

to open the folder where Matlab.exe is located.

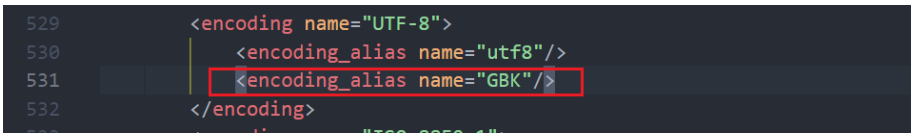
（3） In the folder shown in step (2), find the two files lcdata.xml and lcdata_utf8.xml, rename lcdata.xml to lcdata_old.xml to back up the original lcdata.xml.

（4） Copy lcdata_utf8.xml to the same folder and rename the newly copied file lcdata_utf8.xml to lcdata.xml.

（5） Open lcdata.xml and delete the GBK-related code shown in the image below.



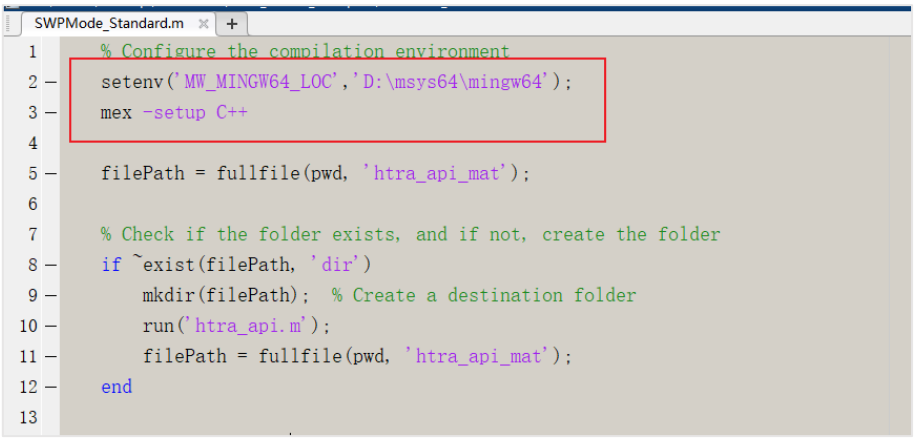
（6） Find the "UTF-8" section, add the code from the marked line in the image to the corresponding position in the image, save lcdata.xml, and then close the file.



（7） After restarting Matlab, the garbled Chinese characters will return to normal.

2. Method 1 for configuring the compilation environment: configure in the script

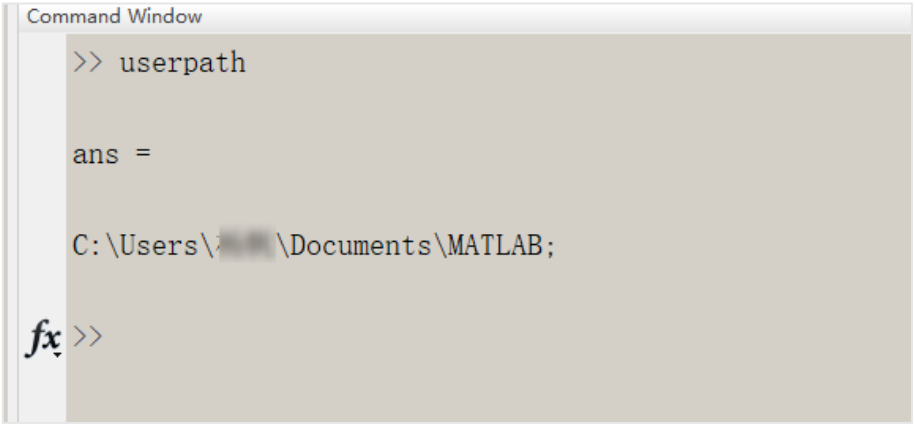
While running the script, execute the commands setenv('MW_MINGW64_LOC', 'D:\msys64\ucrt64') and mex -setup C++ to configure the compilation environment for the C++ programming language.



Note: D:\msys64\ucrt64 is the folder where the compilation environment is located. Please check if there are files such as c++.exe, g++.exe, and gcc.exe in the bin folder at this address. If they exist, this address is the compilation environment address; if not, please find the correct address of the compilation environment.

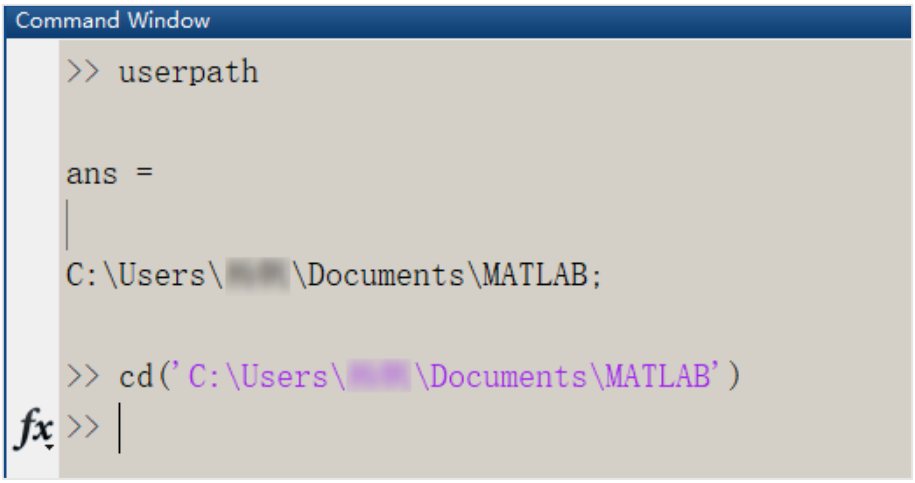
3. Method 2 for configuring the compilation environment: modify the startup.m file

（1） In the Matlab terminal, input: userpath, and the command line window will output a result similar to: C:\Users\YourUsername\Documents\MATLAB

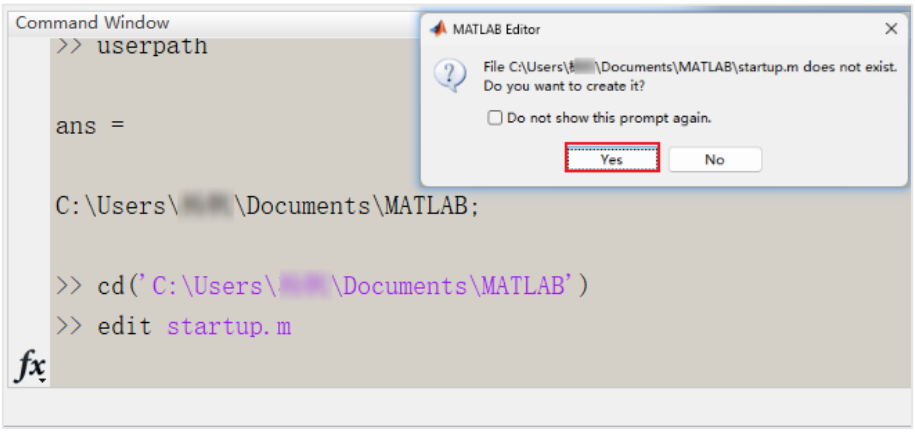


（2） Check if the startup.m file exists at C:\Users\YourUsername\Documents\MATLAB. If it does not exist, create a new startup.m file at this location. The steps for creation are as follows:

1） Matlab terminal input: cd('C:\Users\YourUsername\Documents\MATLAB'), switch the working directory to C:\Users\YourUsername\Documents\MATLAB.

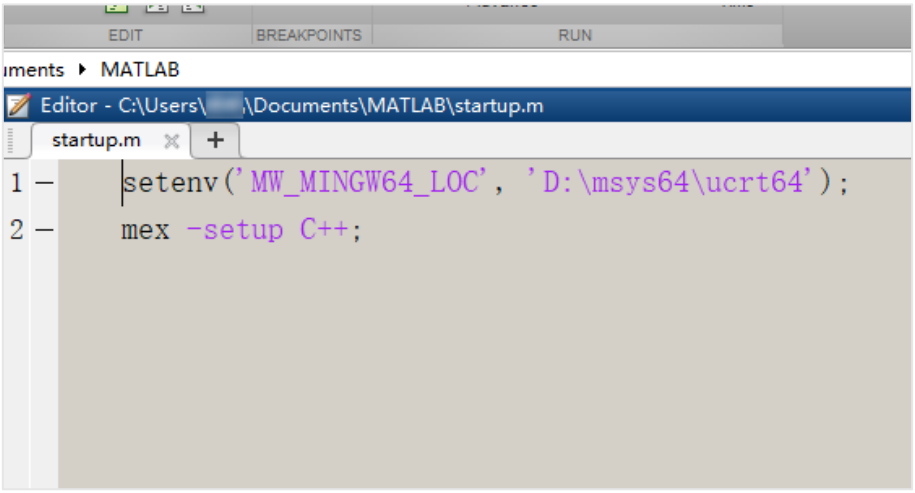


2) Matlab terminal input: edit startup.m, select "Yes" in the pop-up window to create the startup.m file.



3) Add commands in the startup.m file.

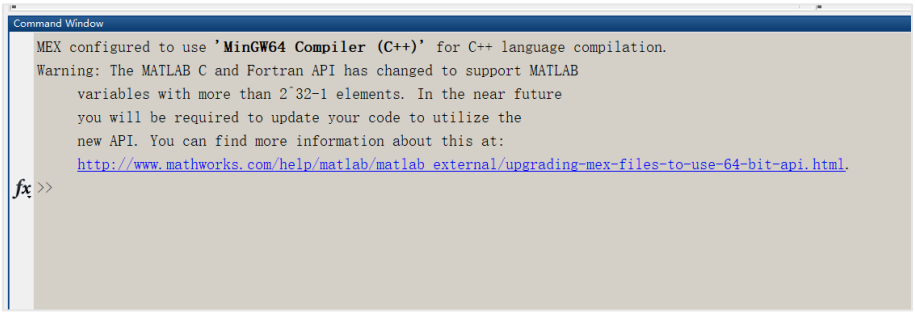
setenv('MW_MINGW64_LOC', 'D:\msys64\ucrt64'); andmex -setup C++.



4) startup.m After editing the file, save and close it.

5) Restart Matlab, and observe that the command line window

appears as shown in the figure below, indicating that the configuration is complete.



5.1.3 Call htra_api.dll Description

1. loadlibrary

The loadlibrary function can load dynamic link libraries. loadlibrary('.\htra_api\htra_api.dll','.\htra_api\htra_api.h'); Ensure that the file paths for .dll and .h are correct.

```
%Loadhtra_api.dll
if not(libisloaded('htra_api.dll'))
    %The file paths for .dll and .h files should be carefully noted
    loadlibrary('.\htra_api\htra_api.dll','.\htra_api\htra_api.h');
end
```

2. libfunctions

libfunctions('htra_api'); This is used to view all available functions in htra_api.dll.

```
%View all functions in the API
libfunctions('htra_api');
```

3. libpointer

libpointer allows the creation of data type pointers in Matlab and passes them to external library functions.

```
%Open the device
%Create a Device pointer
Device = libpointer;
DevNum = 0;
Status = 0;
```

4. libstruct

libstruct is used to define structure types in Matlab and pass them to external library functions.

```
%Create a BootProfile structure.
BootProfile = libstruct('BootProfile_TypeDef');
save(fullfile(folderPath,'BootProfile.mat'), 'BootProfile');

%Create the DeviceInfo structure
DeviceInfo = libstruct('DeviceInfo_TypeDef');
save(fullfile(folderPath,'DeviceInfo.mat'), 'DeviceInfo');
```

5. get

The get function is used to retrieve the property values of a structure.

```
%Call the Device_Open function
Status = calllib('htra_api', 'Device_Open', Device, DevNum, BootProfile_p, BootInfo_p);
get(BootInfo_p); %Print the value of BootInfo_p
```

6. calllib

calllib is the command in Matlab used to call functions in htra_api.dll.

```
%Call the Device_Open function
Status = calllib('htra_api', 'Device_Open', Device, DevNum, BootProfile, BootInfo);
get(BootProfile);%Prints the value of BootInfo_p
```

7. load

load is used to load the .mat structure files generated in htra_api.m.

```
% Load the BootProfile_TypeDef structure directly.
load(fullfile(filePath, 'BootProfile.mat'));
% Load the BootInfo_TypeDef structure directly
load(fullfile(filePath, 'BootInfo.mat'));
```

8. fullfile

fullfile is a function in Matlab used to generate complete file paths. It loads the BootProfile.mat and BootInfo.mat files from the filePath.

```
% Load the BootProfile_TypeDef structure directly.
load(fullfile(filePath, 'BootProfile.mat'));
% Load the BootInfo_TypeDef structure directly
load(fullfile(filePath, 'BootInfo.mat'));
```

9. unloadlibrary

unloadlibrary is used to unload a previously loaded htra_api library, appearing in pairs with loadlibrary.

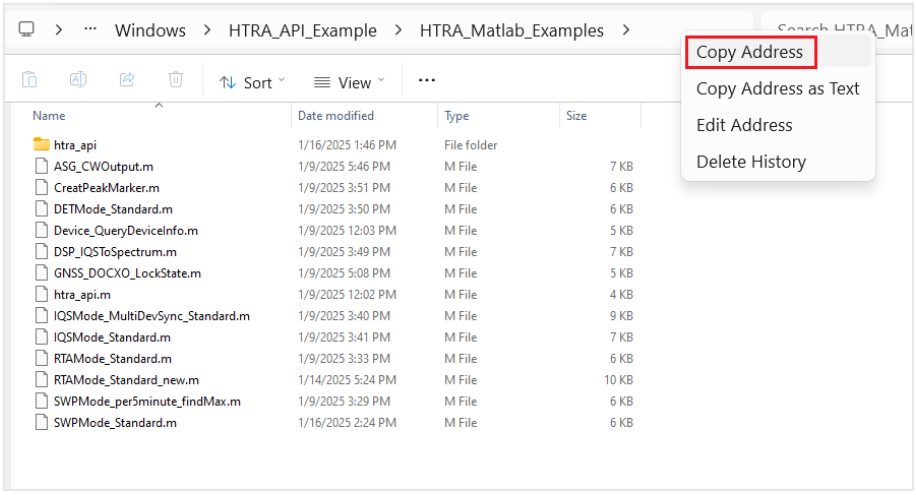
```
%Unload library file
unloadlibrary('htra_api');
disp('Uninstall complete')
```

5.2 Matlab example usage process

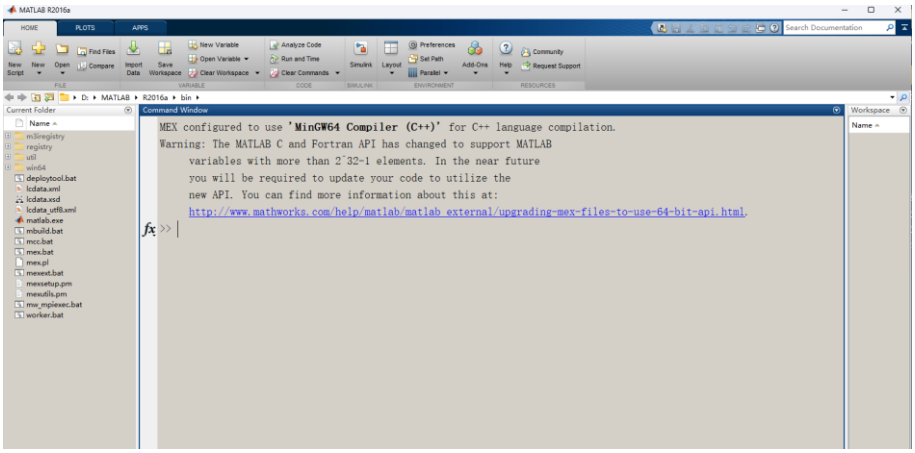
The usage process for the Matlab examples included on the USB drive is as follows:

1. Open the Windows\HTRA_API_Example\HTRA_Matlab_Examples folder on the USB drive, and double-click any .m file to open the example. For instructions on how to run the example, please refer directly to step 4.

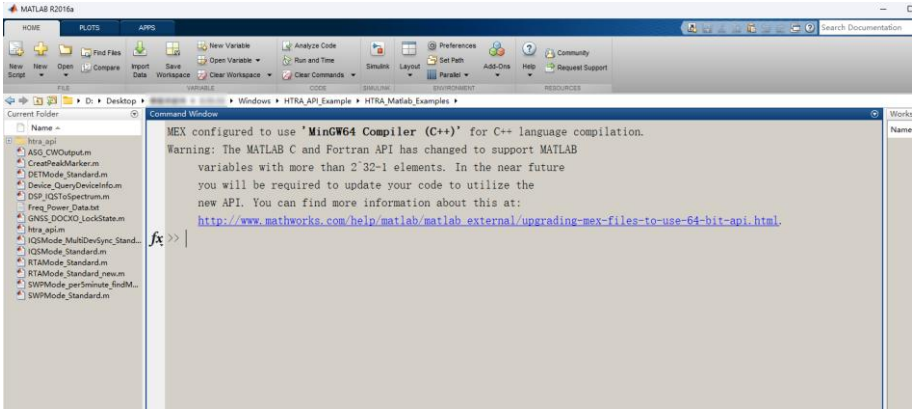
2. If you cannot open the example in step 1, please continue to this step and copy the address Windows\HTRA_API_Example\HTRA_Matlab_Examples.



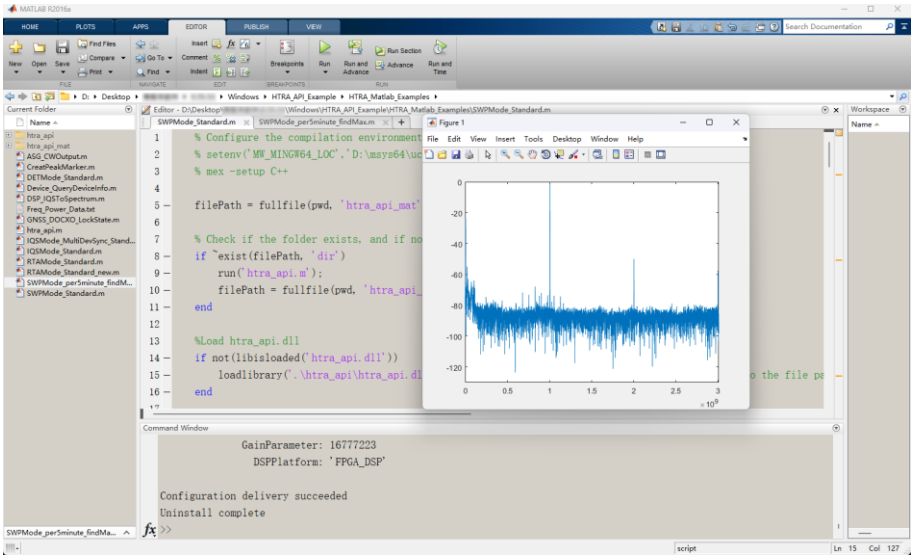
3. Open the Matlab software installed on your system.



4. After pasting the copied address into the file address box, press Enter to navigate to the \Windows\HTRA_API_Example\HTRA_Matlab_Examples folder included with the materials.



5. Click on the .m file on the left as needed, click "Run," and wait for the Figure 1 window to appear, indicating that the example has run successfully. For the functional descriptions of each example, please refer to section 5.2 Matlab Example Description.


% 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
Figure 1
File Edit View Insert Tools Desktop Window Help
GainParameter: 16777223
DSPPlatform: 'FPGA_DSP'
Configuration delivery succeeded
Uninstall complete
>>

5.3 Introduction to Accompanying Examples

5.3.1 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.

5.3.2 Obtain Standard Spectrum Data

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

5.3.3 Create multiple cursors to display the frequency and power of the cursors.

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

5.3.4 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.

5.3.5 Obtain continuous stream or fixed number of IQ data.

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

5.3.6 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.

5.3.7 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.

5.3.8 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.

5.3.9 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.

5.3.10 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.

5.3.11 Multi-machine synchronization

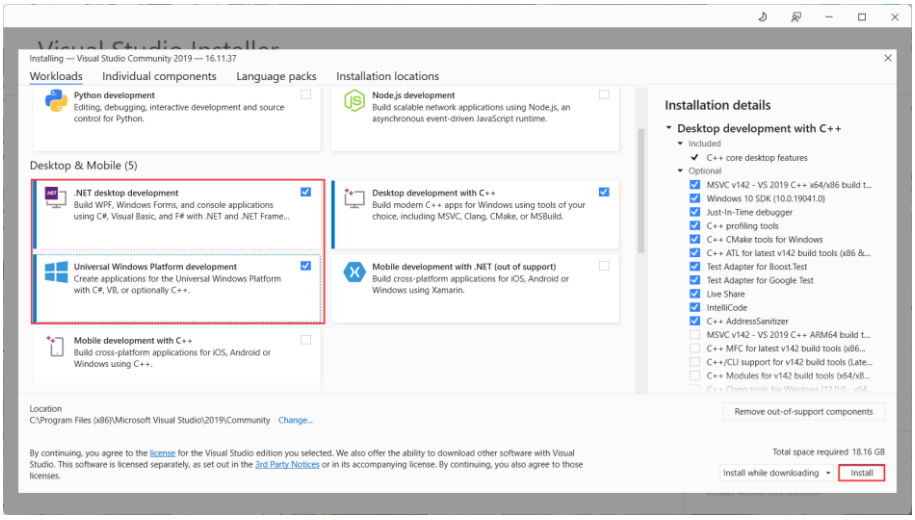
IQSMode_MultiDevSync_Standard.m: When using the same reference clock source input and the same trigger source input, two devices simultaneously acquire IQ data, allowing for the observation of the synchronization of their collected data.

6. C#

6.1 Configure Development Environment

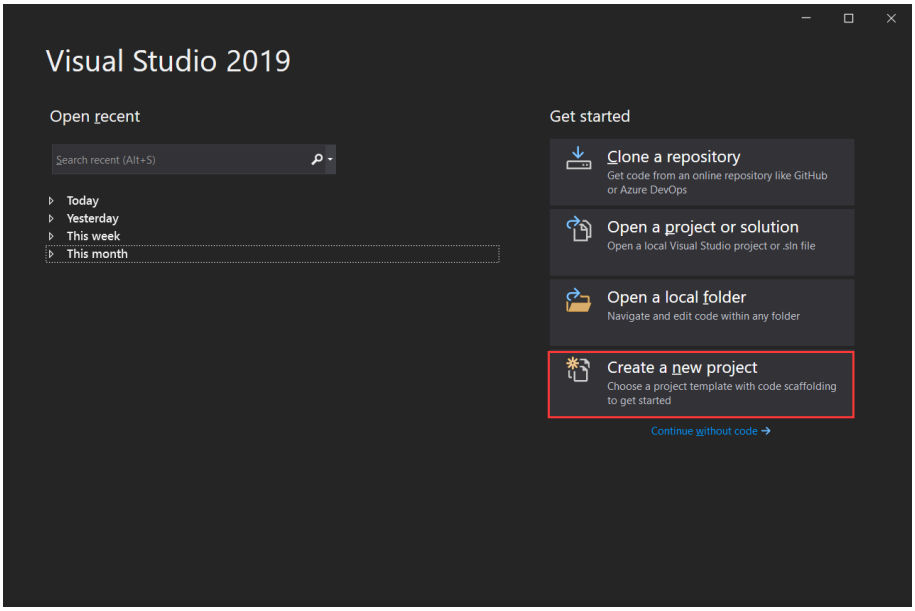
6.1.1 Development Environment Confirmation

Open Visual Studio Installer, check the .NET desktop development components and Universal Windows Platform development components, and click Modify to ensure that Visual Studio 2019 has the C# development environment.

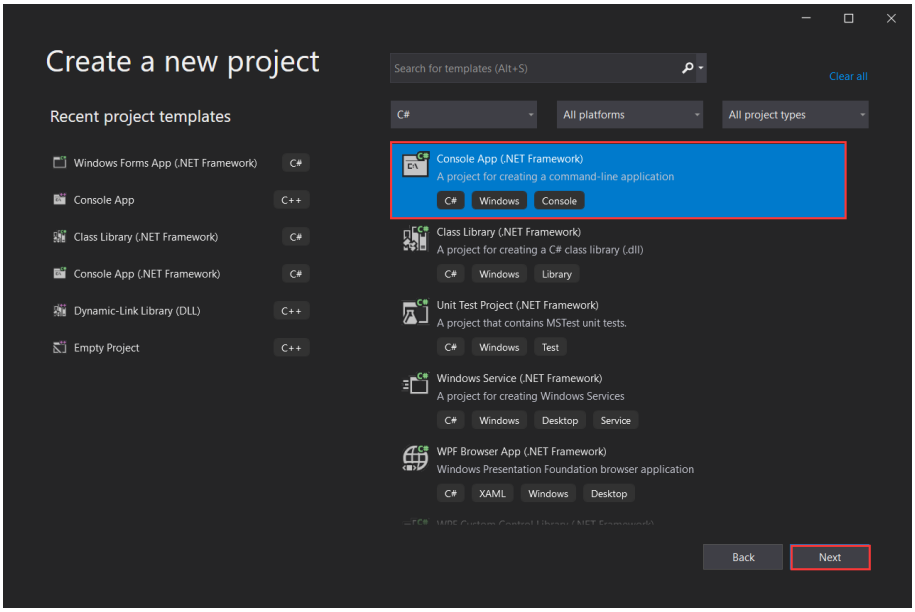


6.1.2 Project Setup

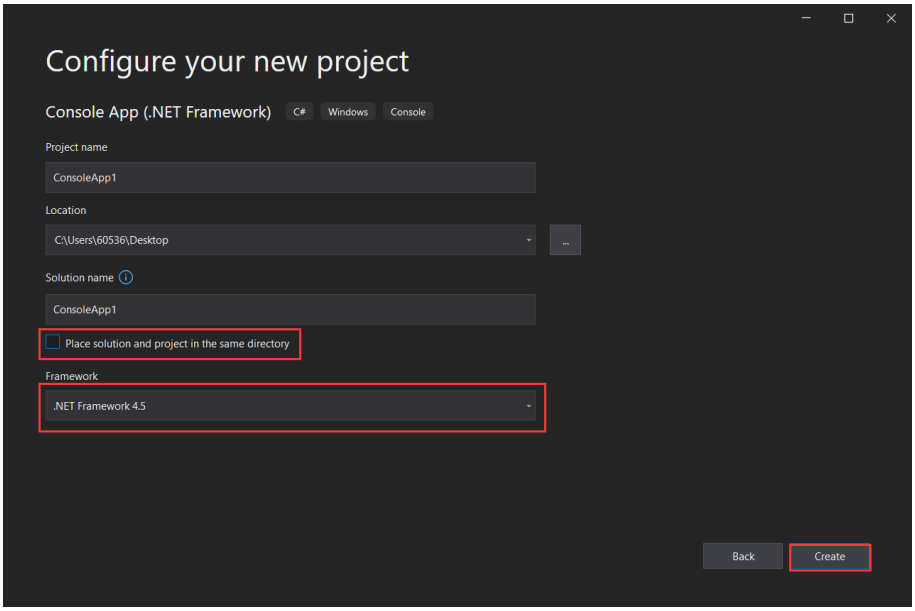
1. Open Visual Studio 2019 and click on Create a New Project.



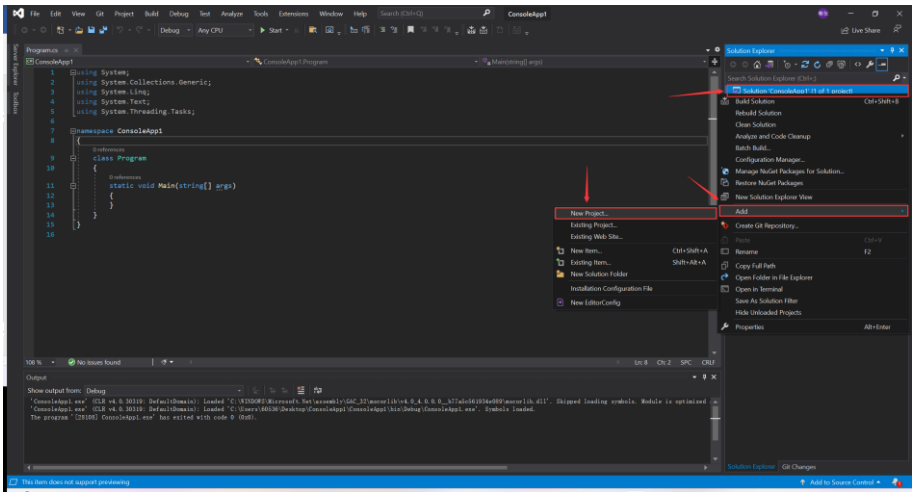
2. Select C# Console Application and click Next.



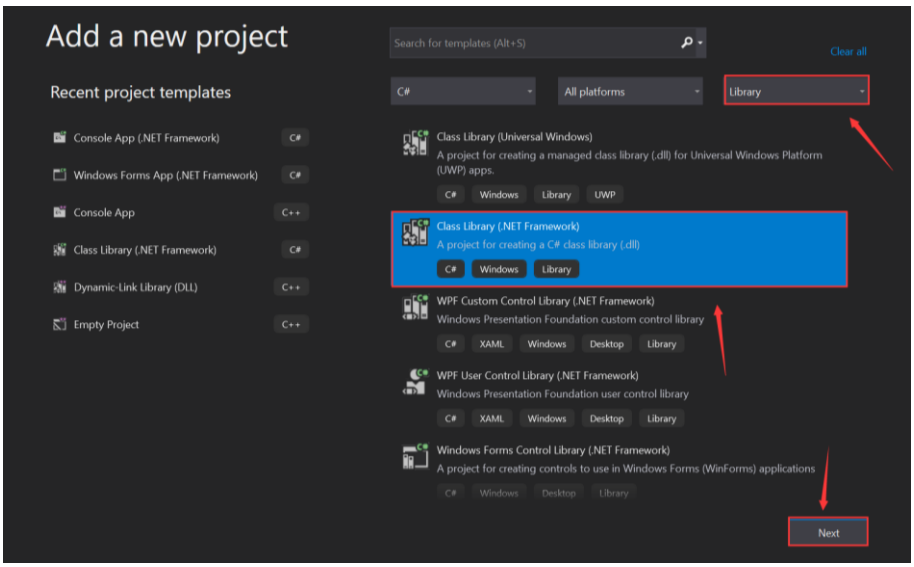
3. Enter the project name and storage location, uncheck the option to place the solution and project in the same directory. Select .NET Framework 4.5 as the framework, and finally click Create.



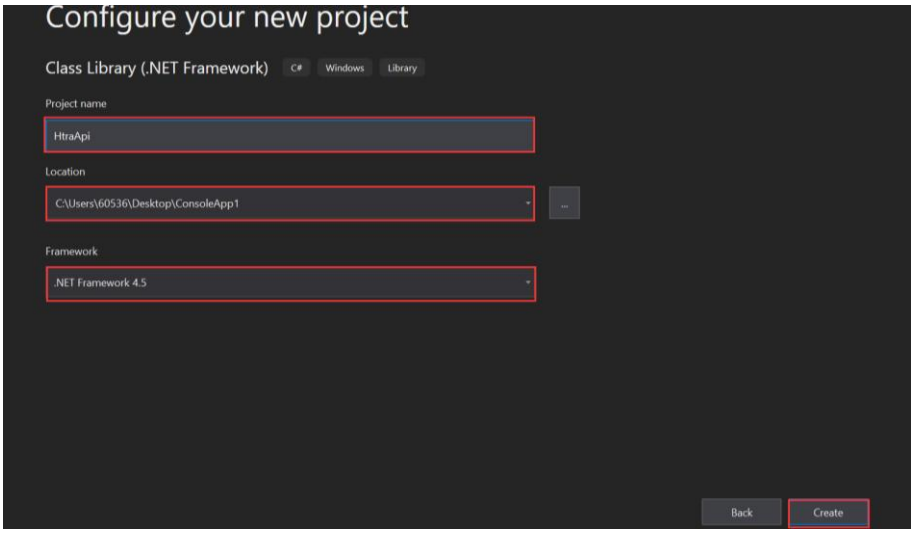
4. Once the creation is complete, open the project, right-click on the solution, and select Add New Project.

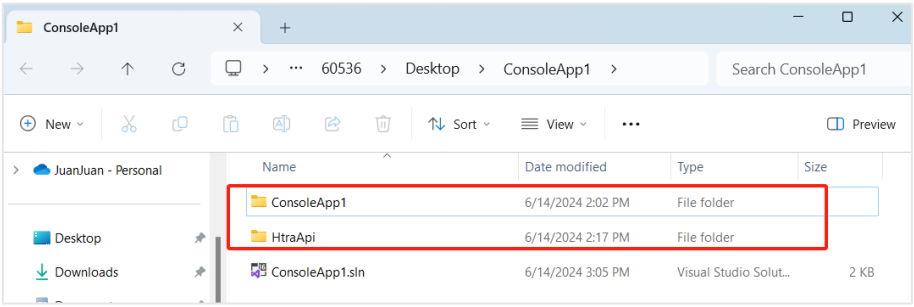


5. Select Class Library (.NET Framework) under Library type for the project type, and click Next.

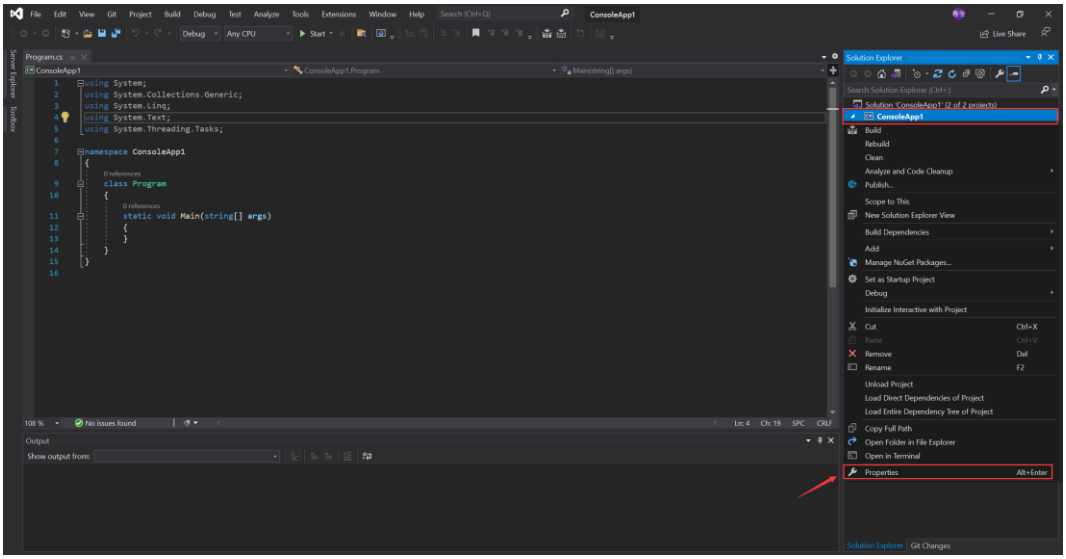


6. The library name can be modified as needed, such as HtraApi, and the location should not be changed, remaining at the same directory level as the solution. The result is shown in the figure, select the .NET Framework 4.5 framework, and click Create.

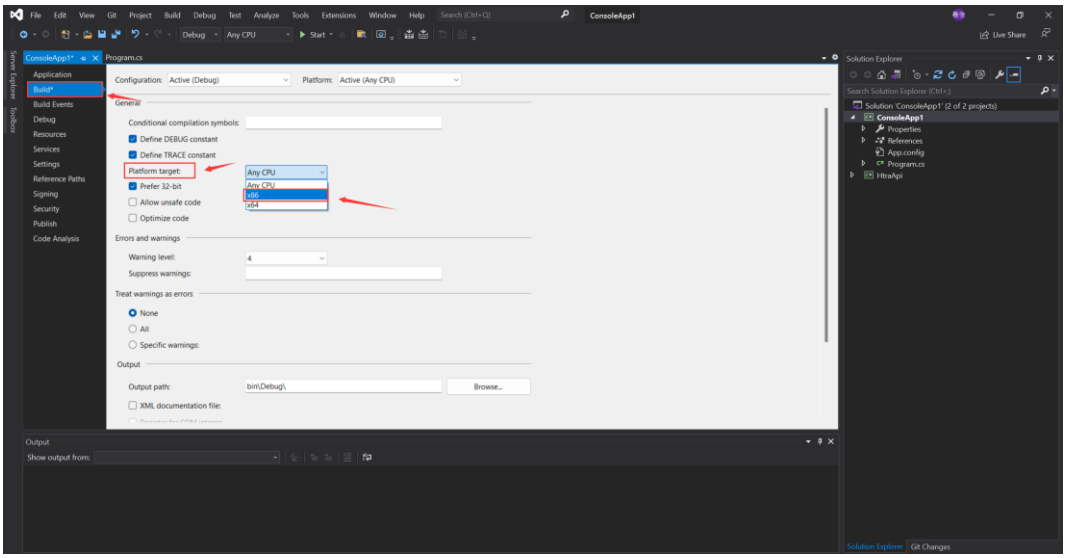


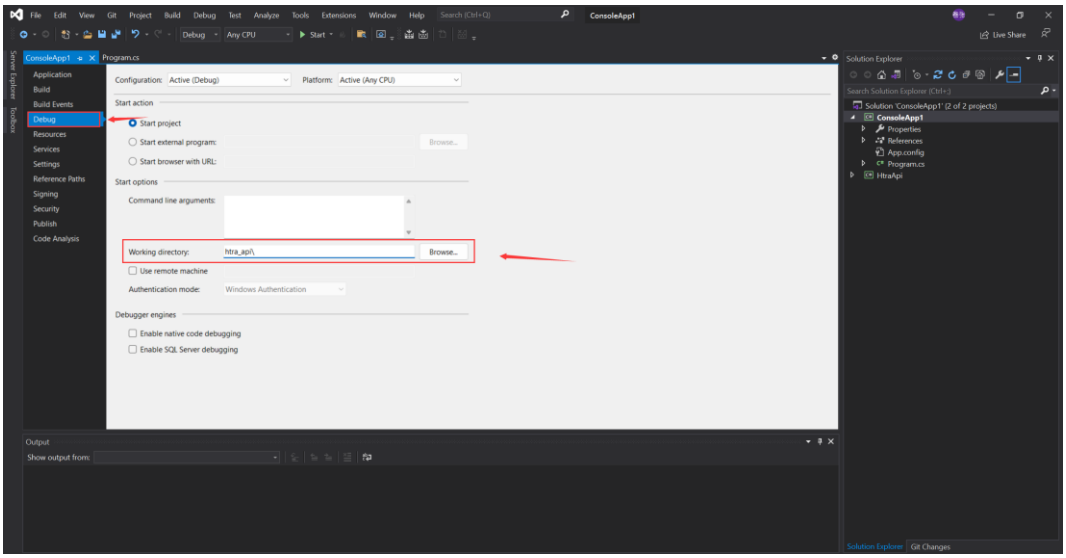


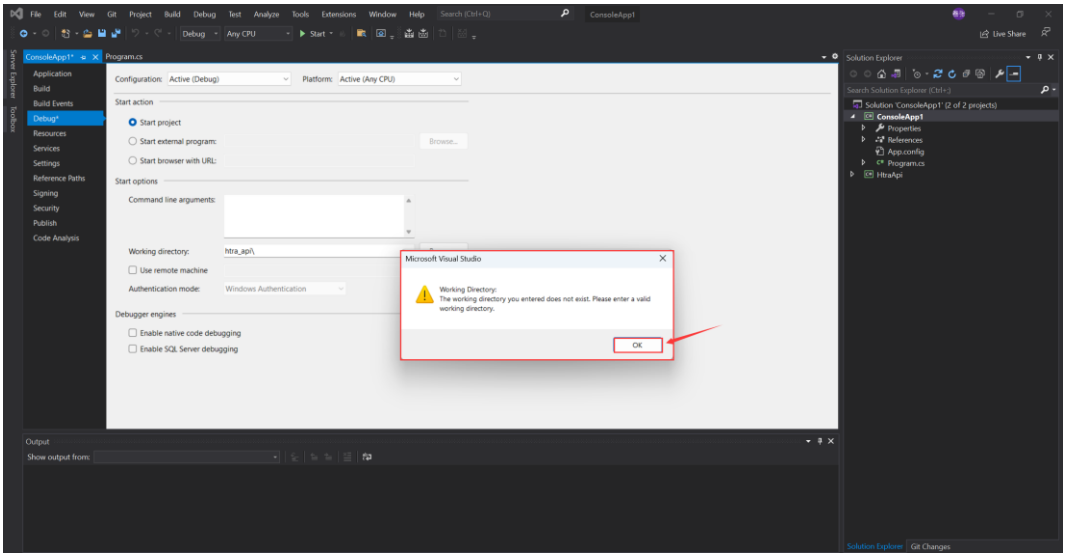
7. Right-click on ConsoleApp1 and select Properties.



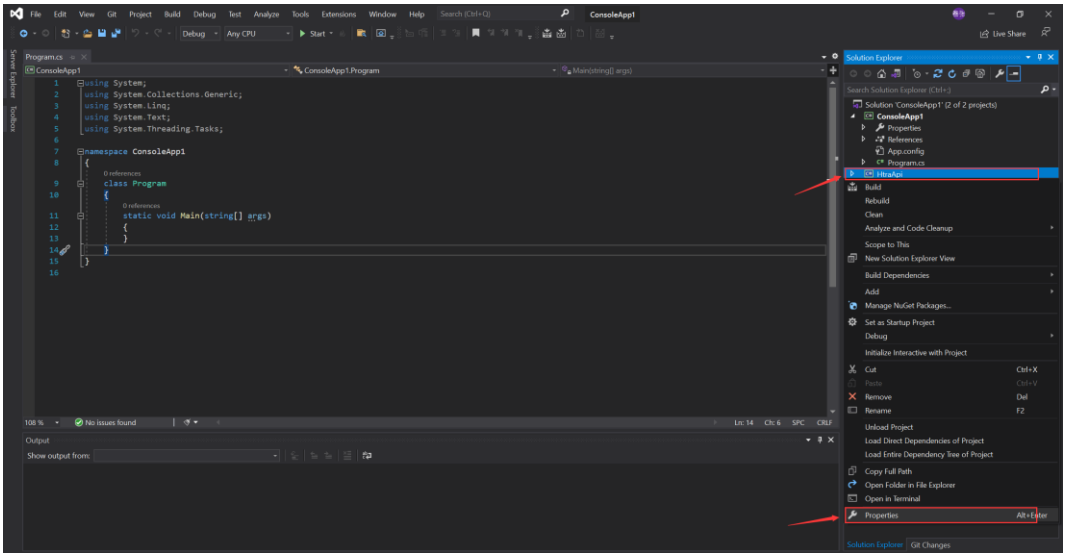
8. View the project's build properties, change the target platform to x86, click Debug, enter htra_api\ in the working directory, and save. If the situation in Figure 12 occurs, click OK and save again.



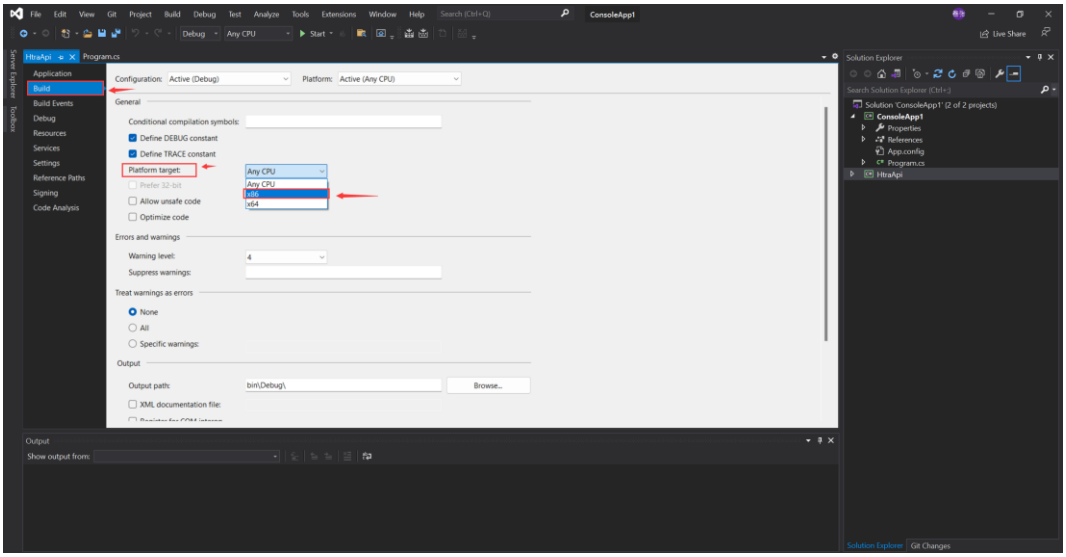




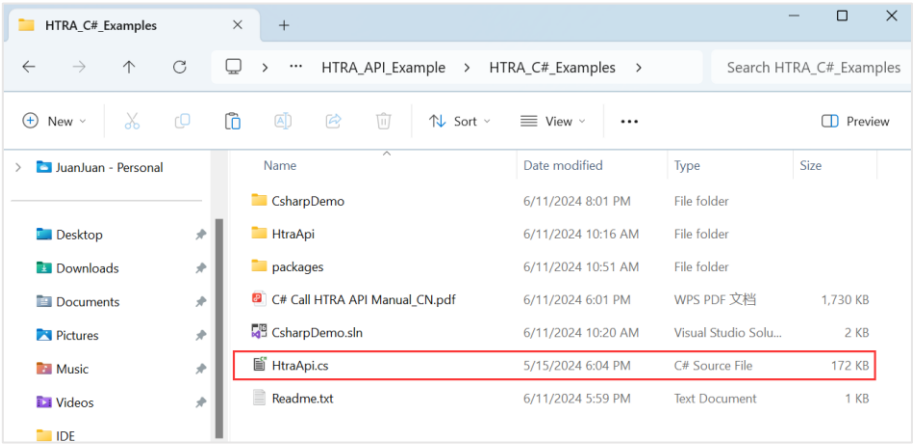
9. Right-click on the library HtraApi and select Properties.

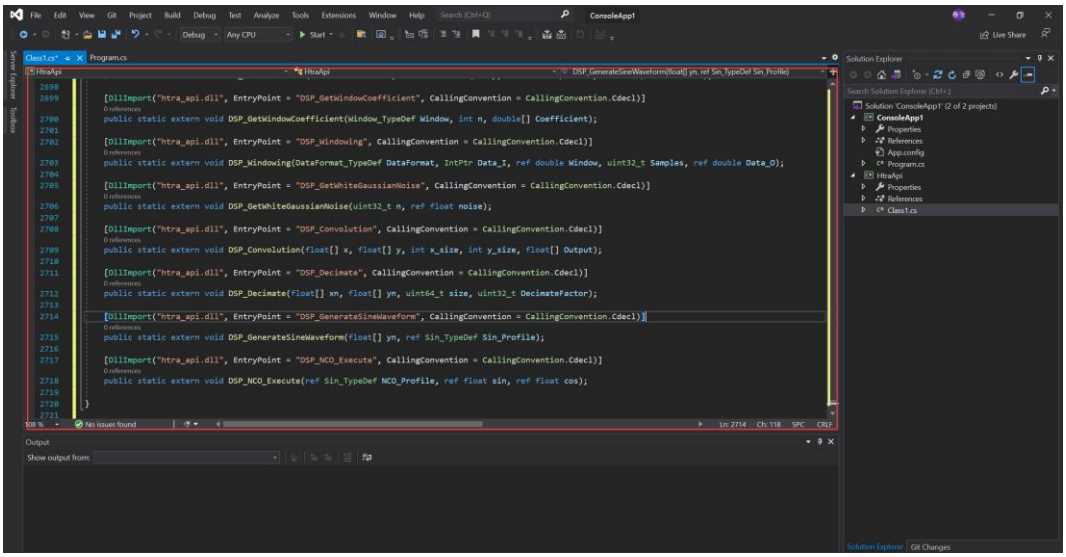


10. View the library's build properties, change the target platform to x86, and save.

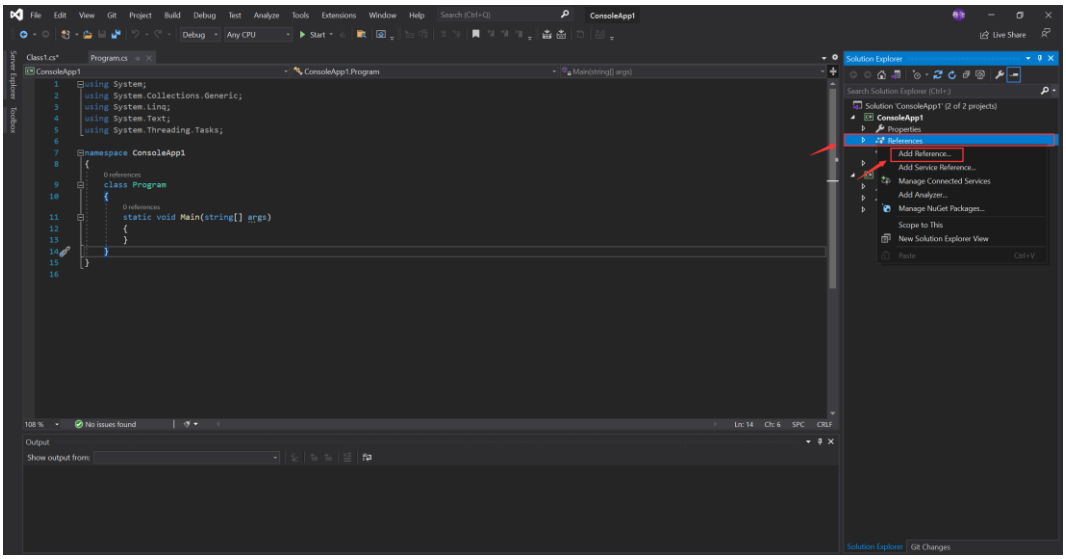


11. Copy the contents of the HtraApi.cs file from the folder \Windows\HTRA_API_Example\HTRA_C#_Examples included in the accompanying materials to the Class1.cs file in the project library and save.

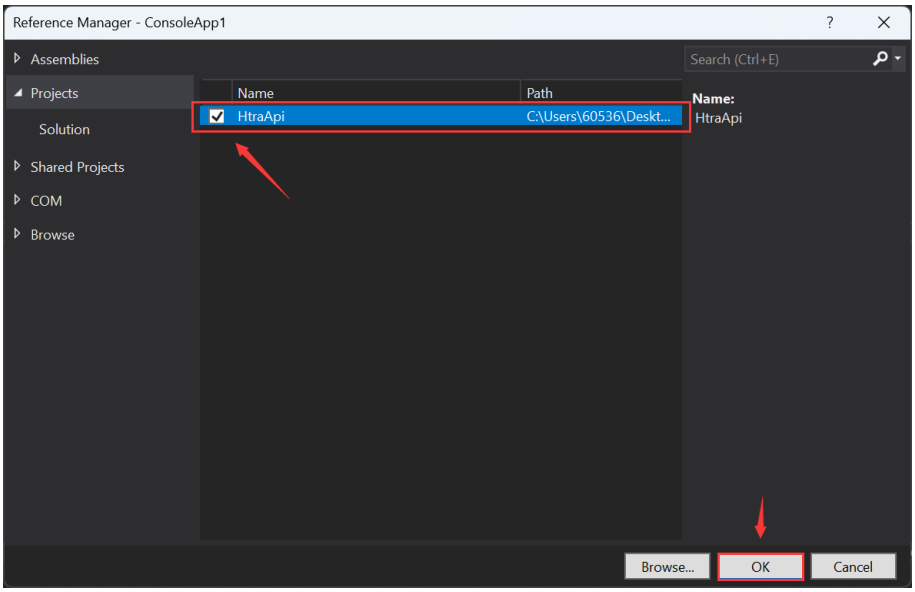




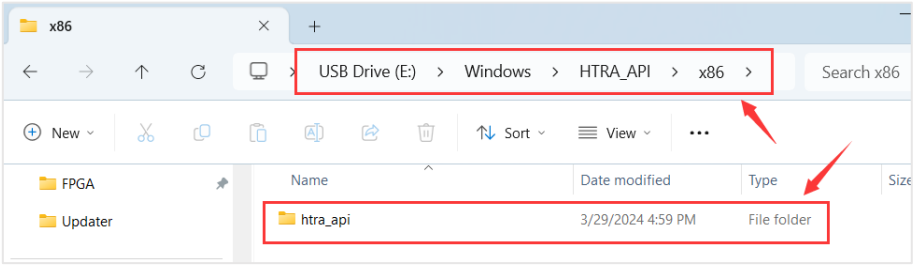
12. Select the ConsoleApp1 project, right-click on References, and choose Add Reference.

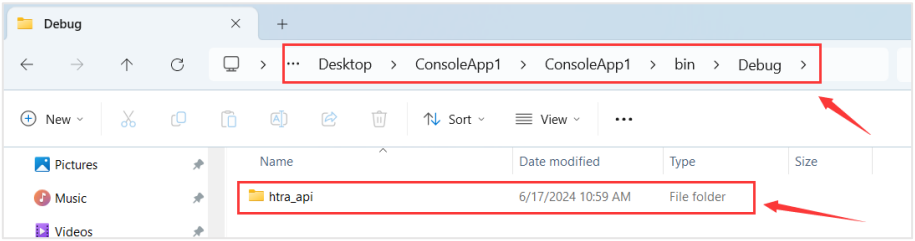


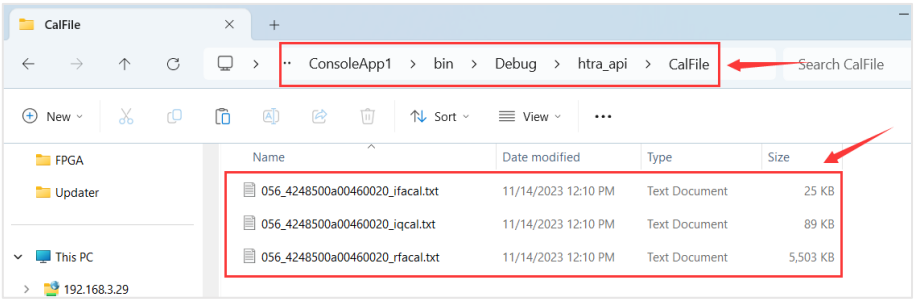
13. Select the HtraApi library and confirm the addition of the library reference.



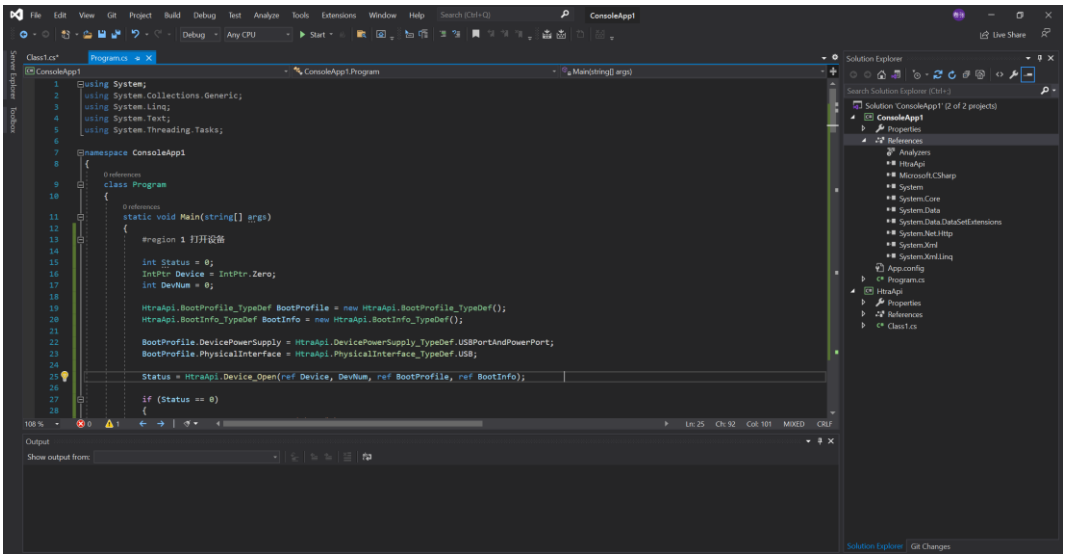
14. Copy the htra_api folder from \Windows\HTRA_API\x86 on the accompanying USB drive to the Debug folder under the project's bin folder, and ensure that the CalFile folder within the htra_api folder contains the calibration files.







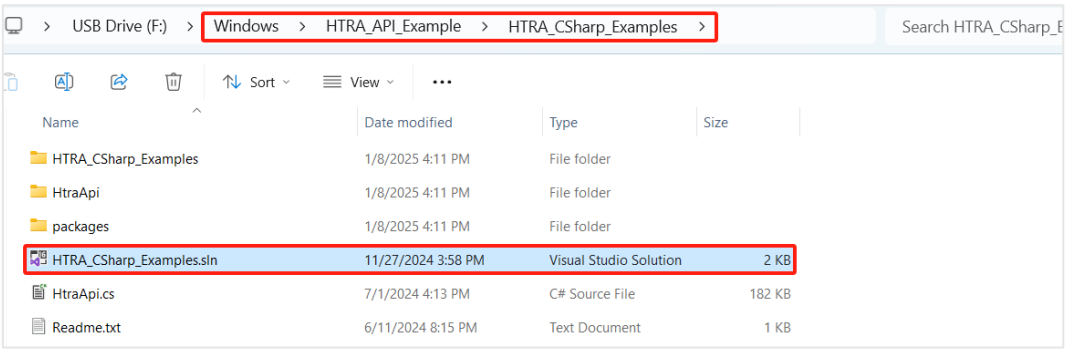
15. Write code normally. You can refer to the C# examples included in the accompanying USB drive, specifically the projects in \Windows\HTRA_API_Example\HTRA_C#_Examples.


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
ConsoleApp1
Debug
Any CPU
Start
Live Share
Class1.cs*
Program.cs
ConsoleApp1
ConsoleApp1.Program
Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp1
{
0 references
class Program
{
0 references
static void Main(string[] args)
{
#region 1 打开设备
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)
{
Ln: 25 Ch: 92 Col: 101 MIXED CRLF
Output
Show output from:
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApp1' (2 of 2 projects)
ConsoleApp1
Properties
References
Analyzers
HtraApi
Microsoft.CSharp
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Net.Http
System.Xml
System.Xml.Linq
App.config
Program.cs
HtraApi
Properties
References
Class1.cs
Solution Explorer
Git Changes

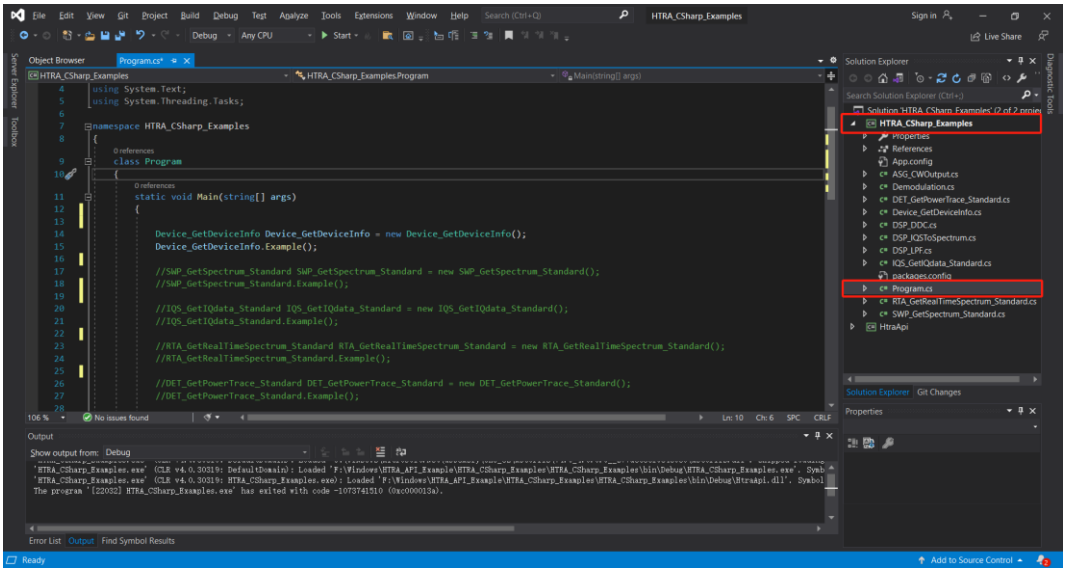
6.2 C# Example Usage Process

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

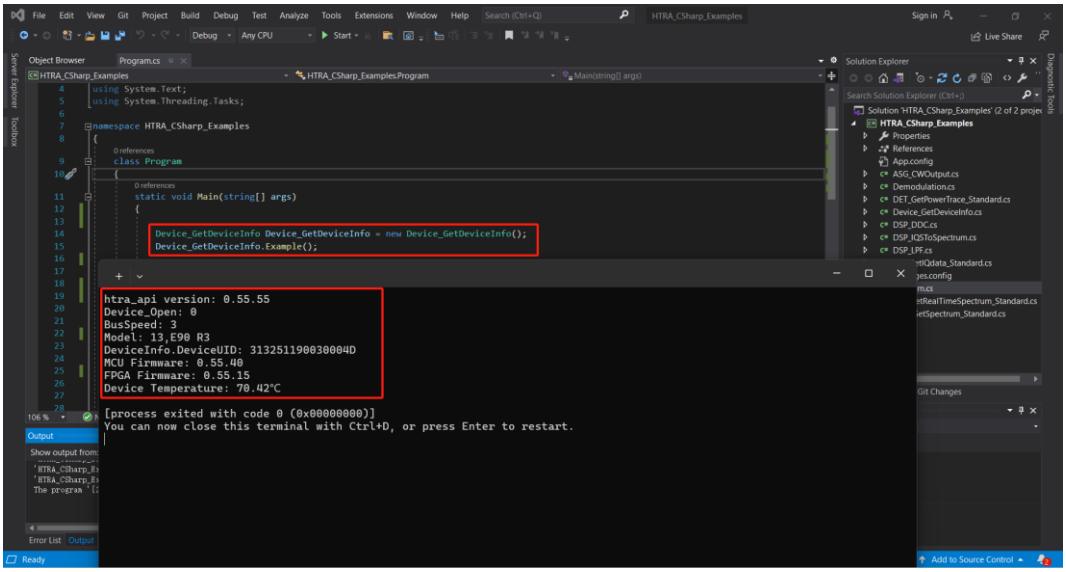
1. Open the USB drive using Visual Studio and navigate to the folder Windows\HTRA_API_Example\HTRA_CSharp_Examples, then open the solution file HTRA_CSharp_Examples.sln.



2. Click on the HTRA_CSharp_Examples project on the right side, and then click on the Programe.cs file.



3. Since each example in the C# included examples is encapsulated in a separate class, you can run the examples by uncommenting them (multiple examples cannot be used simultaneously). For instance, when testing the Device_GetDeviceInfo example, uncomment it and click run; as shown in the figure, the device is running normally.


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();
htra_api version: 0.55.55
Device_Open: 0
BusSpeed: 3
Model: 13,E90 R3
DeviceInfo.DeviceUID: 313251190030004D
MCU Firmware: 0.55.40
FPGA Firmware: 0.55.15
Device Temperature: 70.42°C
[process exited with code 0 (0x00000000)]
You can now close this terminal with Ctrl+D, or press Enter to restart.

6.3 C# Example Descriptions

6.3.1 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.

6.3.2 Obtain Standard Spectrum Data

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

6.3.3 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.

6.3.4 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.

6.3.5 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.

6.3.6 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.

6.3.7 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.

6.3.8 IQ to Spectrum Data

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

6.3.9 Low-pass filtering

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

6.3.10 Digital Downconversion

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

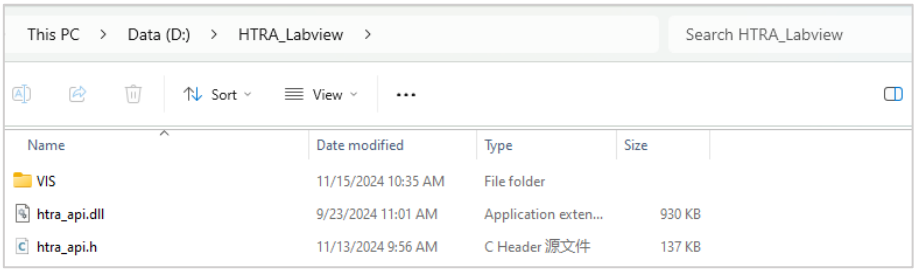
7. Java (to be supplemented)

8. Labview

8.1 Configure Development Environment

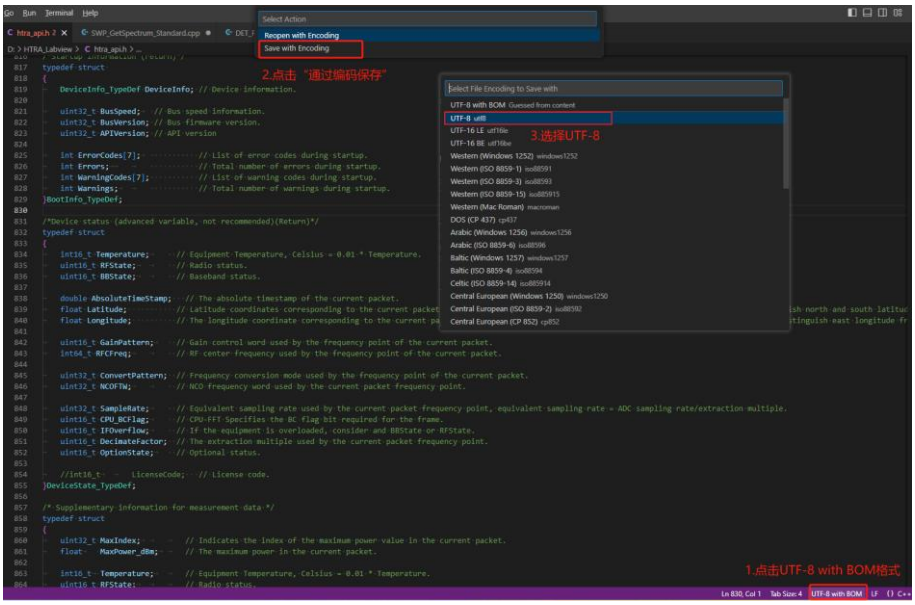
8.1.1 Export library functions from htra_api.dll using LabVIEW

1. Create a folder (e.g., HTRA_Labview) and copy the htra_api folder from the USB drive located at Windows\HTRA_API\x86 into this folder. Then create another folder (e.g., VIS) to place the exported VIs.

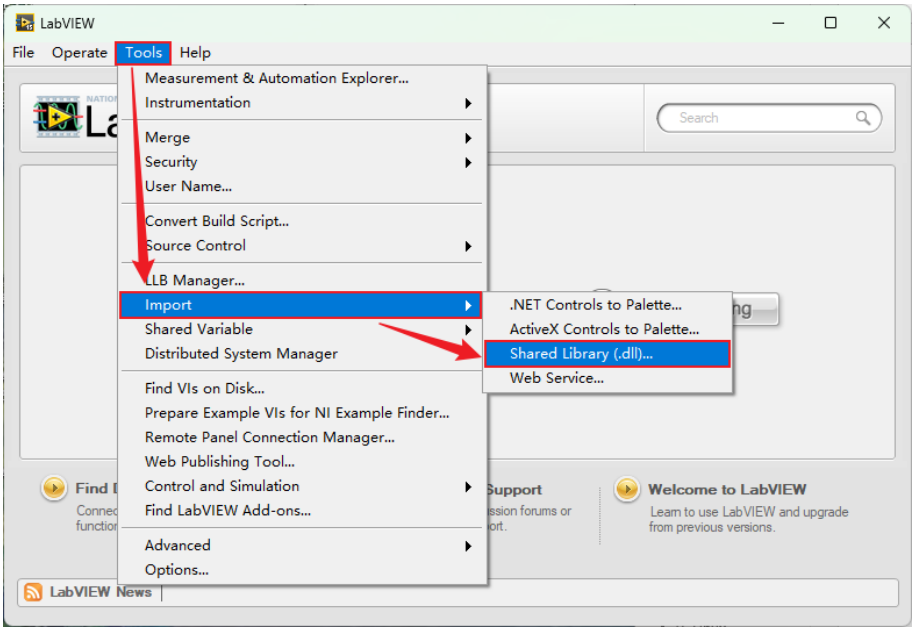


2. LabVIEW does not recognize the uint64_t and int64_t data types during import, so before importing, all parameters of type uint64_t and int64_t need to be changed to double. Note that after exporting the functions, these modified parameter types should be changed back to uint64_t or int64_t in the VIs.

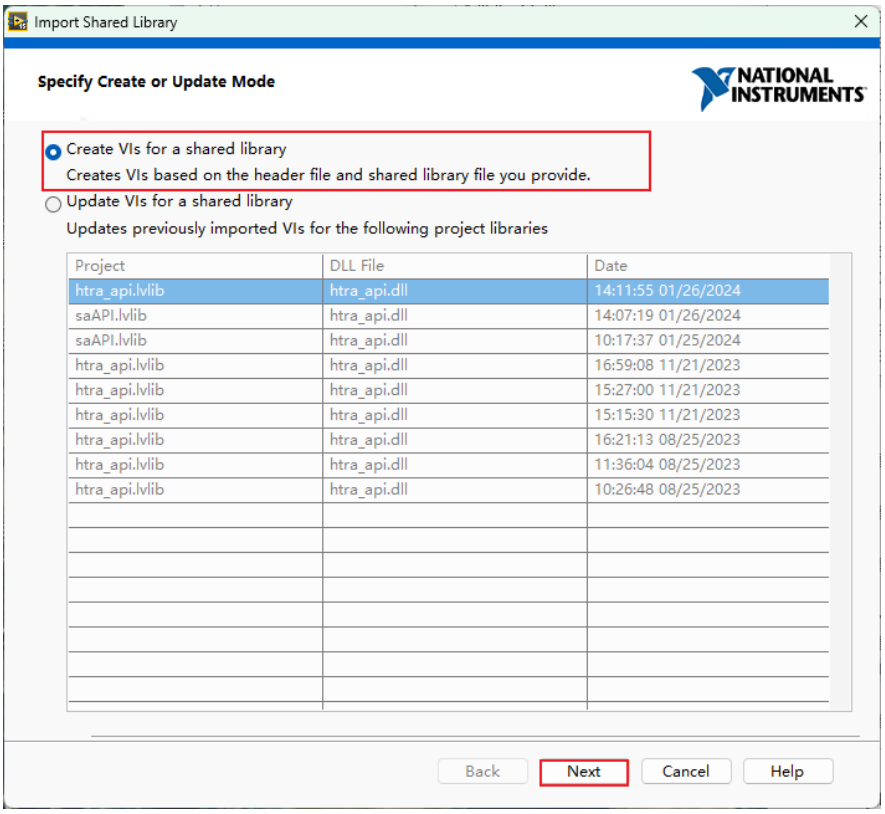
3. Change the encoding format of htra_api.h to UTF-8.



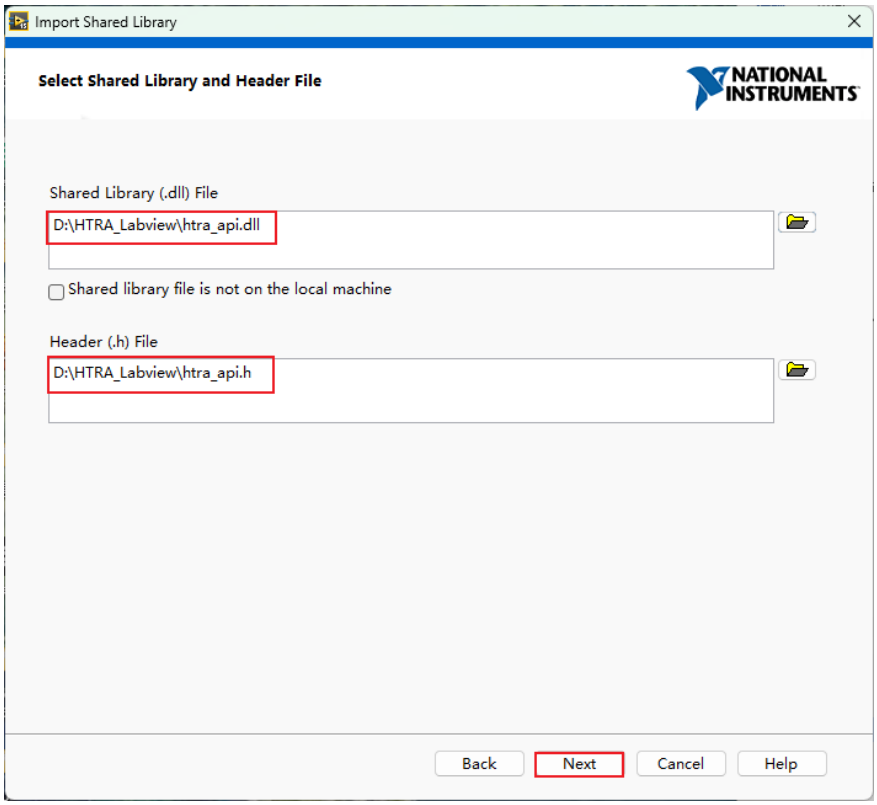
4. Open LabVIEW, and select “Tools--->Import--->Shared Library.”



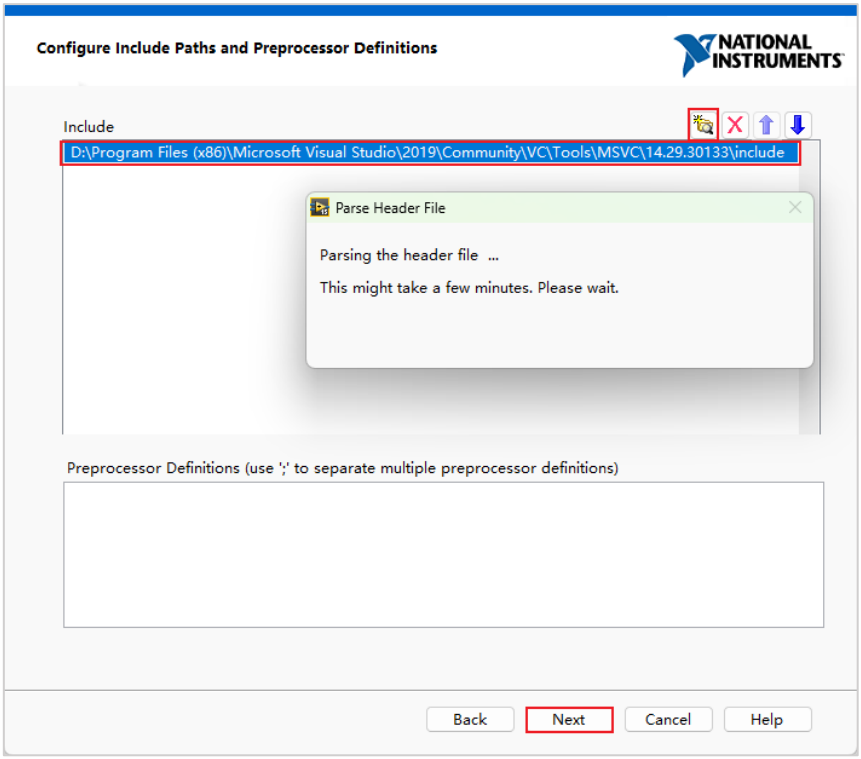
5. Select “Create VIs for a shared library” and click “Next.”



6. In the “Shared Library (.dll) File” and “Header (.h) File” sections, select the corresponding library files from the previously created folder. After selecting the shared library file path, the header file path can be automatically recognized and does not need to be selected again. Then click “Next.”

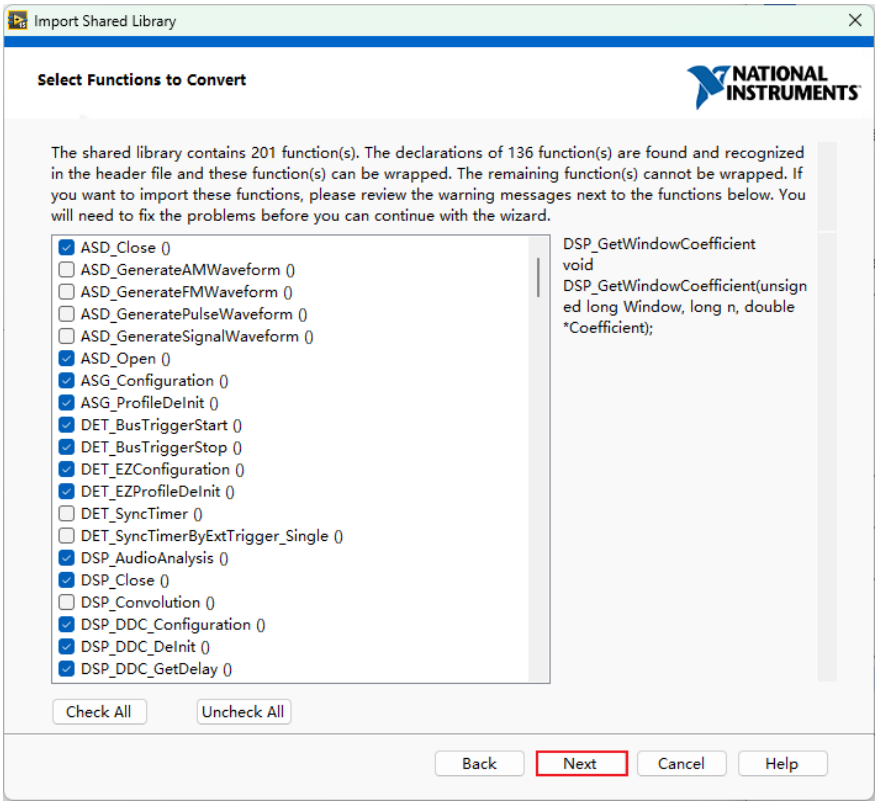


7. In the “Include” section, configure the paths for other dependency files, which are generally located in the folder where Visual Studio is installed, as shown in the image below, and then click “Next” to wait for parsing.

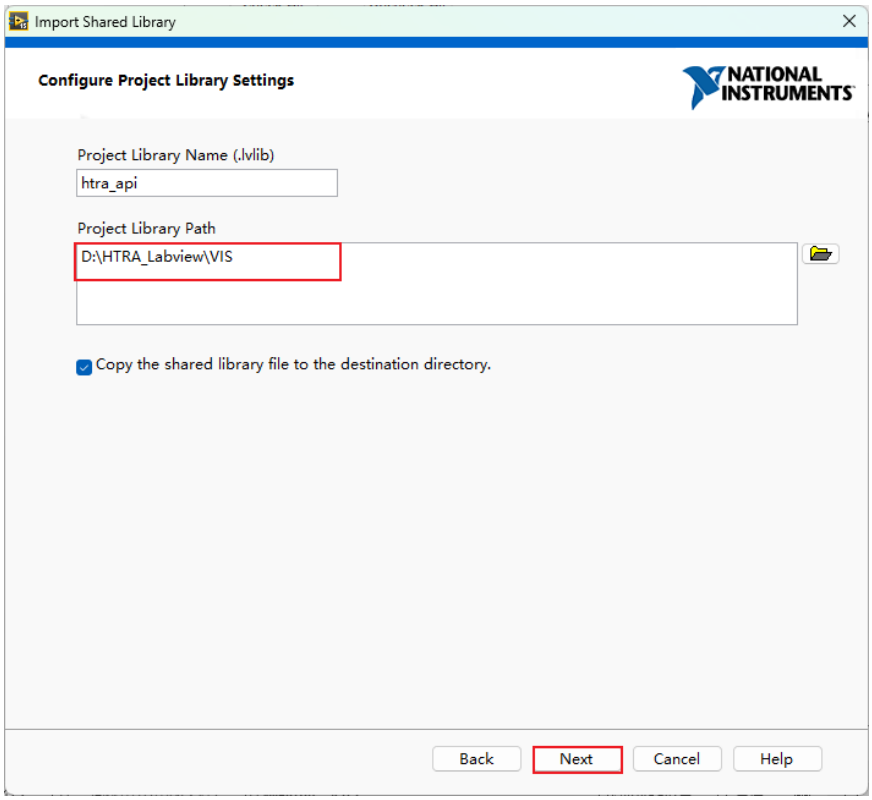


8. Select the functions to be exported; some functions have been deprecated or

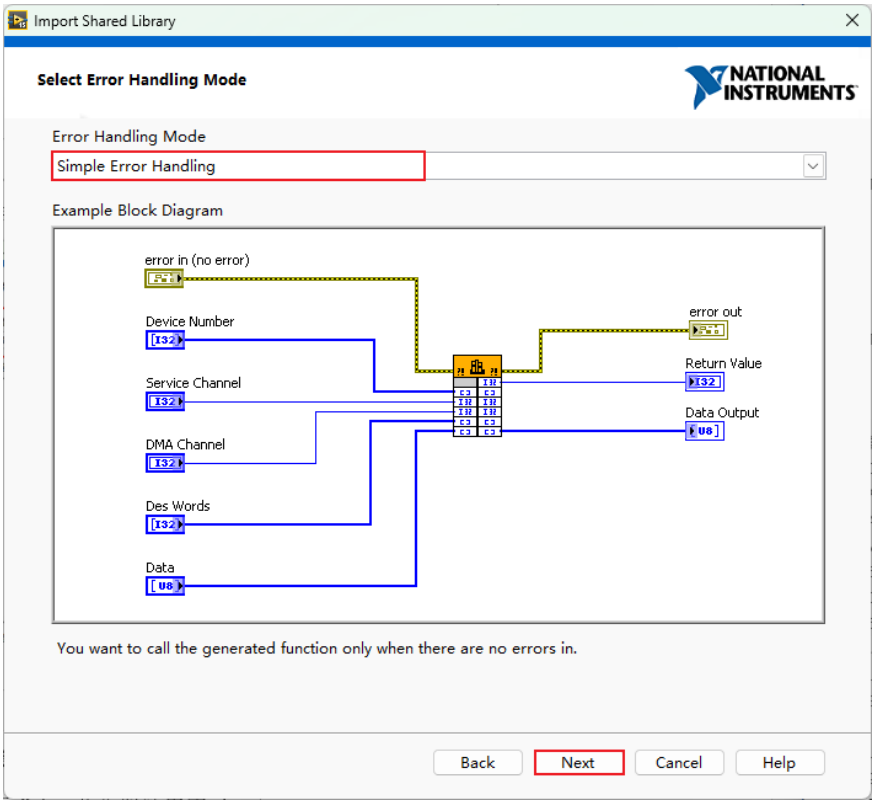
are not yet available. You can refer to htra_api.h for selection, and after making your selections, click “Next.”



9. For “Project Library Path,” select the VIS folder in the library and header file paths, and then click “Next.”

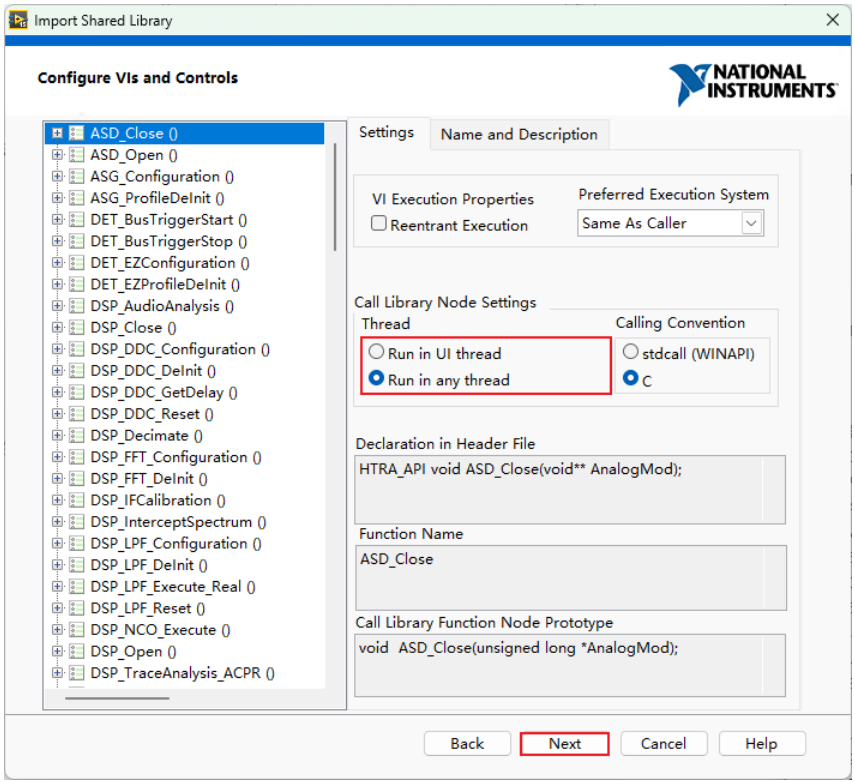


10. For “Error Handling Mode,” it is recommended to select “Simple Error Handling,” and then click “Next.”

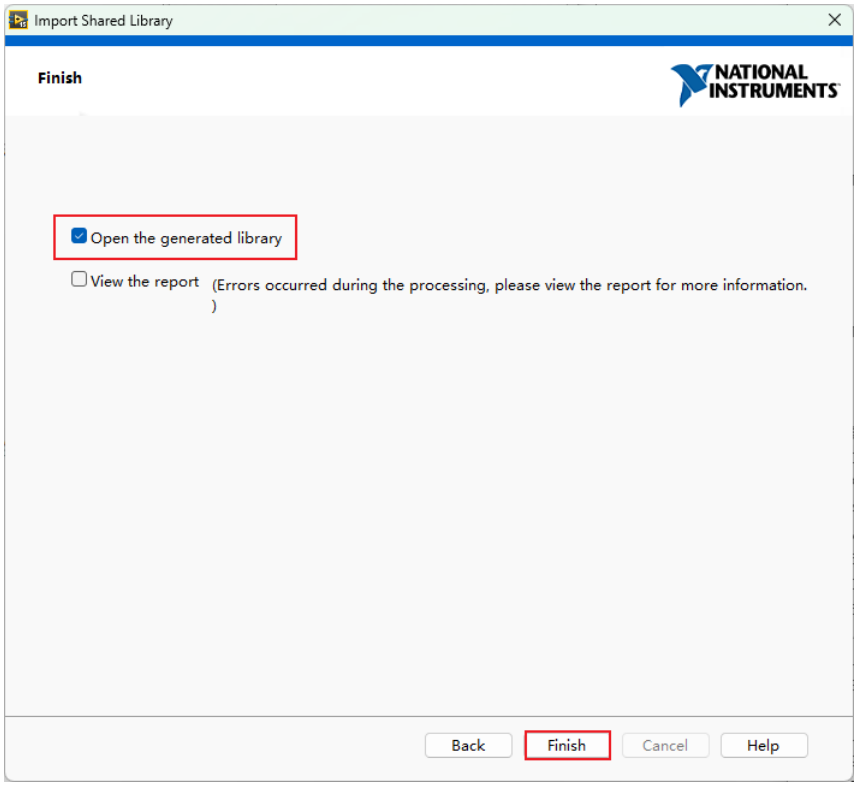


11. Set the call library node for each function to "Run in any thread." After all

settings are complete, click "Next" and wait for the VI to be generated.

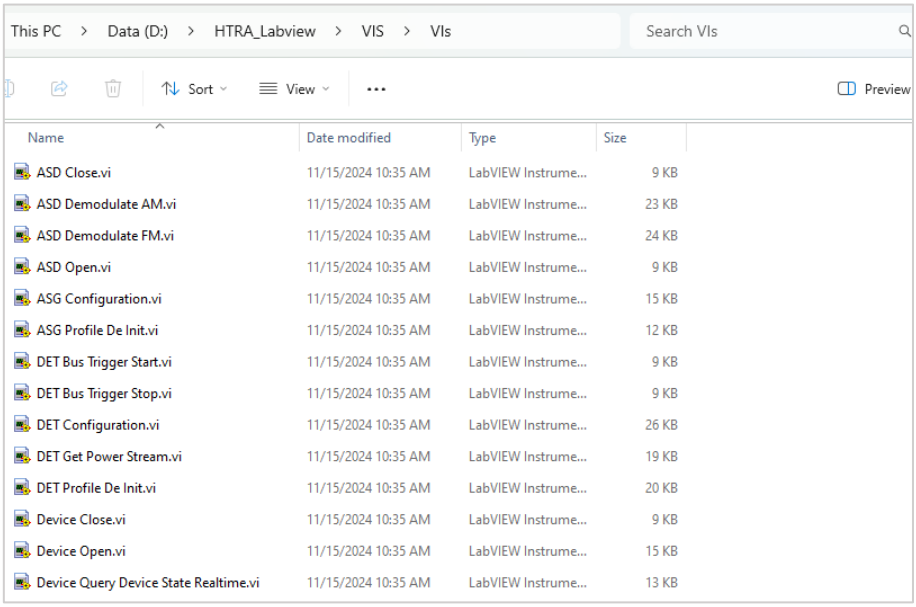


12. Check "Open the generated library," and "View the report" is optional; then click "Finish."



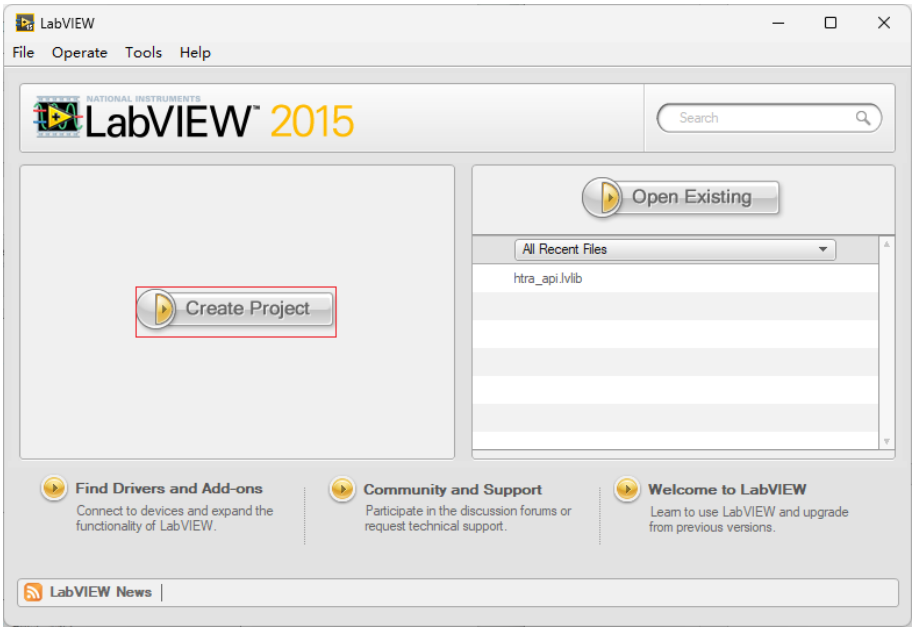
13. The VI in the VIS folder is the exported API function. Thus, the export of the

API function is complete.

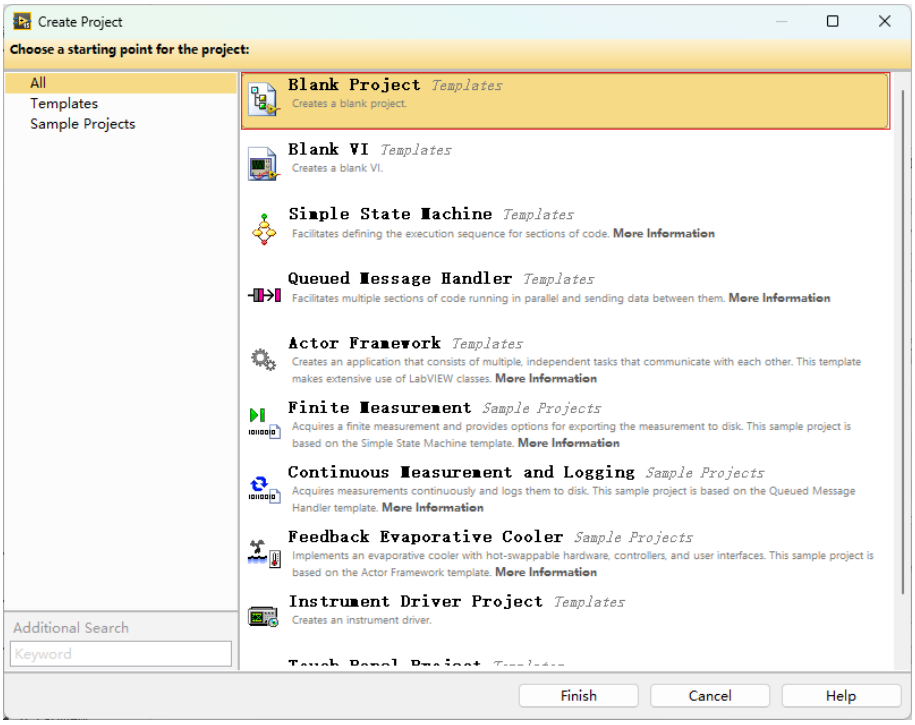


8.1.2 Using the API in the LabVIEW environment

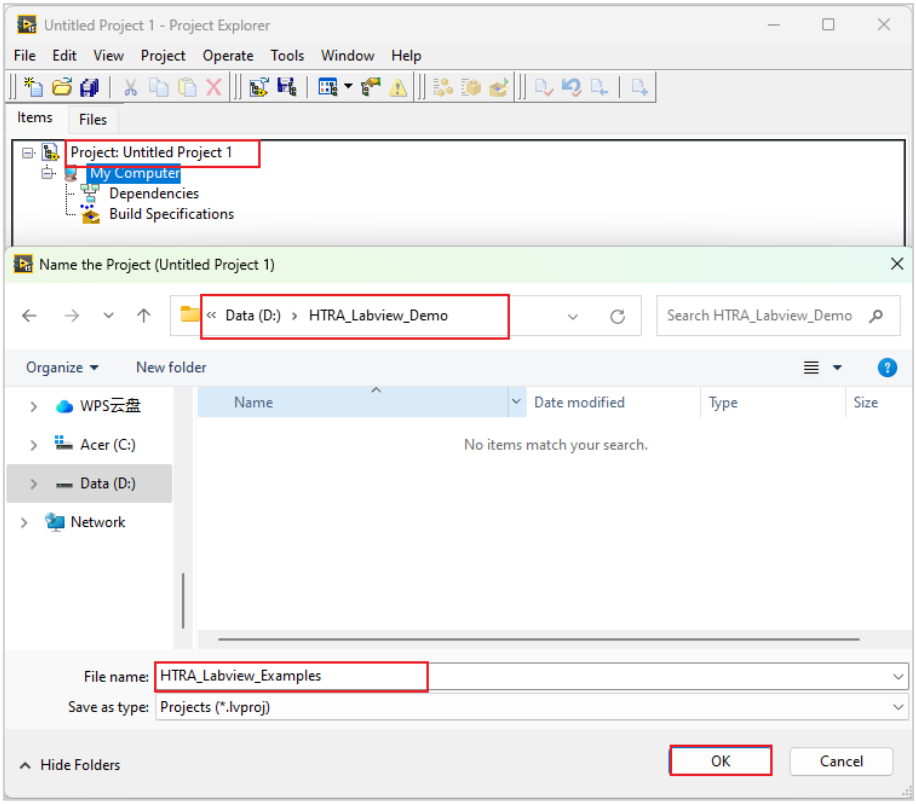
1. Open LabVIEW and click "Create Project."



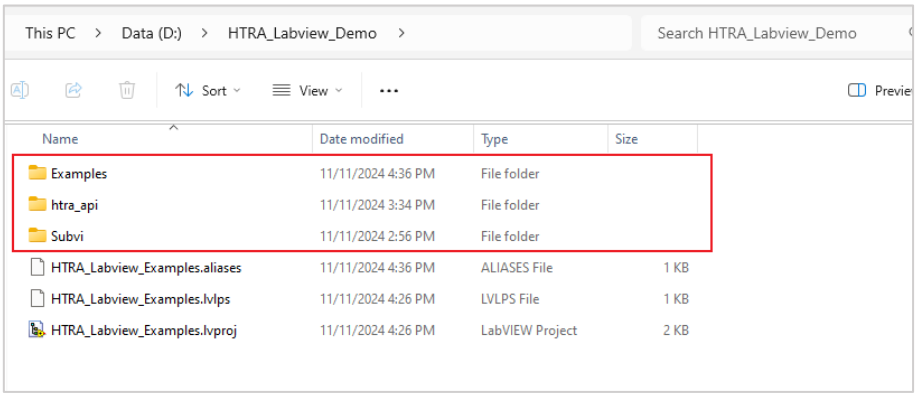
2. Select "Blank Project" and click "Finish."



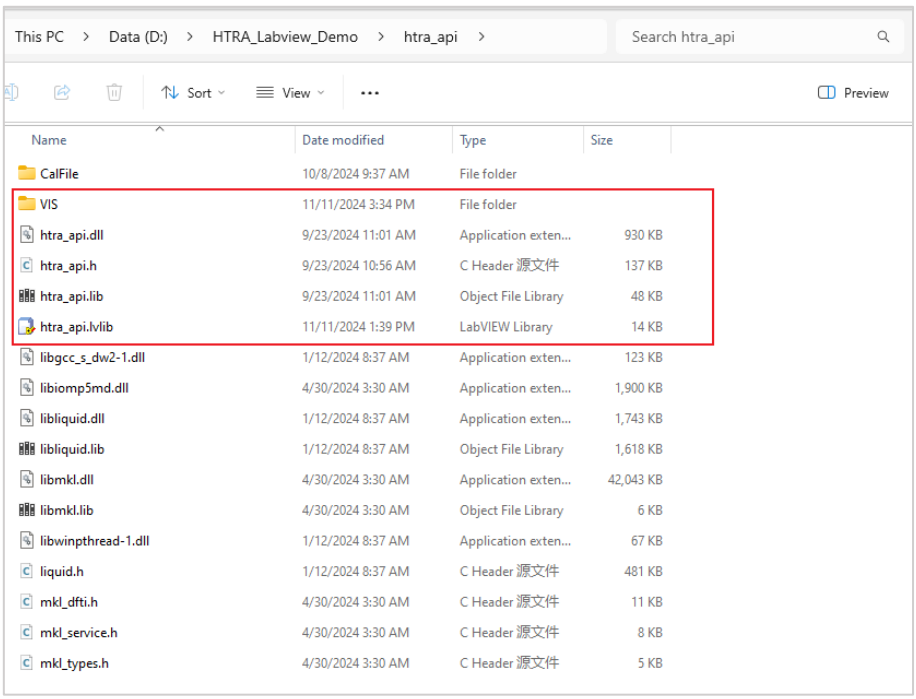
3. A blank unnamed project will appear; press "Ctrl+S" to save the project, select the project save path, name the project, and then click "OK."



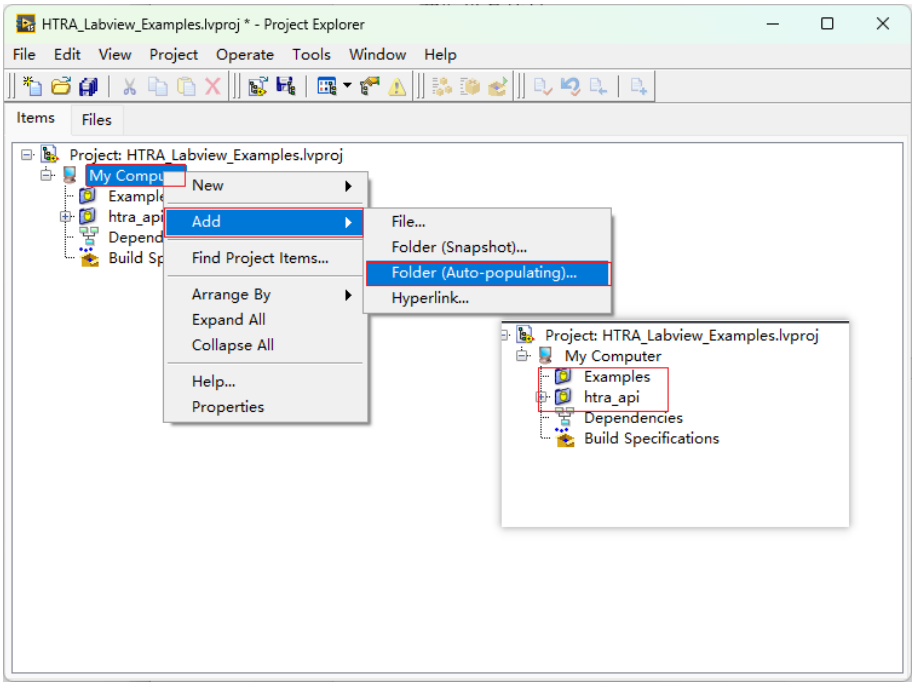
4. Copy the htra_api folder from the \Windows\HTRA_API\x86 folder on the USB drive to the same directory as the project. Additionally, you can create an Examples folder to store examples. If needed, you can also create a folder named Subvi to store sub VIs.



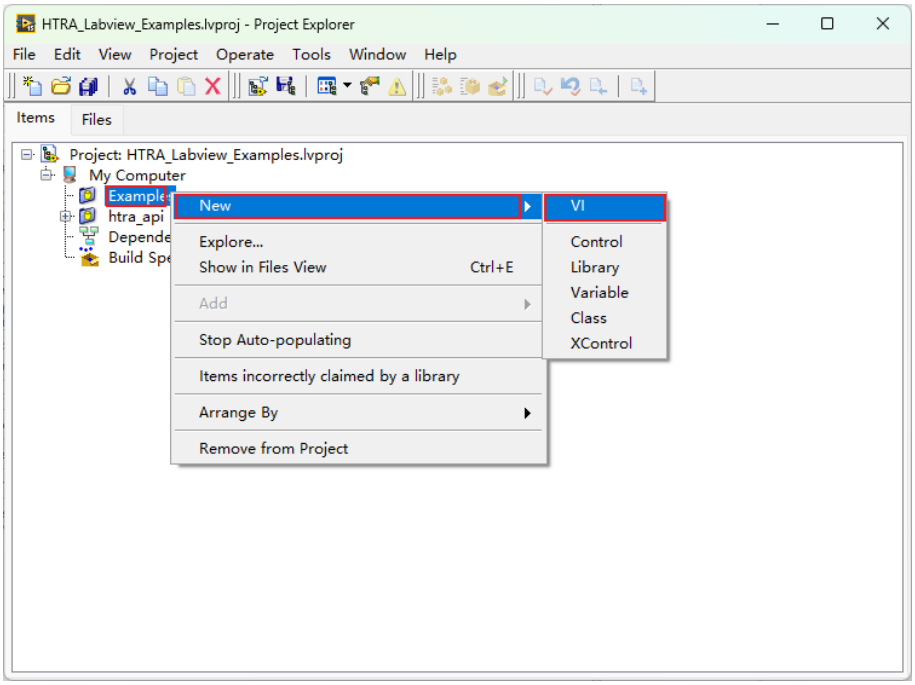
5. Copy the previously exported library functions, specifically the contents of the VIS folder created in section 8.1.1, to the htra_api folder.



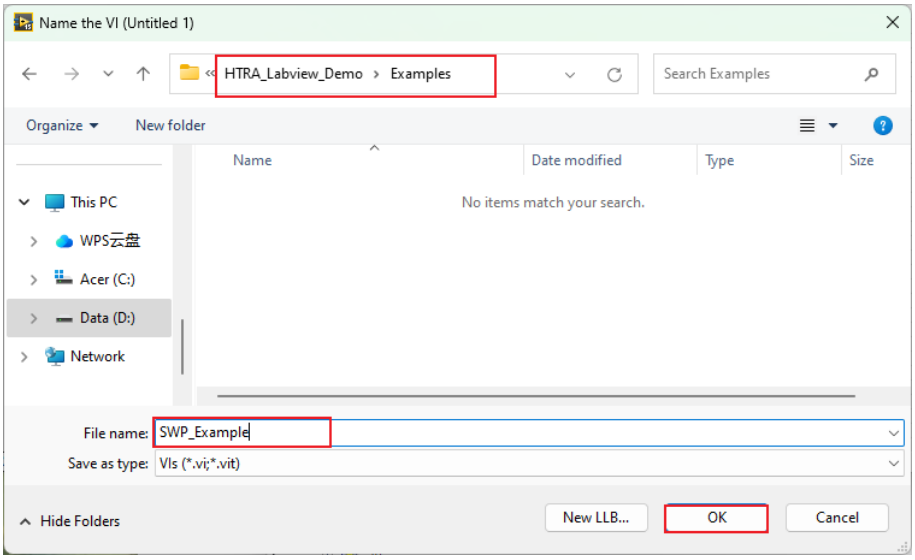
6. In the LabVIEW project, add the newly created Examples folder and the htra_api folder to the project, and then save.



7. A new VI is created in the Examples folder, where the exported Labview API functions can be called within the block diagram page, and the calling process is consistent with that in the C environment.



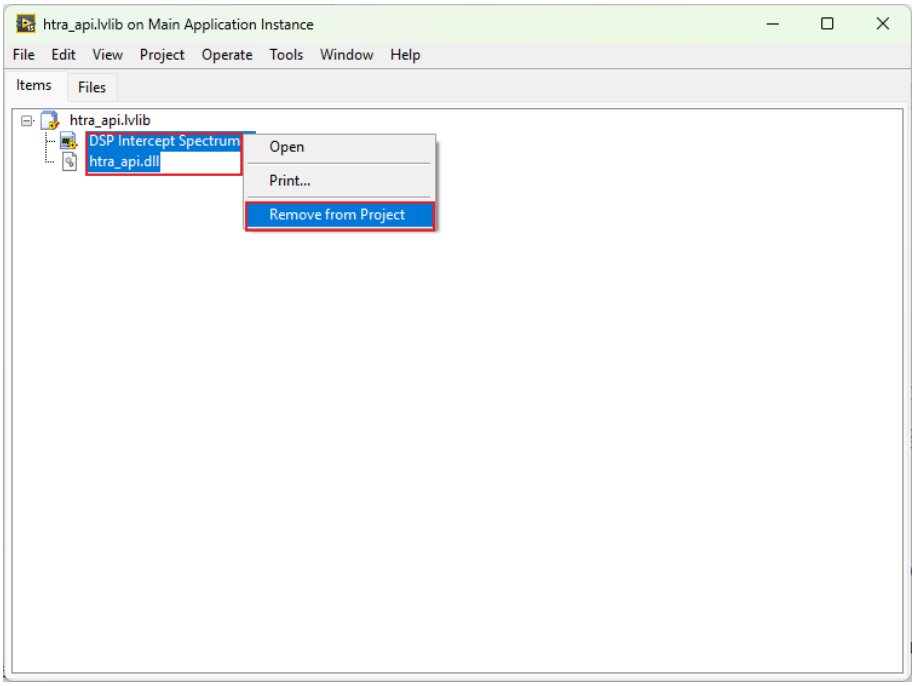
8. Finally, save this program to the Examples folder and rename it.



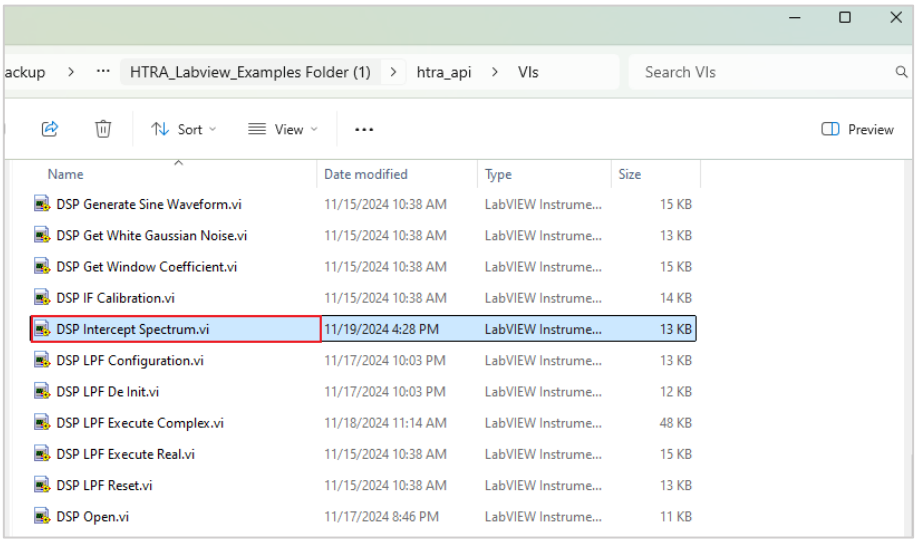
8.1.3 Use the newly exported library functions in an existing project.

Below, we will take the library function DSP_InterceptSpectrum() as an example to explain how to use the newly imported function in an existing project.

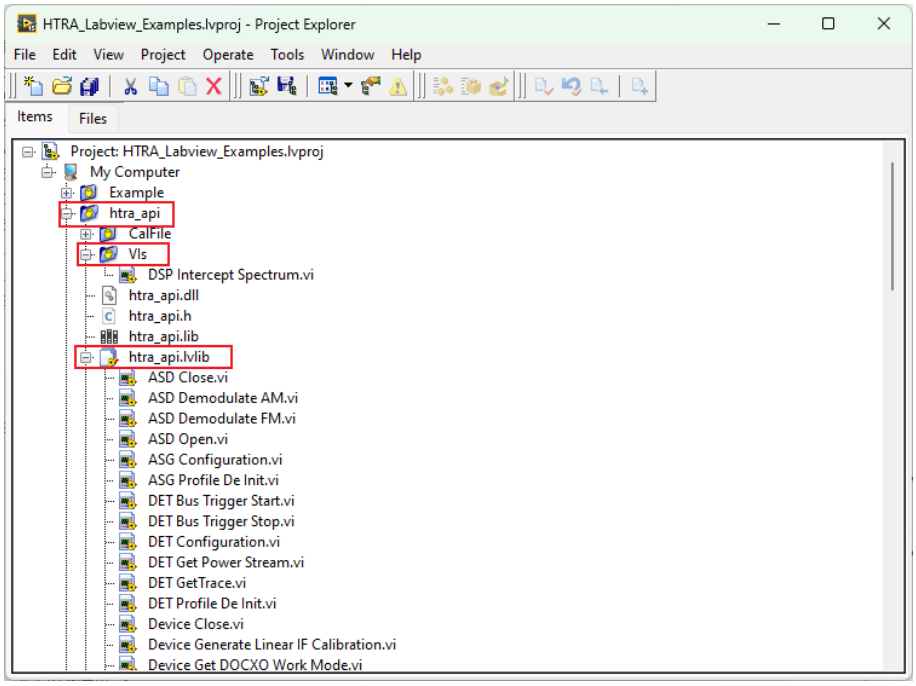
1. For the method of exporting library functions, please refer to section 8.1.1. Note that in the popped-up htra_api.lvlib, you should delete the exported functions and DLL files from the project, and then close htra_api.lvlib, as shown in the figure below.



2. Copy the exported DSP_InterceptSpectrum function VI to the htra_api\V Is folder of the existing project.



3. Open the project with LabVIEW, then open the htra_api\V Is folder and htra_api.lvlib within the project.



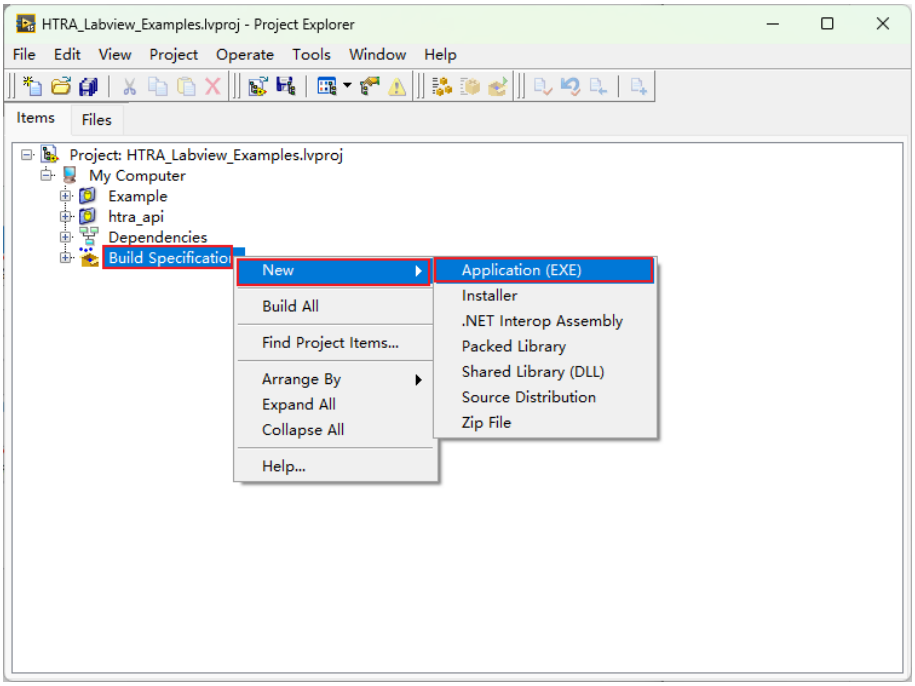
4. Drag the DSP_InterceptSpectrum function VI from the V Is folder into htra_api.lvlib. At this point, you can use the newly added VI function normally.


Project: HTRA_Labview_Examples.lvproj
My Computer
Example
htra_api
CalFile
VIs
htra_api.dll
htra_api.h
htra_api.lib
htra_api.lvlib
ASD Close.vi
ASD Demodulate AM.vi
ASD Demodulate FM.vi
ASD Open.vi
ASG Configuration.vi
ASG Profile De Init.vi
DET Bus Trigger Start.vi
DET Bus Trigger Stop.vi
DET Configuration.vi
DET Get Power Stream.vi
DET GetTrace.vi
DET Profile De Init.vi
Device Close.vi
Device Open.vi
Device Query Device Info Realtime.vi
Device Query Device Info.vi
Device Query Device State Realtime.vi
Device Query Device State.vi
DSP Audio Analysis.vi
DSP Close.vi
DSP DDC Configuration.vi
DSP DDC De Init.vi
DSP DDC Execute.vi
DSP DDC Get Delay.vi
DSP FFT Configuration.vi
DSP FFT De Init.vi
DSP FFT IQS To Spectrum.vi
DSP Intercept Spectrum.vi
DSP LPF Configuration.vi
DSP LPF De Init.vi

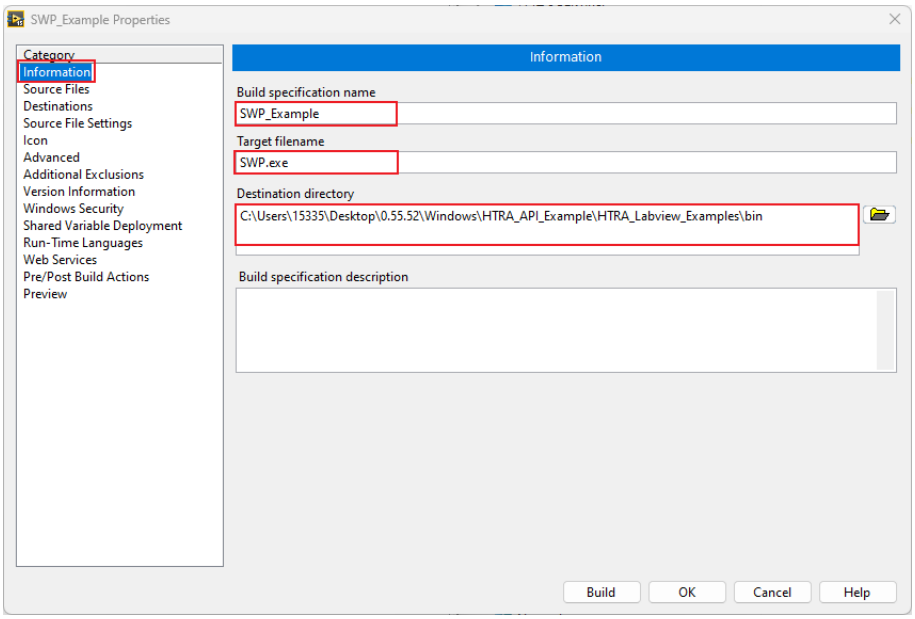
8.1.4 Generate an EXE from the VI in LabVIEW.

Below, we will take the LabVIEW project in the Windows\HTRA_API_Example folder on the USB drive as an example to explain how to generate an EXE from the program VI.

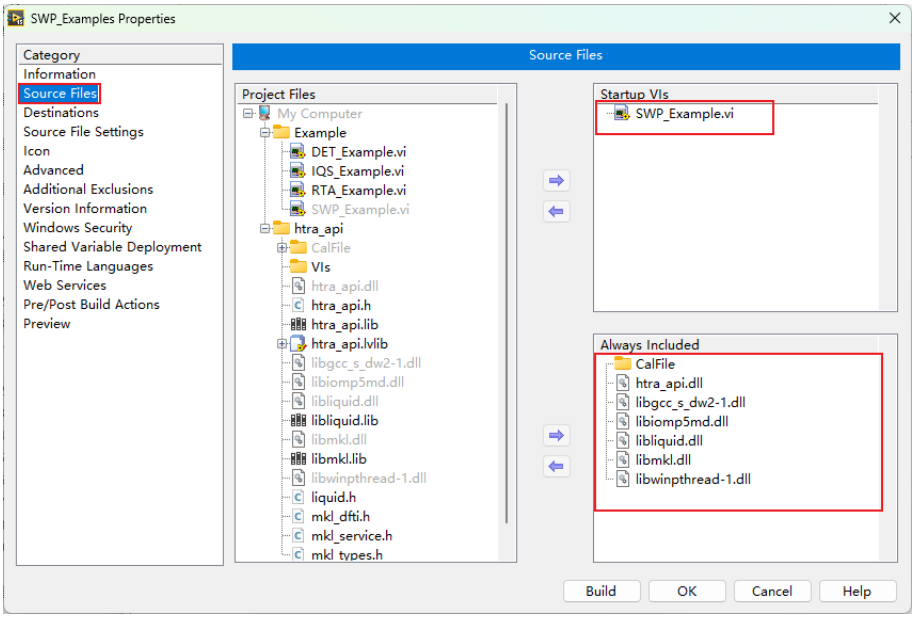
1. Open the LabVIEW project, then select "Build Specifications --> New --> Application (EXE)".



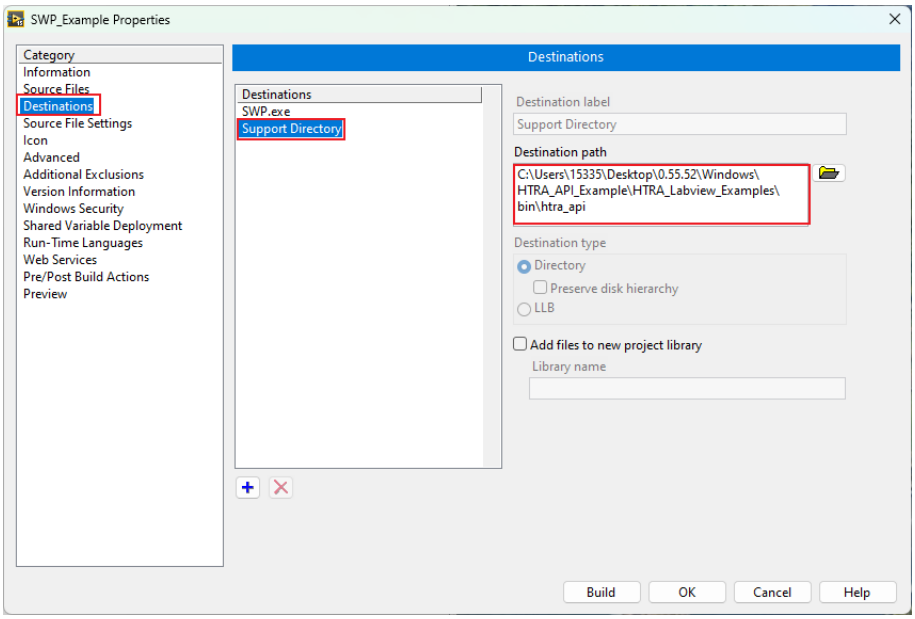
2. Fill in the standard name of the program file generated, the name of the exe, and the target directory path in "Information".



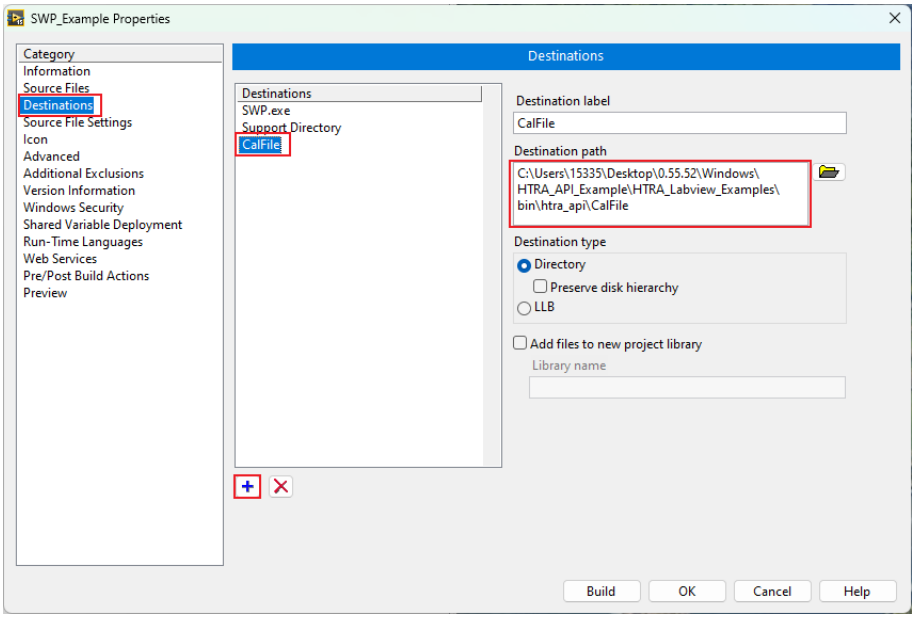
3. Select the vi programs, CalFile folder, and all dependent library files to be exported as exe in "Source Files".



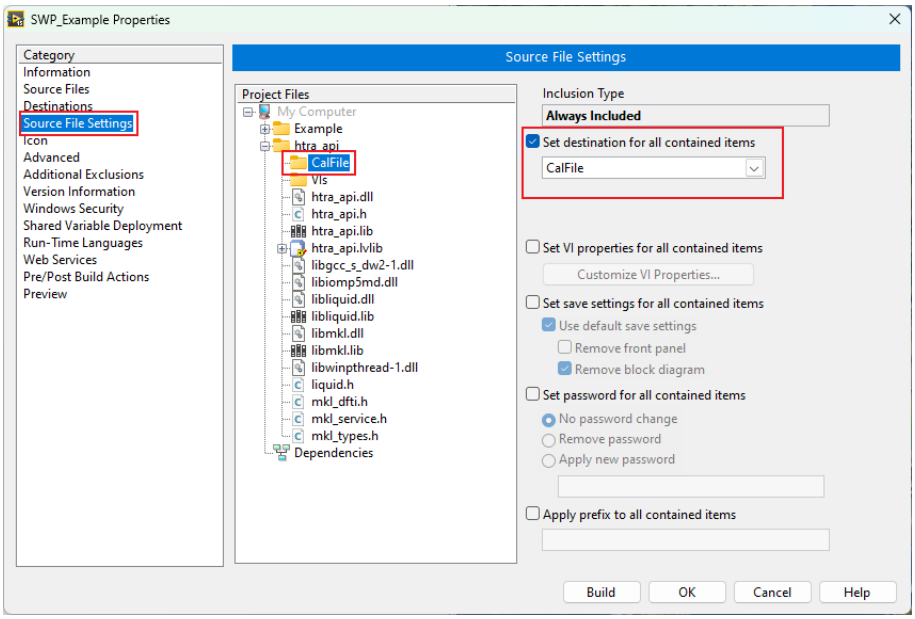
4. Add a layer of htra_api to the target path of "Support Directory" in "Destinations" to store the libraries and calibration folder.



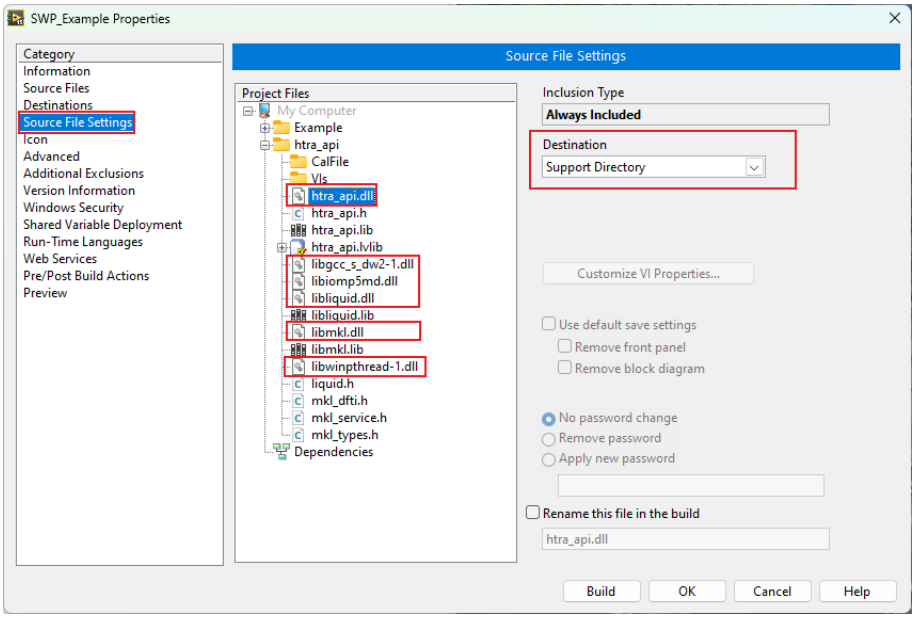
5. Add a CalFile folder to store the device's calibration files, and add the target path of CalFile to the directory under "Support Directory".



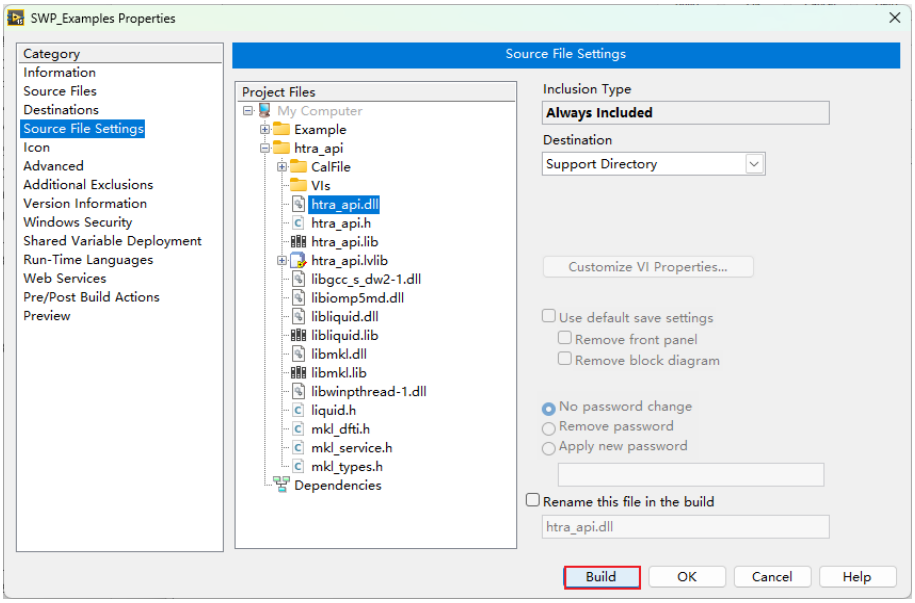
6. In "Source Files Settings", select the target path of the calibration files to "CalFile".



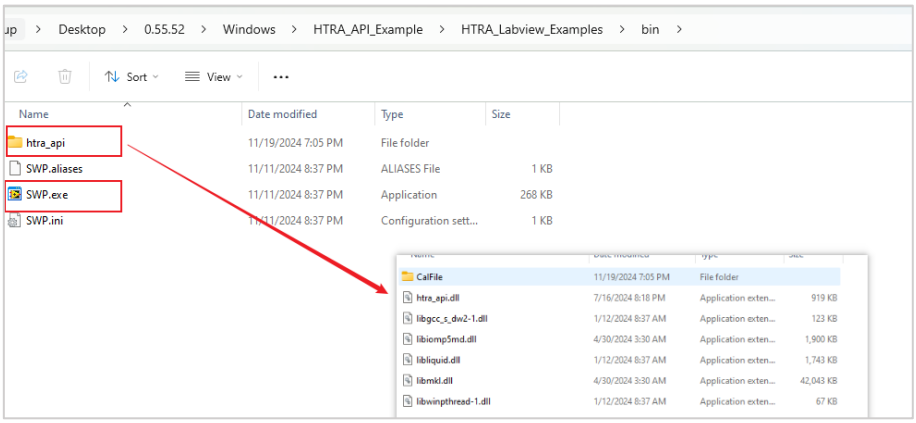
7. Select the target paths of all dependent libraries to the "Support Directory".



8. Click "Build".



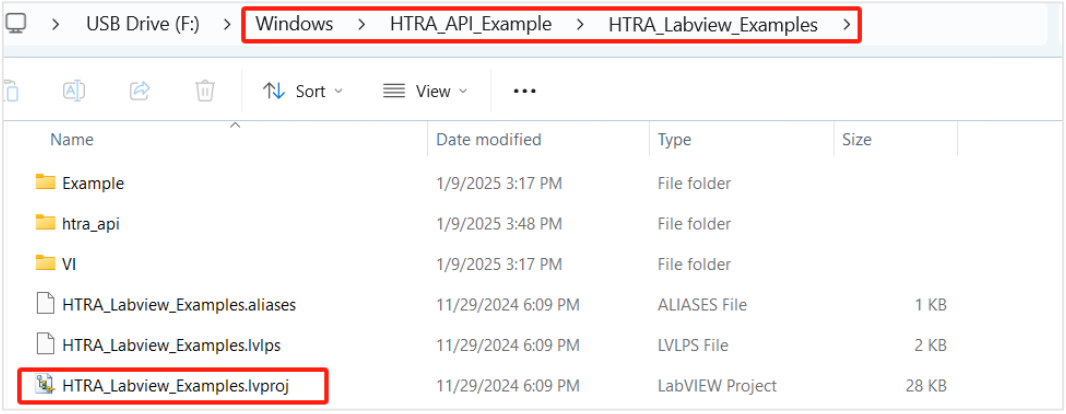
9. At this point, the generation of the exe program from the vi program in Labview is complete.



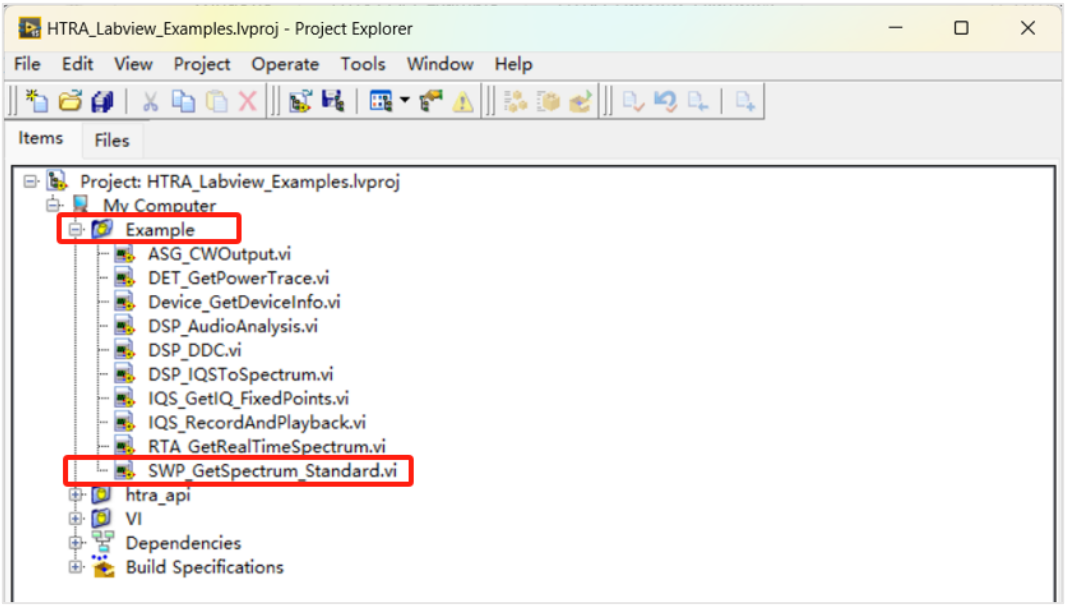
8.2 The usage process of Labview examples

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

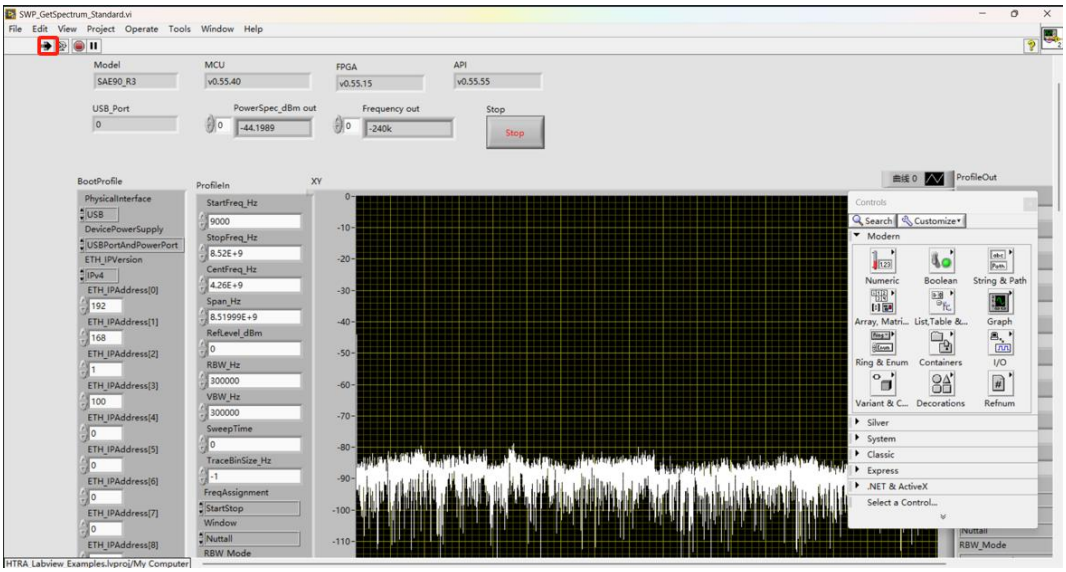
1. Open the accompanying USB drive using LabVIEW in the folder Windo ws\HTRA_API_Example\HTRA_Labview_Examples, and open the project HTRA_La bview_Examples.lvproj.



2. After opening the project, double-click to open any routine vi in the Example folder. For example, here we open SWP_GetSpectrum_Standard.vi to use the SWP mode routine.



3. Once the program is opened, simply click the run button in the upper left corner, as shown in the image where the program is running normally.


SWP_GetSpectrum_Standard.vi
File Edit View Project Operate Tools Window Help
Model
SAE90_R3
MCU
v0.55.40
FPGA
v0.55.15
API
v0.55.55
USB_Port
PowerSpec_dBm out
-44.1989
Frequency out
-240k
Stop
BootProfile
ProfileIn
XY
ProfileOut
StartFreq_Hz
9000
StopFreq_Hz
8.52E+9
CentFreq_Hz
4.26E+9
Span_Hz
8.51999E+9
RefLevel_dBm
RBW_Hz
300000
VBW_Hz
300000
SweepTime
TraceBinSize_Hz
FreqAssignment
StartStop
Window
Nuttall
RBW Mode
Controls
Search
Customize
Modern
Numeric
Boolean
String & Path
Array, Matri...
List,Table &...
Graph
Ring & Enum
Containers
I/O
Variant & C...
Decorations
Refnum
Silver
System
Classic
Express
.NET & ActiveX
Select a Control...
HTRA Labview Examples.lvproj/My Computer

8.3 Labview Example Description

8.3.1 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.

8.3.2 Standard spectrum acquisition

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

8.3.3 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.

8.3.4 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.

8.3.5 IQ to Spectrum Data

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

8.3.6 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.

8.3.7 Audio Analysis

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

8.3.8 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.

8.3.9 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.

8.3.10 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.

9. Linux

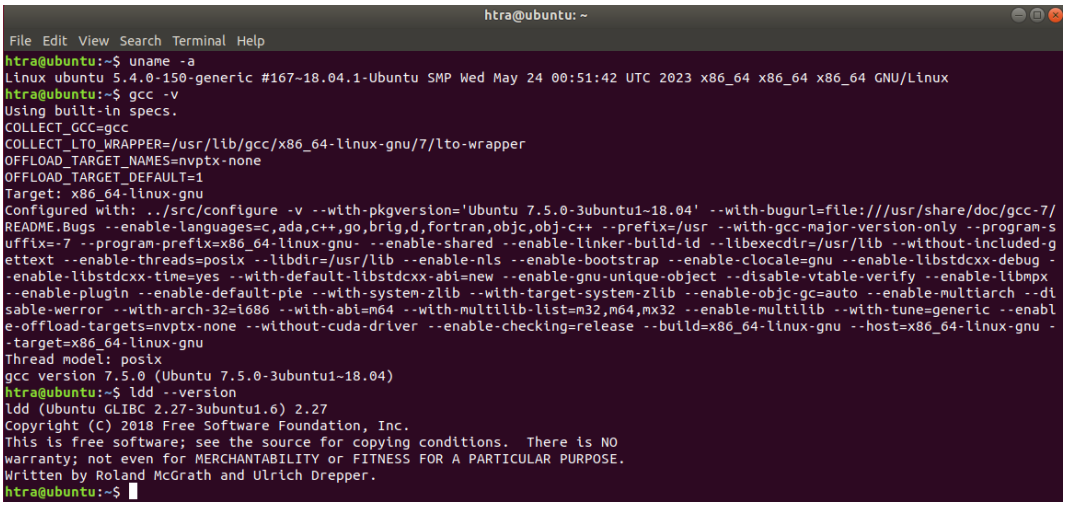
9.1 Environment Version Compatibility Self-Check

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.



4. Confirm whether the current environment is supported according to the terminal information comparison table. If it is not yet supported, please contact technical support personnel.

X86 processor	Supports Intel and AMD processors
ARM processor	aarch64 (armv8), armv7 processors, such as: Raspberry Pi 4b, RK3399, RK3568, RK3588, T507, NVIDIA Jetson TX2
Compilation	gcc4.8, glib2.17 and above

environment	
Distribution	Customized system for Raspberry Pi 4b, Ubuntu 18.04, etc.

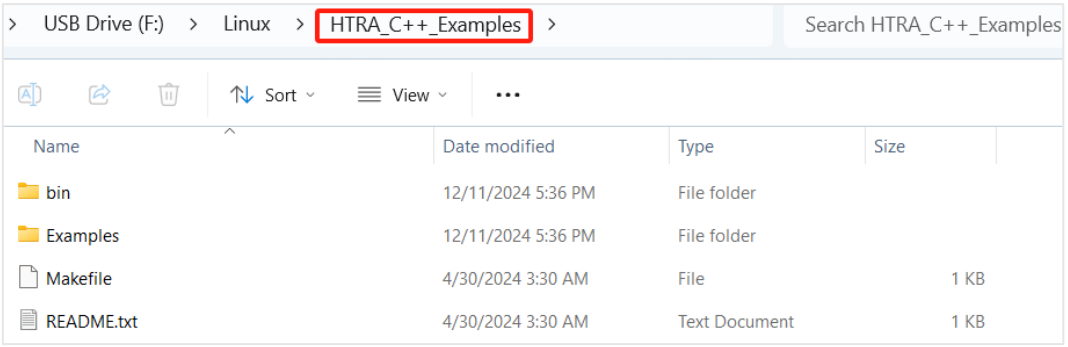
9.2 Accompanying documentation

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

9.2.1 HTRA_C++_Examples

The HTRA_C++_Examples folder contains:

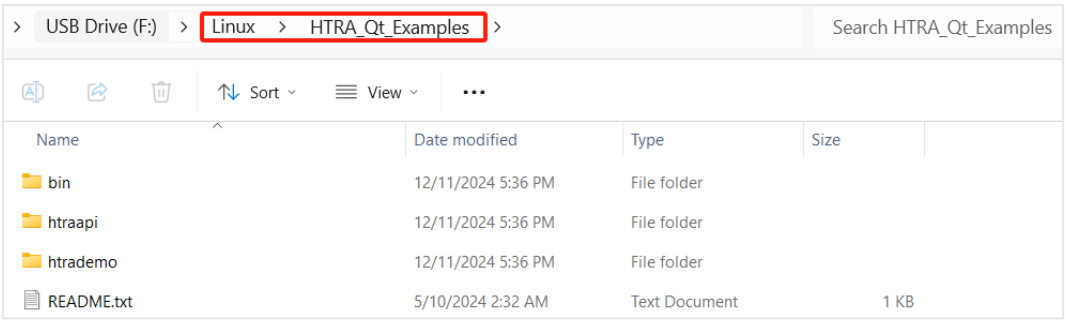
1. Examples folder: C++ example programs (see section 9.4 for usage).Chapter9.4
2. Makefile: A build script used to compile the example programs into executable files.
3. bin folder: Used to store device calibration files and executable files generated from the example programs.



9.2.2 HTRA_Qt_Examples

The HTRA_Qt_Examples folder contains:

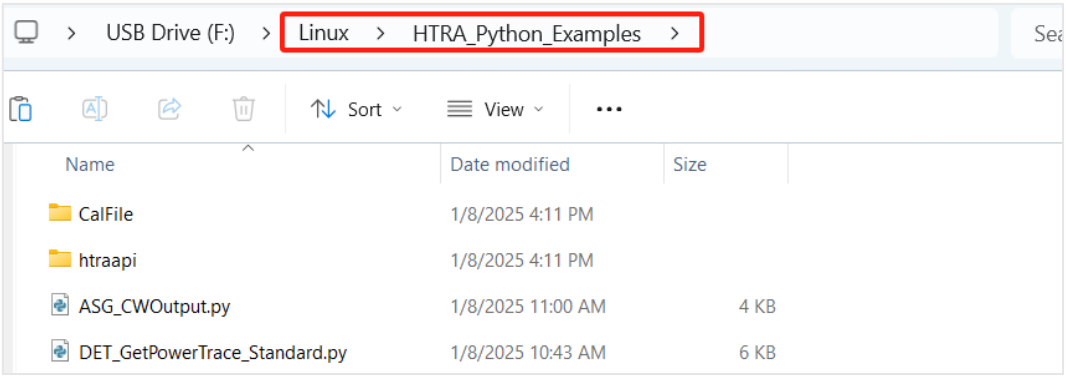
1. htrademo folder: Qt examples and pro files (see section 9.5 for usage).Chapter9.5
2. bin folder: Used to store device calibration files and executable files compiled from the example programs.
3. htraapi folder: Used to store dynamic link libraries.



9.2.3 HTRA_Python_Examples

HTRA_Python_Examples folder specifically contains:

1. Python example programs (see section 9.6 for usage).Chapter9.6
2. CalFile folder: stores device calibration files.
3. Htraapi folder: used to store dynamic link libraries.

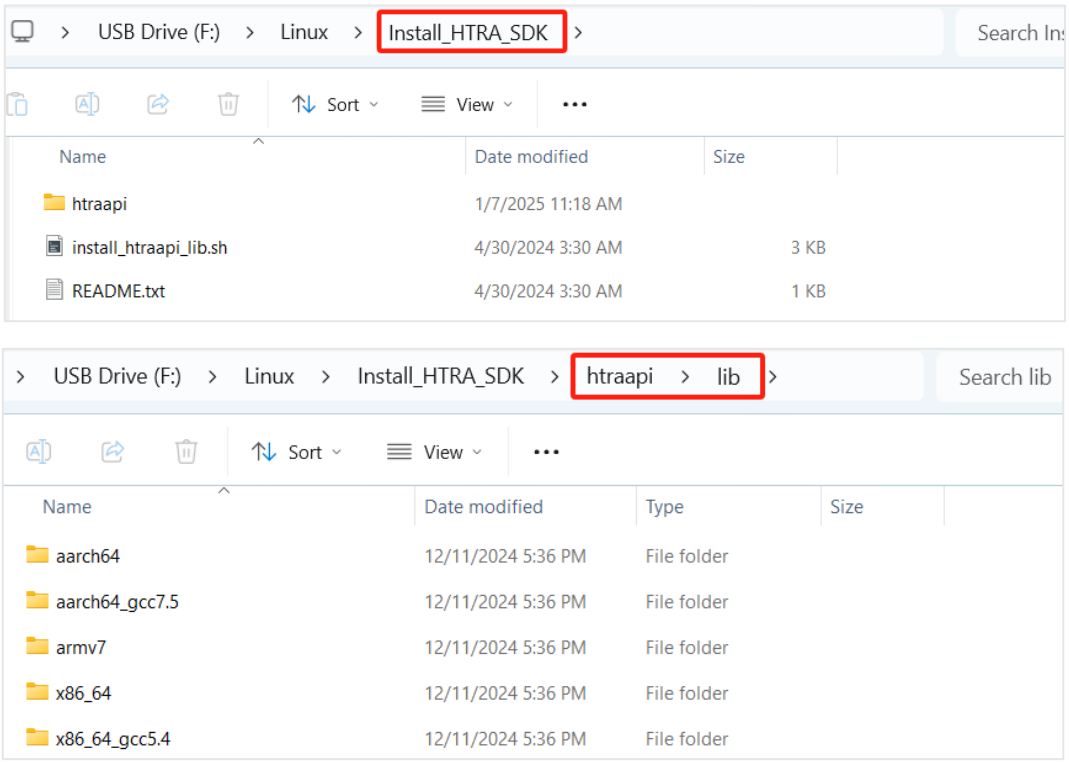


9.2.4 Install_HTRA_SDK

Install_HTRA_SDK folder contains:

1. install_htraapi_lib.sh: driver configuration script.
2. Install_HTRA_SDK\htraapi\configs folder: driver configuration files.
3. Install_HTRA_SDK\htraapi\inc folder: header files.
4. Install_HTRA_SDK\htraapi\lib\arrch64 folder: arrch64 architecture dynamic link libraries.
5. Install_HTRA_SDK\htraapi\lib\arrch64_gcc7.5 folder: Dynamic link library for arrch64 architecture with more efficient FFT (requires system gcc version higher than 7.5).

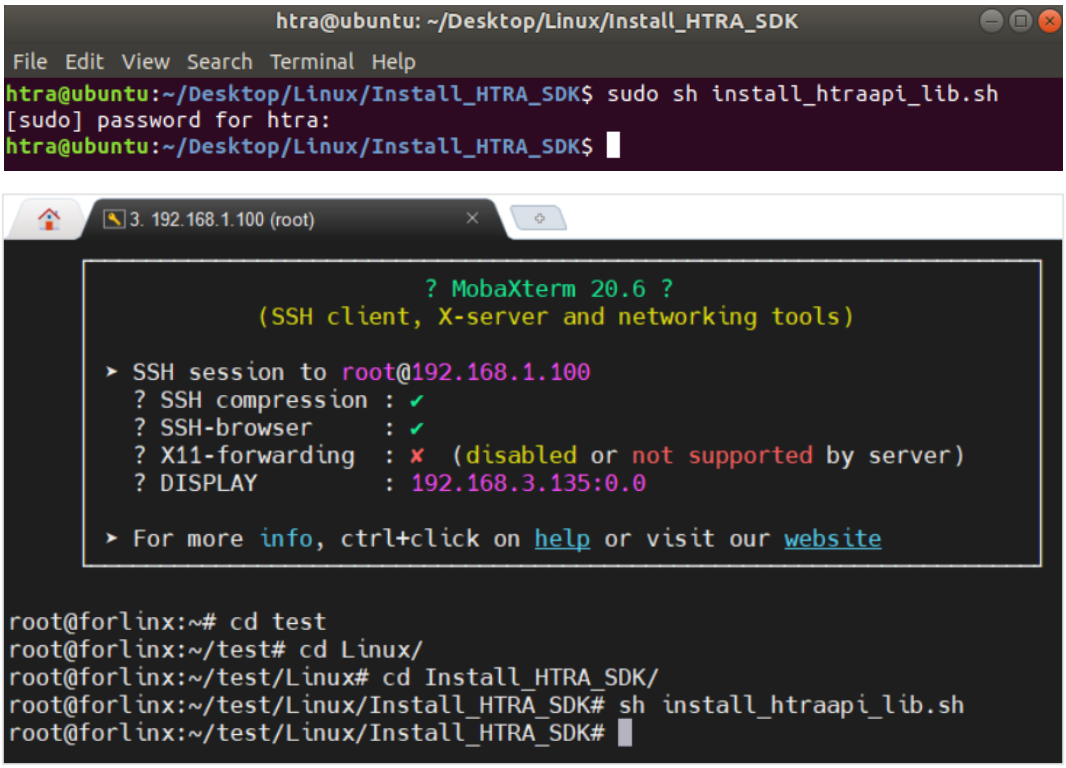
6. Install_HTRA_SDK\htraapi\lib\x86_64 folder: Dynamic link library for x86_64 architecture.
7. Install_HTRA_SDK\htraapi\lib \ x86_64_gcc5.4 folder: Dynamic link library for x86_64 architecture with more efficient FFT (requires system gcc version higher than 5.4).
8. Install_HTRA_SDK\htraapi\lib\armv7 folder: Dynamic link library for armv7 architecture.



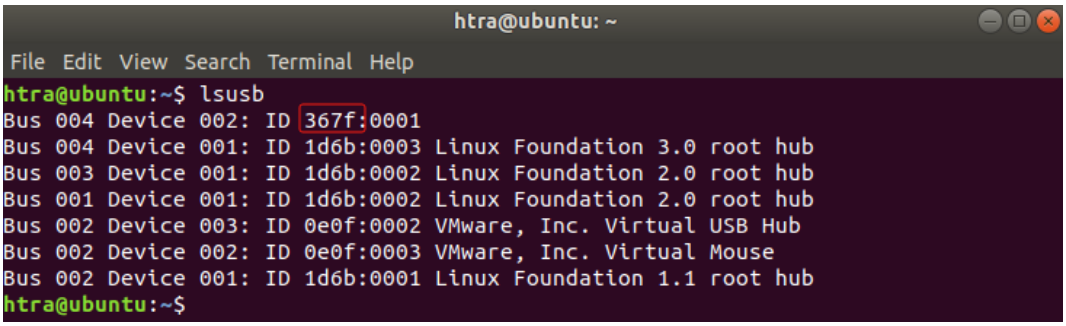
9.3 Driver file configuration

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

1. Driver file configuration: First, drag the Install_HTRA_SDK folder into the Linux host computer, then open a terminal in the Install_HTRA_SDK folder and enter “sudo sh install_htraapi_lib.sh” to configure the driver file. If the special development board does not have the sudo command, simply enter “sh install_htraapi_lib.sh”.



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.

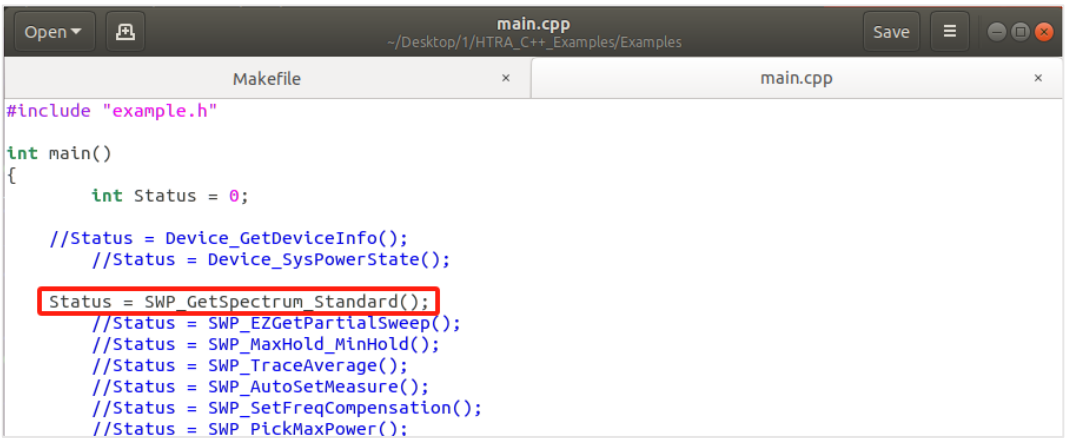


9.4 C++ example usage and project creation

9.4.1 C++ example usage

Under the premise that the device is properly connected and the driver files have been correctly configured as per Section 9.3, if you wish to use the C++ examples included on the USB drive, you can refer to the following process (the specific functions of the examples can be directly viewed in the description in Chapter 1):Chapter9.3Chapter One

1. Select the program to compile: First, copy the Linux\ HTRA_C++_Examples folder from the USB drive to the host computer. Double-click to open main.cpp in the Examples folder, and uncomment the example you need to test (the following steps will take the SWP_GetSpectrum_Standard example as an example), as shown in the figure to uncomment.



2. Refer to the system architecture in section 9.1, and based on the selected test example, open the terminal in the HTRA_C++_Examples folder, follo

wing the host computer's system architecture.Chapter9.1

For x86_86 system, input (the example routine here is SWP_GetSpectrum_Standard):

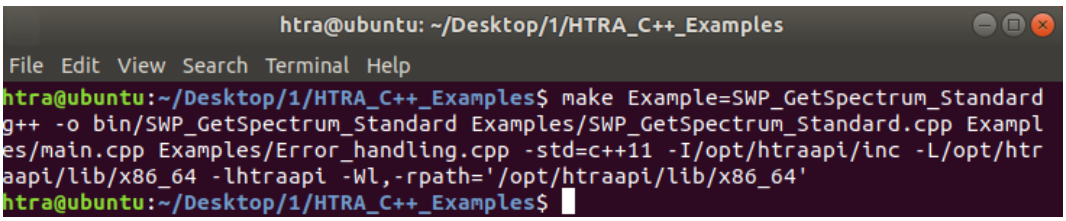
make Example=SWP_GetSpectrum_Standard

For aarch64 system, input:

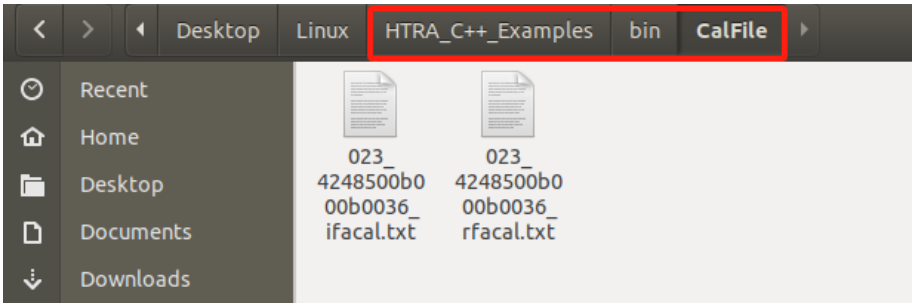
make TARG=aarch64 Example=SWP_GetSpectrum_Standard

For armv7 system, input:

make TARG=armv7 Example=SWP_GetSpectrum_Standard



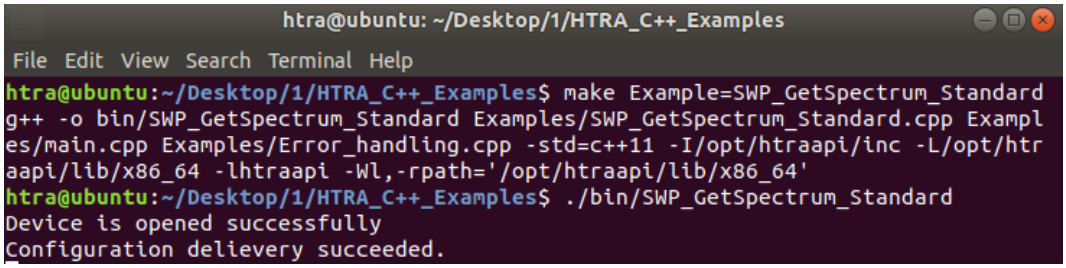
3. Verify the calibration file: Open the HTRA_C++_Examples\bin\CalFile folder, and ensure that the folder contains the device calibration files, as shown in the figure.



4. Run the program: After the program compiles successfully and the calibration is confirmed to be correct, open the terminal and input:

./bin/SWP_GetSpectrum_Standard

The example shown in the figure is a normal operation example.



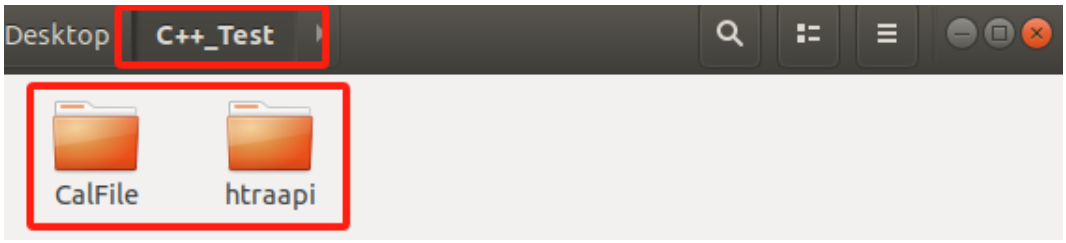
9.4.2 C++ Project Creation and Compilation

Assuming that the driver files have been correctly configured as per Section 9.3, if you want to create a C++ project for compilation, please refer to the following process:Chapter9.3

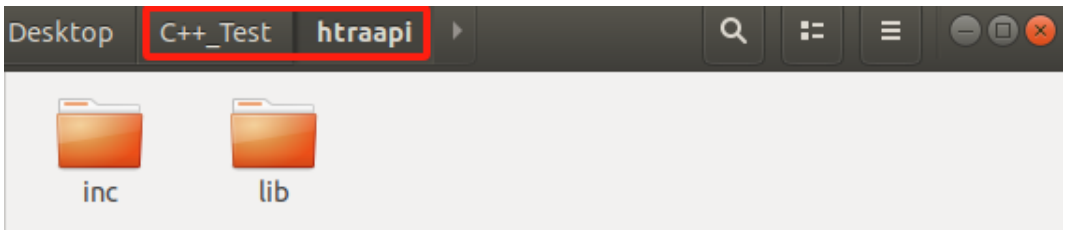
Write Code: Since the Linux dynamic link library provided with the USB drive is identical to that in Windows, the code only needs to comply with the API programming guidelines.

Compile and Run:

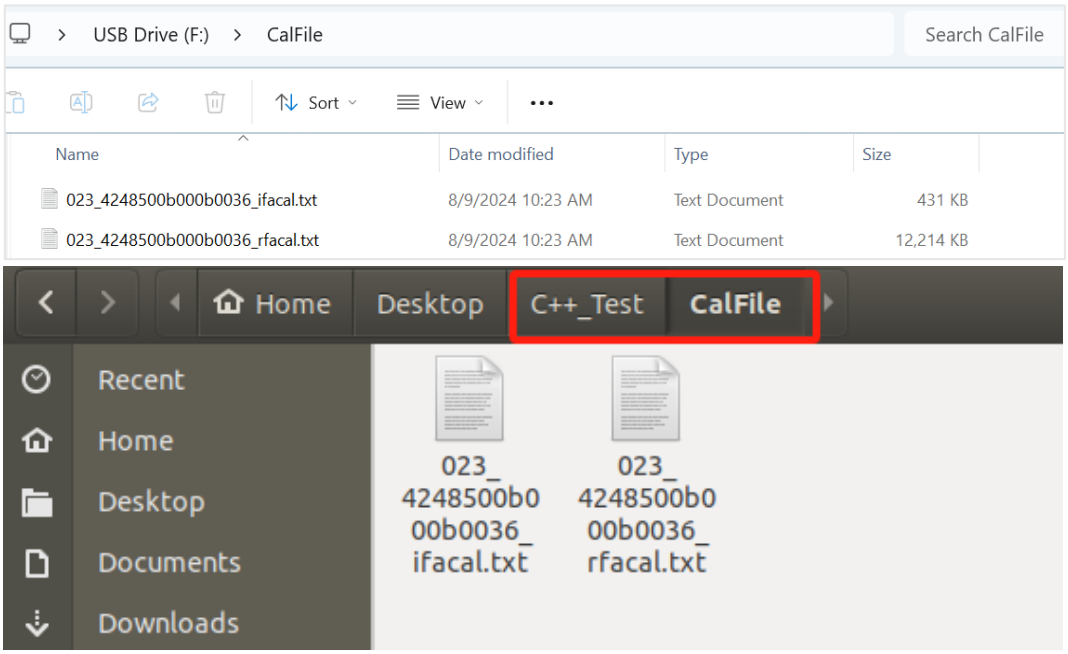
1. As shown in the figure, first create a new folder to store the entire project (taking C++_Test as an example), then create a CalFile folder within the folder to store calibration files, and create an htraapi folder to store header files and dynamic link libraries.



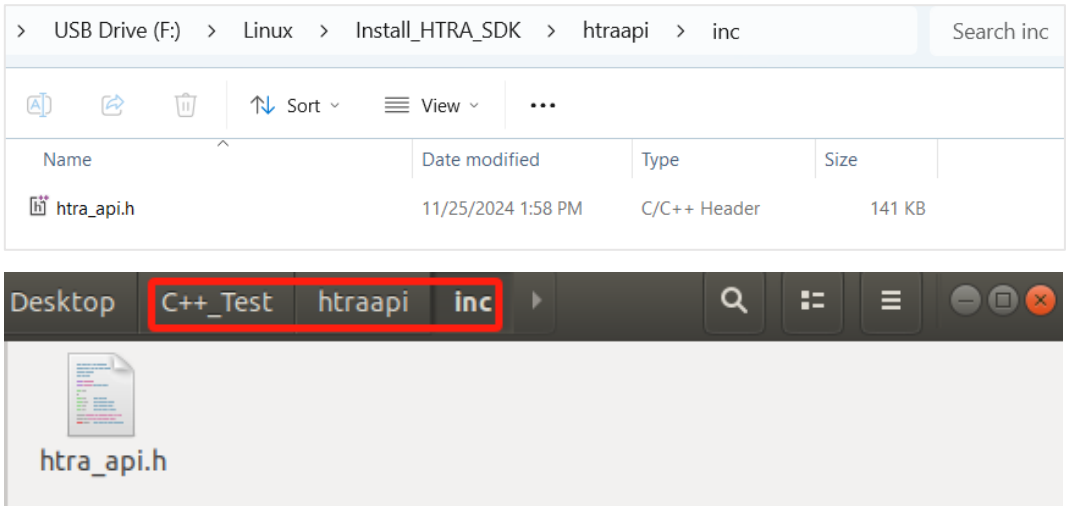
2. Create an inc folder under the htraapi folder to store header files, and a lib folder to store dynamic link libraries.



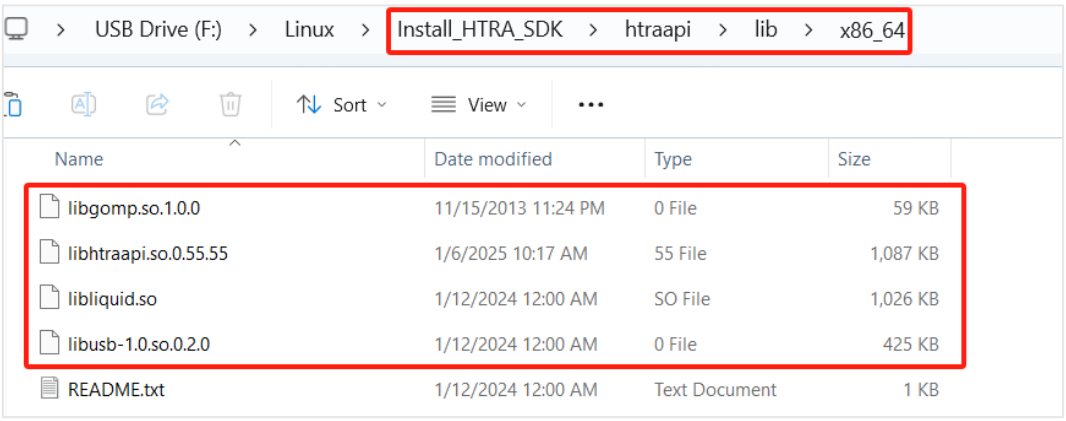
3. Copy the files from the CalFile folder on the provided USB drive to the newly created C++_Test\CalFile folder.

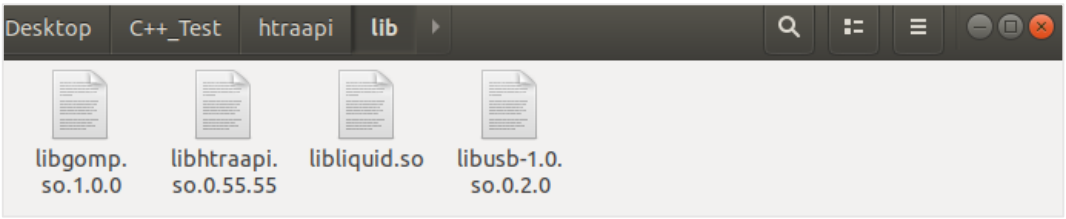


4. Copy the header files from the Linux\Install_HTRA_SDK\htraapi\inc folder on the provided USB drive to the newly created C++_Test\htraapi\inc folder.



5. Refer to the system architecture in section 9.1, and then according to the instructions in section 9.2.4, copy the dynamic link libraries corresponding to the architecture from the Linux\Install_HTRA_SDK\htraapi\lib folder to the newly created C++_Test\htraapi\lib folder (taking the x86_64 architecture host computer as an example).Chapter9.1Chapter9.2.4





6. Open the terminal in the lib folder location and enter the following commands to create soft links for the copied dynamic link libraries (the commands for the dynamic link libraries of the three architectures are the same):

ln -sf libhtraapi.so.0.55.5 libhtraapi.so.0 (Here, the libhtraapi.so library is taken as an example with version 0.55.55; modify the version number for other versions.)

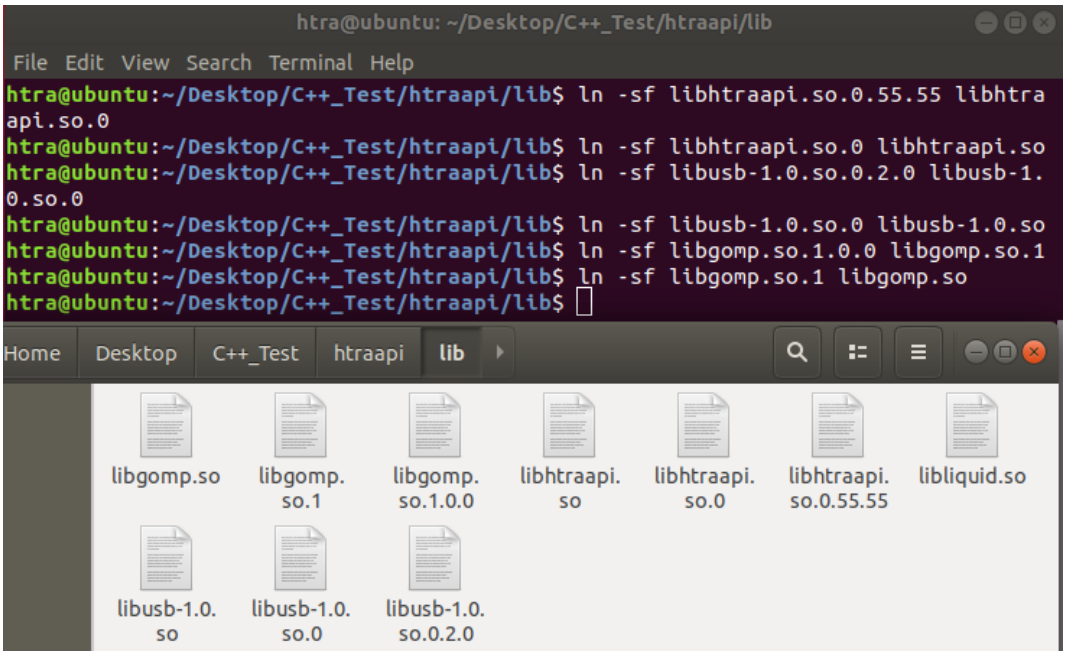
ln -sf libhtraapi.so.0 libhtraapi.so

ln -sf libusb-1.0.so.0.2.0 libusb-1.0.so.0

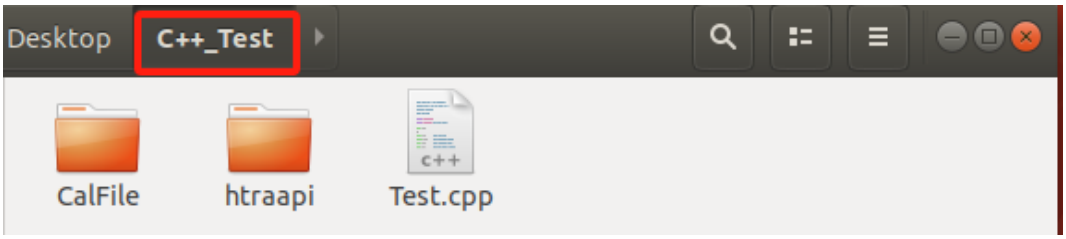
ln -sf libusb-1.0.so.0 libusb-1.0.so

ln -sf libgomp.so.1.0.0 libgomp.so.1

ln -sf libgomp.so.1 libgomp.so



7. Store the written code files in the outermost folder of C++_Test.



8. Compile to generate the executable file: First, check the system architecture according to the process in section 9.1, then open the terminal in the C++_Test folder (the image below takes the x86_64 system as an example), and input according to the host system architecture.Chapter9.1

For x86_64 system input (example test.cpp here):

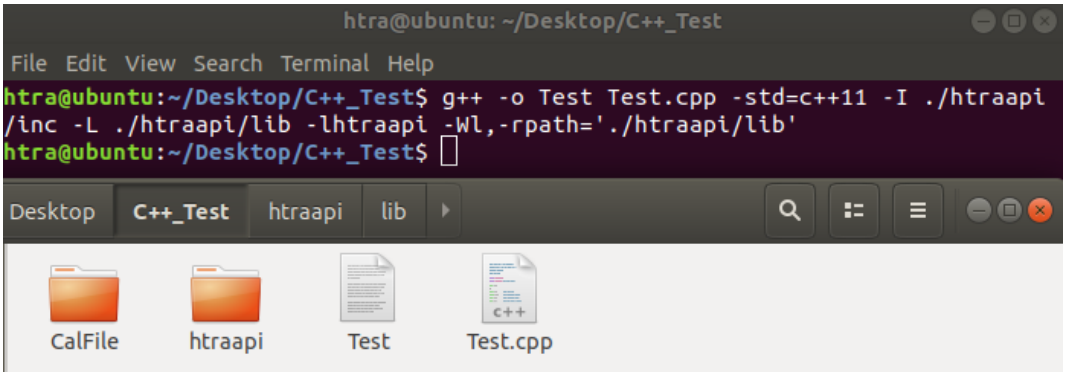
g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L ./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'

Input for arrch64 system:

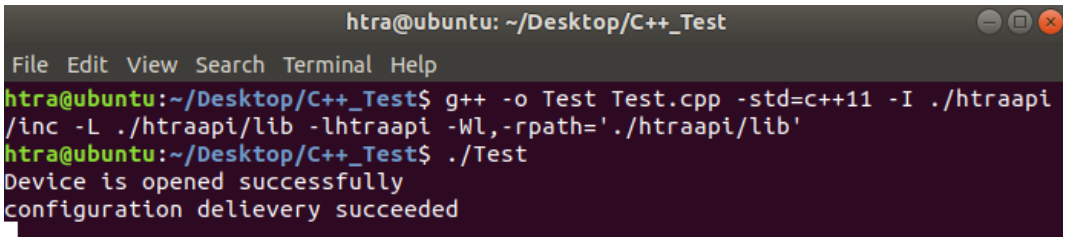
aarch64-linux-gnu-g++-o Test Test.cpp -std=c++11 -I ./htraapi/inc -L ./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'

For armv7 system, input:

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



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



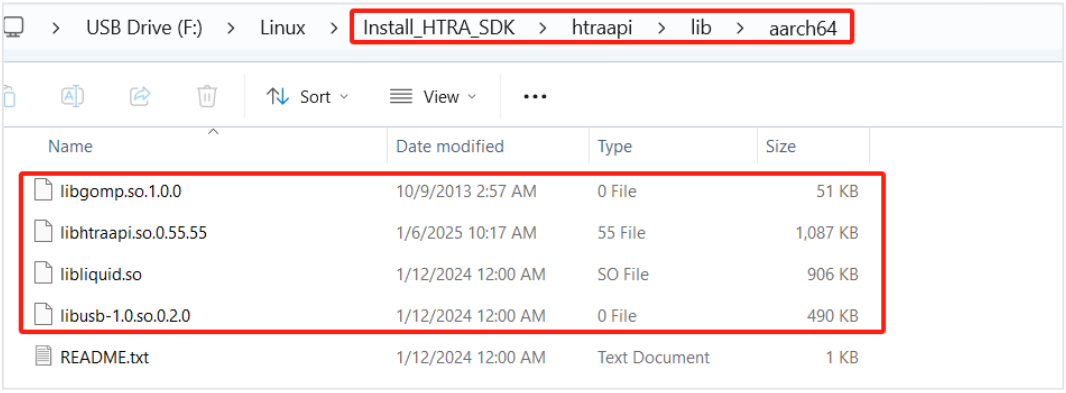
9.4.3 C++ Project Cross-Compilation

Under the premise that the host computer has a cross-compilation toolchain, if you want to cross-compile to use the device, please refer to the following process (taking the cross-compilation of an aarch64 executable program on an x86_64 host computer as an example):

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

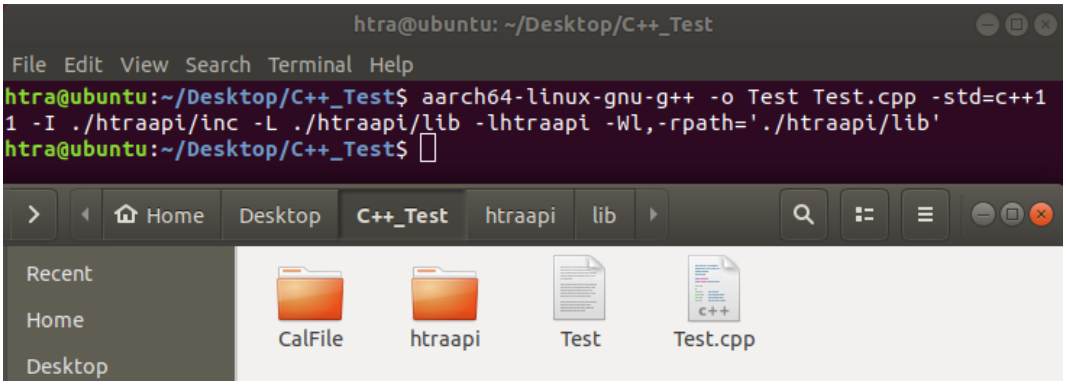
(1) Create a project and place the calibration files and header files according to steps 1-4 in the compilation and running method in section 9.4.2.Chapter9.4.2

(2) Place the cross-compiled target architecture library files as per step 5. For example, when cross-compiling an executable for the arrch64 architecture, please place the library files for the arrch64 architecture.

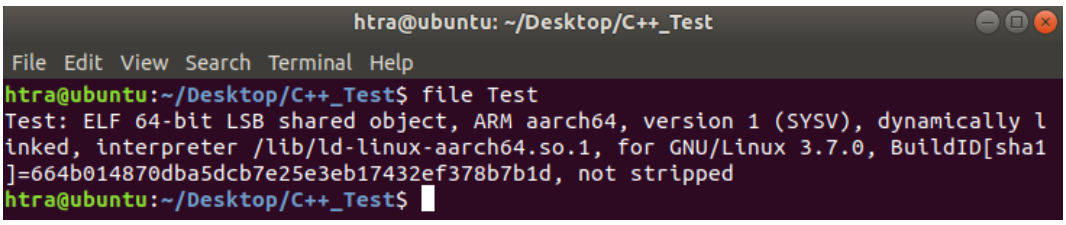


(3) Perform soft linking and program writing storage according to steps 6-7.

(4) Compile the executable file using the target architecture compilation command as described in step 8. The compilation command for the aarch64 architecture is expected to be used when compiling the executable file for the aarch64 system, as shown in the figure below.



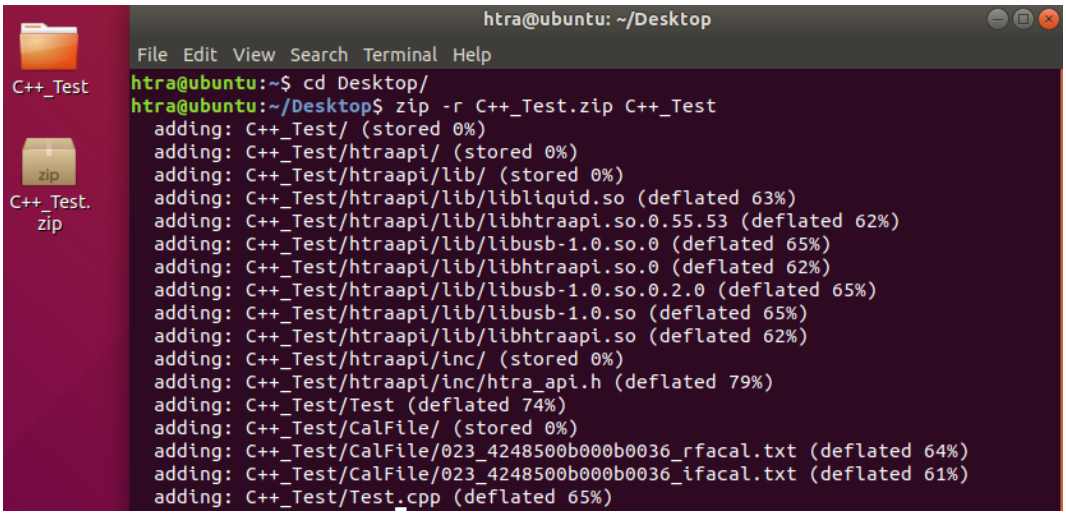
(5) After generating the executable file, input 'file Test' to check the architecture of the executable program, and you will see that the current executable program architecture is aarch64.



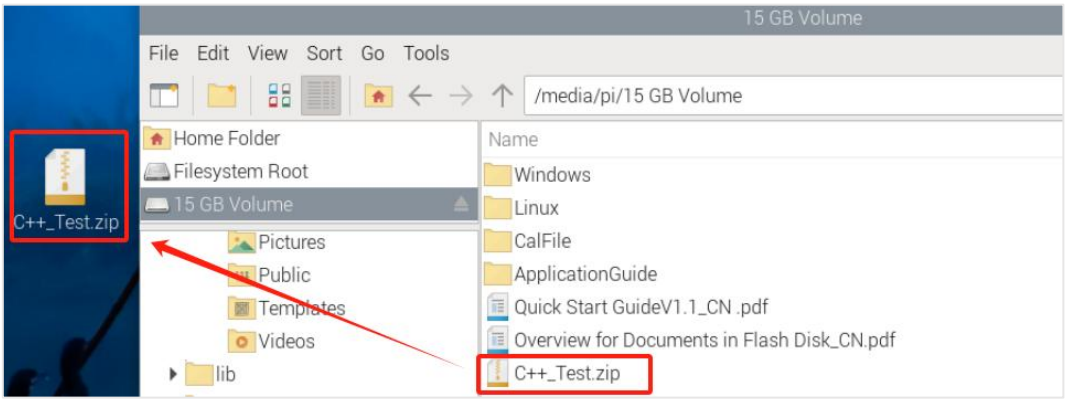
2. At this point, the executable program has been successfully generated, and you can now run the program on the aarch64 host machine using the device:

(1) Navigate to the project directory (the example project is located on the desktop, so input 'cd Desktop/'), and compress the entire folder into a zip file, for

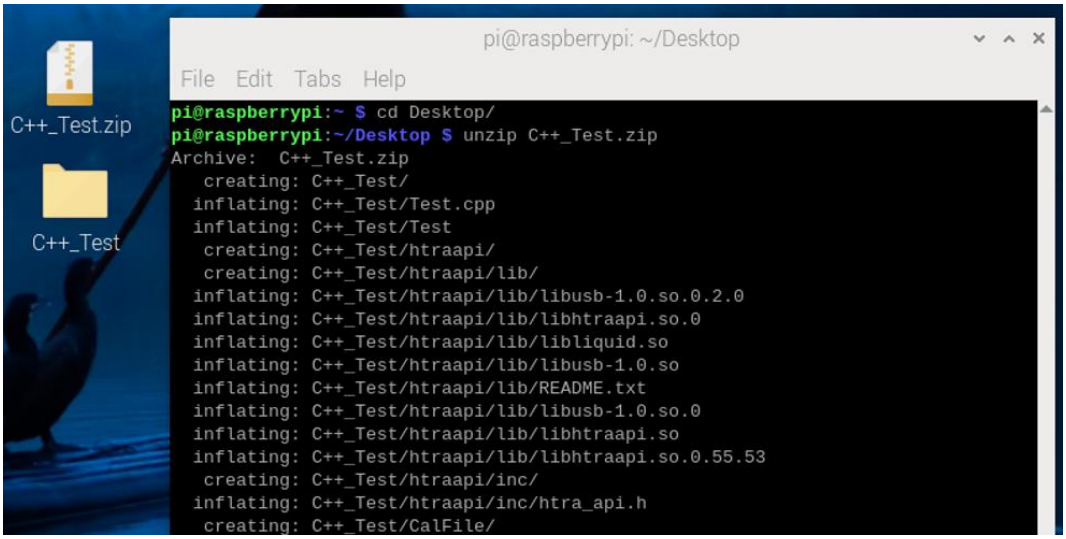
example, input 'zip -r C++_Test.zip C++_Test' to create the zip archive.



(2) Copy the zip file to the aarch64 host machine.



(3) Navigate to the location where the zip file is stored (in this example, the zip file is on the desktop, so input 'cd Desktop/'), and extract the project (input 'unzip C++_Test').



(4) Configure the driver files on the aarch64 host machine as per the steps in section 9.3.Chapter9.3

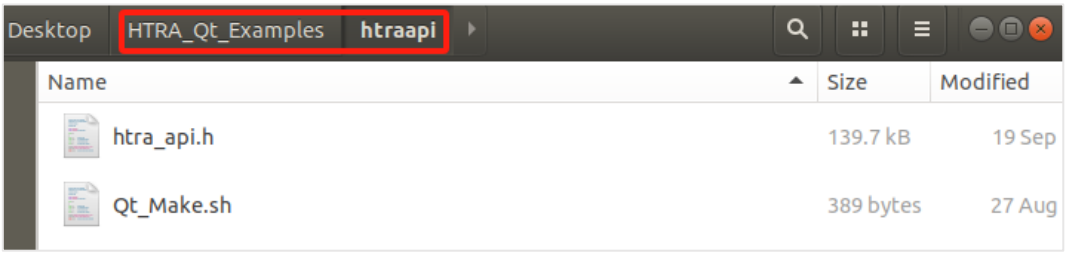
(5) After configuring the driver files, input 'cd C++_Test/' to enter the folder, and then simply input './Test' to run the program.

9.5 Using Qt Examples and Project Creation

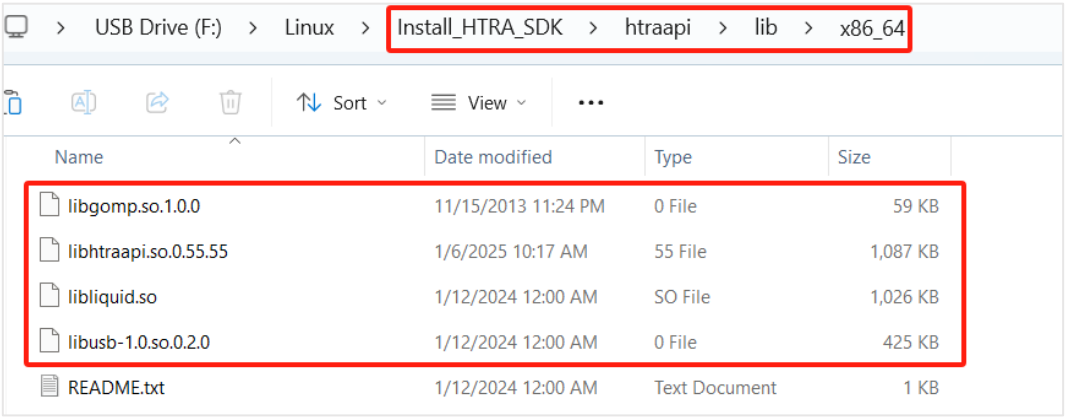
9.5.1 Using Qt Examples

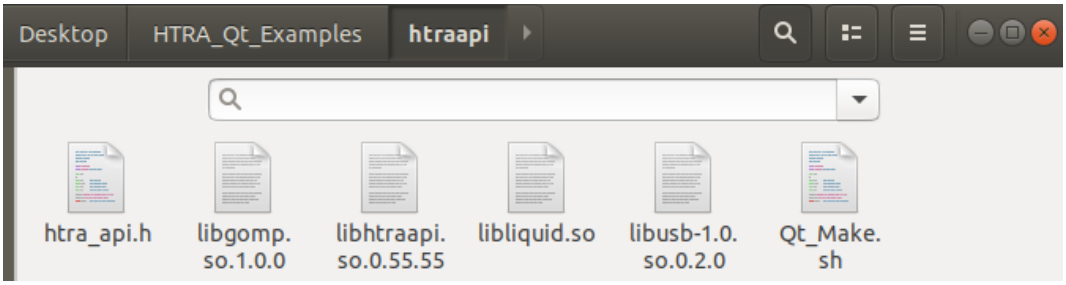
To use the Qt examples included on the USB drive, provided that the device is properly connected and the driver files have been configured correctly as per Section 9.3, please refer to the following process (the purpose of this Qt example is to obtain complete spectrum data within a specified frequency band):Chapter9.3

1. Copy the Linux\HTRA_Qt_Examples folder from the included USB drive to the host computer, and then navigate to the HTRA_Qt_Examples\htraapi subfolder as shown in the figure.

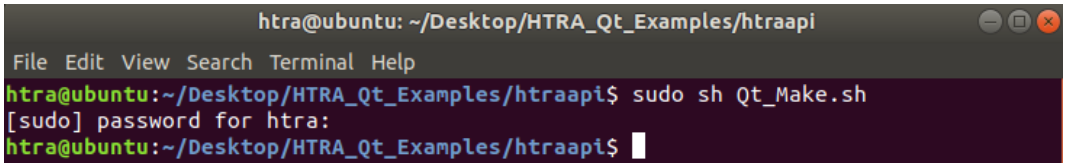


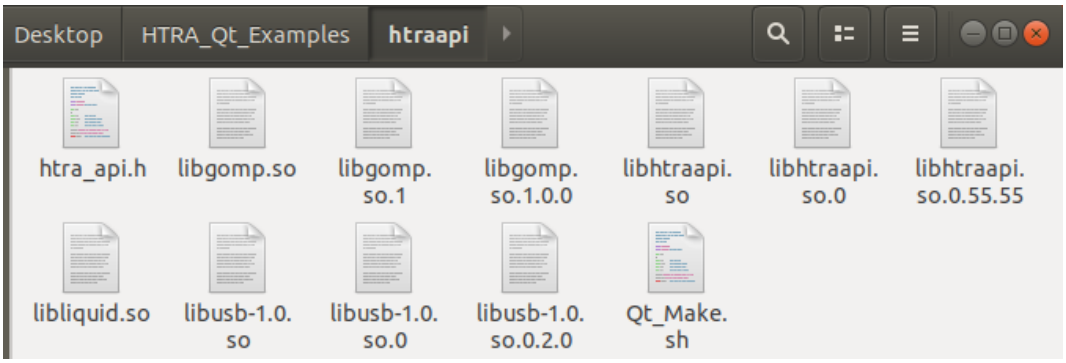
2. Refer to the system architecture in the process outlined in Section 9.1, and then follow the instructions in Section 9.2.4 to copy the dynamic link libraries corresponding to the system architecture from the Linux\Install_HTRA_SDK\htraapi\lib folder on the USB drive to the HTRA_Qt_Examples\htraapi folder (this example uses an x86_64 architecture host computer).Chapter9.1Chapter9.2.4





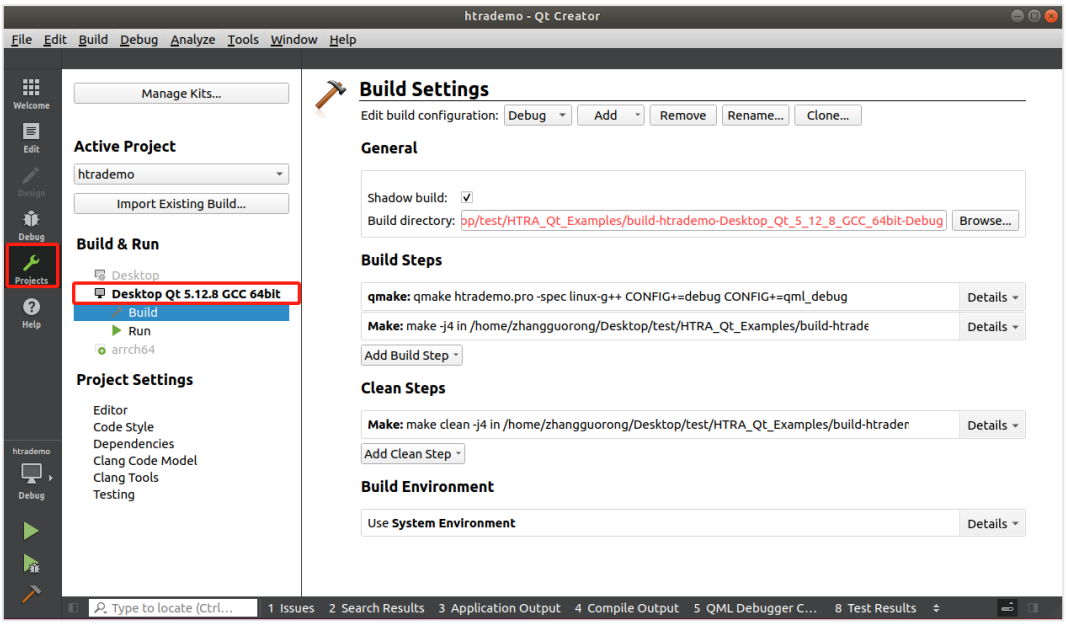
3. Open a terminal in the current folder, enter “sudo sh Qt_Make.sh”, and then follow the prompts to enter the sudo password to grant permission for creating soft links to the libraries.



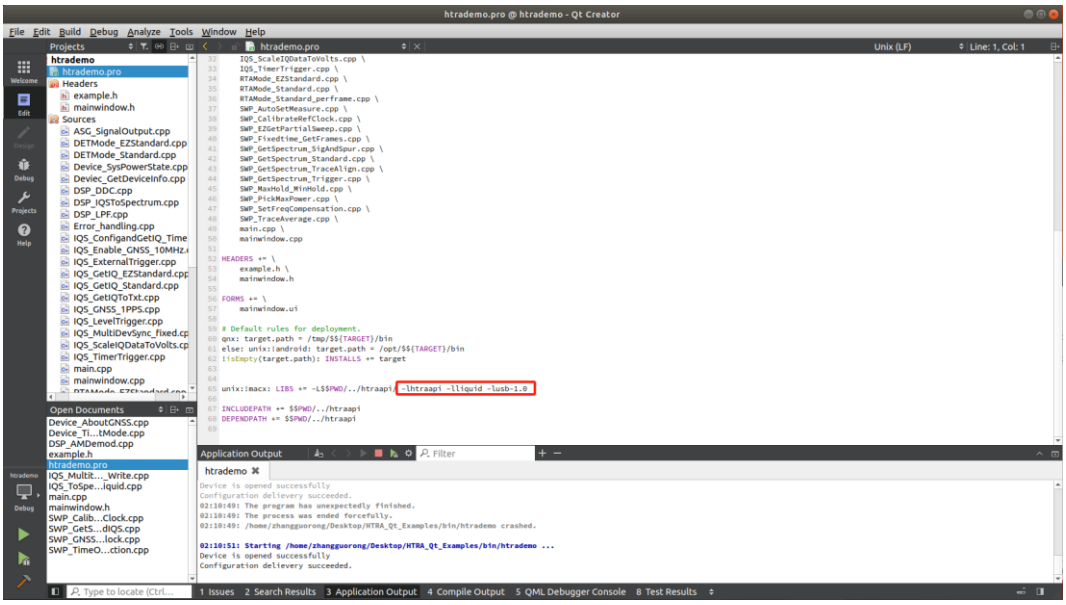


4. After running the script, use Qt Creator to open the htrademo.pro file in the htrademo folder.

5. First, select the build environment for the project.

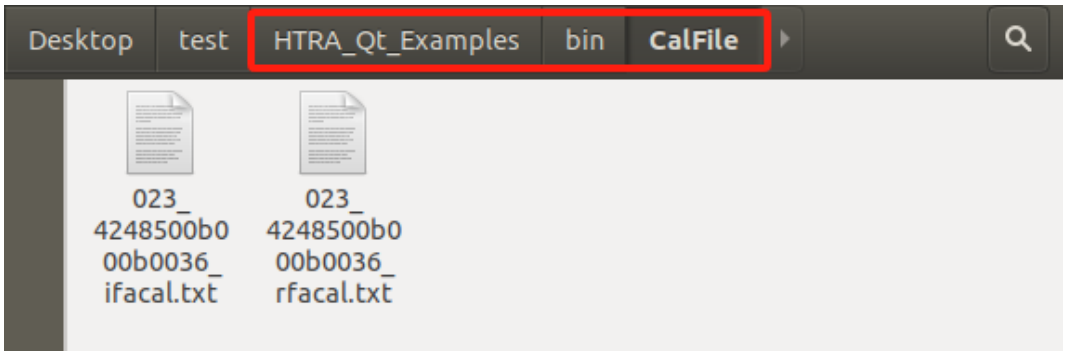


6. Once the build environment is configured successfully, click on Edit, then click on the htrademo.pro file. As shown in the figure below, fill in the corresponding library file names at the location indicated by the red arrow based on the library files selected in Step 2. (This example only uses the libraries libhtraapi.so, libliquid.so, and libusb-1.0.so, so it is sufficient to ensure -lhtraapi -lliquid -lusb-1.0 are included. If the libiomp5.so library is used, -liomp5 should be added later; if the libmkl.so library is used, -lmkl should be added; if libgomp.so is used, -lgomp should be added; if the libarmral.so library is used, -larmral should be added.)

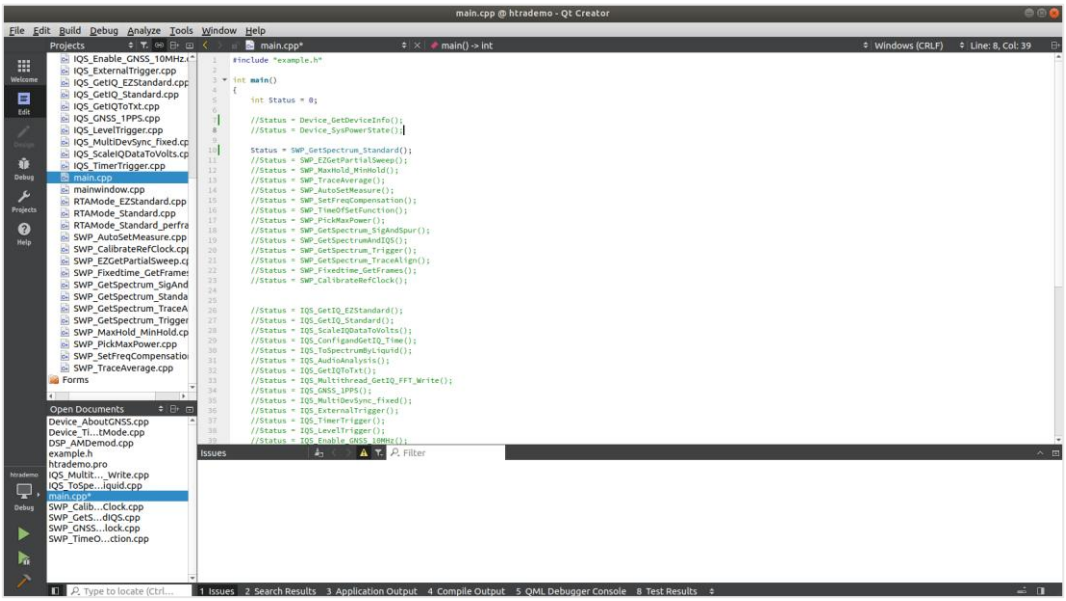


7. Open the HTRA_Qt_Examples\bin\CalFile folder and ensure that there is a

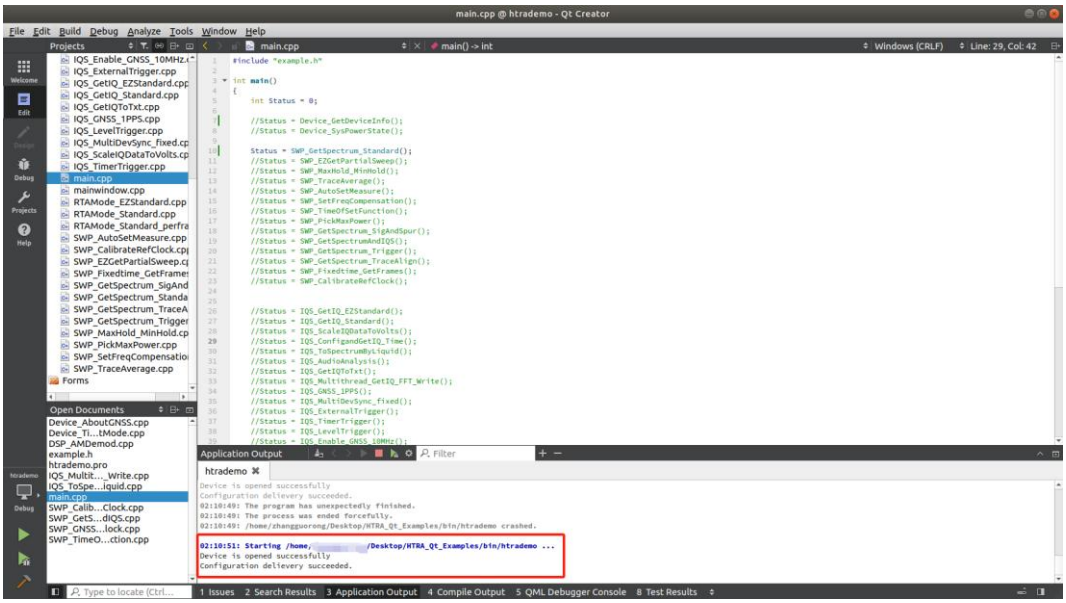
device calibration file in the folder.



8. After confirming that there is a calibration file, select any example in main.cpp.



9. Click Run, and as shown in the figure, the device is functioning normally.



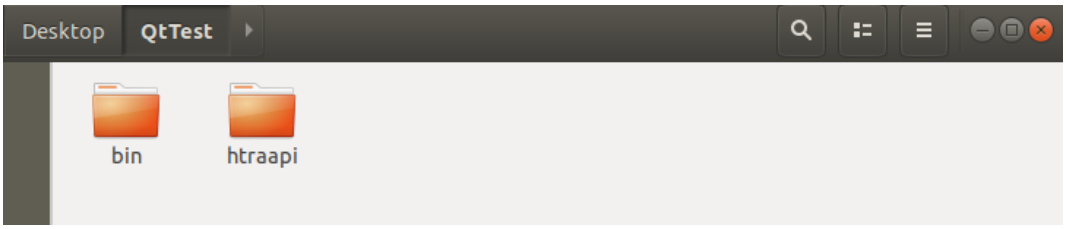
9.5.2 Qt Project Creation and Compilation

Under the premise that the driver files have been correctly configured according to section 9.3, if you want to create a Qt project for compilation, please refer to the following process:Chapter9.3

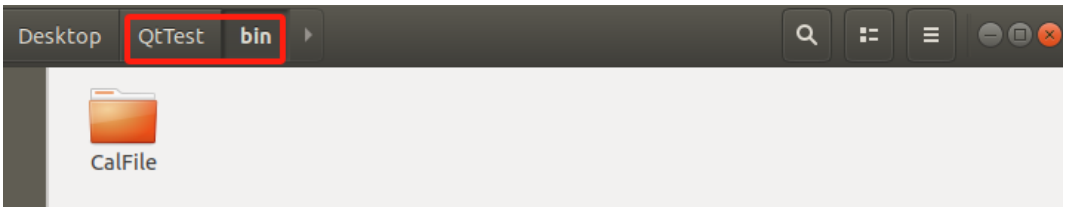
First, when writing code, since the Linux dynamic link libraries provided with the USB drive are identical to those for Windows, the code only needs to comply with the API programming guidelines.

Next, the process for creating the project is as follows:

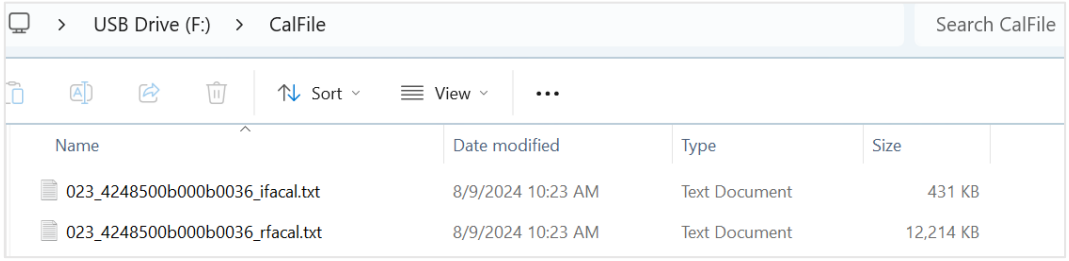
1. As shown in the figure, first create a new folder to store the entire project (taking QtTest as an example), then create a bin folder within this folder to store calibration files and the generated executable files, and create an htraapi folder to store header files and dynamic link libraries.

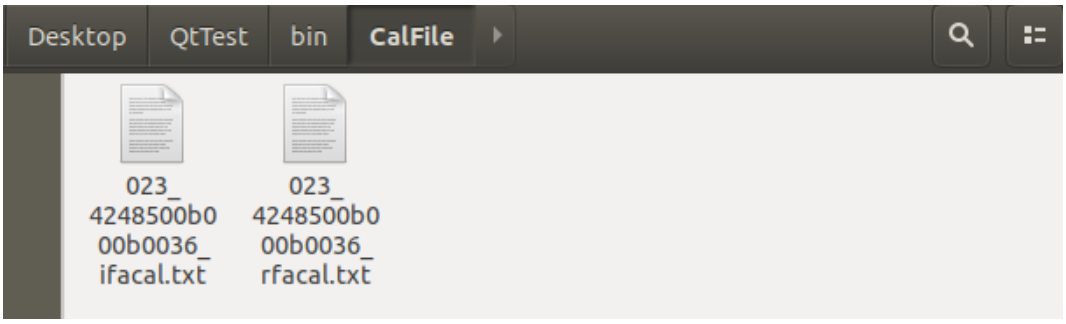


2. Create a CalFile folder under the bin folder to store the device calibration files.

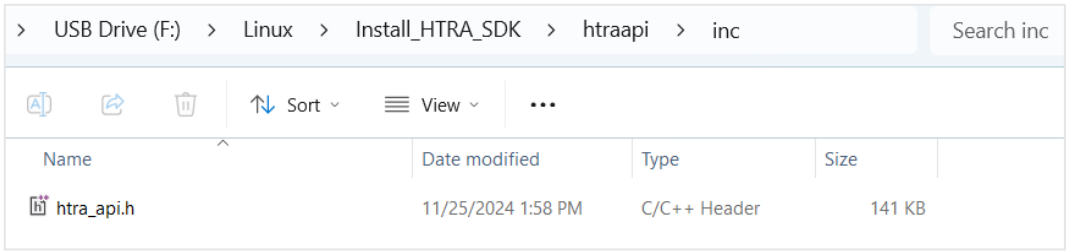


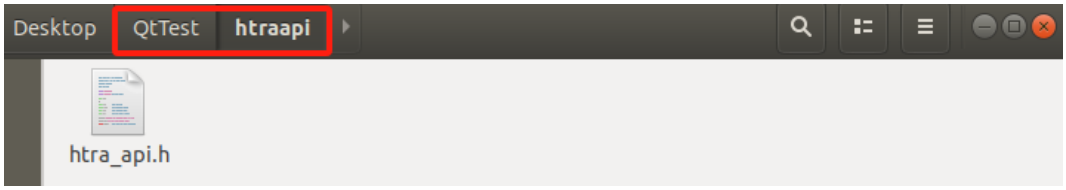
3. Copy the files from the CalFile folder on the provided USB drive to the newly created QtTest\bin\CalFile folder.



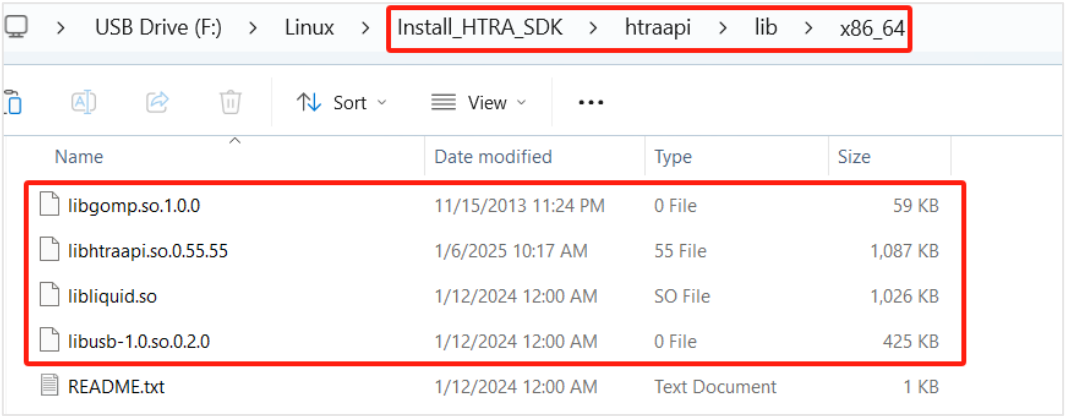


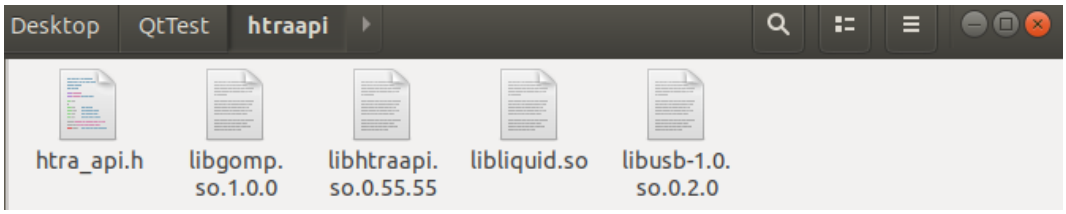
4. Copy the header files from the Linux\Install_HTRA_SDK\htraapi\inc folder on the provided USB drive to the newly created QtTest\htraapi folder.





5. Refer to the system architecture in section 9.1, and then according to section 9.2.4, copy the dynamic link libraries corresponding to the system architecture from the Linux\Install_HTRA_SDK\htraapi\lib folder on the provided USB drive to the newly created QtTest\htraapi folder (taking the x86_64 architecture for the host computer as an example).Chapter9.1Chapter9.2.4





6. Open a terminal at the location of the htraapi folder and enter the following command to create a soft link for the copied dynamic link libraries (the soft link command for different architectures is the same):

ln -sf libhtraapi.so.0.55.5 libhtraapi.so.0 (Here, the libhtraapi.so library is taken as an example with version 0.55.55; modify the version number for other versions.)

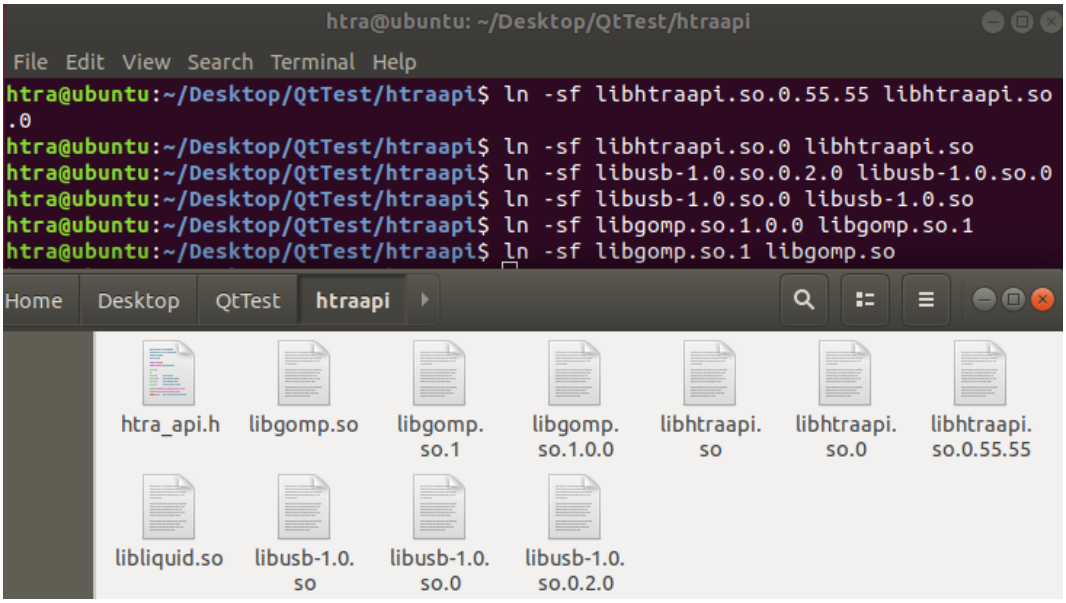
ln -sf libhtraapi.so.0 libhtraapi.so

ln -sf libusb-1.0.so.0.2.0 libusb-1.0.so.0

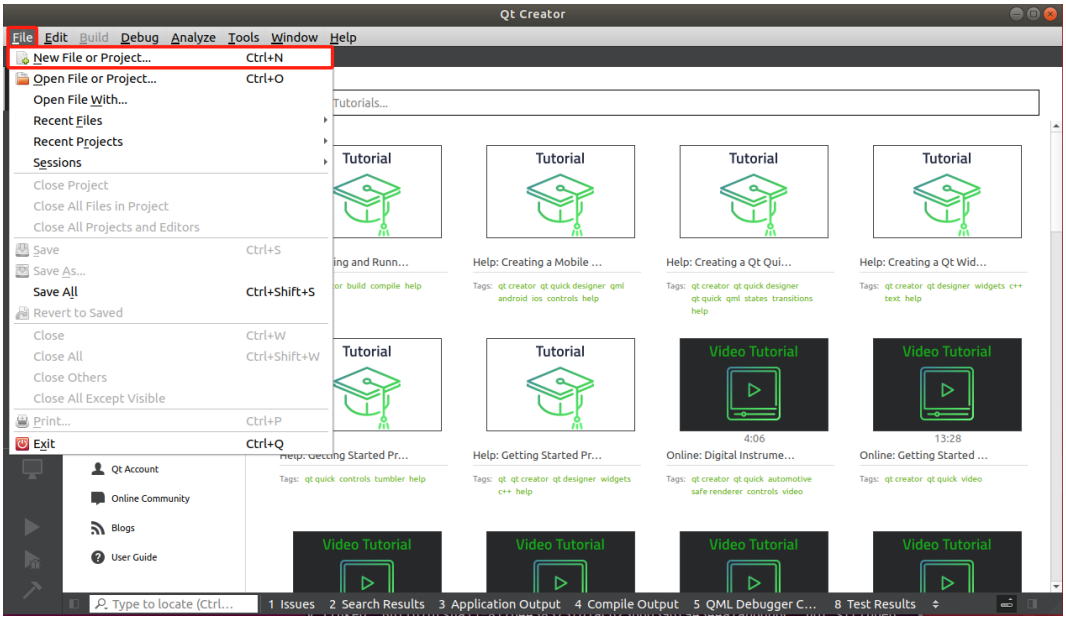
ln -sf libusb-1.0.so.0 libusb-1.0.so

ln -sf libgomp.so.1.0.0 libgomp.so.1

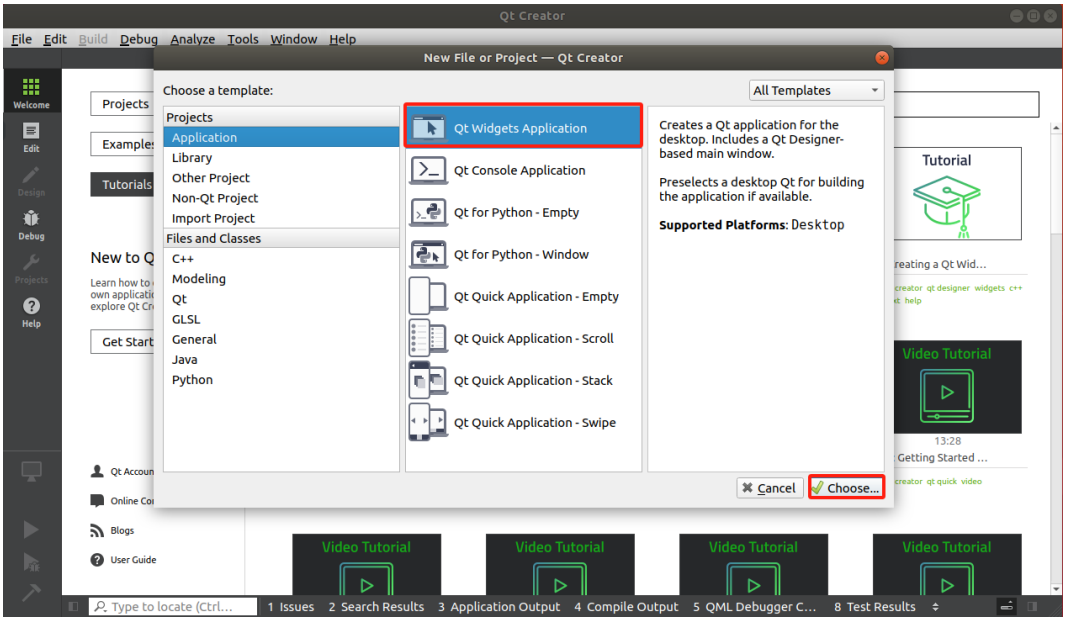
ln -sf libgomp.so.1 libgomp.so



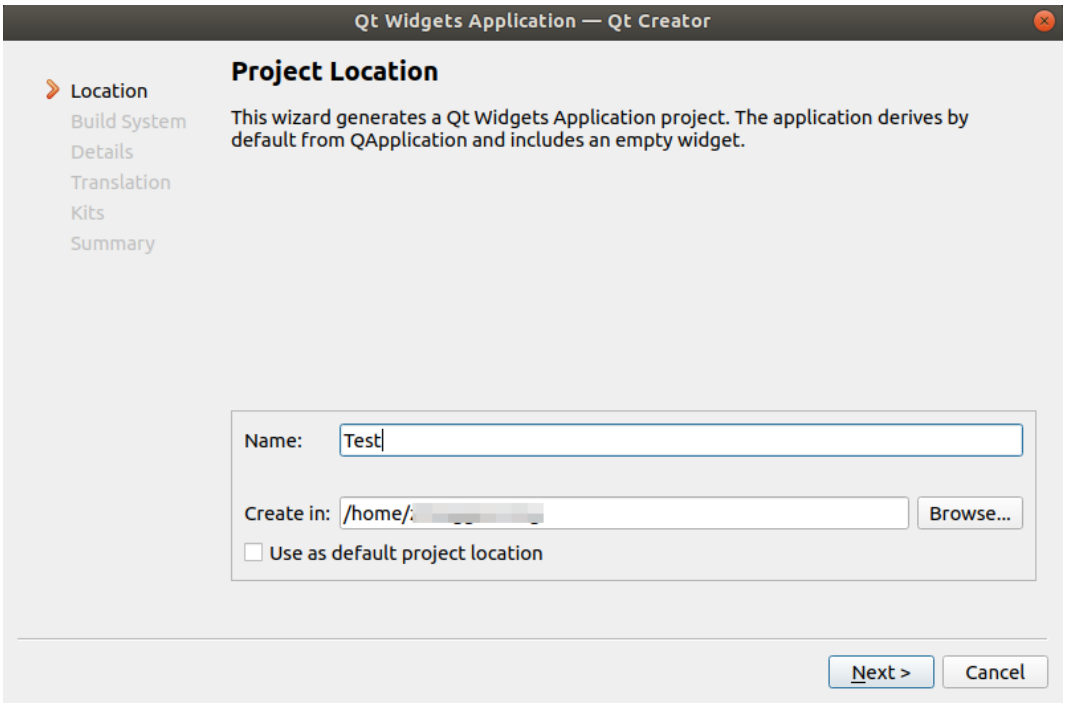
7. Open Qt Creator, click on File, and select New File or Project.



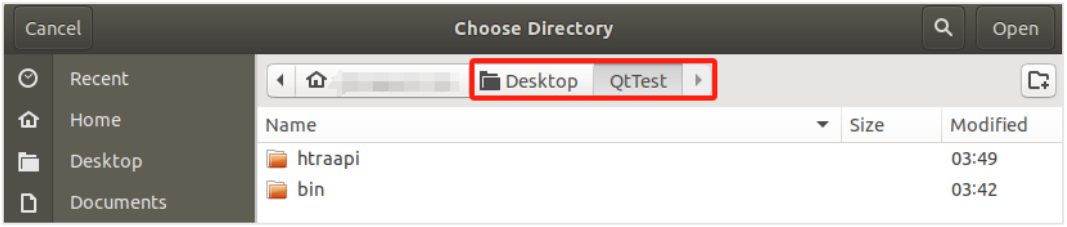
8. Select Create Form Application.



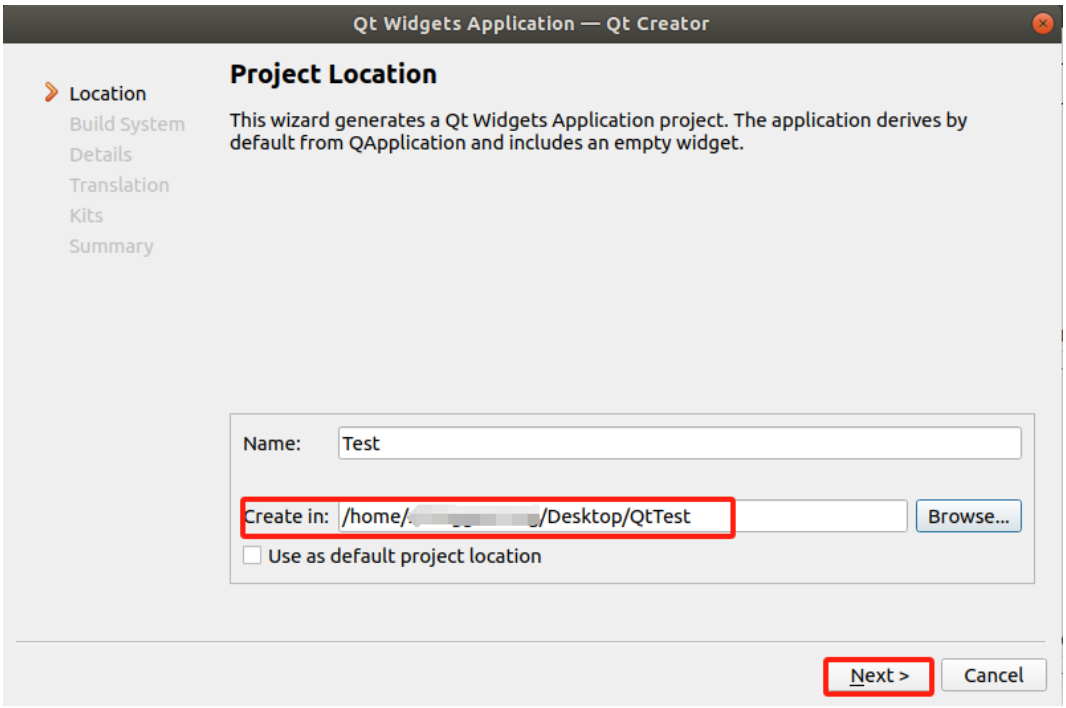
9. After filling in the project name, click Browse to change the project path.



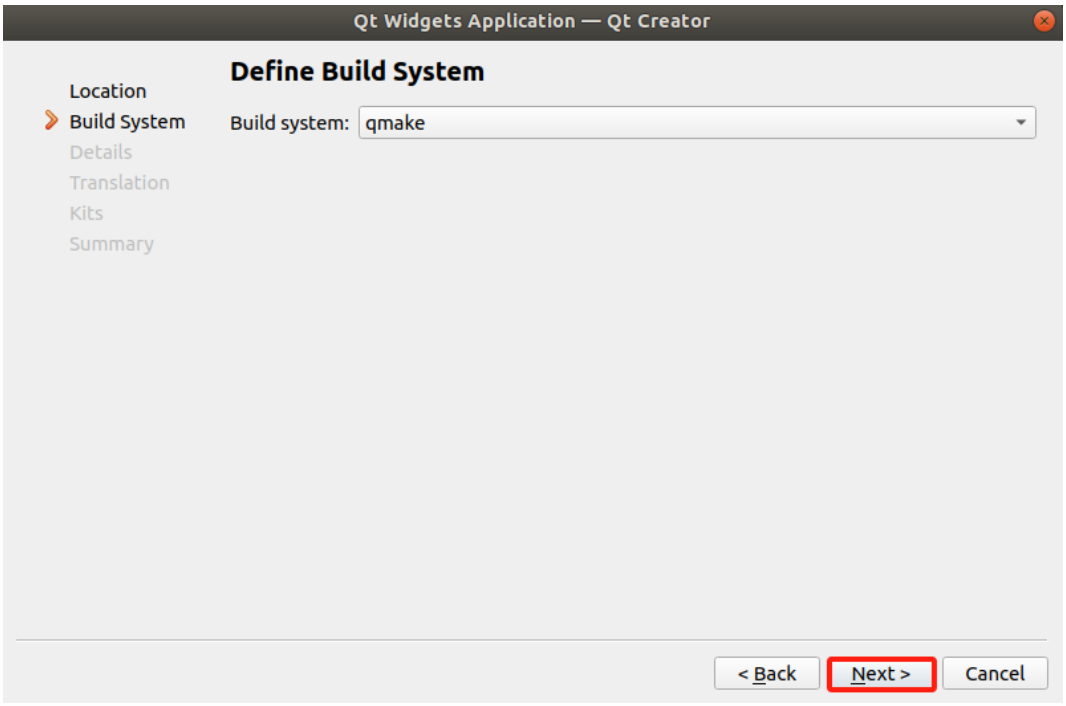
10. Select the directory as the QtTest address created in the first step and click Open.



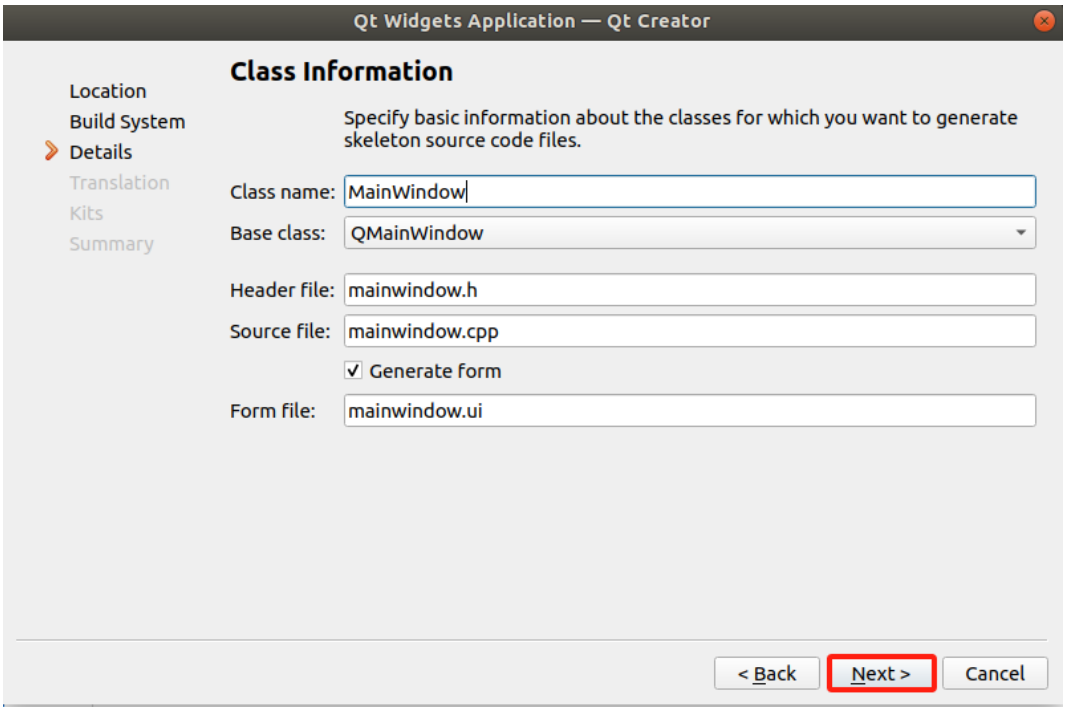
11. After selecting the path, click Next.



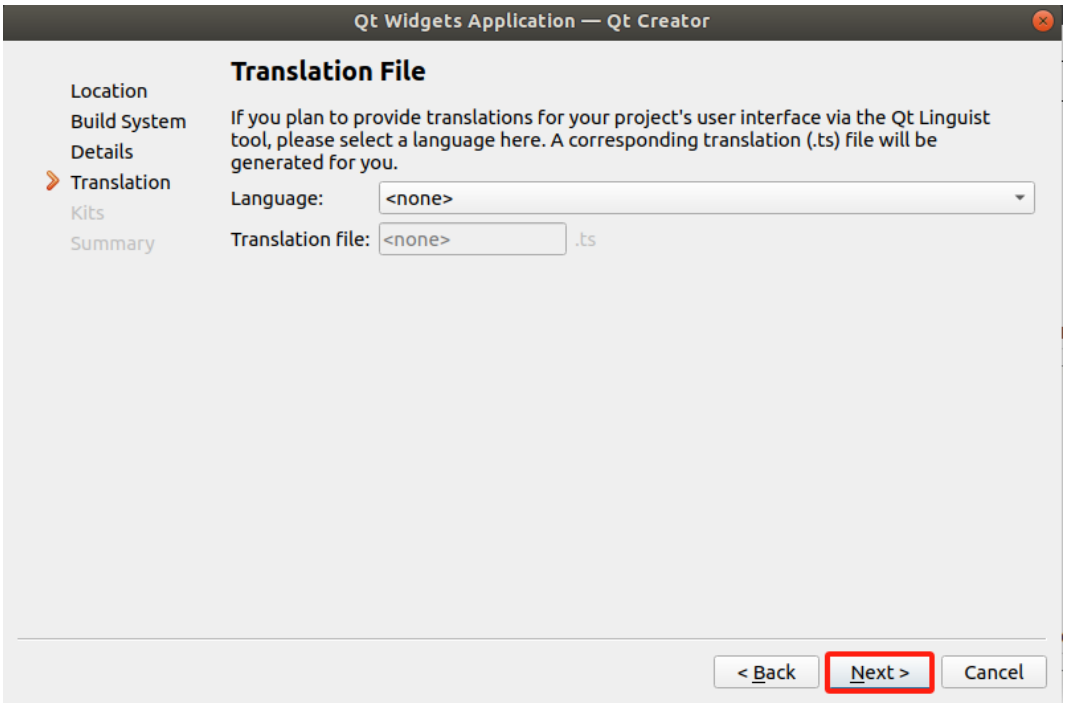
12. Click Next to continue.



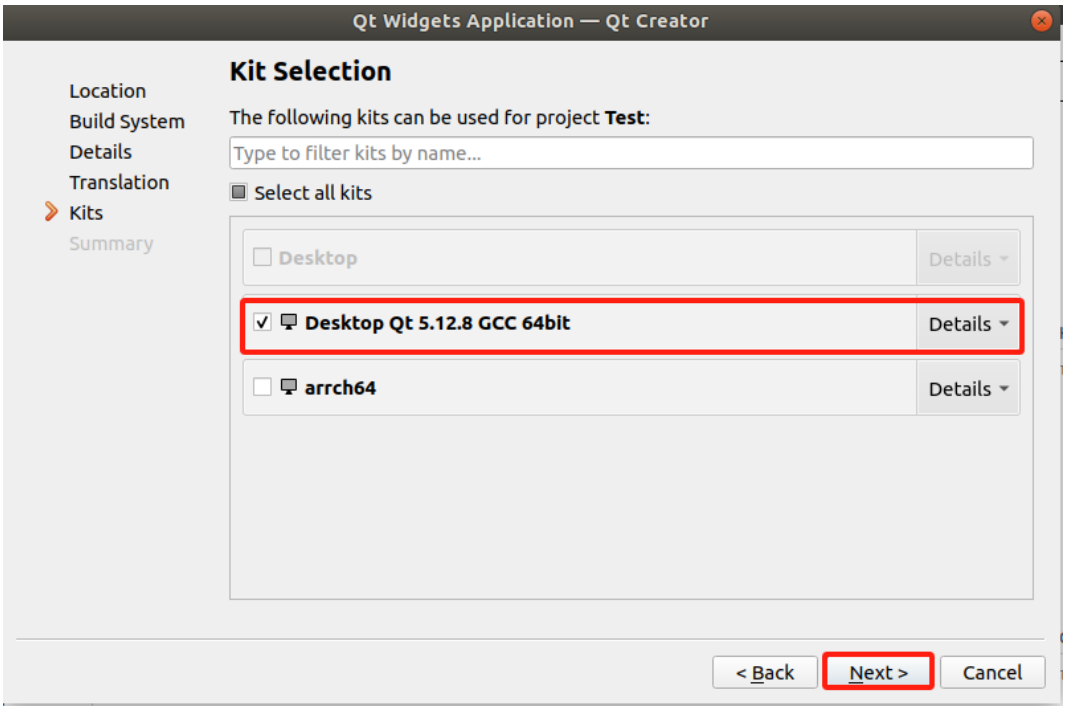
13. Click Next to continue.



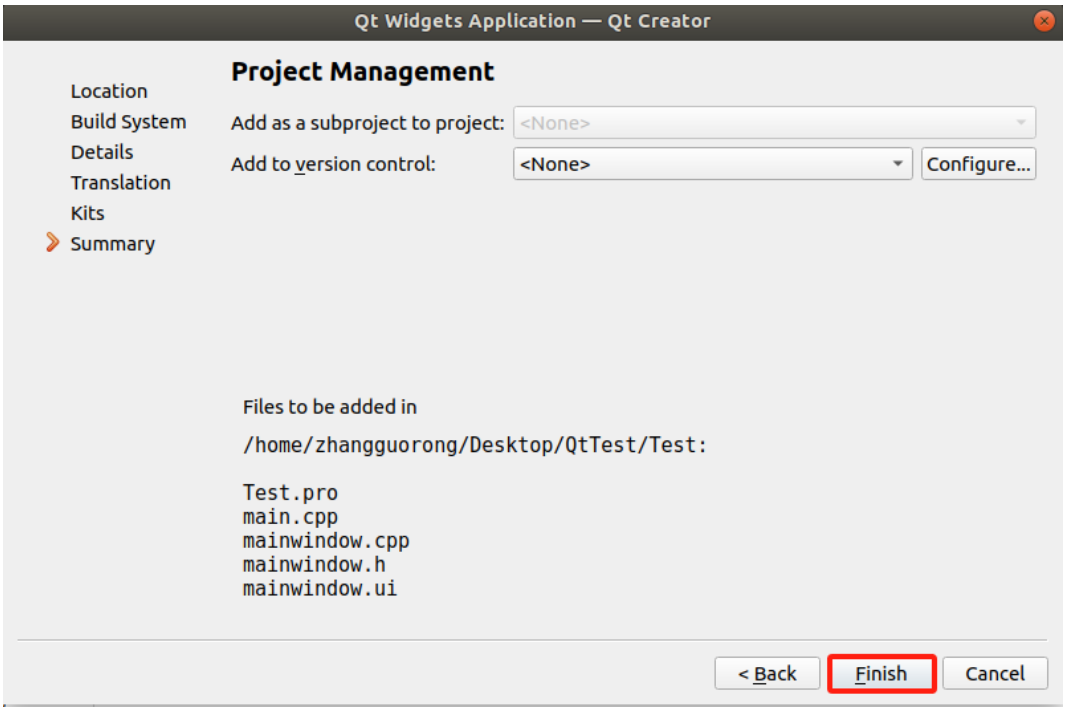
14. Click Next to continue.



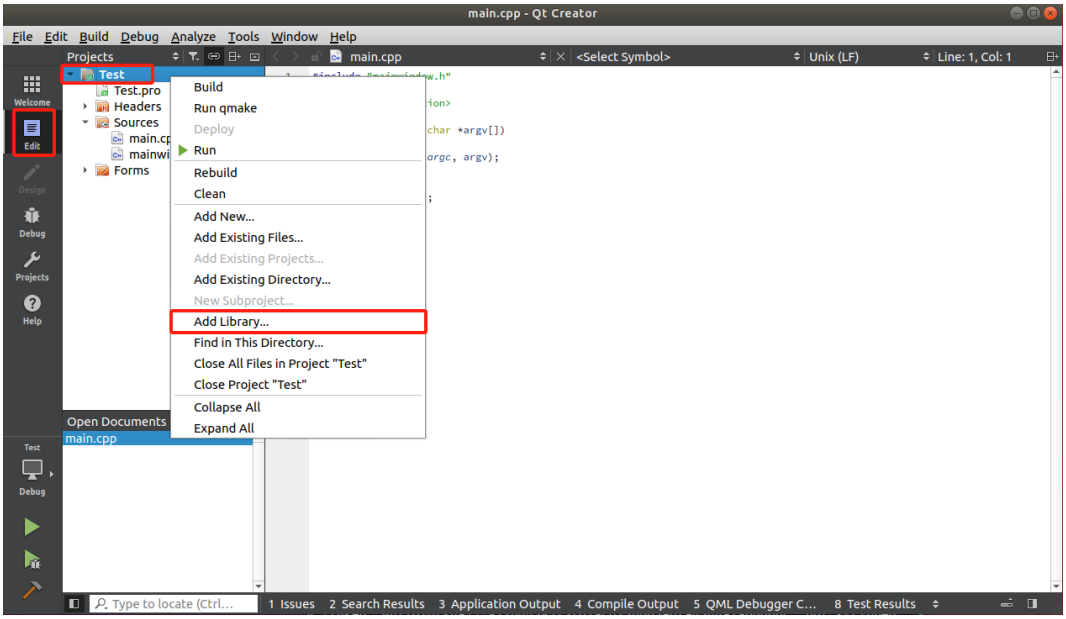
15. Select the x86_64 build environment for the project and then continue by clicking next.



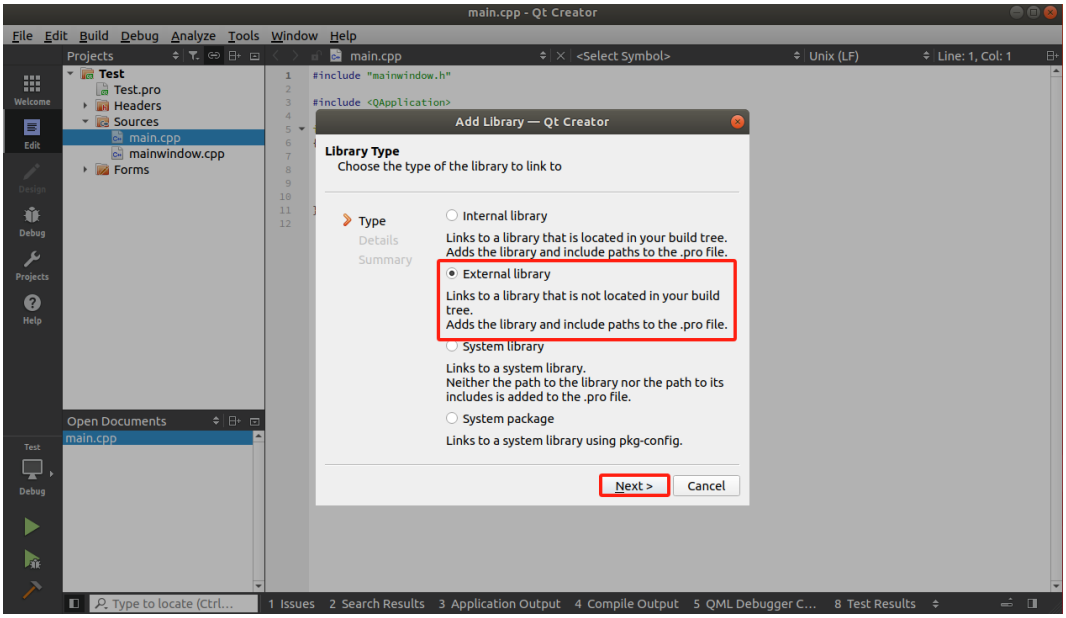
16. Click Finish to create the project.



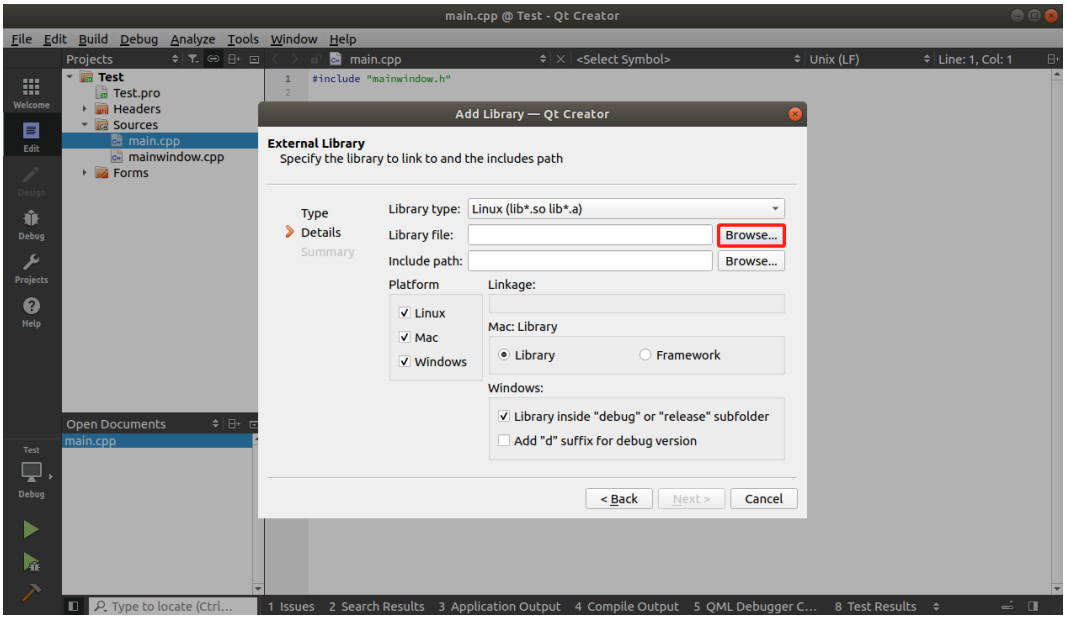
17. Click Edit, right-click the Test project, and click Add Library.



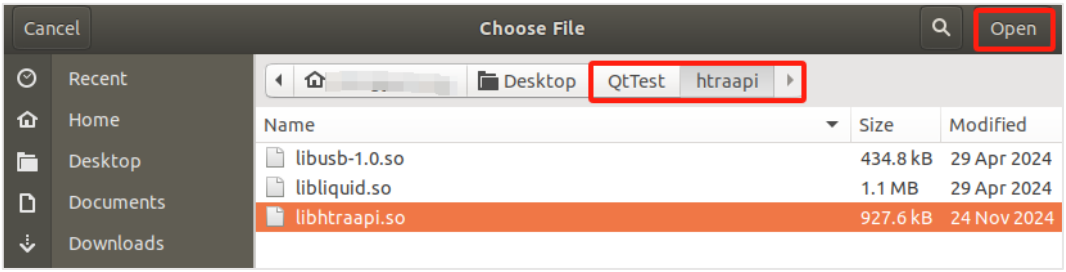
18. Select External Library and click Next.



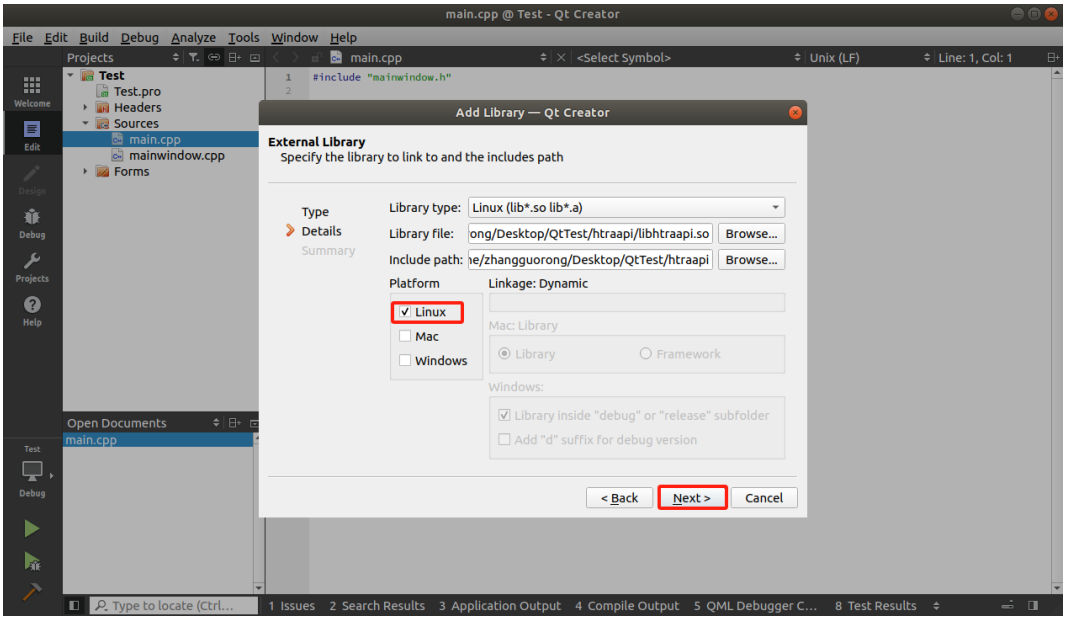
19. Click Browse Library File.



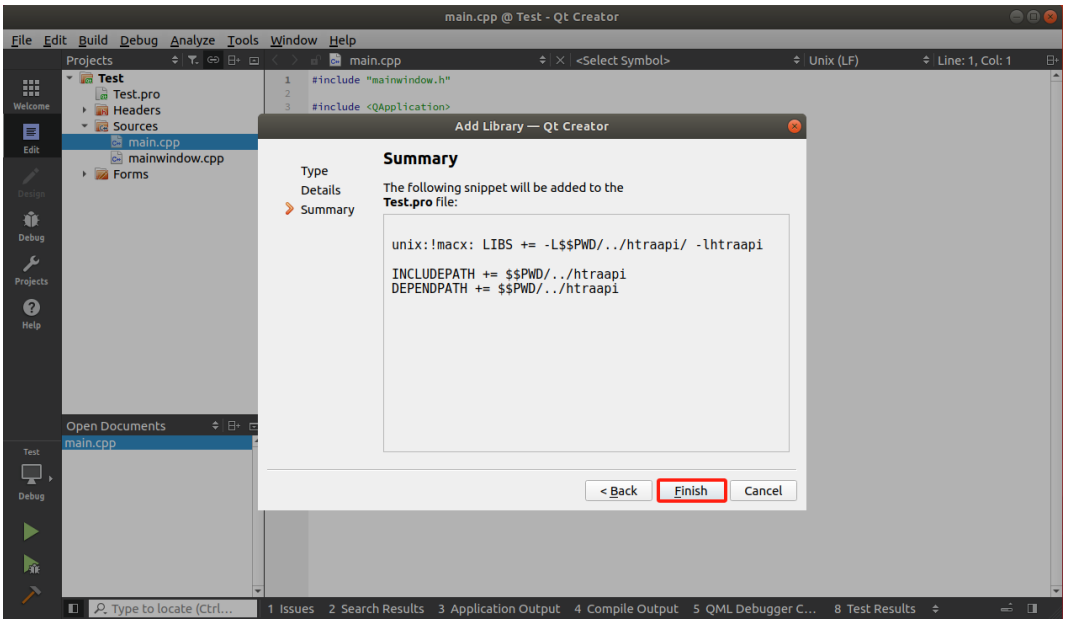
20. Select Q Test\htraapi, the library previously copied and soft-linked, and click Open.



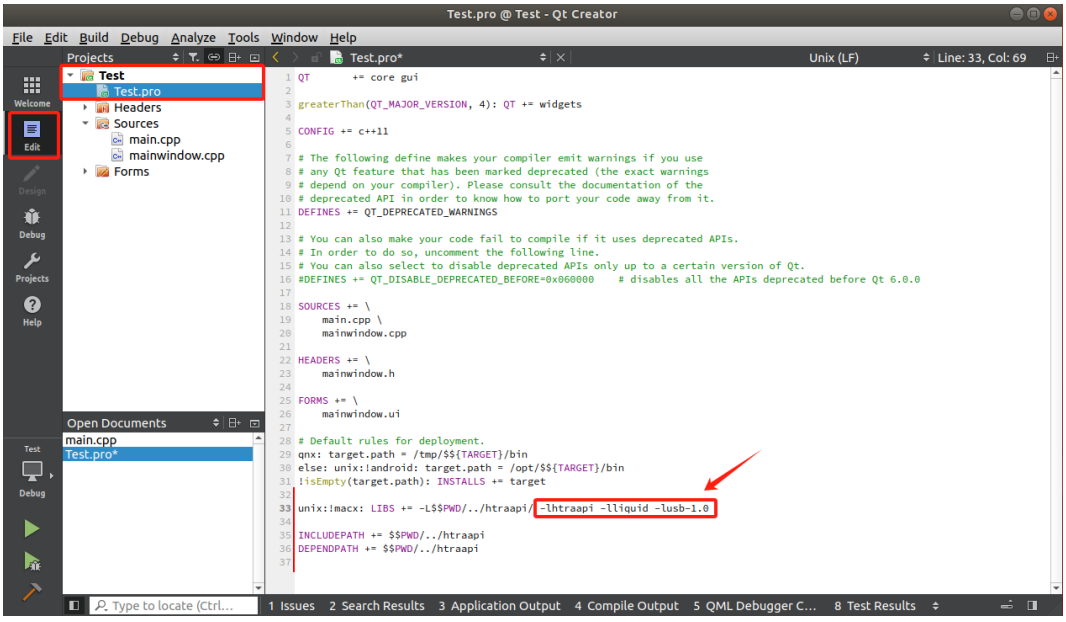
21. Select the Linux platform and click Next.



22. Click Finish to add the external library.



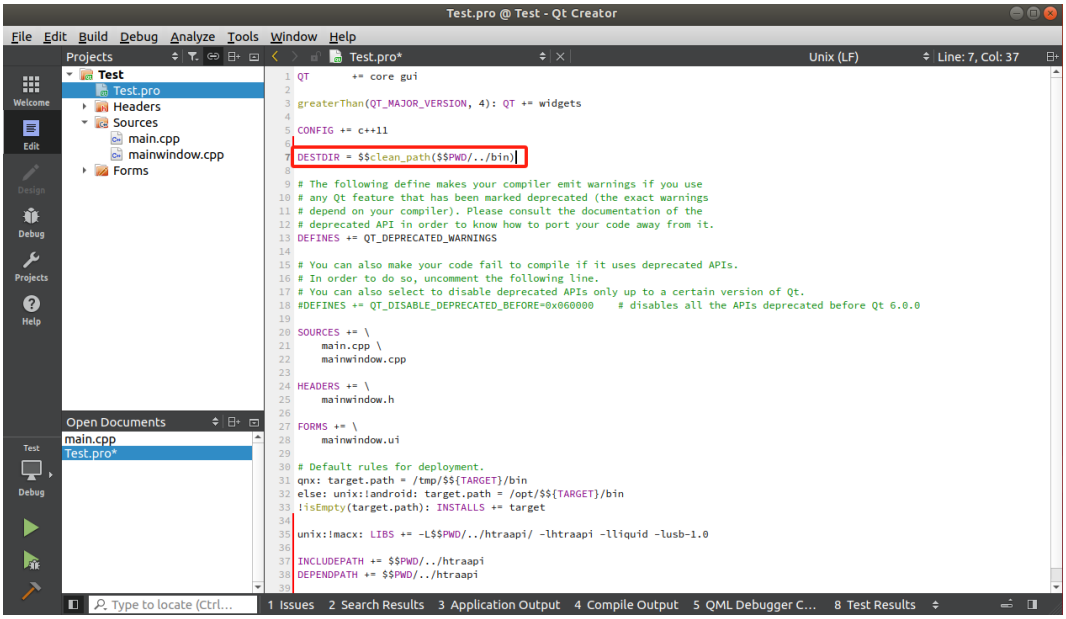
23. Click Edit, then click the Test.pro file. As shown in the figure below, fill in the corresponding library file names at the position indicated by the red arrow based on the library files selected in the previous step (5). (This example only uses libhtraapi.so, libliquid.so, and libusb-1.0.so, so it is sufficient to ensure -lhtraapi -lliquid -lusb-1.0 are included. If the libiomp5.so library is used, -liomp5 should be added later; if libmkl.so is used, -lmkl should be added; if libgomp.so is used, -lgomp should be added; if libarmral.so is used, -larmral should be added.)



24. As shown in the figure below, add

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

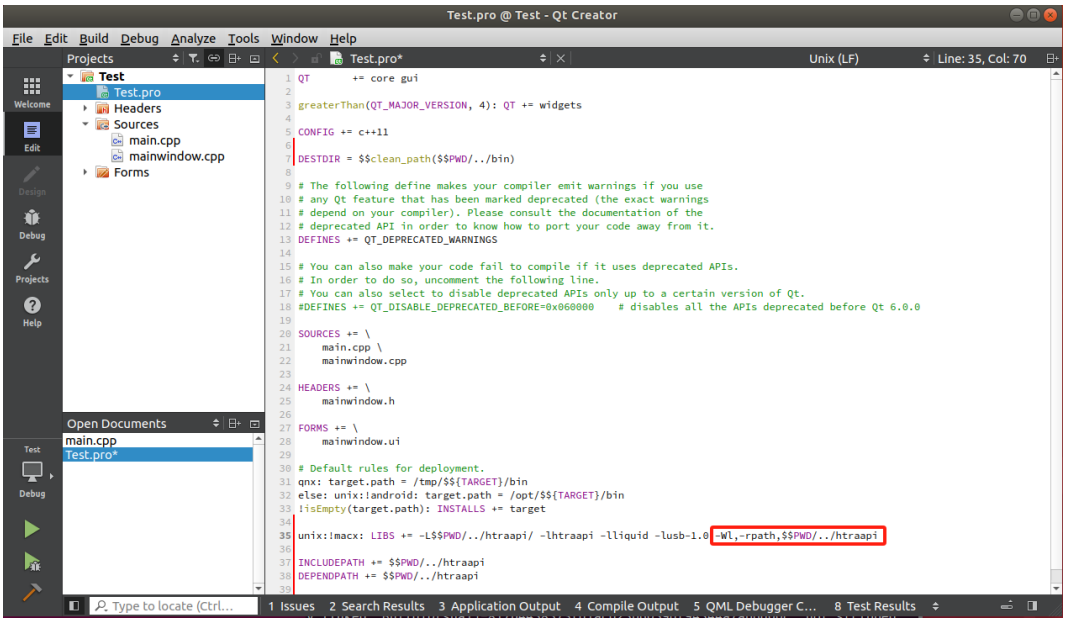
(This step specifies the location for generating the executable program.)



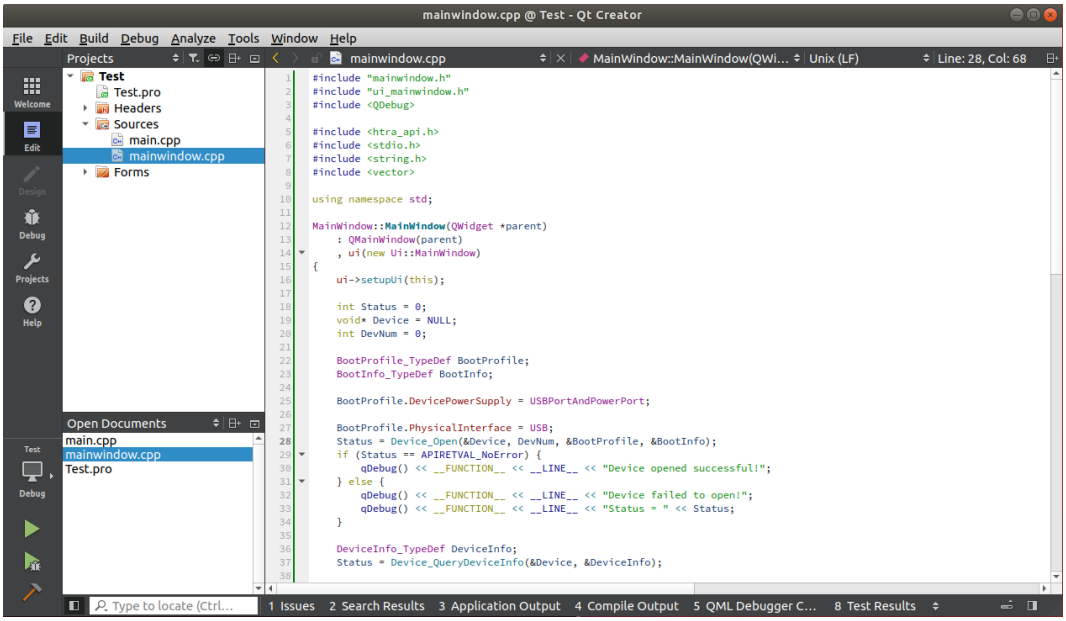
25. As shown in the figure below, after the library files, add

-Wl,-rpath,$$PWD/../htraapi

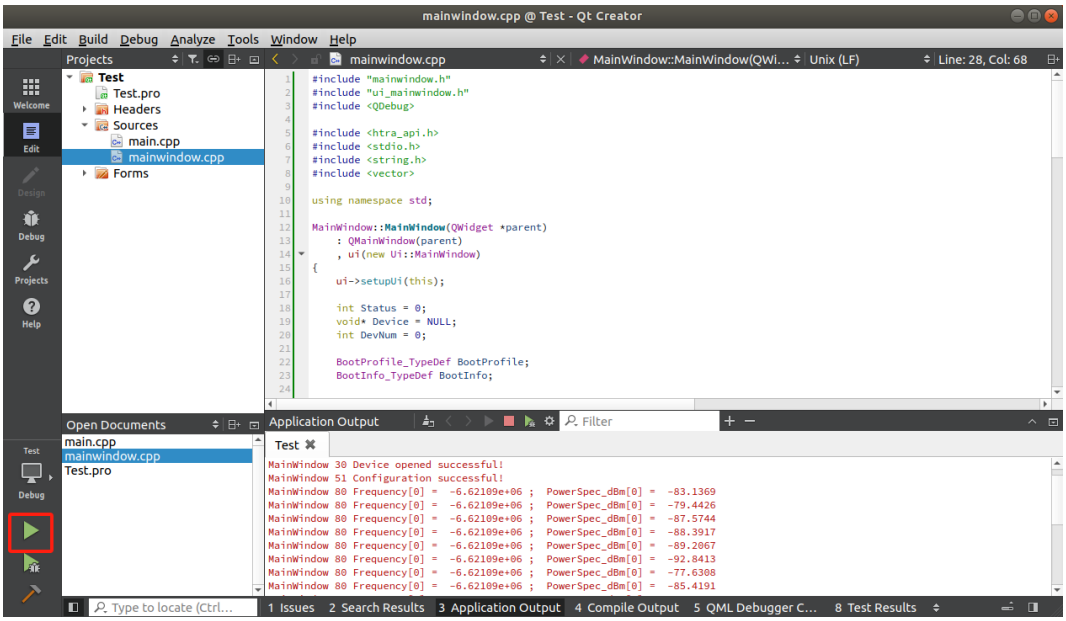
(This step specifies the library path for linking the executable program.)



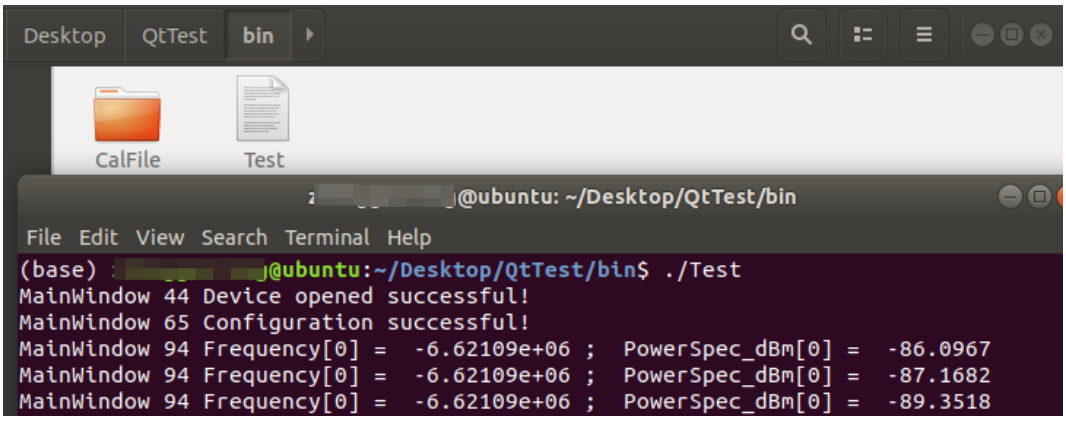
26. Save the Test.pro file, and then write the code in mainwindow.cpp.



27. After writing the code correctly, click run. As shown in the figure, you can see that the program is running normally in the application output.



28. Close Qt Creator, enter the QtTest\bin folder, open the terminal, and type ./Test to run the executable program.



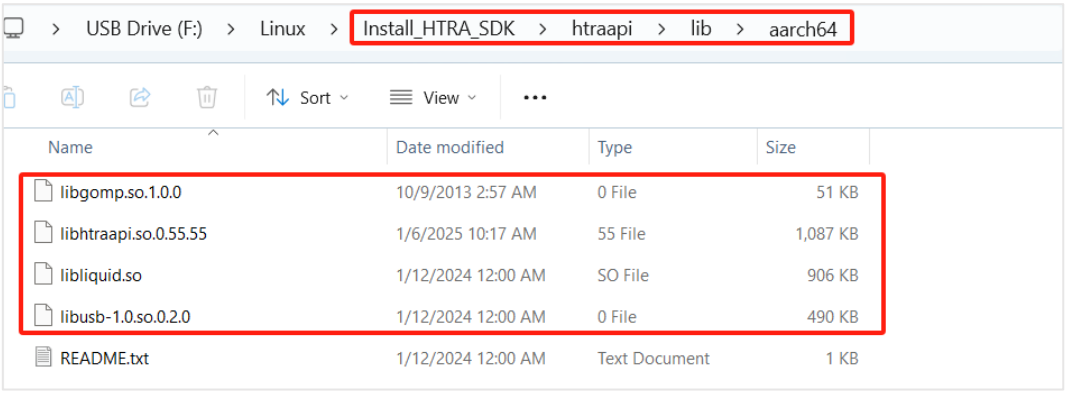
9.5.3 Cross-compiling Qt projects

Under the premise that the host computer has a cross-compilation toolchain, if you want to cross-compile to use the device, please refer to the following process (taking the cross-compilation of an aarch64 executable program on an x86_64 host computer as an example):

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

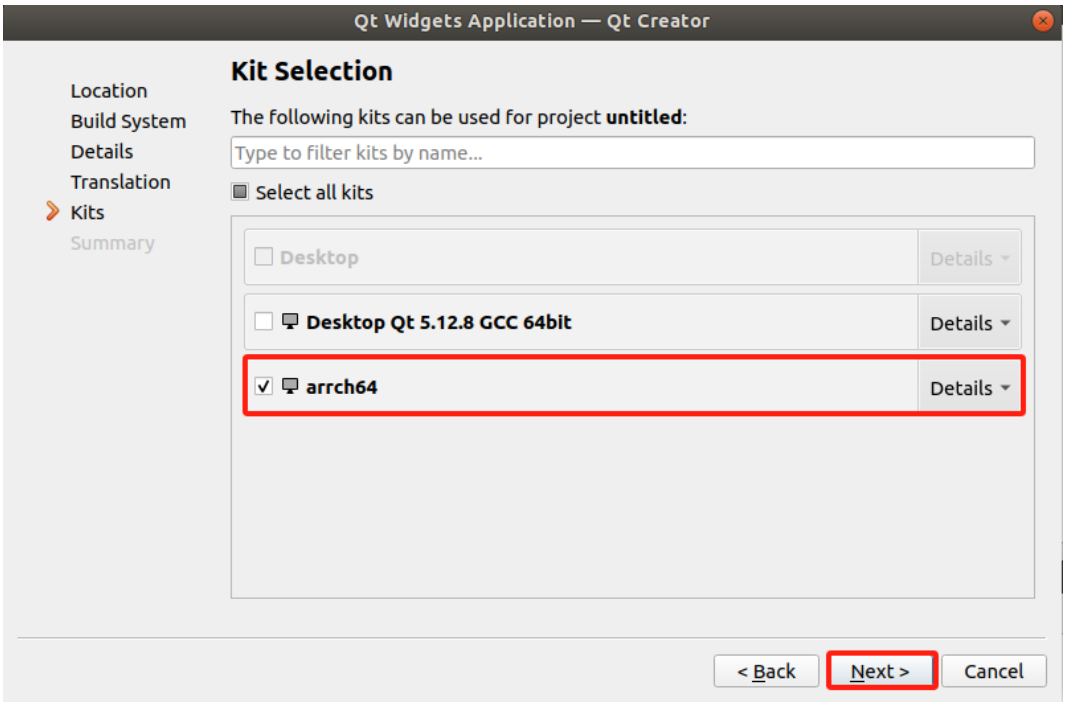
(1) Create the project and place the calibration files and header files according to steps 1-4 of the form program creation process.QtProject Creation and Compilation

(2) Place the cross-compilation target architecture library files according to step 5 of the form program creation process. For example, if you expect to cross-compile an executable for the arrch64 architecture, please place the library files for the arrch64 architecture.



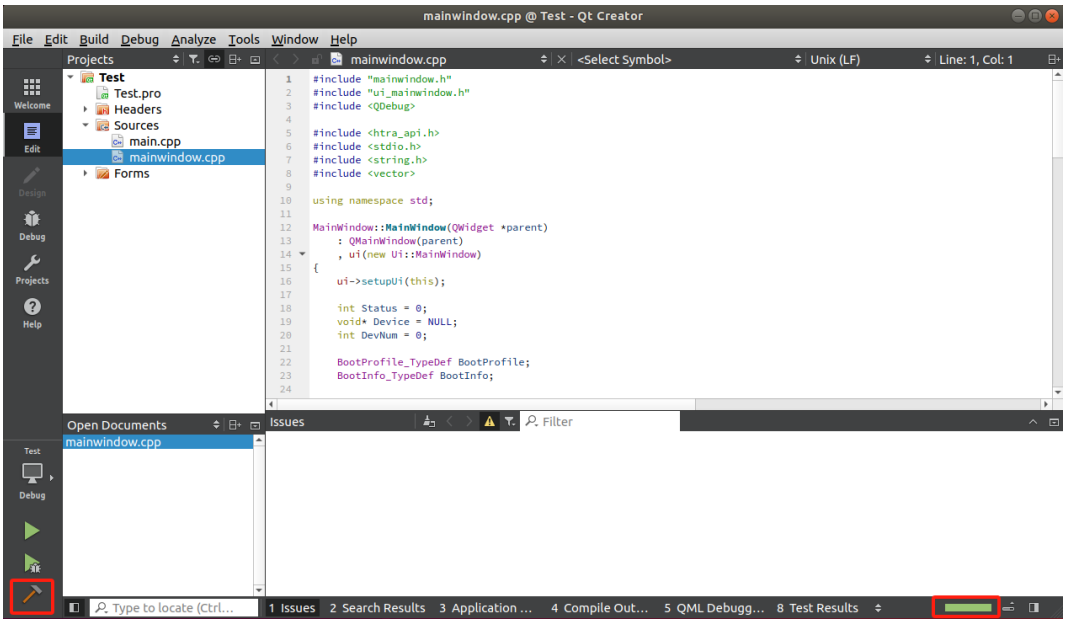
(3) Create a soft link for the dynamic link library according to step 6 of the form program creation process.

(4) Create the program according to steps 7-13 of the form program creation process, and in step 14, select the build kit for the cross-compilation target architecture (for example, in this example, select the build kit for the arrch64 architecture).

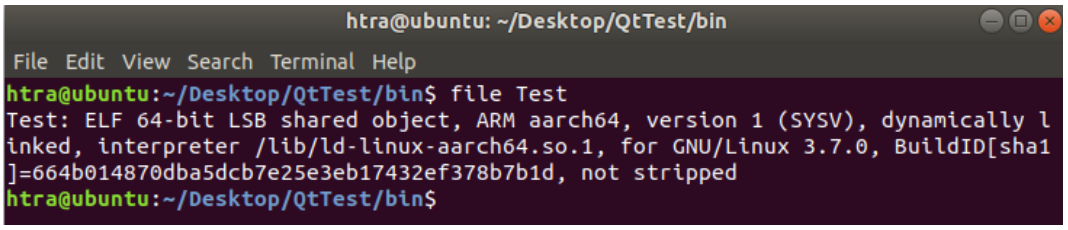


(5) Follow steps 16-26 of the form program creation process to create the project, reference the libraries, modify the location for generating the executable program, and write the project.

(6) Click the build button in the lower left corner to build the executable program.

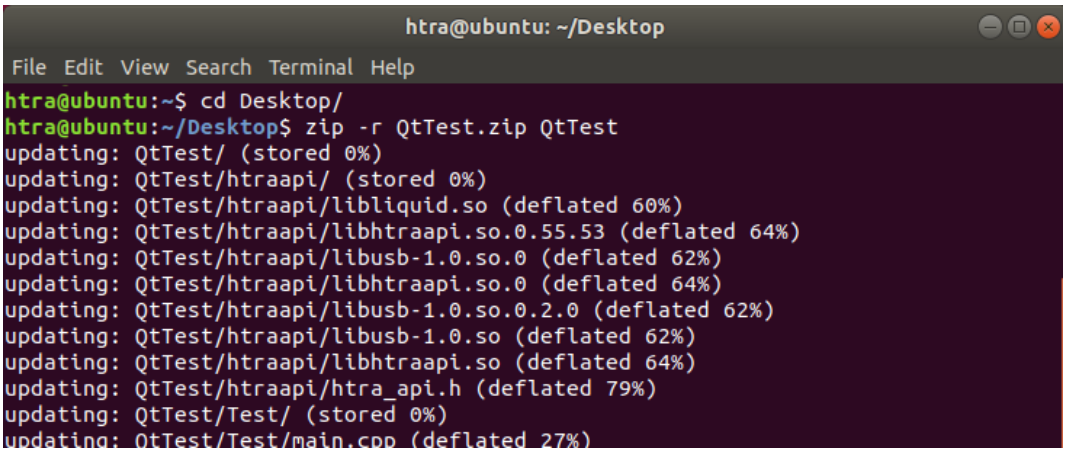


(7) After building the executable program, open the terminal in the QtTest\bin folder and type file Test to check the architecture of the executable program (the name of the executable program here is Test, so type file Test).

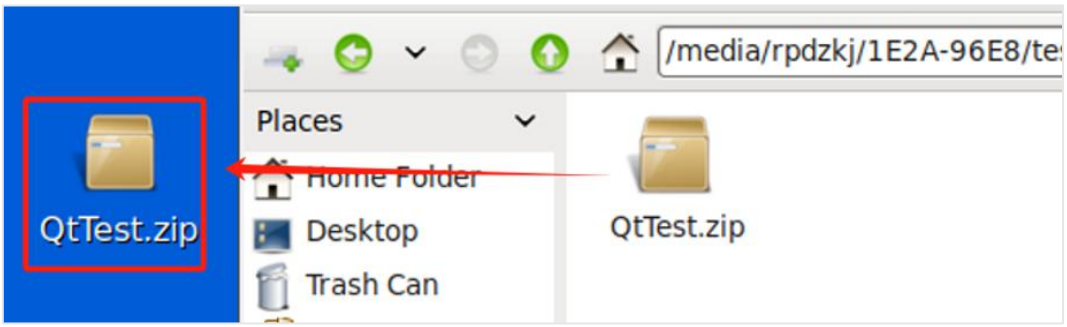


2. At this point, the executable program has been successfully generated, and you can now run the program on the aarch64 host machine using the device:

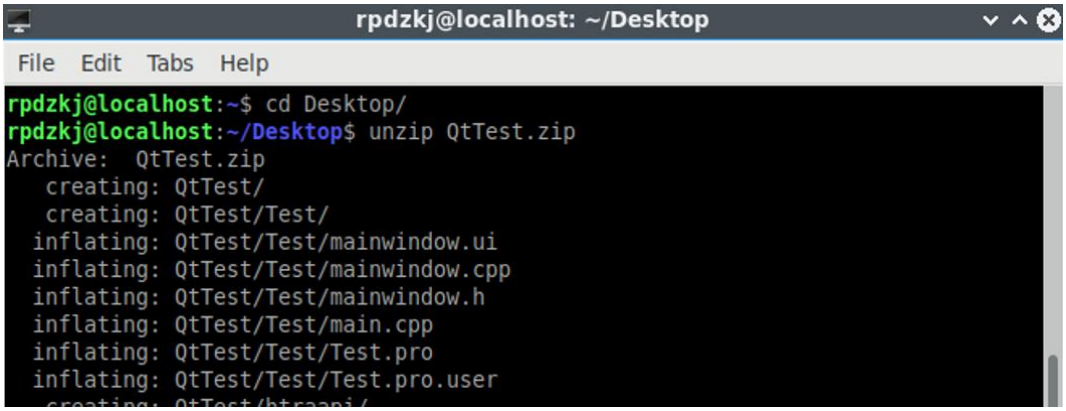
(1) Navigate to the project location (in this example, the project is located on the desktop, so type cd Desktop/), and compress the entire folder into a zip file, for example, type zip -r Qt Test.zip Qt Test to create a zip archive.



(2) Copy the zip file to the aarch64 host machine.

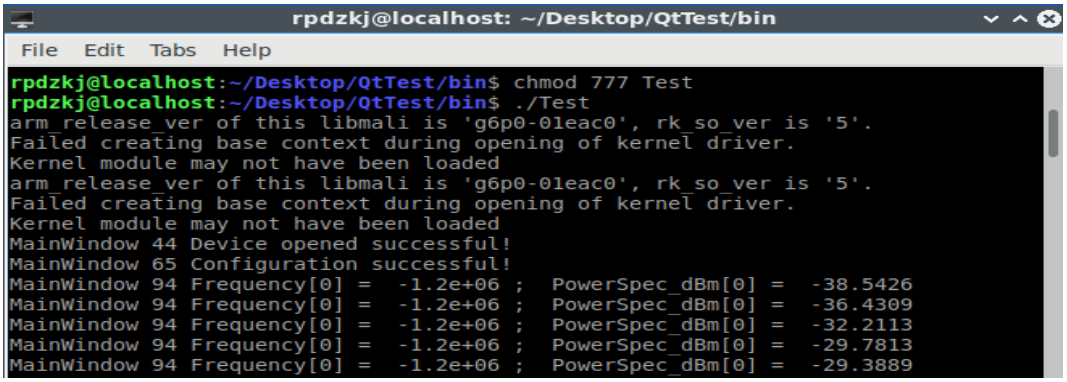


(3) Navigate to the location of the compressed package (in this example, the compressed package is located on the desktop, so input cd Desktop/), and unzip the project (input unzip QtTest here).



(4) Configure the driver files on the aarch64 host machine as per the steps in section 9.3.Chapter9.3

(5) After configuring the driver files, input cd QtTest/ to enter the folder, then input chmod 777 Test to provide execution permissions for the program, and finally input ./Test to run the program.

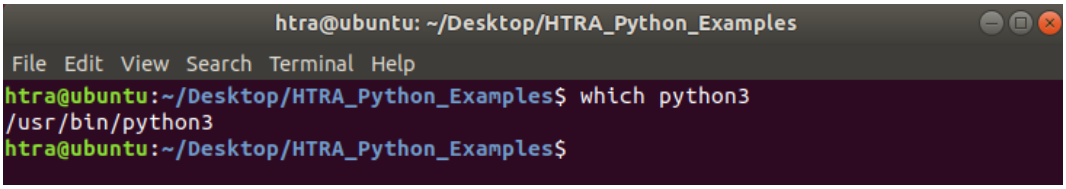

rpdzkj@localhost: ~/Desktop/QtTest/bin
File Edit Tabs Help
rpdzkj@localhost:~/Desktop/QtTest/bin$ chmod 777 Test
rpdzkj@localhost:~/Desktop/QtTest/bin$ ./Test
arm_release_ver of this libmali is 'g6p0-01eac0', rk_so_ver is '5'.
Failed creating base context during opening of kernel driver.
Kernel module 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

9.6 Usage of Python examples and project creation

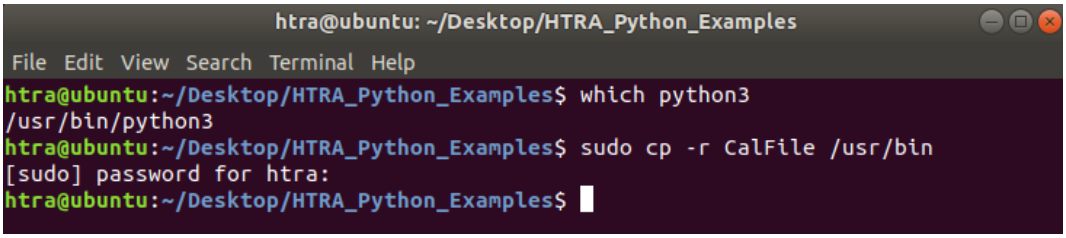
9.6.1 Usage of Python examples

Assuming the device is properly connected and the driver files have been configured correctly as per section 9.3, if you want to use the Python examples included on the USB drive, you can refer to the following process (the specific functions of the Python examples can be directly viewed in section 4.2):Chapter9.3

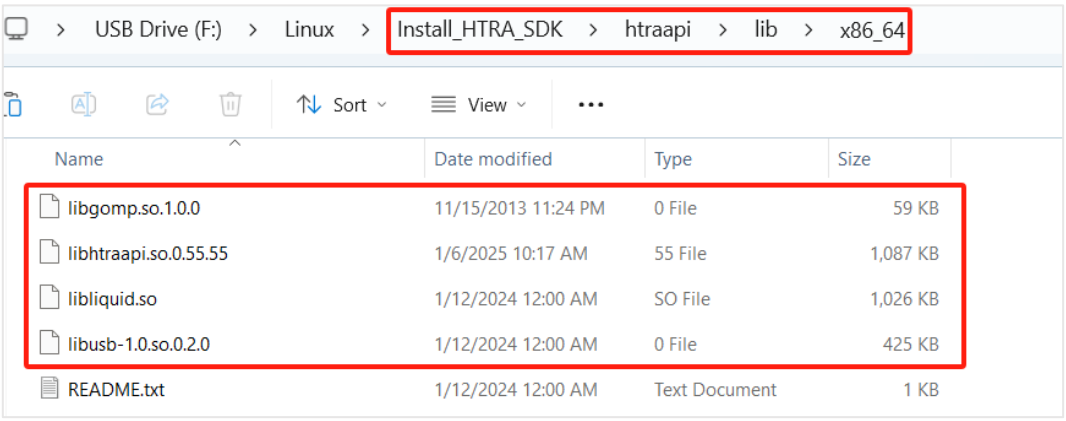
1. Copy the Linux\ HTRA_ Python_ Examples folder from the included USB drive to the host computer, then open a terminal in the HTRA_ Python_ Examples folder and input which python3 to check the location of the Python interpreter (for example, it may be located at /usr/bin).

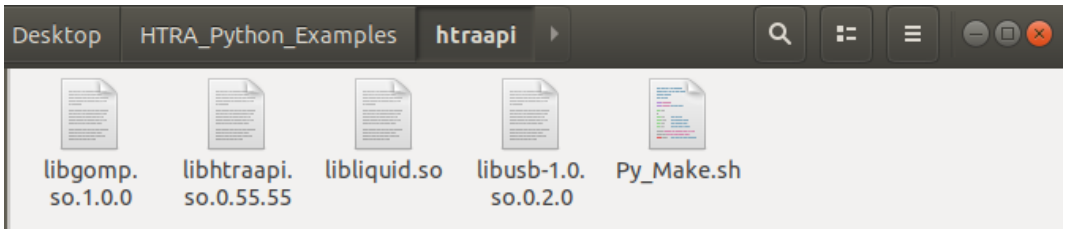


2. Based on the interpreter address obtained earlier, input sudo cp -r CalFile /usr/bin to copy the device calibration file to the same directory as the interpreter (Python3) (for example, in this case, the same directory is /usr/bin; if the actual interpreter location is different, modify the address accordingly).

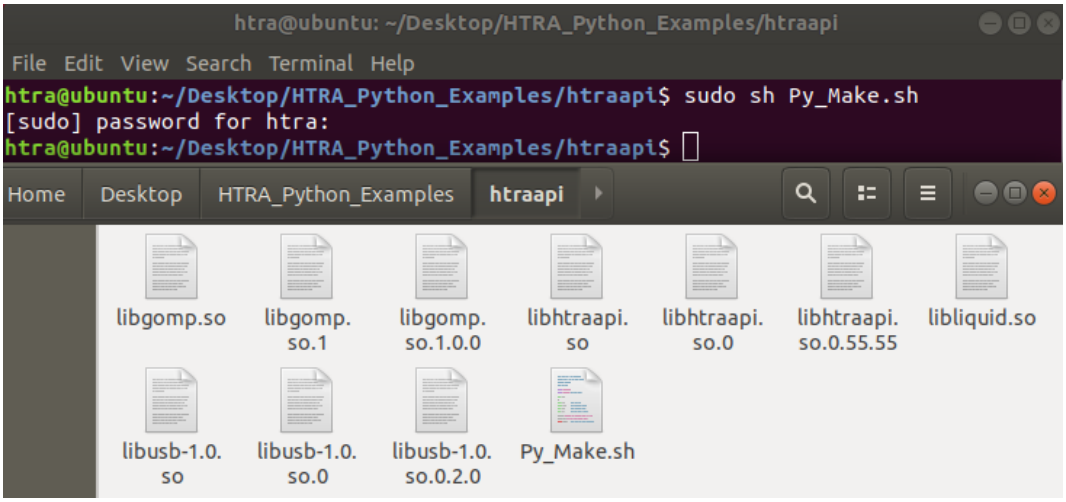


3. First, refer to the system architecture in section 9.1, then according to section 9.2.4, copy the dynamic link libraries from the corresponding system architecture folder under the included USB drive Linux\Install_HTRA_SDK\htraapi\lib to the HTRA_Python_Examples\htraapi folder (this example uses the x86_64 architecture host computer).Chapter9.1Chapter9.2.4

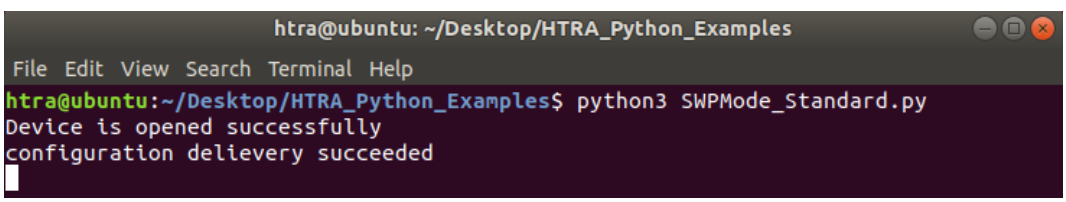




4. Open a terminal in the current folder and input “sudo sh Py_Make.sh”, then follow the prompts to enter the sudo password to provide permissions for creating soft links to the libraries.



5. In the HTRA_ Python_ Examples location, open a terminal and input python3 SWPMode_Standard.py to run the example. (This example uses SWPMode_Standard.py as an example)



9.6.2 Python Project Creation

Under the premise that the driver files have been correctly configured according to section 9.3, if you want to create and write a Python project, please refer to the following process:Chapter9.3

First, when writing code, since the Linux dynamic link library provided with the USB drive is identical to that in Windows, the code only needs to comply with the API programming guidelines.

Secondly, when running the program, simply place the program in the example folder and follow the process outlined in the Python example usage section.PythonExample Usage

9.7 Java (to be supplemented)

It is important to note that when writing programs in Java to control devices on a Linux system, the CalFile folder needs to be placed in the same directory as the Java interpreter.