API 范例使用指南

API Version 0.55.61

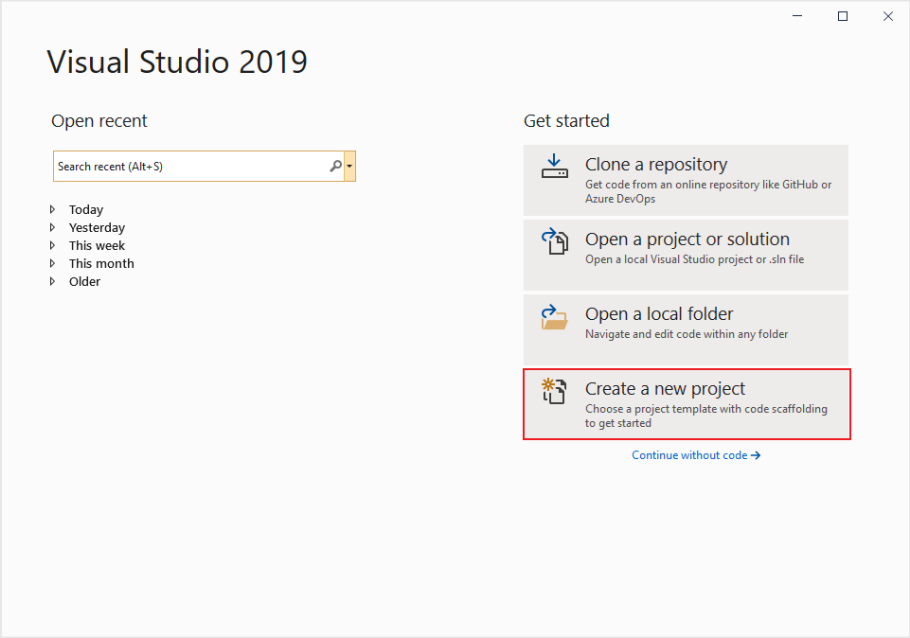
2025-8-18

目录

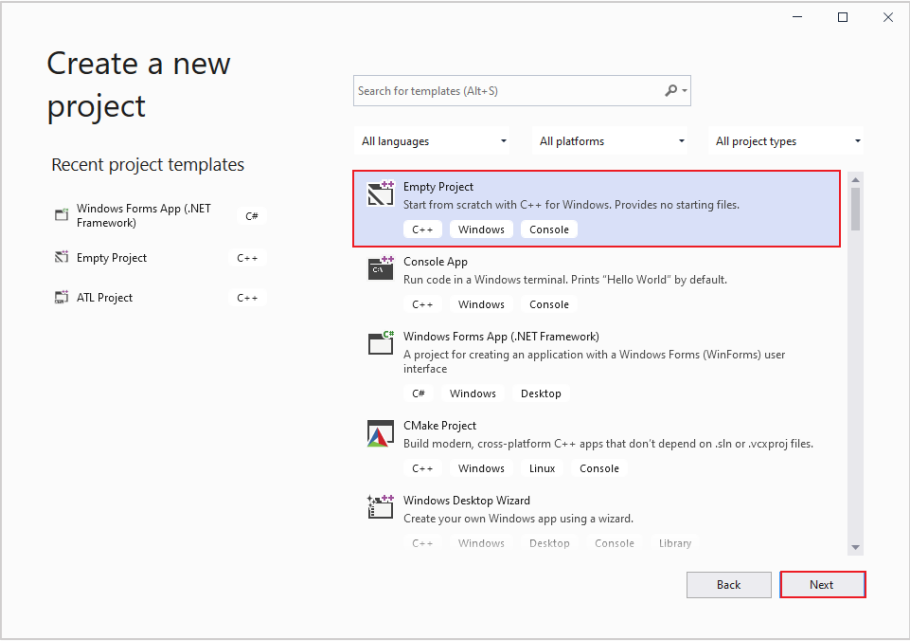
1. C/C++

1.1 配置开发环境

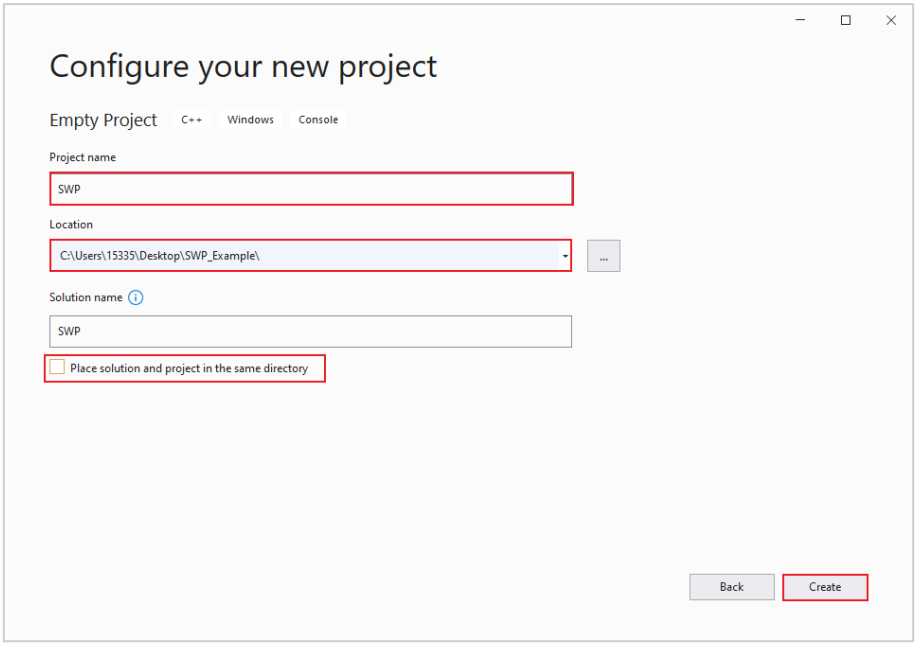
1. 打开 VS Studio 2019，创建一个新项目。



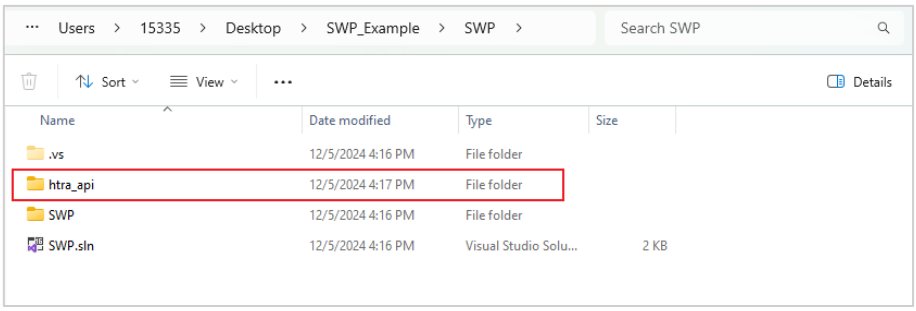
2. 选择空项目，并点击下一步。



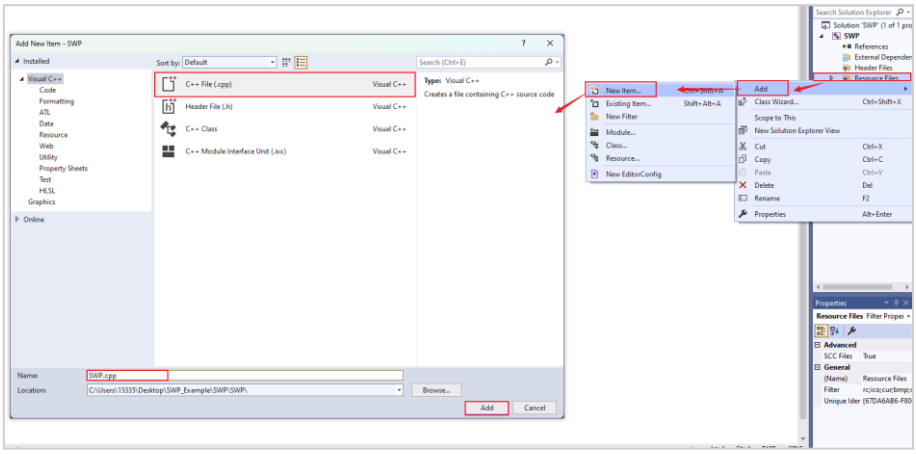
3. 填写项目名称与存放地址，取消“将解决方案和项目放在同一目录中”，然后点击创建。



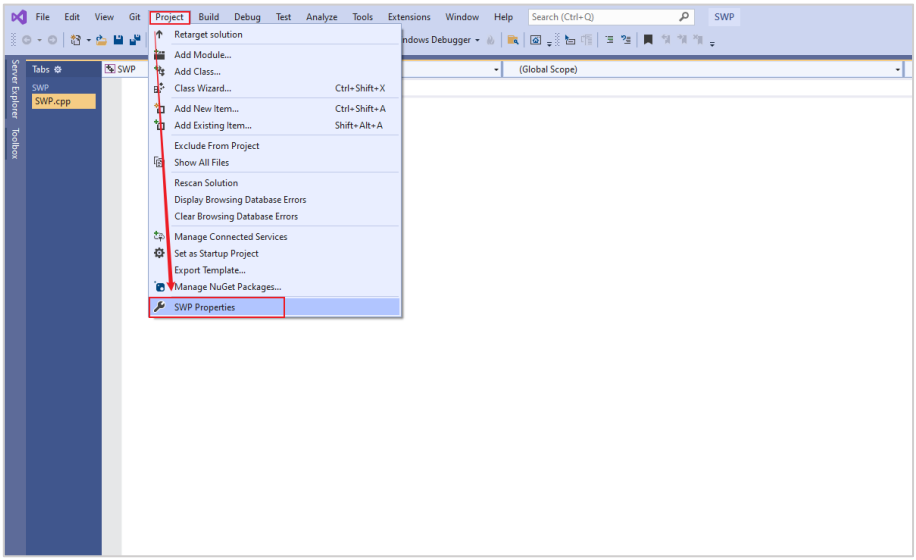
4. 创建完成后，将发货 U 盘 Windows\HTRA_API\x86 中的 htra_api 文件夹拷贝到工程的平级目录下（此处以配置 x86 架构项目为例，若想配置 x64 架构项目则拷贝 Windows\HTRA_API\x64\htra_api 文件夹）。



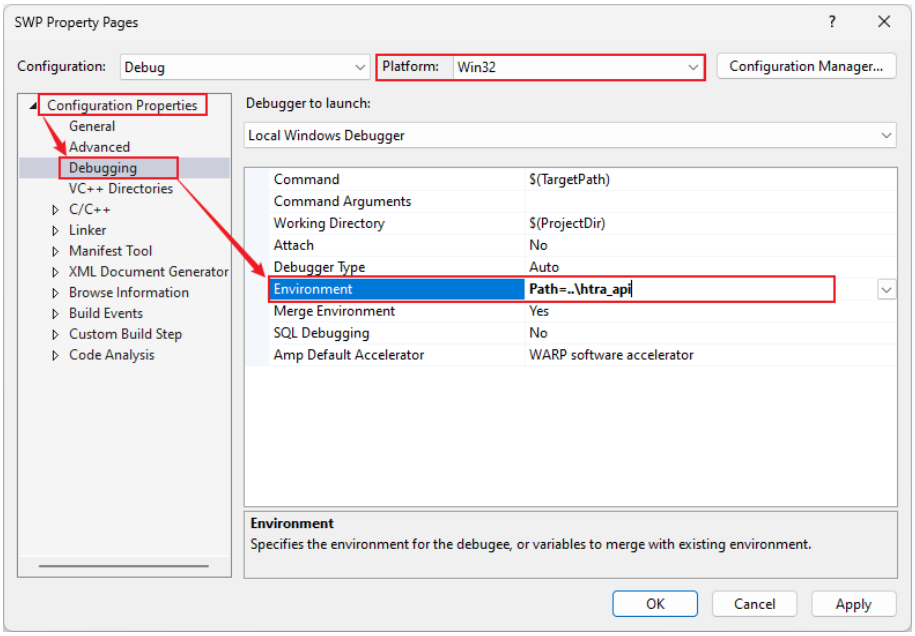
5. 双击打开 SWP.sln，在源文件中新建一个 SWP.cpp 文件。



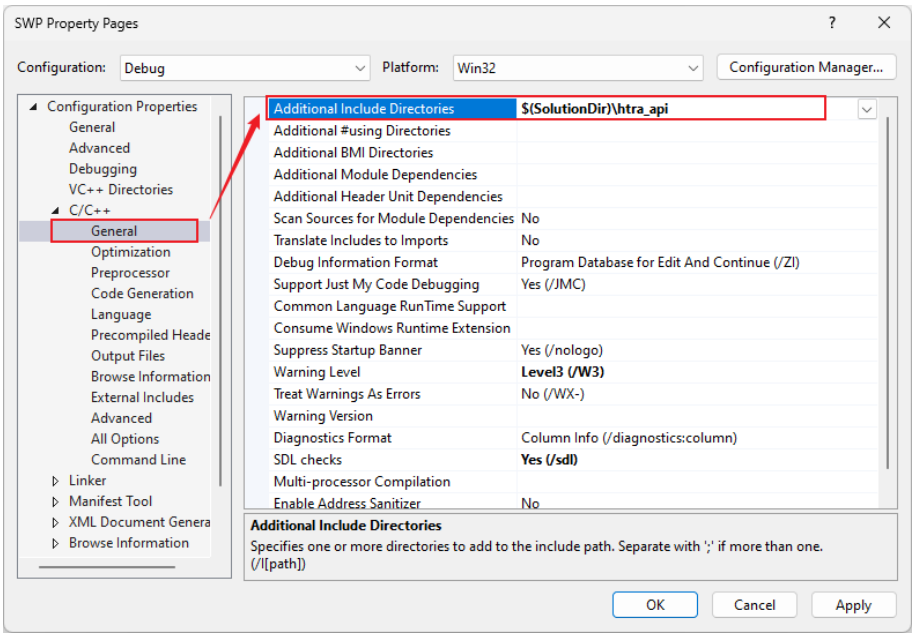
6. 点击菜单栏中的“项目”，选择“属性”。



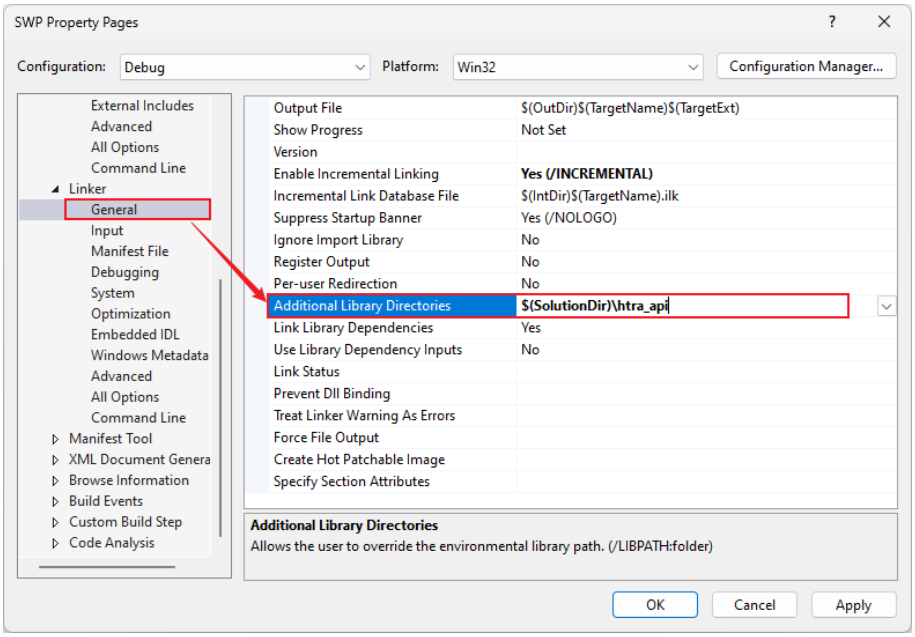
7. 配置平台选择“Win32”，并将配置属性 -> 调试中的环境变量设置为 Path=..\htra_api（配置 x64 架构项目时，配置平台选择“x64”，除此之外 7-10 步的配置流程与 x86 架构（Win32）相同）。



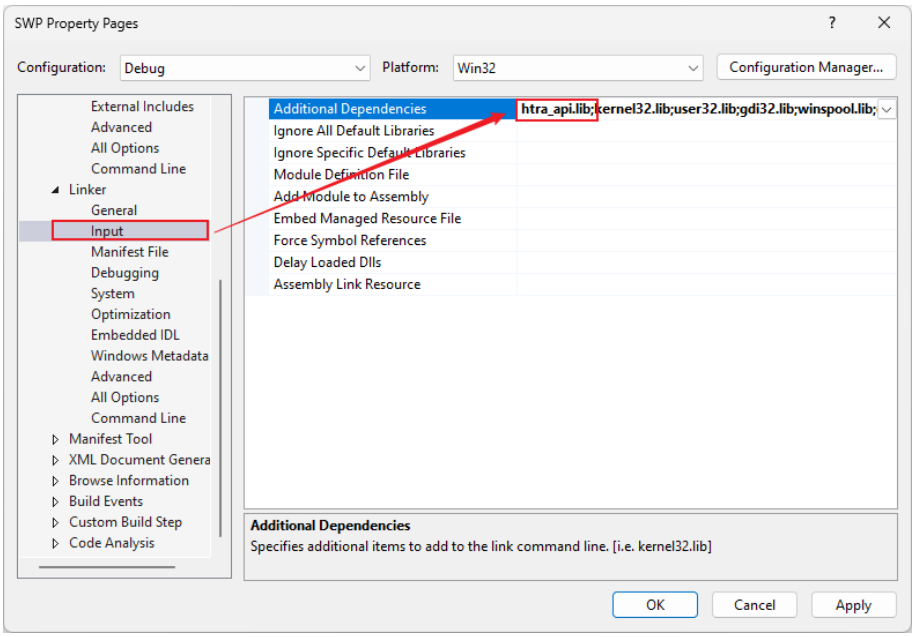
8. 将配置属性 -> C/C++ -> 常规中的附加包含目录设置为 $(SolutionDir)\htra_api。



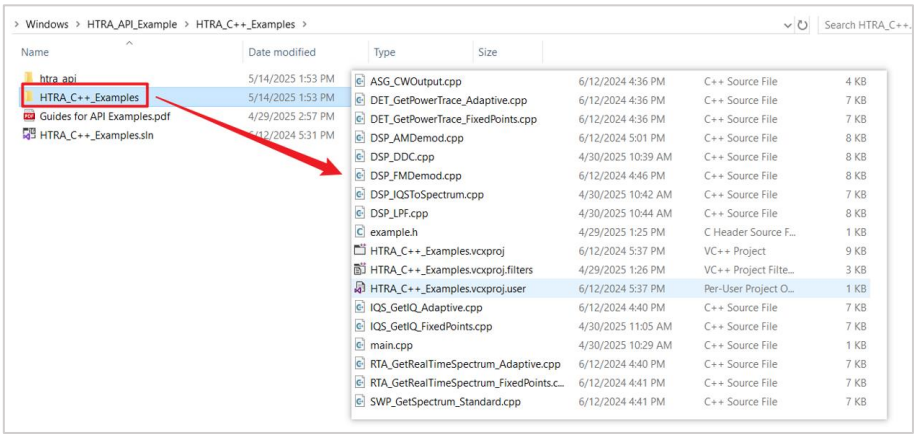
9. 将配置属性 -> 链接器 -> 常规中的附加库目录设置为 $(SolutionDir)\htra_api。



10. 给配置属性 -> 链接器 -> 输入中的附加依赖项新增 htra_api.lib。



11. 至此，环境配置结束，可以进行编程开发。此处可参照随寄 U 盘中 C/C++ 范例，即 Windows\HTRA_API_Example\HTRA_C++_Examples\HTRA_C++_Examples 中项目。

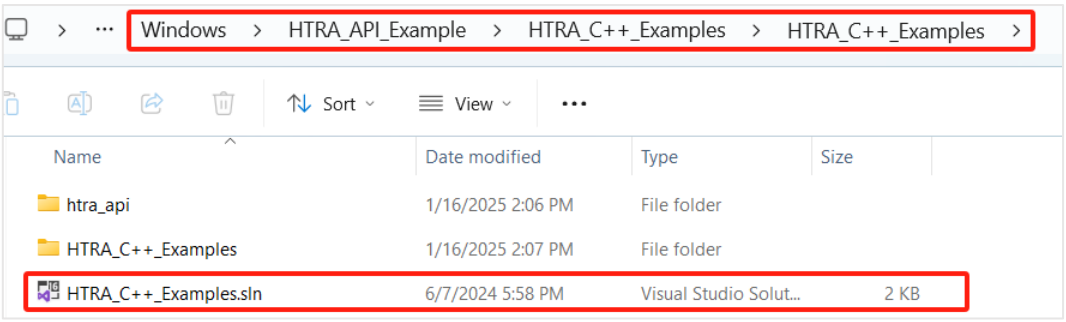


1.2 C++范例使用流程

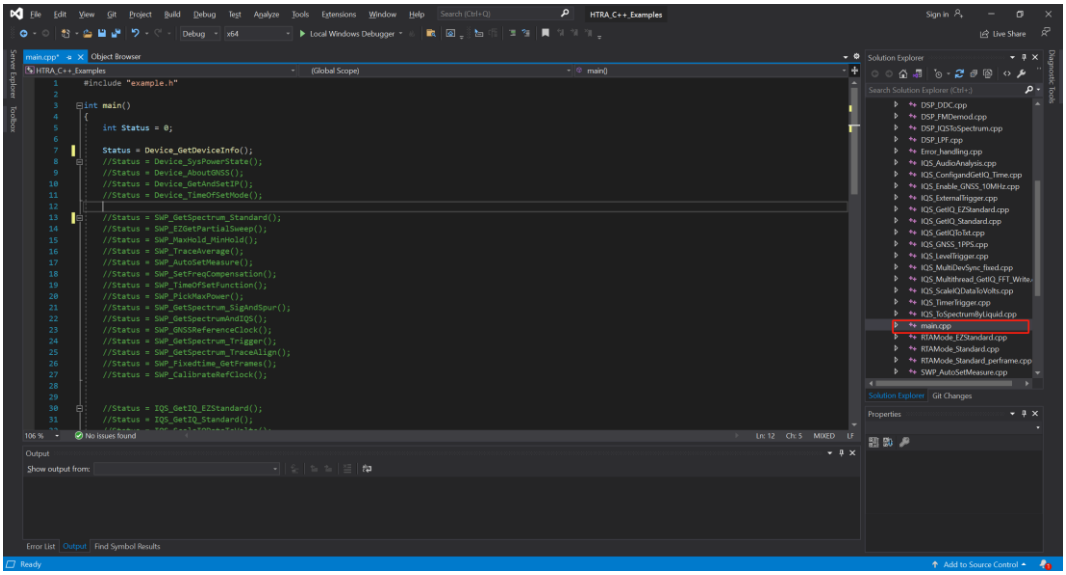
1.2.1 常规 C++范例的使用

随寄 U 盘中常规 C++范例使用流程如下：

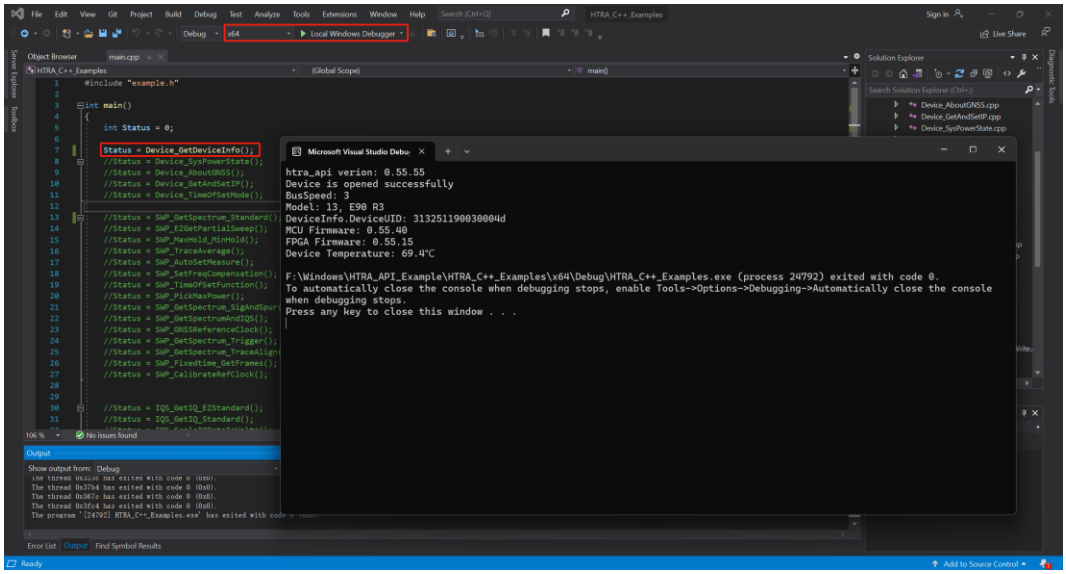
1. 使用 Visual Studio 打开随寄 U 盘 Windows\HTRA_API_Example\HTRA_C++_Examples\HTRA_C++_Examples 文件夹下解决方案 HTRA_C++_Examples.sln。



2. 点击右侧 HTRA_C++_Examples 项目，点击其中的 main.cpp 文件。

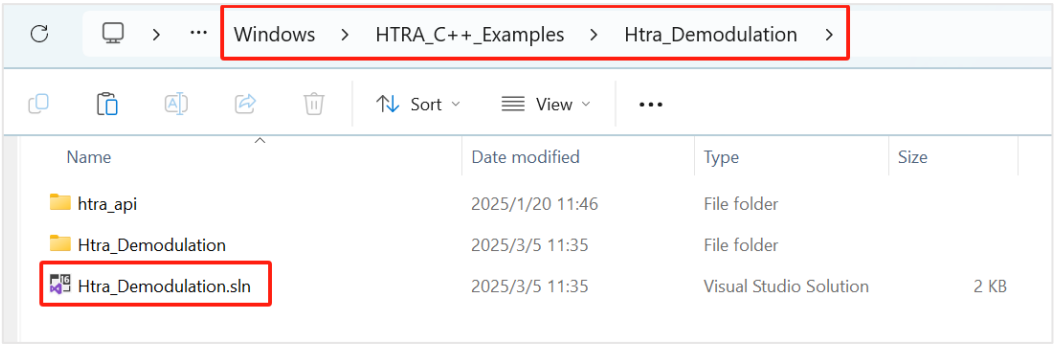


3. C++随寄范例每个例程都封装在单独的函数中，使用范例时取消注释即可（同一时间只可以使用一个范例）。例如测试使用 Device_GetDeviceInfo 例程时，取消注释并保存，选择编译架构（x86 与 x64 均可），点击运行，如图所示为正常运行设备。



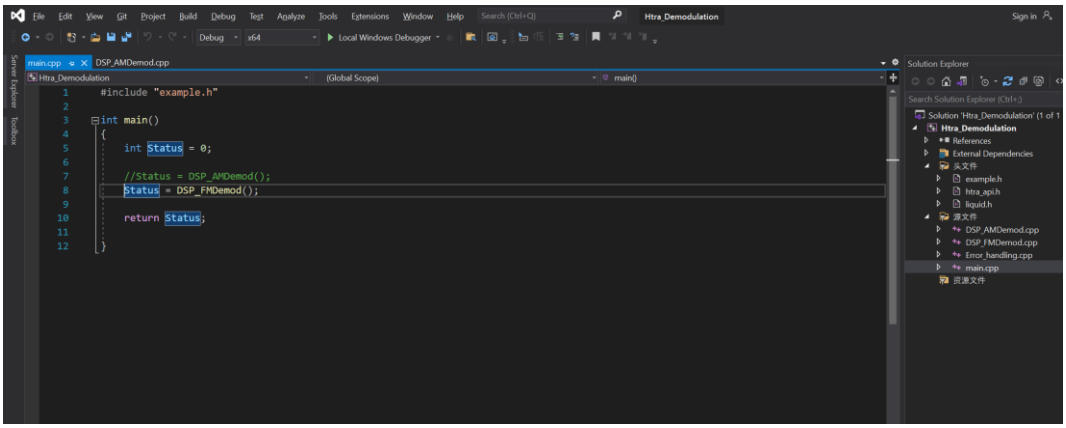
1.2.2 AM/FM 解调范例的使用

1. 使用 Visual Studio 打开随寄 U 盘 Windows\HTRA_C++_Examples\Htra_Demodulation 文件夹下解决方案 Htra_Demodulation.sln。



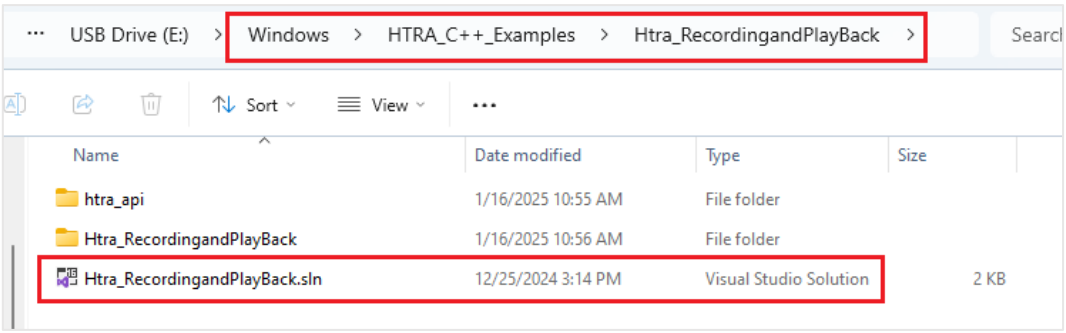
2. 点击右侧 HTRA_C++_Examples 项目，点击其中的 main.cpp 文件。

3. C++随寄范例每个例程都封装在单独的函数中，使用范例时取消注释即可（不可以同时使用多个范例）。例如测试使用 DSP_FMDemod 例程时，取消注释并保存，选择编译架构（x86 与 x64 均可），点击运行。



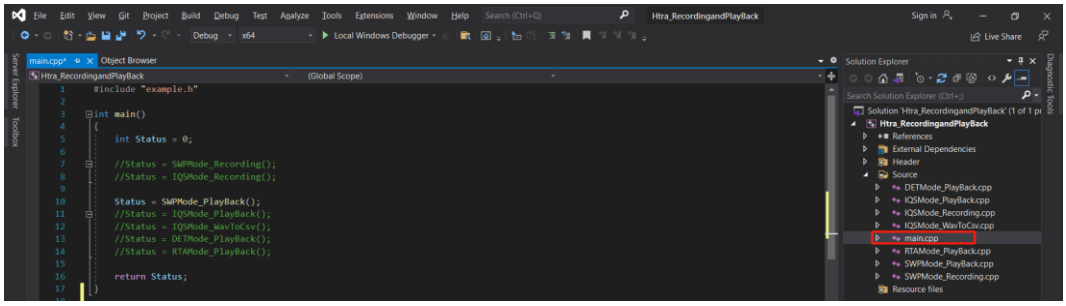
1.2.3 记录与读取范例的使用

1. 使用 Visual Studio 打开随寄 U 盘 Windows\HTRA_API_Example\Htra_RecordingandPlayBack 文件夹下解决方案 Htra_RecordingandPlayBack.sln。



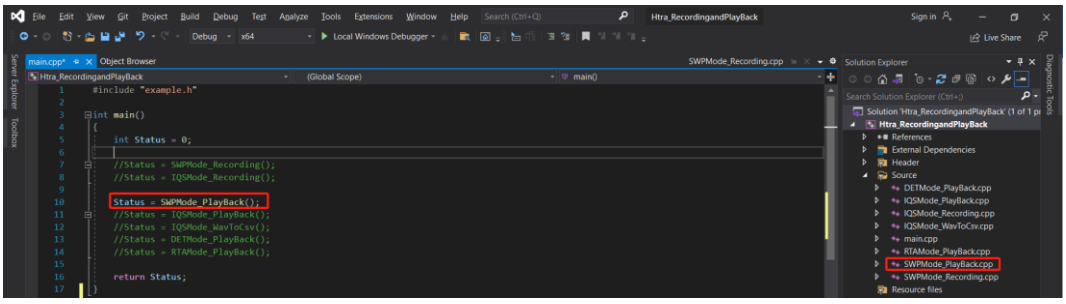
2. 点击右侧 Htra_RecordingandPlayBack 项目，点击其中的 main.cpp 文件。

3. 记录与读取范例每个例程都封装在单独的函数中，使用范例时取消注释即可（不可以同时使用多个范例）。

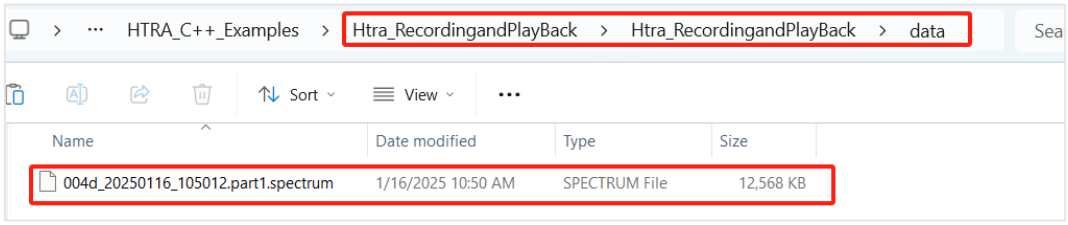


（1）读取范例的使用

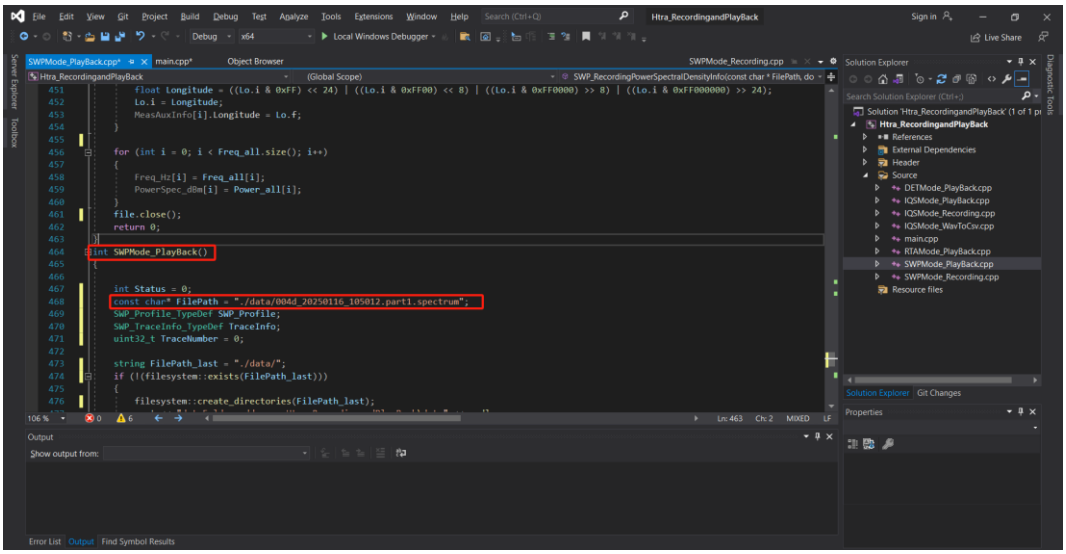
1) 例如读取 SWP 模式流盘数据时，将 SWPMode_PlayBack 函数取消注释并保存。



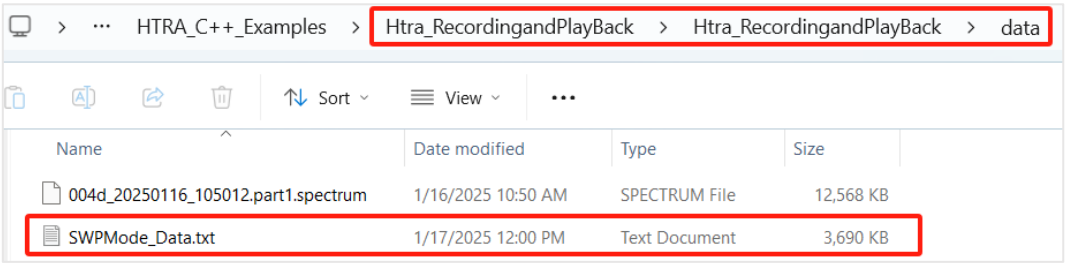
2) 将 SWP 模式下的记录文件数据放至“Windows\HTRA_C++_Examples\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data”文件夹中。



3) 点击进入 SWPMode_PlayBack.cpp，修改 SWPMode_PlayBack()函数中记录文件的名称；

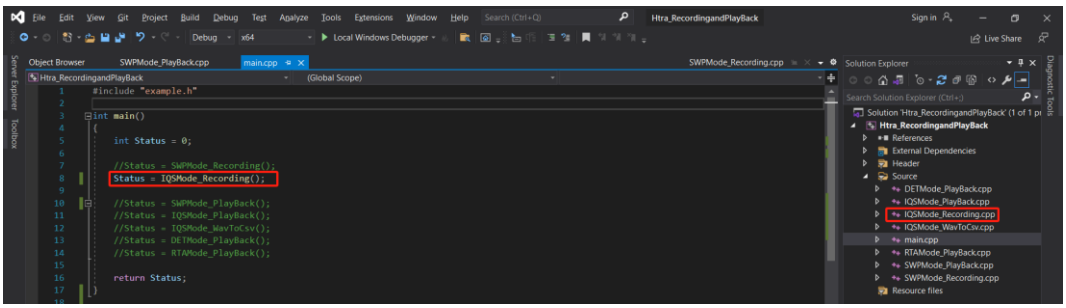


4) 运行程序即可在 data 文件夹中得到 SWP 模式下记录文件的解析数据“SWPMode_Data.txt”。

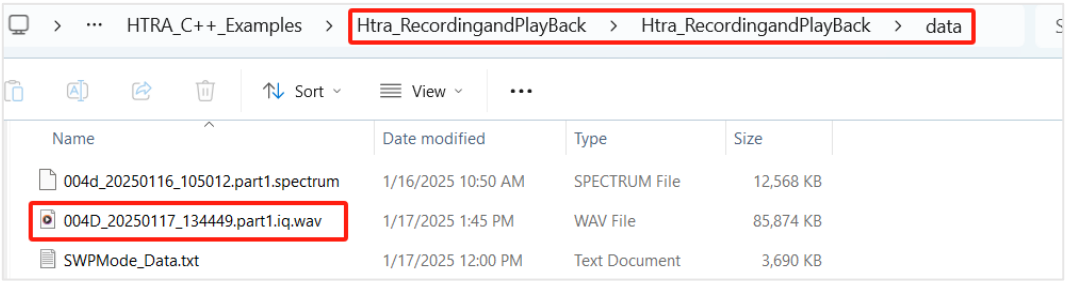


（2）记录范例的使用

1) 例如测试使用 IQSMode_Recording 例程时，取消注释并保存。



2) 点击进入 IQSMode_Recording()函数，配置参数后运行程序，可以在“Windows\HTRA_C++_Examples\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data”文件夹中得到 IQS 模式下的记录文件数据。



1.3 Device 相关

1.3.1 获取设备信息

Device_GetDeviceInfo.cpp：获取设备信息，包括：API 版本、上位机与设备相连的 USB 版本、设备型号、设备 UID、MCU 版本、FPGA 版本和设备温度。

1.3.2 设备待机

Device_SysPowerState.cpp：设置设备待机状态的范例，可以设置为正常工作状态和射频处于下电状态（低功耗）。

1.3.3 GNSS 相关

Device_AboutGNSS.cpp：获取 GNSS 模块获取的经纬度、海拔和时间等信息，获取 SWP 模式下 MeasAuxInfo 中 GNSS 相关的经纬度和时间信息，获取 IQS 模式下 IQStream.DeviceState 中的经纬度和时间信息。

1.3.4 获取和修改 NX 设备的 IP 地址

Device_GetAndSetIP.cpp：获取设备 IP 地址，并通过设备 UID 或者设备当前的 IP 修改 IP 地址。

1.3.5 模式切换耗时

Device_MeasureModeSwitchTime.cpp：获取当前上位机切换不同模式所需要的时间。

1.4 SWP 模式

1.4.1 标准频谱获取

SWP_GetSpectrum_Standard.cpp：通过调用函数接口获取频谱数据。

1.4.2 简洁配置模式

SWP_EZGetPartialSweep.cpp：使用简洁配置快速获取频谱数据。

1.4.3 最大和最小保持

SWP_MaxHold_MinHold.cpp：将迹线模式设置为 MaxHold 或 MinHold，并使用 SWP_ResetTraceHold 重置保持。

1.4.4 迹线平均

SWP_TraceAverage.cpp：对获取迹线进行平均处理。

1.4.5 自动配置测量

SWP_AutoSetMeasure.cpp：根据具体 SWP 应用，自动配置相关参数，通过下发自动配置参数，完成测量。

1.4.6 频率补偿

SWP_SetFreqCompensation.cpp：当存在外接衰减器时，可对相应频段进行补偿，使得测试结果依然准确。

1.4.7 函数耗时、扫描速度和吞吐量

SWP_TimeOfSetFunction.cpp：获取 SWP_Configuration、SWP_GetPartialSweep 和 SWP_GetFullSweep 函数的调用耗时与当前配置下的扫描速度、吞吐量。

1.4.8 获取频谱峰值

SWP_PickMaxPower.cpp：获取当前频谱的最大功率点与其对应的频率点。

1.4.9 信号和杂散

SWP_GetSpectrum_SigAndSpur.cpp：可以在获取频谱数据后区分信号和杂散。

1.4.10 同时获取频谱和 IQ

SWP_GetSpectrumAndIQS.cpp：同时获取频谱数据与 IQ 数据。

1.4.11 读取随寄软件的 SWP 流盘数据

SWPMode_PlayBack.cpp：可以读取随寄软件在 SWP 模式下的记录文件数据，并将读取的频谱数据写入 SWPMode_Data.txt。

1.4.12 使用 GNSS 的 10MHz 参考时钟

SWP_GNSSReferenceClock.cpp：SWP 模式下使用高品质 GNSS 模块的 10MHz 参考时钟。

1.4.13 外部触发模式

SWP_GetSpectrum_Trigger.cpp：获取触发源设置为外触发时的频谱数据。

1.4.14 迹线对齐方式

SWP_GetSpectrum_TraceAlign.cpp：在迹线对齐方式为对齐至起始频率或对齐至中心频率时，获取频谱数据。

1.4.15 测试一定时间内可获取的频谱帧数

SWP_Fixedtime_GetFrames.cpp：循环 50 次获取 10s 频谱数据，得出在 10s 内能获取的平均频谱帧数。

1.4.16 外触发标定内部 10MHz 参考时钟

SWP_CalibrateRefClock.cpp：设备通过GNSS-1PPS或者通过Ext触发进行Clock校准的范例。

1.4.17 相位噪声测量

SWP_Meas_PhaseNoise.cpp：测量接收信号在用户指定频偏处的单边带相位噪声（单位：dBc/Hz），支持多频点配置。

1.4.18 信道功率测量

SWP_Meas_ChannelPower.cpp：测量接收信号的信道功率。

1.4.19 邻道功率比测量

SWP_Meas_ACPR.cpp：测量接收信号的邻道功率比。

1.4.20 百分比占用带宽测量

SWP_Meas_OBW.cpp：以百分比方式测量接收信号的占用带宽。

1.4.21 XdB 占用带宽测量

SWP_Meas_XdBBW.cpp：以 XdB 方式测量接收信号的占用带宽。

1.4.22 IM3 测量

SWP_Meas_IM3.cpp：接收信号的 IM3 测量。

1.4.23 频率间隔匹配 RBW

SWP_RBW_Spaced_Trace.cpp：使迹线两点之间的频率间隔接近所设置的 RBW 值。

1.4.24 使用外部 10MHz 参考时钟

SWP_RefCLKSource_External.cpp：使用外部 10MHz 参考时钟。（以 SWP 模式下使用外部 10MHz 参考时钟为例，其他模式下使用方法相同）。

1.5 IQS 模式

1.5.1 获取固定点数或连续流的 IQ 数据

IQS_GetIQ_Standrad.cpp：在专业配置下获取固定点数或连续流的 IQ 数据。

1.5.2 简洁配置模式

IQS_GetIQ_EZStandard.cpp：使用简洁配置快速获取 IQ 数据。

1.5.3 IQ 数据转化为电压 V 单位

IQS_ScaleIQDataToVolts.cpp：将获取的 IQ 数据转换为以 V 为单位的数据。

1.5.4 下发配置和获取 IQ 的耗时

IQS_ConfigandGetIQ_Time.cpp：获取 IQS_Configuration、IQS_GetIQStream_PM1 函数的调用耗时。

1.5.5 IQ 转频谱数据

DSP_IQSToSpectrum.cpp：将获取到的时域 IQ 数据通过频谱分析方法转换为频谱数据。

1.5.6 IQ 转频谱（调用 liquid 库版本）

IQS_ToSpectrumByLiquid.cpp：调用 liquid 库将获取到的时域 IQ 数据通过频谱分析方法转换为频谱数据。

1.5.7 FM 解调

DSP_FMDemod.cpp：对 IQ 数据进行 FM 解调，并播放解调后的音频。

1.5.8 AM 解调

DSP_AMDemod.cpp：对 IQ 数据进行 AM 解调，并播放解调后的音频。

1.5.9 数字下变频

DSP_DDC.cpp：对获得的 IQ 数据进行重采样。

1.5.10 数字低通滤波器

DSP_LPF.cpp：对获得的 IQ 数据进行低通滤波。

1.5.11 音频分析

IQS_AudioAnalysis.cpp：对解调之后的 IQ 数据进行音频分析，得到音频电压、音频频率、信纳德和总谐波失真。

1.5.12 读取随寄软件的 IQS 流盘数据

IQSMode_PlayBack.cpp：解析随寄软件在 IQS 模式下的记录文件数据，并将读取的频谱数据写入 IQSMode_Data.txt 文件。

1.5.13 将 IQ 数据记录为.wav 格式

IQSMode_Recording.cpp：将获得的 IQ 数据以.wav 的格式存储。

1.5.14 .wav 换为.csv

IQSMode_WavToCsv.cpp：将 IQS 模式下的.wav 记录文件数据解析并提取 I 路和 Q 路数据，转换为.CSV 格式文件进行保存。

1.5.15 流盘和读取 IQ 数据

IQS_GetIQToTxt.cpp：将获得的 IQ 数据写入.txt 文件中。

1.5.16 多线程获取、处理和流盘 IQ 数据

IQS_Multithread_GetIQ_FFT_Write：同时获取 IQ 数据、做 FFT 和将 IQ 数据写入.txt 文件。

1.5.17 GNSS1PPS 触发

IQS_GNSS_1PPS.cpp：配置触发源为系统内 GNSS 提供的 1PPS 信号。

1.5.18 IQS 多机同步

IQS_MultiDevSync_fixed.cpp：多台设备在同一时刻同时采集同一信号。

1.5.19 外触发

IQS_ExternalTrigger.cpp：配置触发源为外部触发。

1.5.20 定时器触发

IQS_TimerTrigger.cpp：配置触发源为定时器触发。

1.5.21 电平触发（预触发）

IQS_LevelTrigger_PreTriggerr.cpp：配置触发源为电平触发，并设置预触发时间。

1.5.22 电平触发（触发延迟）

IQS_LevelTrigger_TriggerDelay.cpp：配置触发源为电平触发，并设置触发延迟时间。

1.5.23 使用 GNSS 的 10MHz 参考时钟

IQS_Enable_GNSS_10MHz.cpp：在 IQS 模式下使用高品质 GNSS 模块的 10MHz 参考时钟。

1.5.24 固定步进多频点 IQ 数据采集

IQS_SetFreqScan：提前配置好 IQ 的起始、终止频率和频点数。一次 config 之后，按照设定的频点依次采集数据。

1.6 DET 模式

1.6.1 获取固定点数或连续流的检波数据

DETMode_Standard.cpp：获取固定点数或连续流的检波数据。

1.6.2 简洁配置模式

DETMode_EZStandard.cpp：通过简洁配置快速获取检波数据。

1.6.3 读取随寄软件的 DET 流盘数据

DETMode_PlayBack.cpp：读取随寄软件在 DET 模式下的记录文件，并将检波数据输出到 DETMode_Data.txt 文件中。

1.6.4 脉冲检测（后续开放）

1.7 RTA 模式

1.7.1 获取固定点数或连续流的实时频谱数据

RTAMode_Standard.cpp：获取固定点数（时长）或连续流的实时频谱数据。

1.7.2 简洁配置模式

RTAMode_EZStandard.cpp:通过简洁配置快速获取实时频谱数据。

1.7.3 读取随寄软件的 RTA 流盘数据

RTAMode_PlayBack.cpp：读取随寄软件在 RTA 模式下的记录文件数据，同时可以指定读取某一包数据，并将读取的频谱数据写入 RTAMode_Data.txt 文件。

1.7.4 RTA 模式每帧数据的耗时

RTAMode_Standard_perframe.cpp：获取 100 帧数据，并计算每帧的平均处理时间。

1.8 ASG 信号源（选件）

1.8.1 输出单音/扫频/功率扫描信号

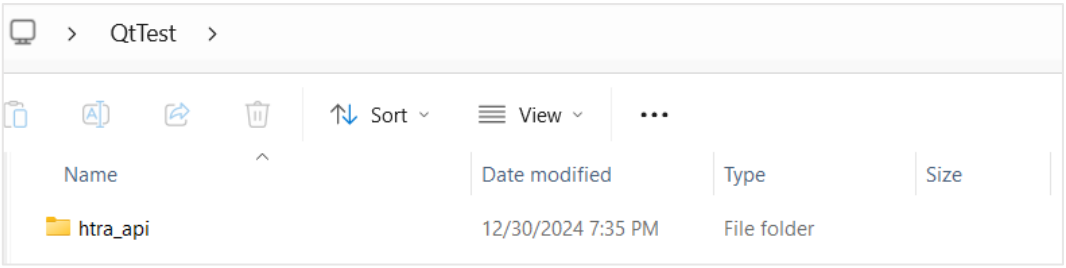
ASG_SignalOutput.cpp：按需输出单音/扫频/功率扫描信号。

2. 数字解调（选件）

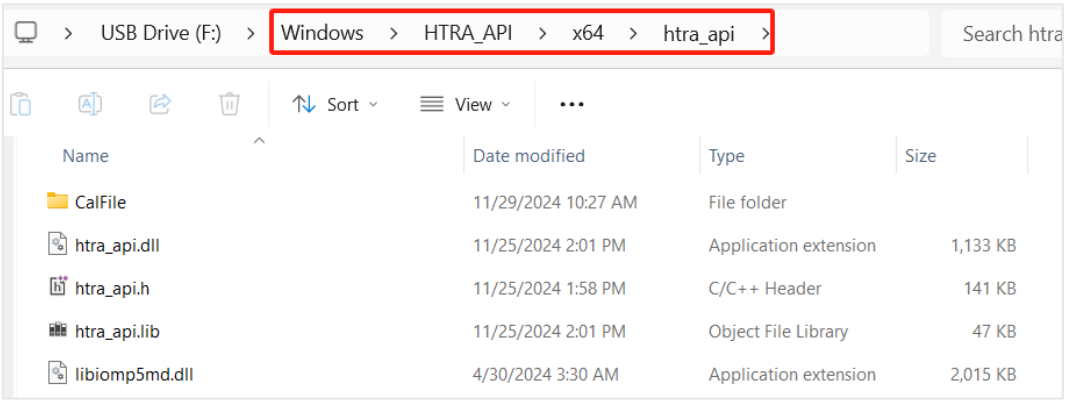
3. Qt

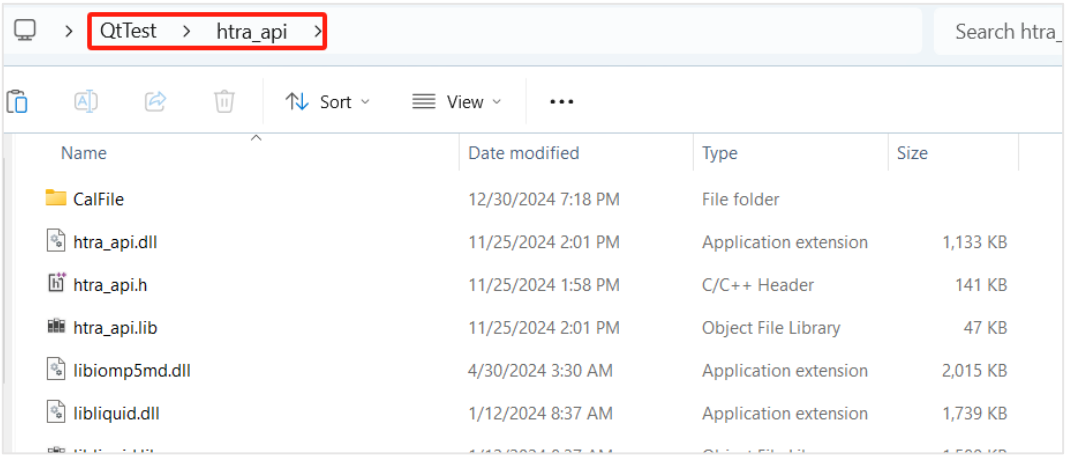
3.1 配置开发环境

1. 如图所示，首先创建一个新文件夹用于存放整个项目（此处以 QtTest 为例，注意不要使用中文路径），然后在文件夹内创建 htra_api 文件夹用于存放动态链接库与校准文件。

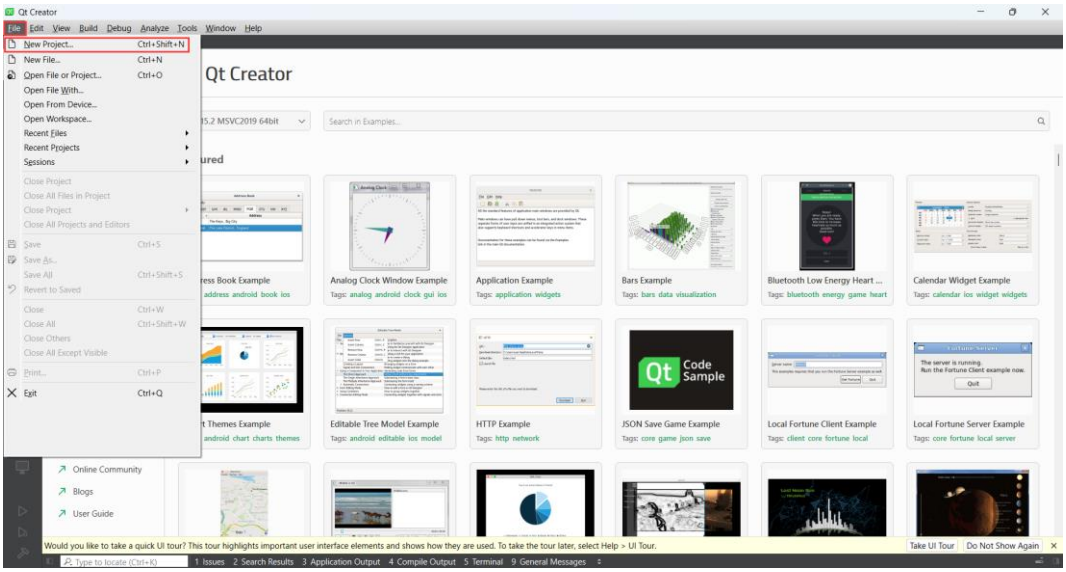


2. 将随寄 U 盘 Windows\HTRA_API\x64\htra_api 文件夹中所有文件复制至刚创建的 QtTest\htra_api 文件夹中（此处以 x64 架构程序为例，x86 架构程序复制 Windows\HTRA_API\x86\htra_api 文件夹中的库即可）。

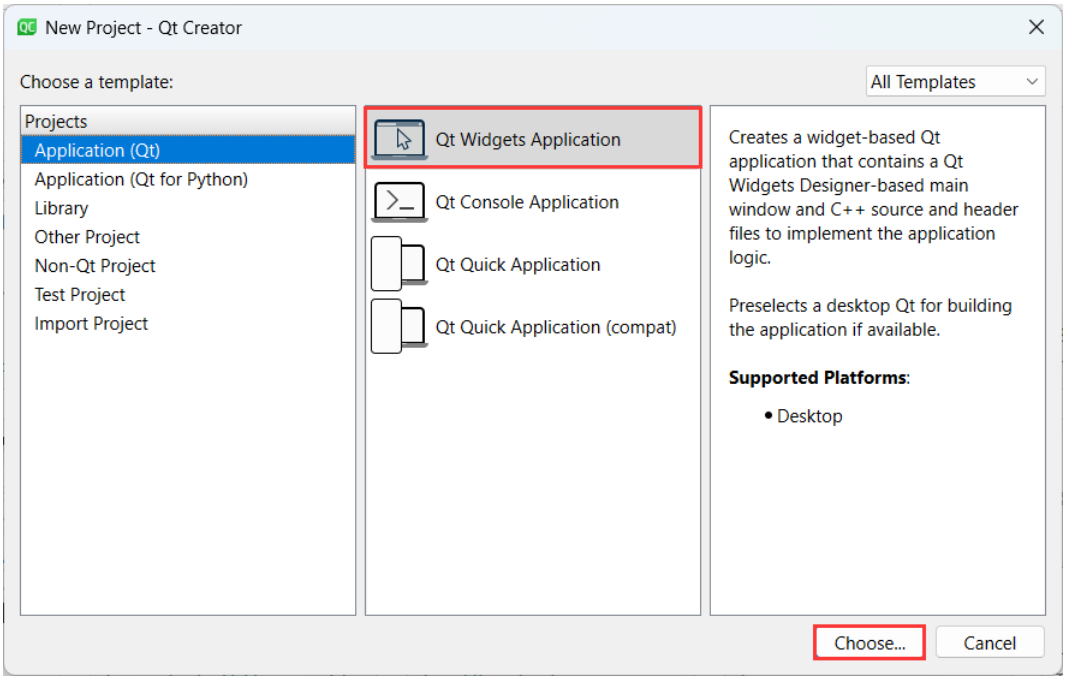




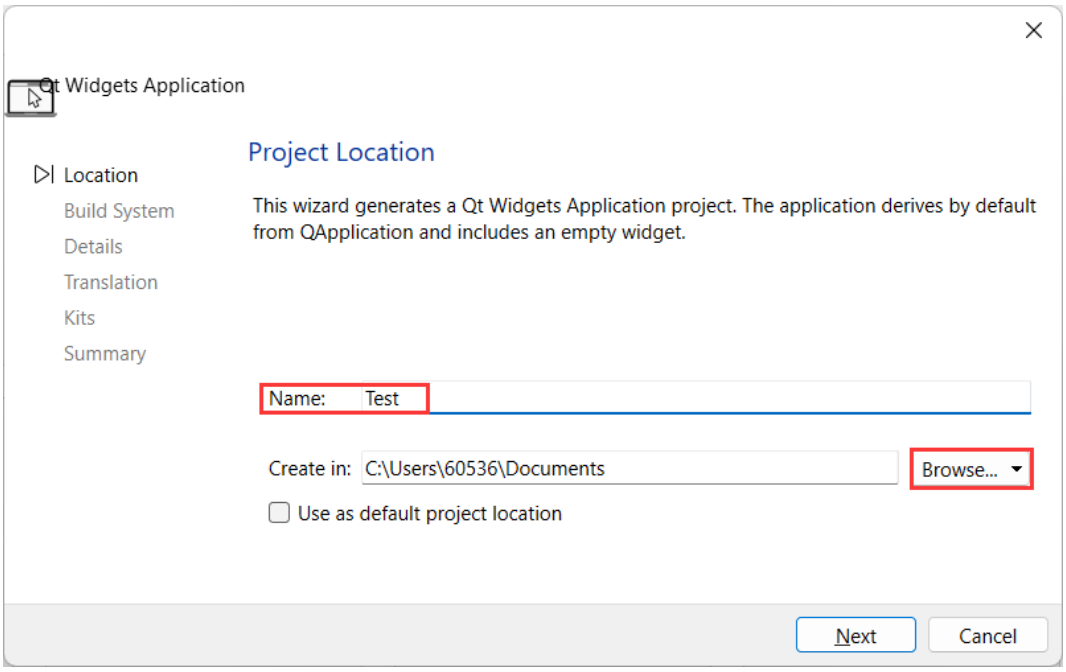
3. 打开 Qtcreater，点击文件，选择新建文件或项目。



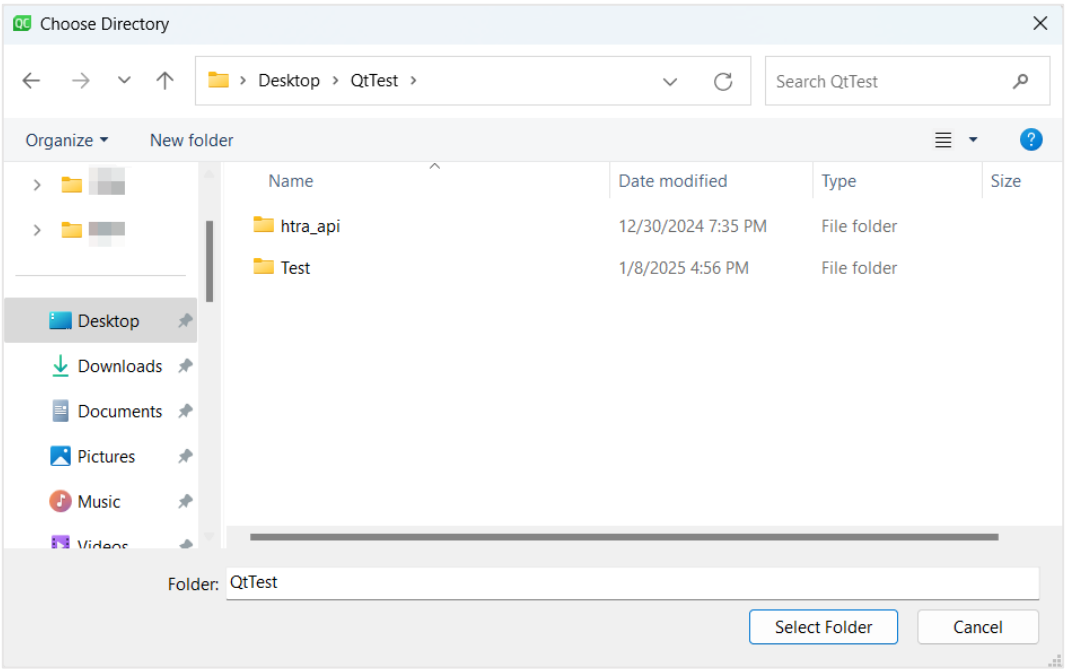
4. 选择创建窗体程序。



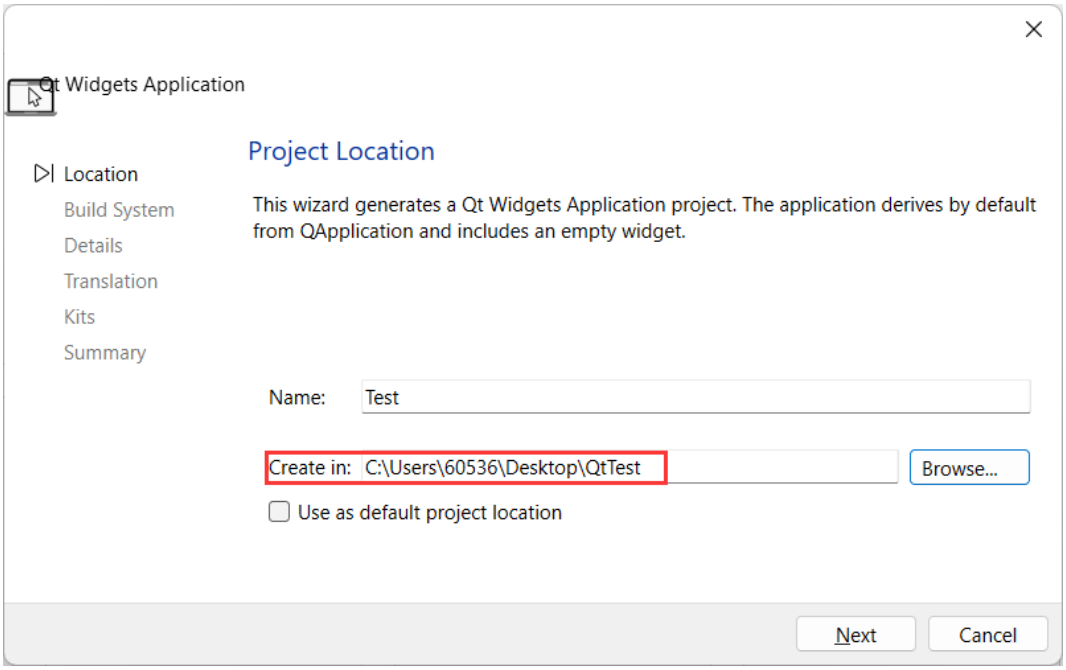
5. 填写项目名称后点击浏览更换项目路径。



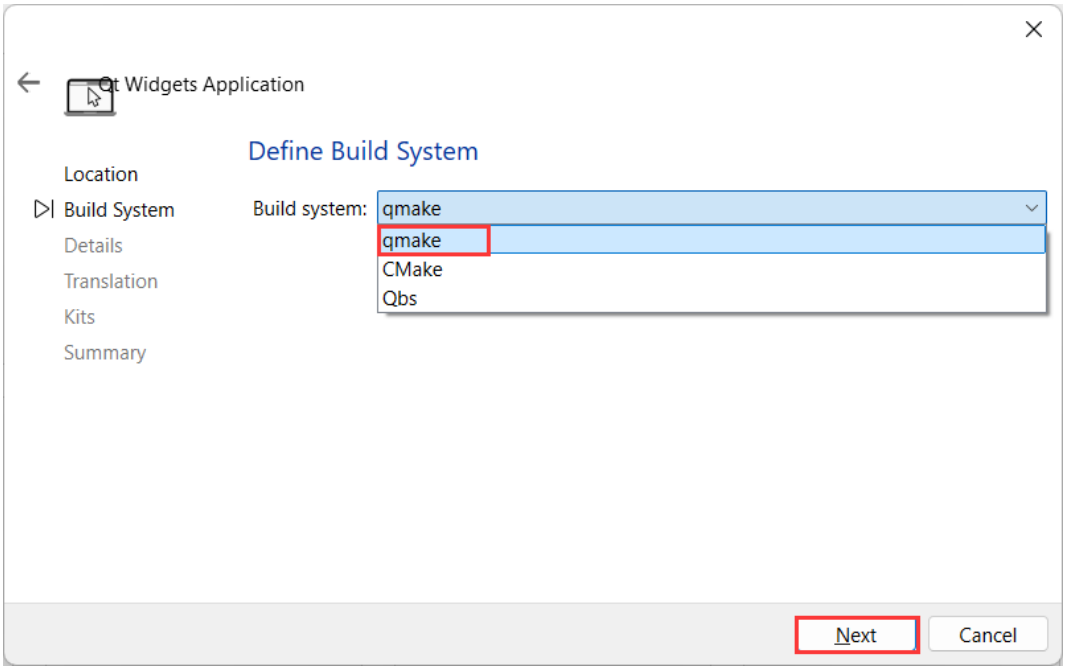
6. 选择目录为刚才创建的 QtTest 地址后点击 Open。



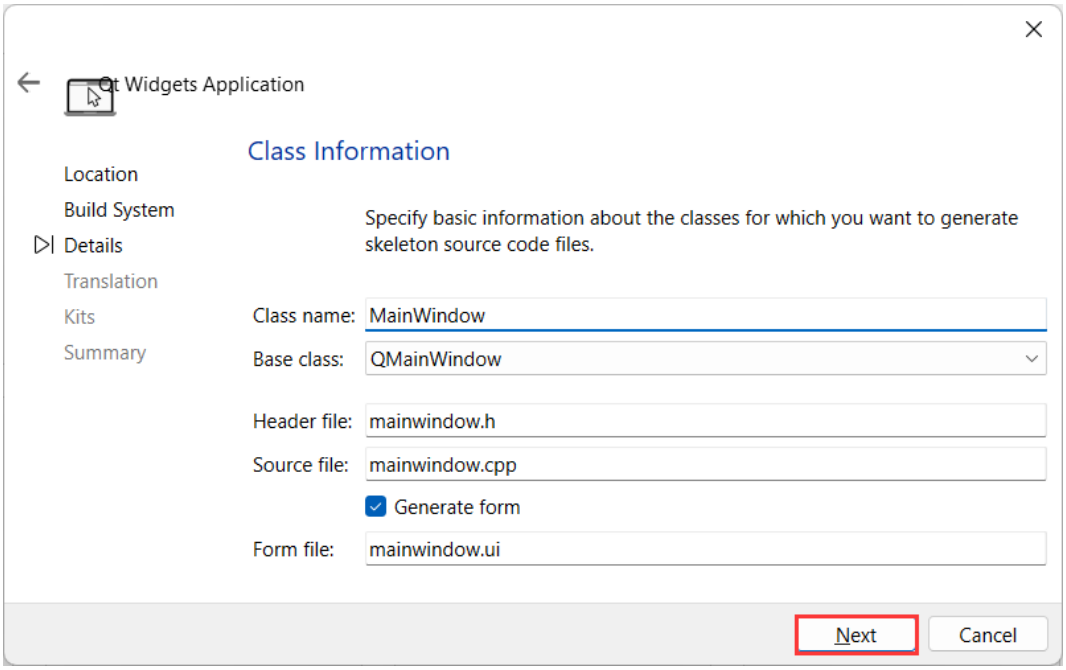
7. 选择好路径后点击下一步。



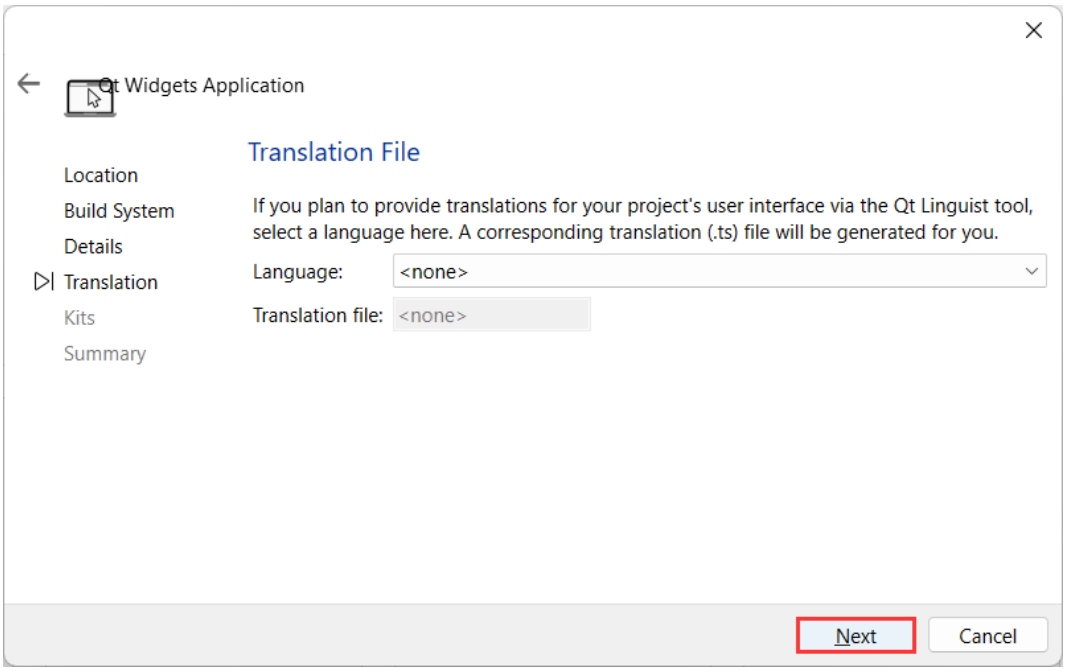
8. 选择 qmake，继续点击下一步。



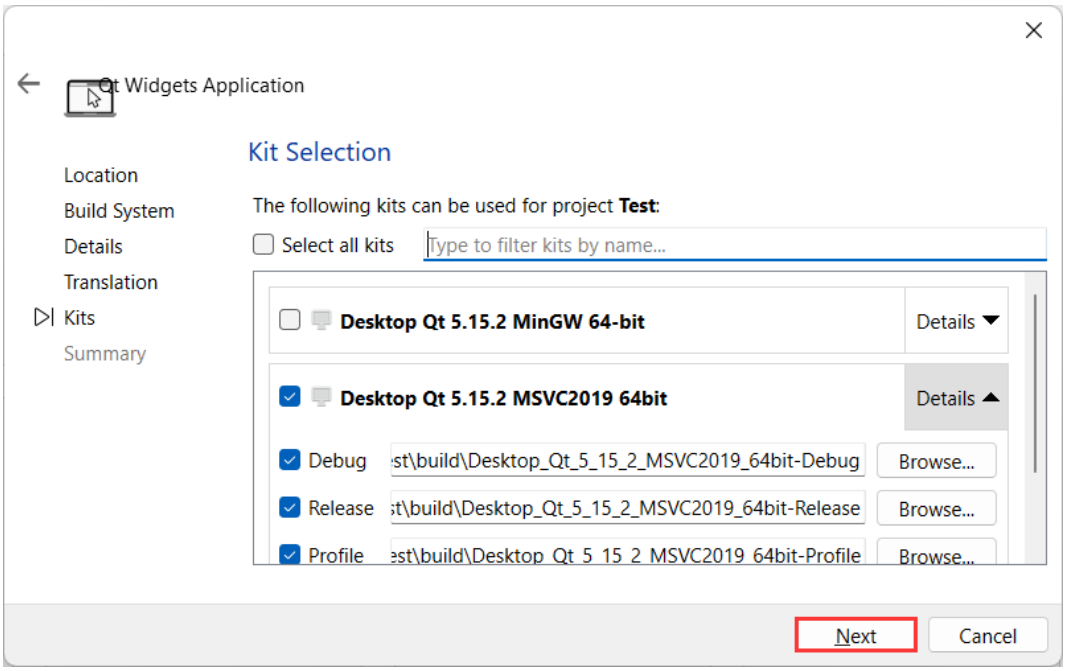
9. 继续点击下一步。



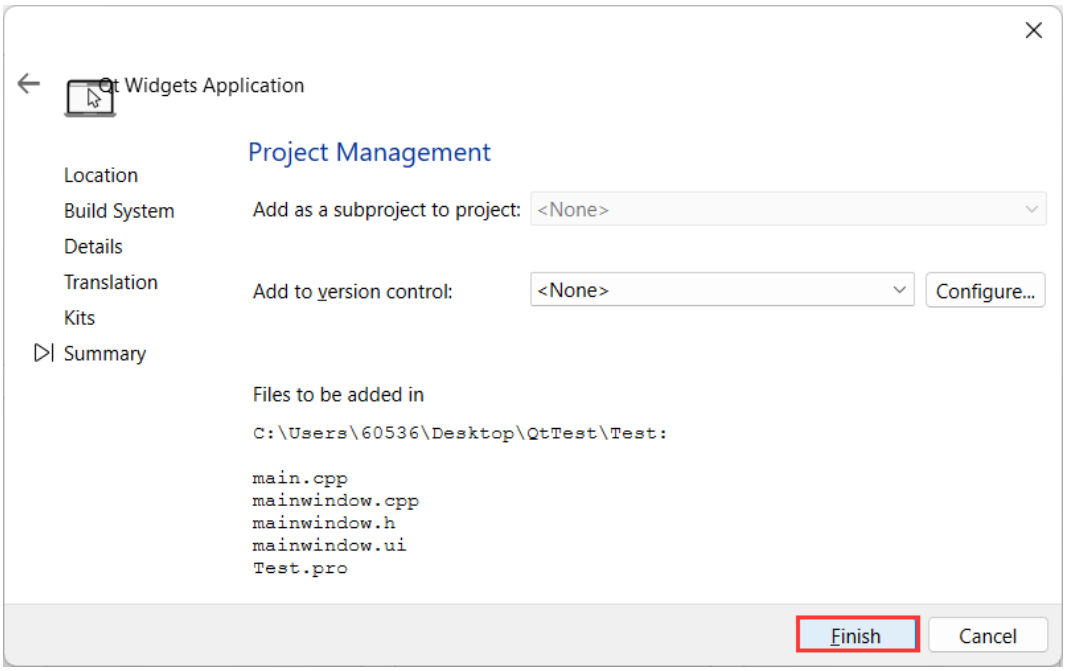
10. 继续点击下一步。



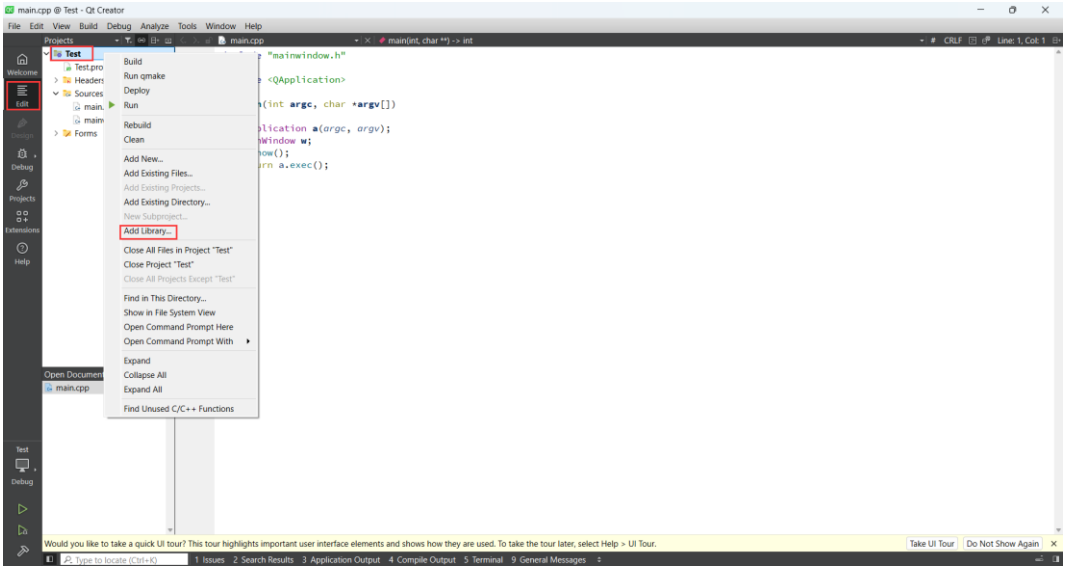
11. 为项目选择一个构建环境后继续点击下一步。



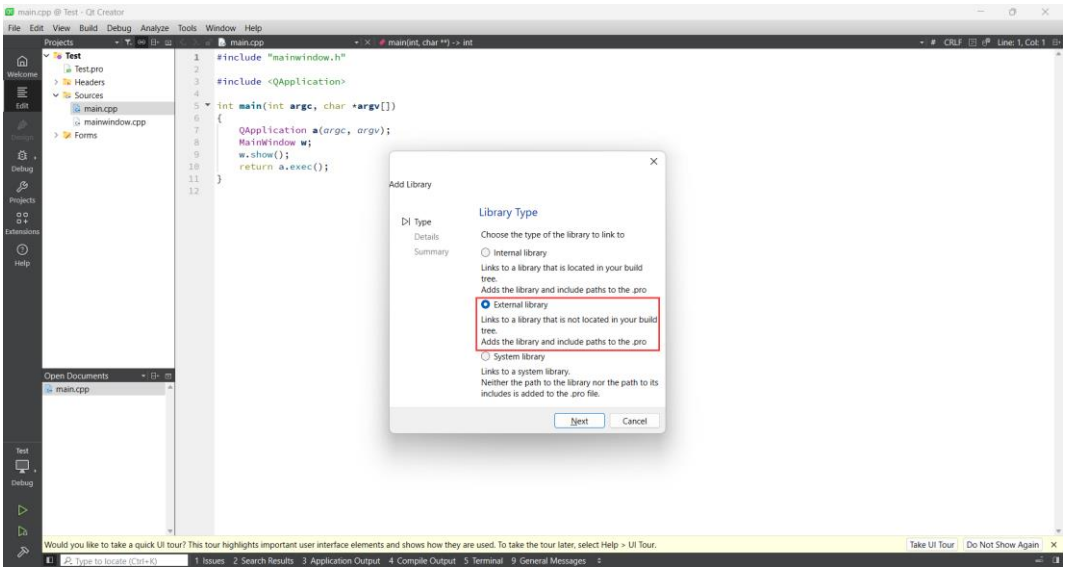
12. 点击完成以创建项目。



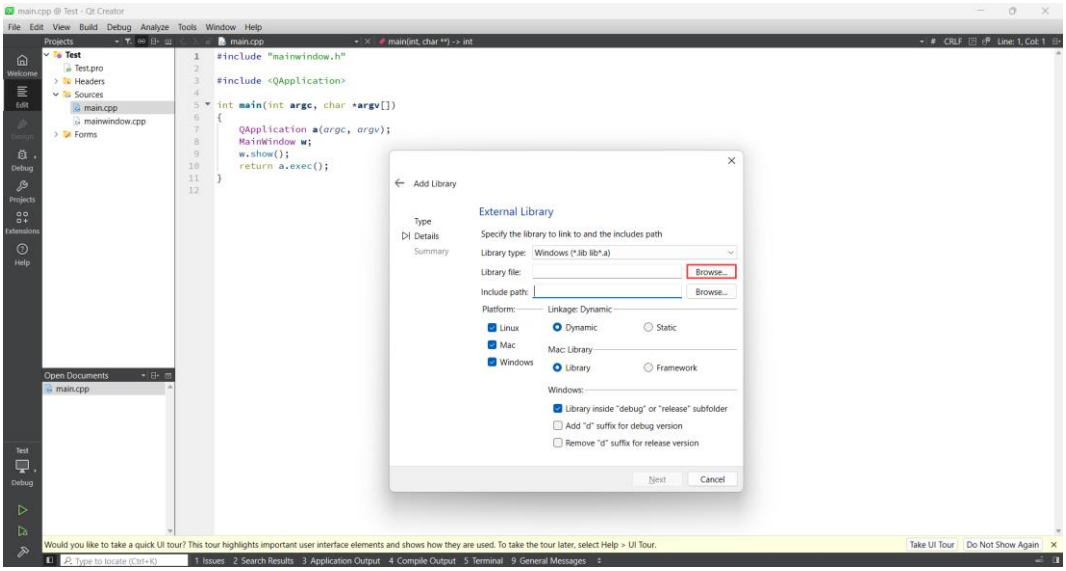
13. 点击编辑，右击 Test 项目，点击添加库。



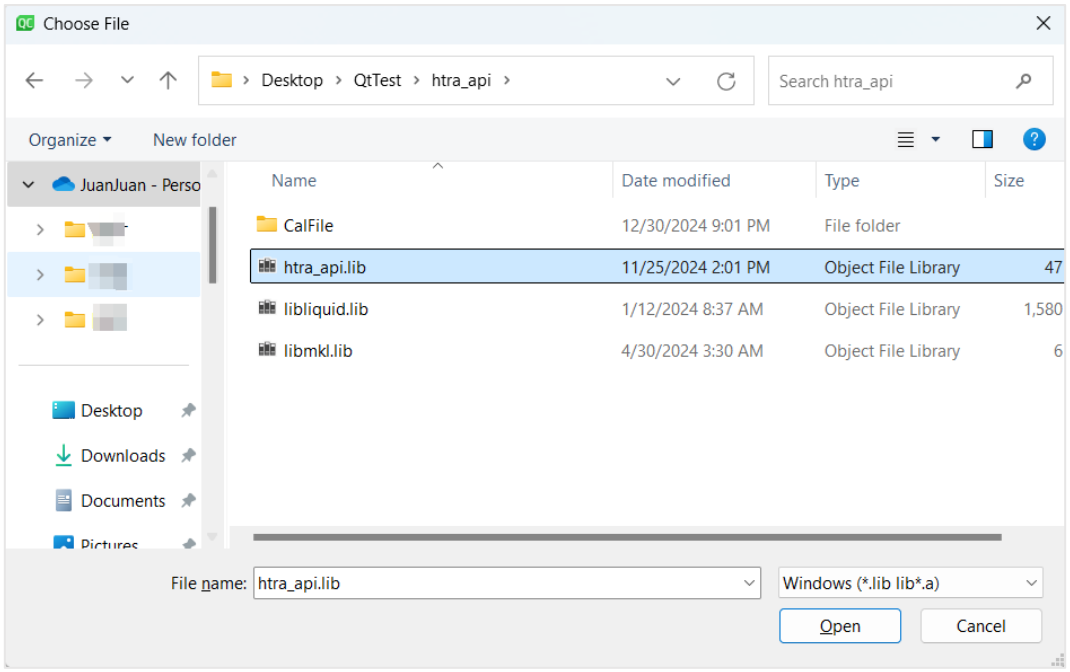
14. 选择外部库，点击下一步。



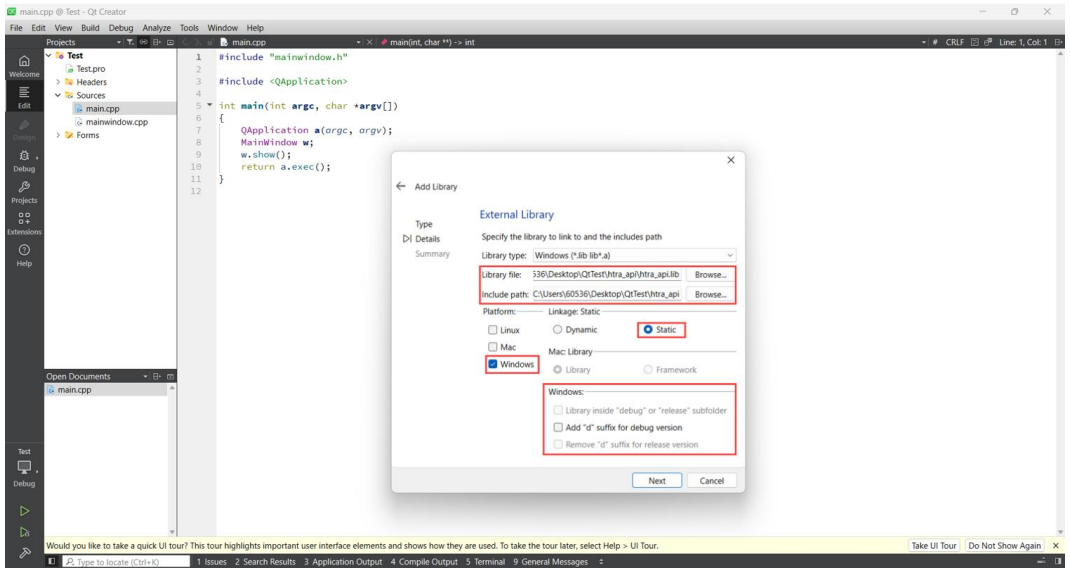
15. 点击浏览库文件。



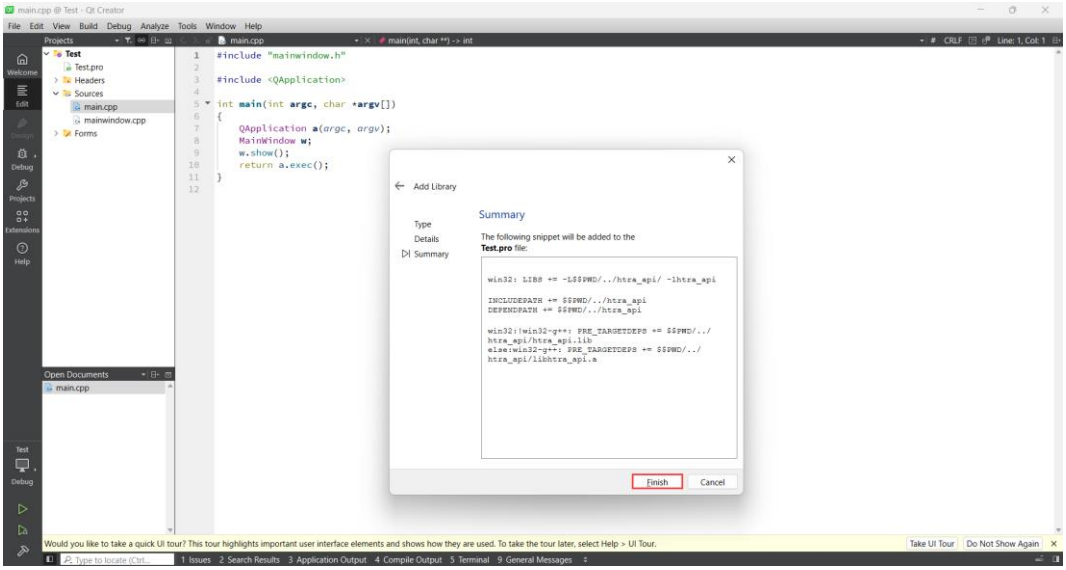
16. 选择 QtTest\htra_api 中的 htra_api.lib 库，点击 Open。



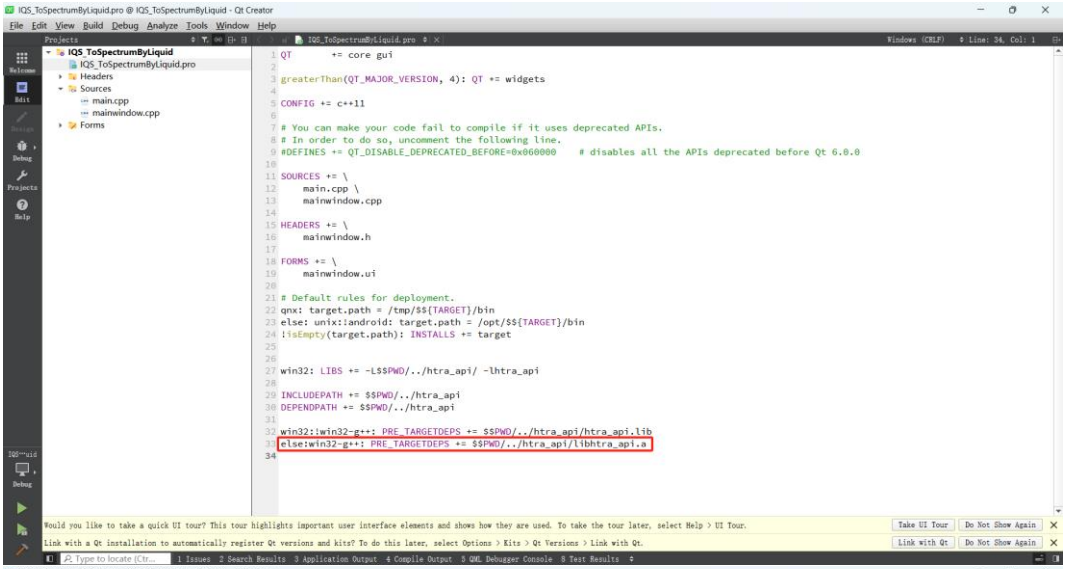
17. 首先取消勾选 Windows 内所有选项，然后点击静态库，最后选择 Windows 平台，点击下一步。



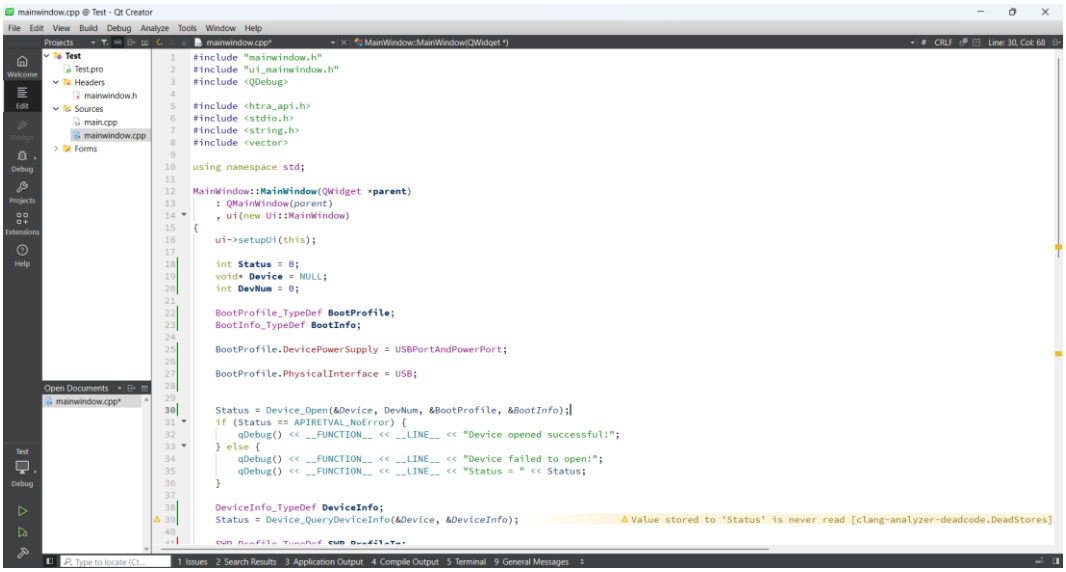
18. 点击完成以添加外部库。



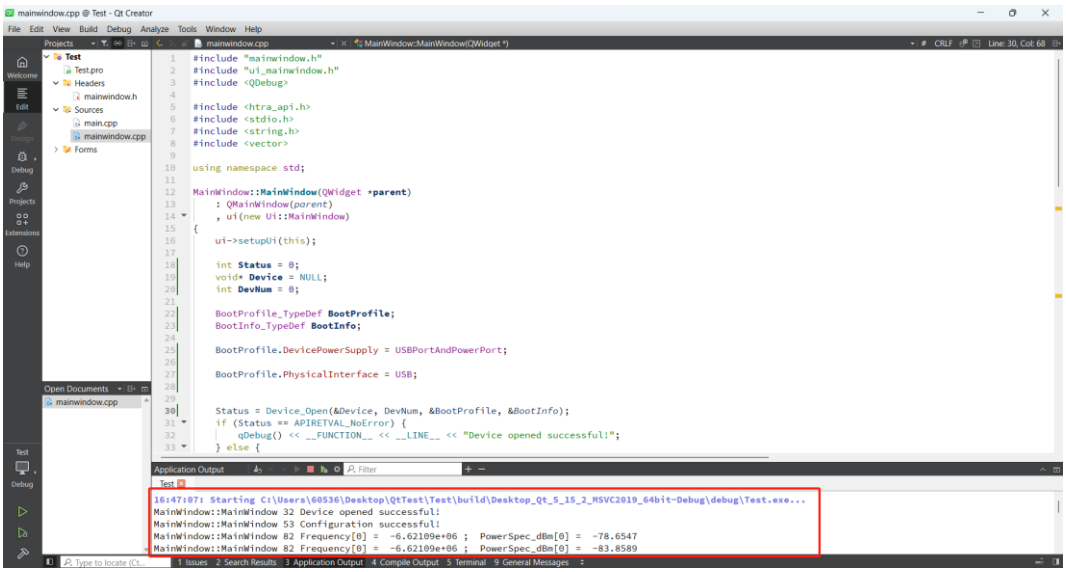
19. 删除最后一行“else：win32-g++：PRE_TARGTDEPS+=$$PWD/../htra_api/libhtra_api.a”整行代码。



20. 保存 Test.pro 文件，之后正常编写代码即可。



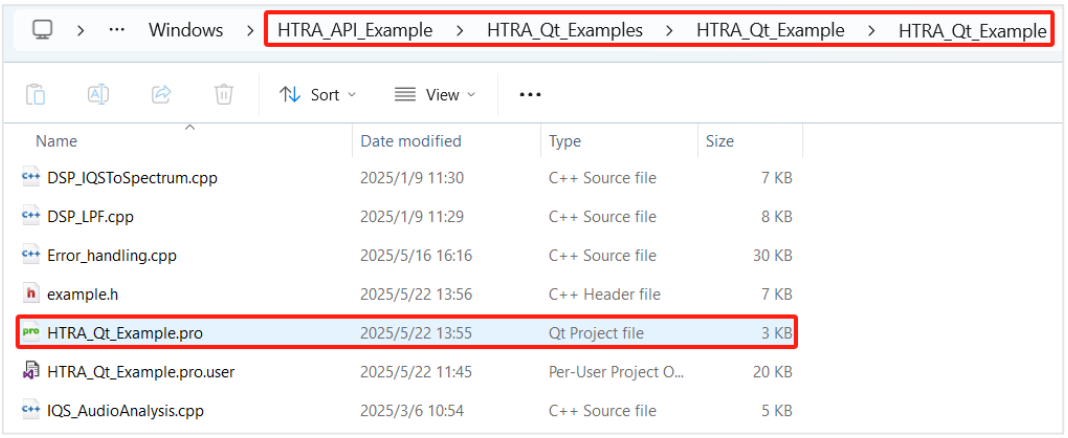
21. 编写完代码后，点击运行。如图所示正常使用设备。



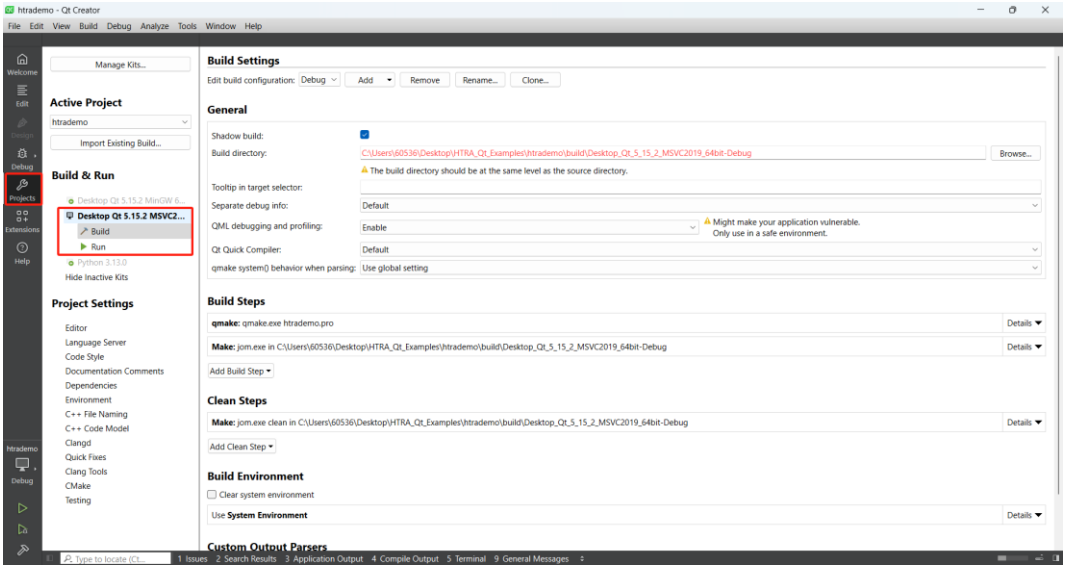
3.2 Qt 范例使用流程

随寄 U 盘中所有 Qt 范例使用流程一致。以 HTRA_Qt_Example 项目为例，具体如下：

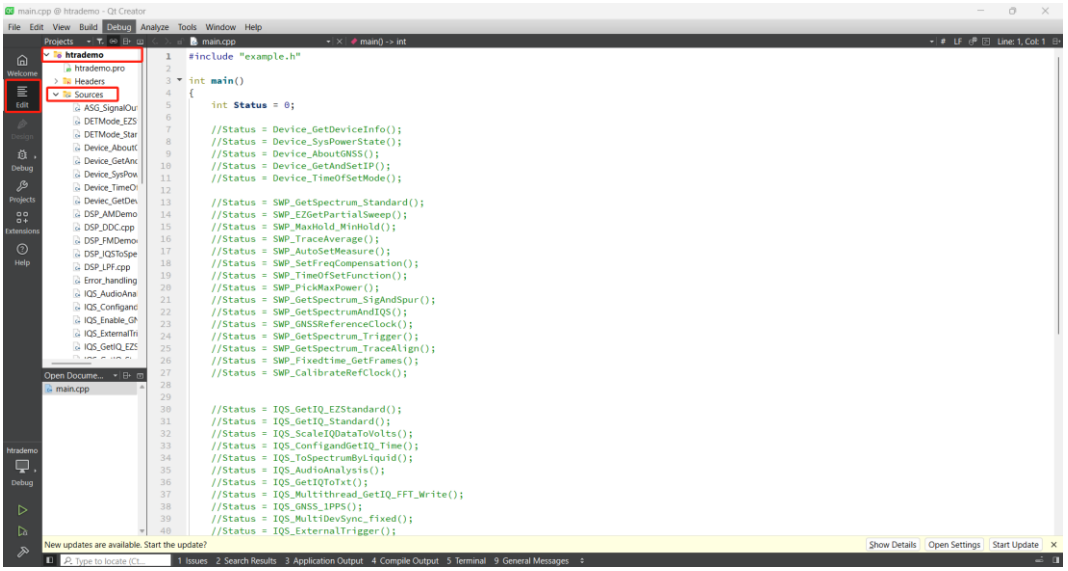
1. 如图所示，使用 Qtcreater 打开随寄 U 盘 Windows\HTRA_API_Example\HTRA_Qt_Examples\HTRA_Qt_Example\HTRA_Qt_Example 文件夹下 HTRA_Qt_Example.pro 文件（项目存放路径请勿包含中文路径！）。



2. 点击项目，为项目配置一个构建环境。

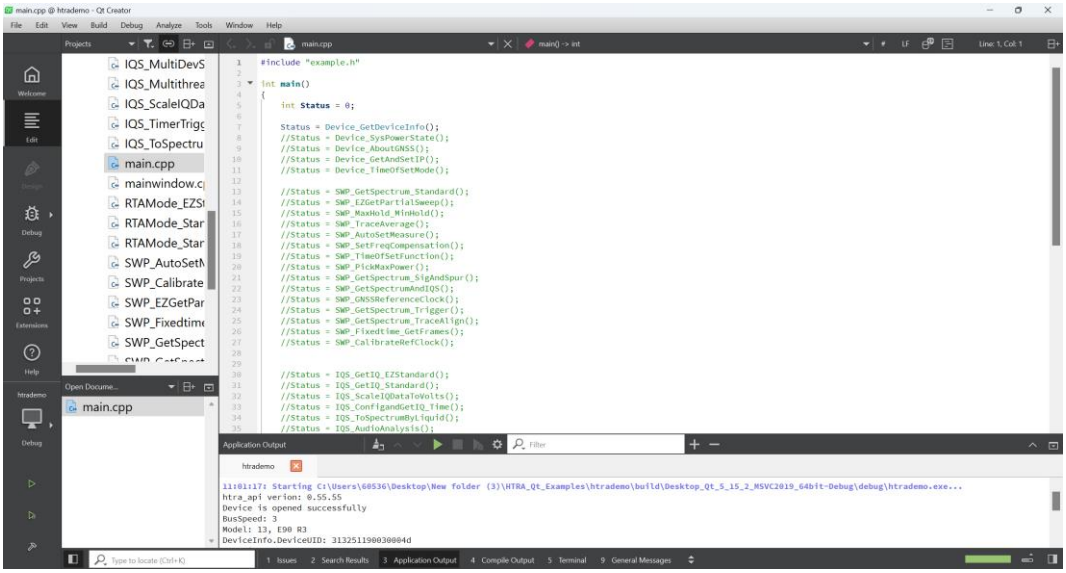


3. 配置完构建环境后，点击编辑，点击 htrademo 项目 Sources 文件夹下的 main.cpp。



4. 因为 Qt 随寄范例每个例程都封装在单独的函数中，所以使用范例时取消

注释即可（不可以同时使用多个范例）。例如测试使用 Device_GetDeviceInfo 例程时，取消注释并保存，点击运行，如图所示为正常运行设备。



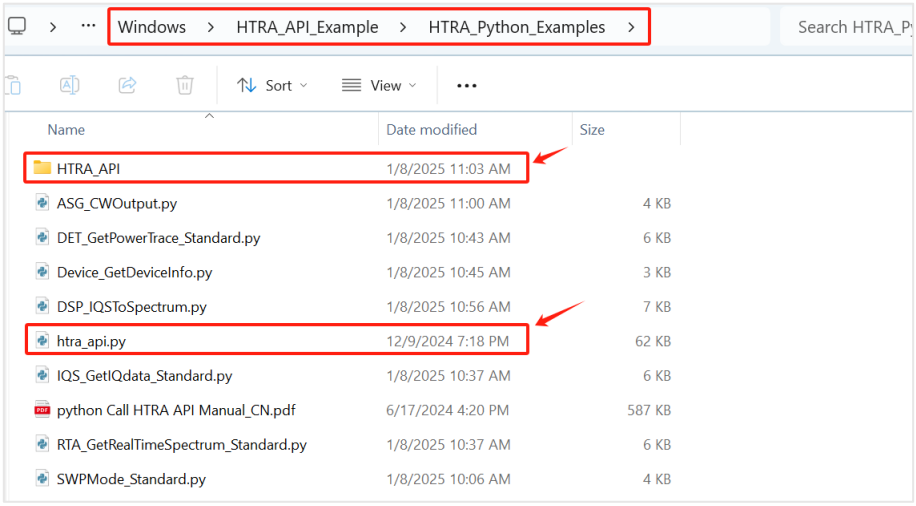
3.3 Qt 范例说明

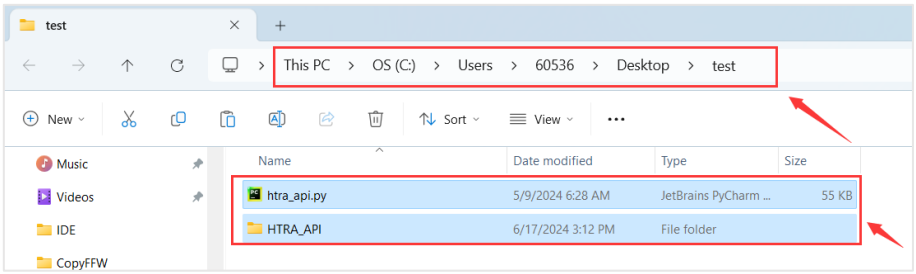
随寄 U 盘中 Qt 范例包含常规范例（HTRA_Qt_Example），FM 与 AM 解调范例（HTRA_Demodulation）以及使用 liquid 库实现 IQ 转频谱范例（IQS_ToSpectrumBuLiquid），具体范例作用参考 C/C++范例章节即可。

4. Python

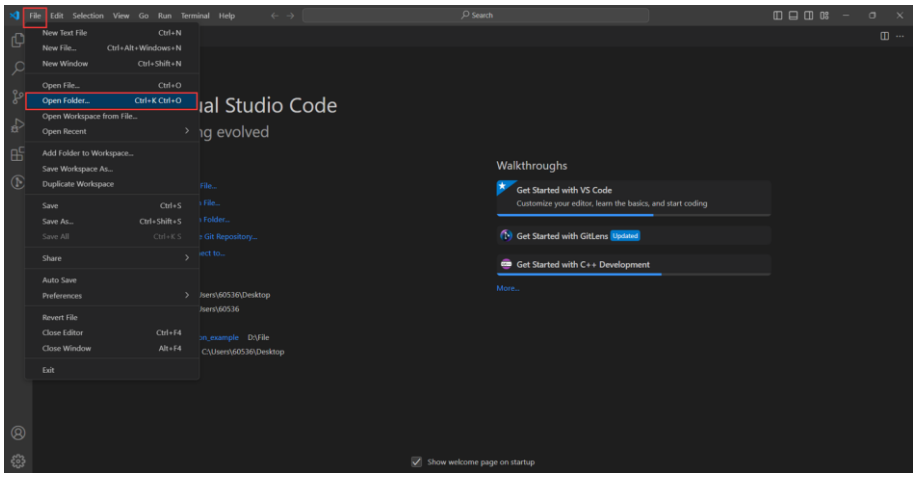
4.1 配置开发环境

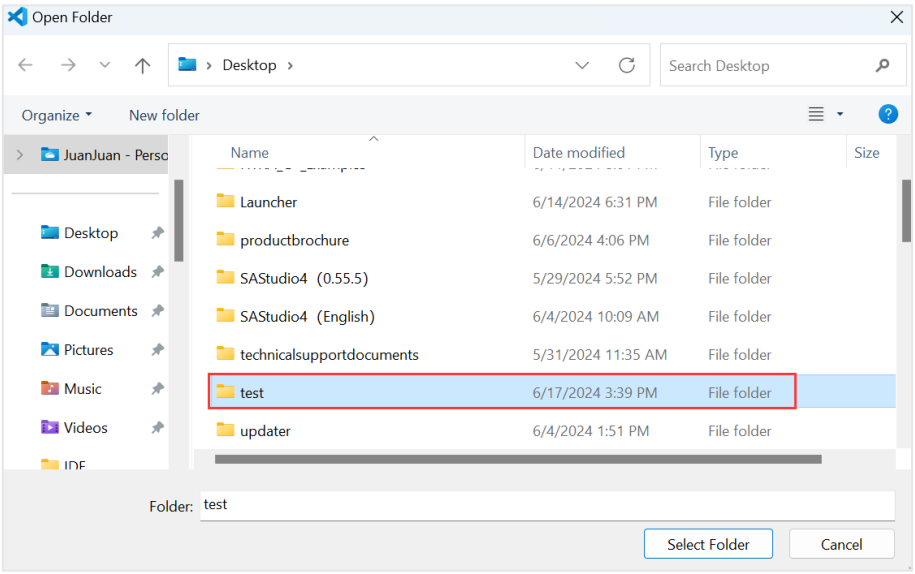
1. 在桌面创建文件夹，命名以 test 为例。打开 U 盘，将随寄 U 盘中“\Windows\HTRA_API_Example\HTRA_Python_Examples”下“HTRA_API”文件夹与“htra_api.py”文件复制至刚创建的文件夹下。



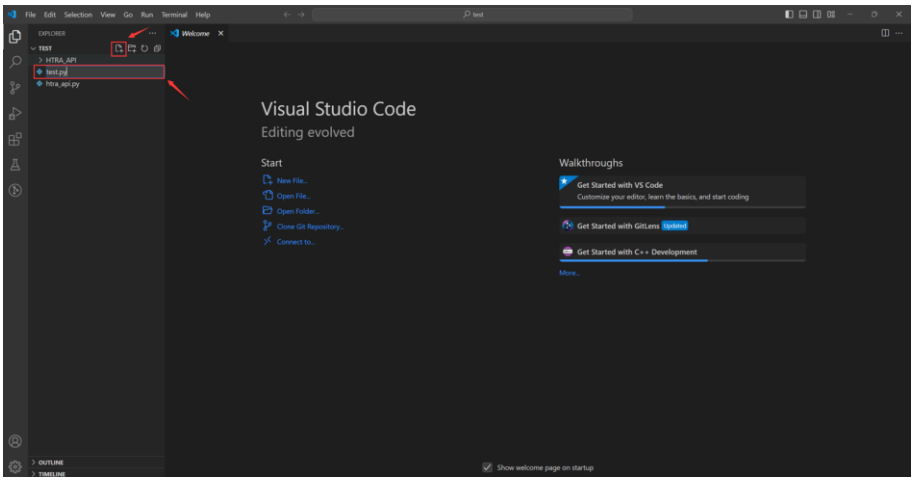


2. 打开 Visual Studio Code，点击 File 下 Open Folder，打开刚创建的文件夹。

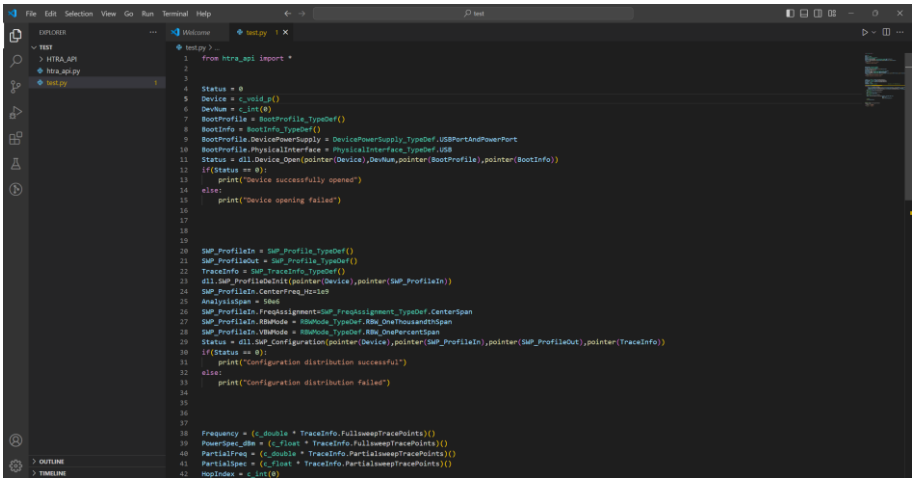




3. 创建新 python 文件。



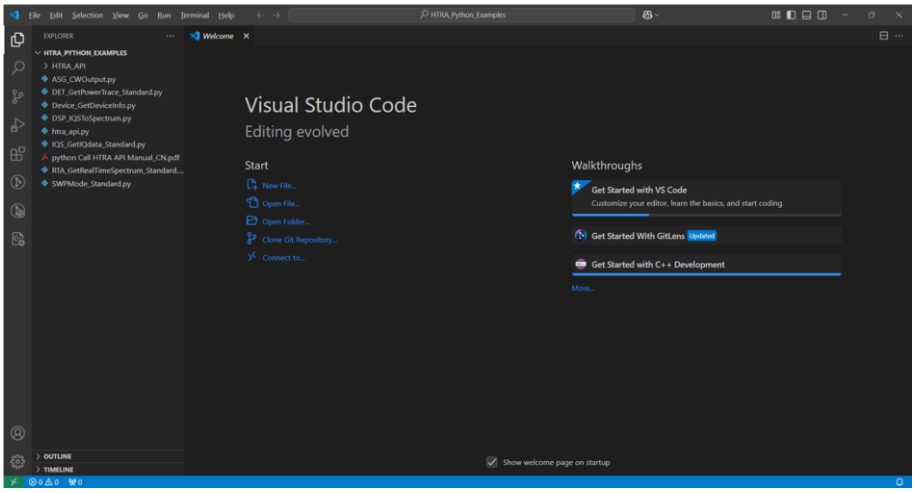
4. 正常编写代码。此处可参照随寄 U 盘中 Python 范例，即“\Windows\HTRA_API_Example\HTRA_Python_Examples”中项目。



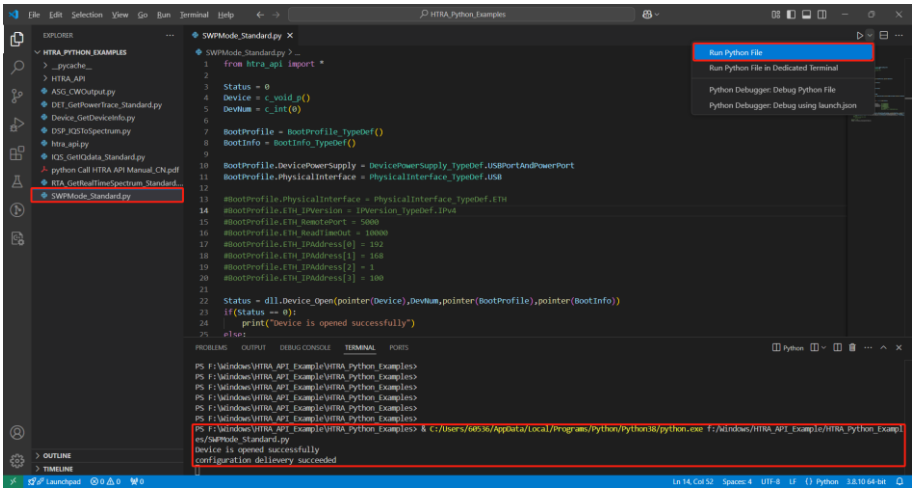
4.2 Python 范例使用流程

随寄 U 盘中 Python 范例使用流程如下：

1. 如图所示，使用 vscode 或其他编译器打开随寄 U 盘 Windows\HTRA_API_Example\HTRA_Python_Examples 整个项目。项目中 htra_api.py 文件为动态链接库在 Python 中的映射文件，其他文件为示例范例（范例作用在后续章节描述）。



2. 选择任意一个示例程序，为其配置 Python 环境，直接运行程序即可。例如使用 SWPMode_Standard.py 例程时，如图所示为正常运行设备。



4.3 Python 范例说明

4.3.1 获取设备信息

Device_GetDeviceInfo.py：获取包括 API 版本、USB 版本、设备型号、设备 UID、MCU 版本、FPGA 版本以及设备温度在内的各种设备信息。

4.3.2 获取标准频谱数据

SWP_GetSpectrum_Standard.py：获取指定频段内完整频谱数据（若上位机包含 matplotlib 库则绘制频谱图）。

4.3.3 获取固定点数或时长的 IQ 数据

IQS_GetIQdata_Standard.py：在 IQS 模式的不同触发模式下获取 IQ 数据。

4.3.4 获取固定点数或时长的检波数据

DET_GetPowerTrace_Standard.py：在 DET 模式的不同触发模式下获取功率检波数据（若上位机包含 matplotlib 库则绘制时间功率图）。

4.3.5 获取固定点数或时长的实时频谱数据

RTA_GetRealTimeSpectrum_Standard.py：在 RTA 模式的不同触发模式下获取实时频谱数据（若上位机包含 matplotlib 库则绘制实时频谱图）。

4.3.6 IQ 转频谱数据

DSP_IQSToSpectrum.py：将 IQS 模式下获取到的 IQ 数据转换成为频谱数据（若上位机包含 matplotlib 库则绘制频谱图与 IQ 时域图）。

4.3.7 GNSS 相关

Device_AboutGNSS.py：获取 GNSS 模块获取的经纬度、海拔和时间等信息，获取 IQS 模式下 MeasAuxInfo 结构体中的经纬度和时间信息。

5. Matlab

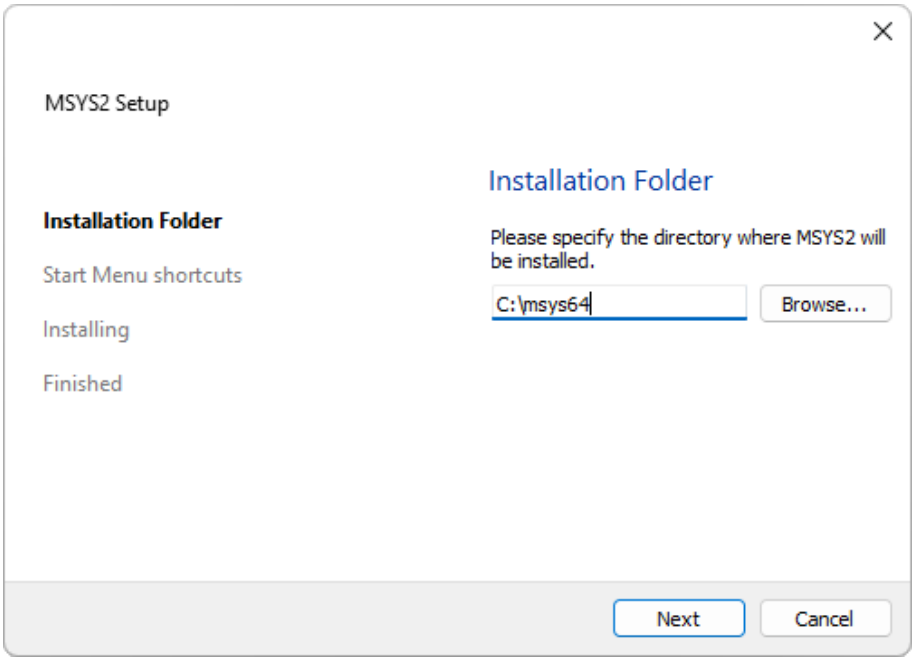
5.1 配置开发环境

32 位的 htra_api 调用方法与 64 位基本相同，所以以下教程以 Matlab2016a 为例说明如何调用 64 位 htra_api。

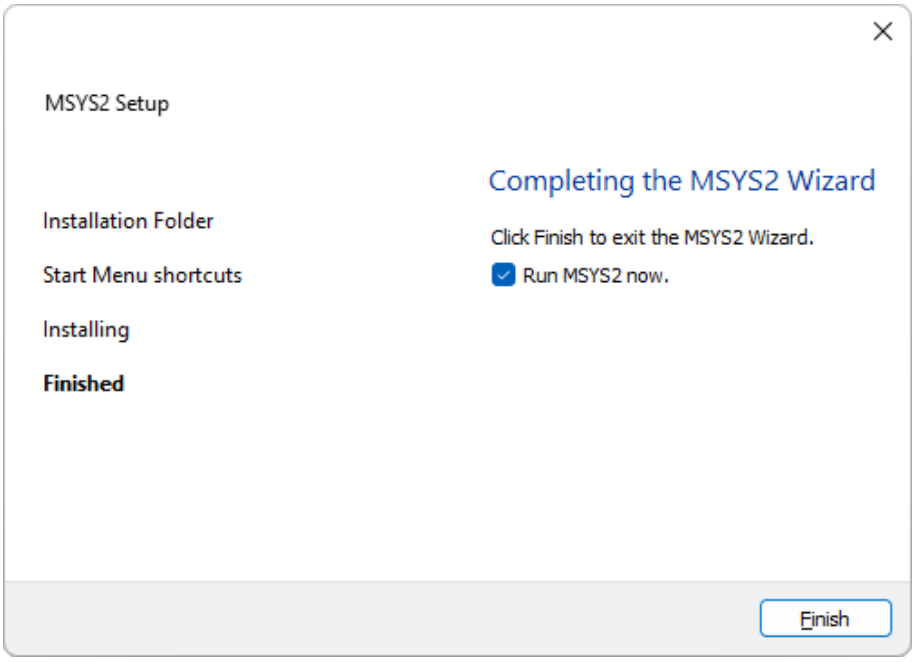
5.1.1 安装 MSYS2

下载和安装链接： https://www.msys2.org/

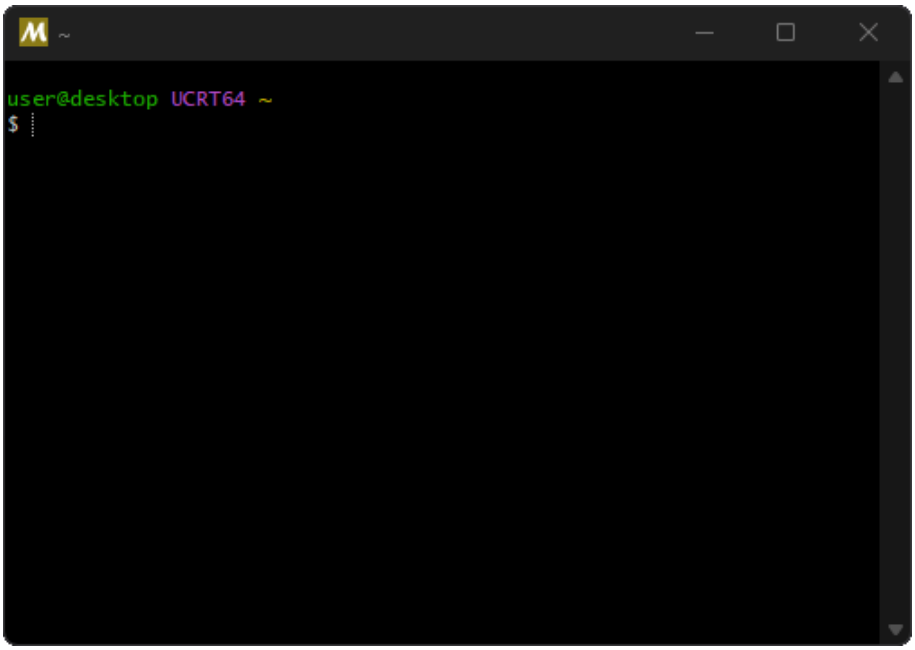
1. 下载安装程序 MSYS2-x86_64-20231026.exe
2. 运行安装程序。MSYS2 需要 64 bit Windows 8.1 及以上版本。
3. 默认为 C:\msys64，可按需选择安装路径。



4. 完成后，单击 Finish。

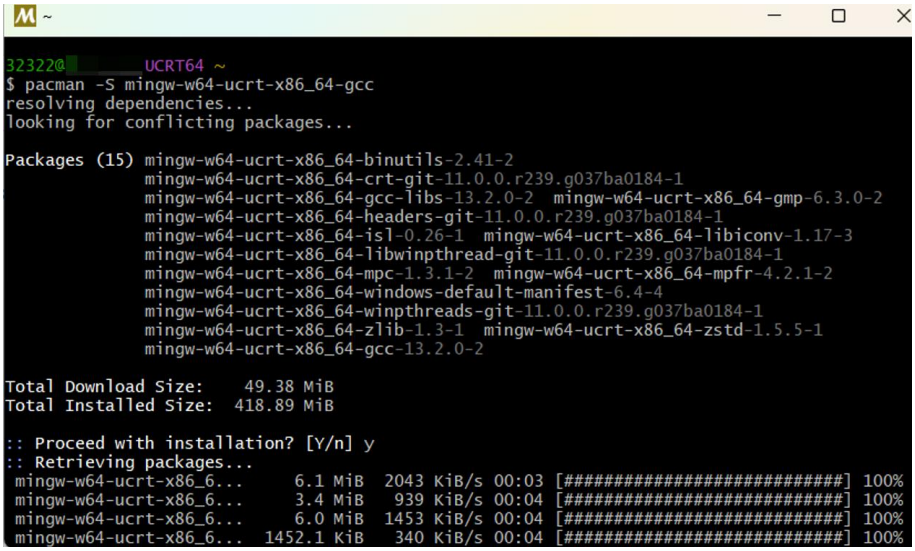


5. 现 MSYS2 已准备就绪，UCRT64 环境的终端启动。

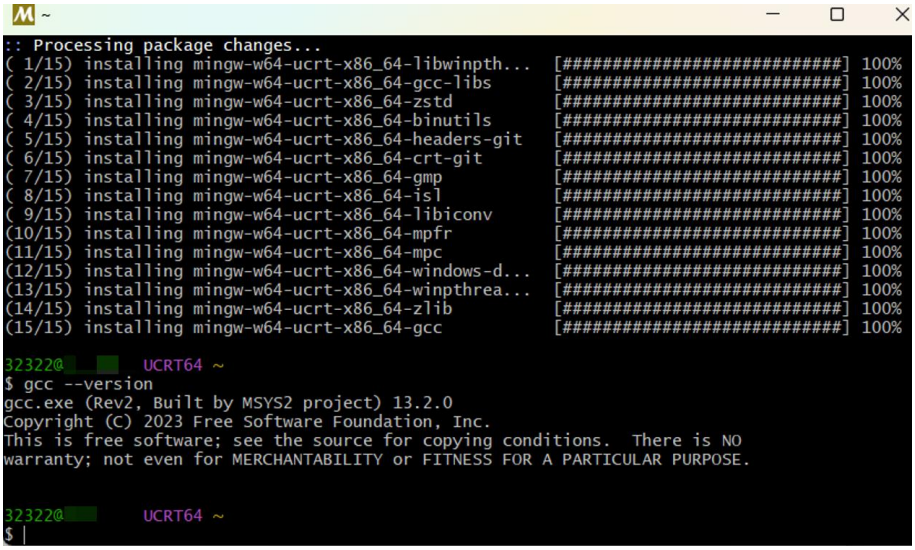


6. 安装 GCC 工具，输入命令：pacman -S mingw-w64-ucrt-x86_64-gcc

7. 终端窗口将显示如下输出。按“Enter”继续



8. 输入命令 gcc --version 查看 GCC 的版本信息



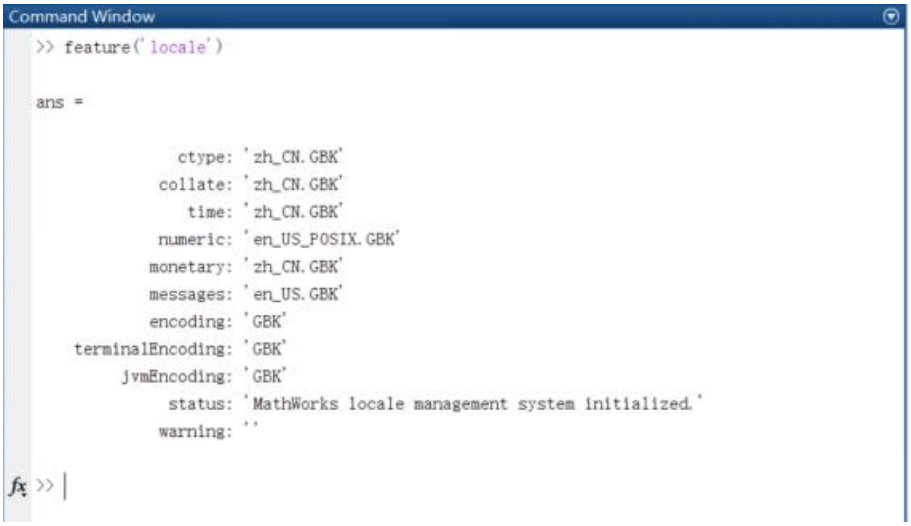
5.1.2 配置 Matlab

1. 解决 Matlab 2016a 打开.m 文件出现中文乱码情况。

注意：若您使用的 Matlab 版本高于 2019a，请忽略此步骤。

（1） 查看当前编码格式：

在 Matlab 命令行中输入：feature('locale')



从图中可以看出，编码格式为 GBK。

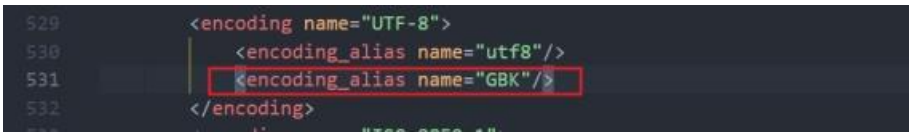
（2） 右击 Matlab2016a 的快捷方式选择“打开文件所在的位置”，打开 Matlab.exe 所在文件夹。

（3） 在步骤（2）所示的文件夹中，找到 lcdata.xml 和 lcdata_utf8.xml 两个文件，将 lcdata.xml 重命名为 lcdata_old.xml，即备份原 lcdata.xml。

（4） 复制一份 lcdata_utf8.xml 放至同级文件夹中，并将新复制的文件 lcdata_utf8.xml 重命名为 lcdata.xml。

（5） 打开 lcdata.xml，删去下图所示的 GBK 相关代码。

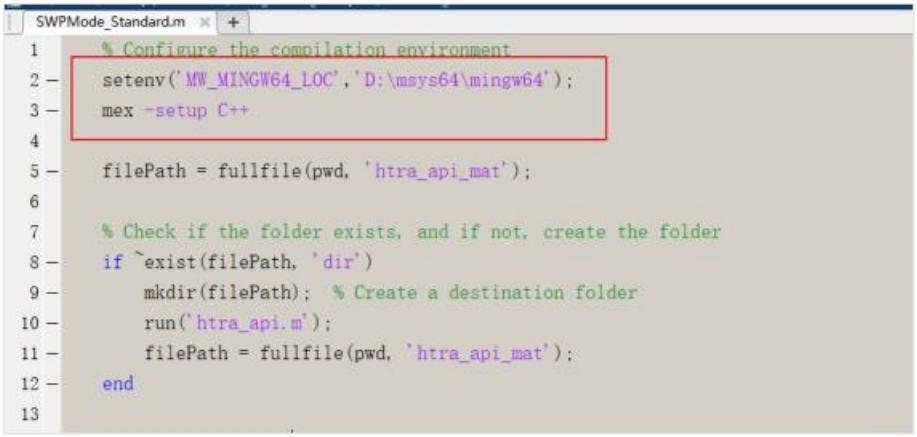
（6） 找到“UTF-8”部分，将图中标注行的代码添加至图中对应位置，保存 lcdata.xml 后关闭文件。



（7） 重启 Matlab 后，乱码的中文恢复正常。

2. 配置编译环境方法一：在脚本中配置

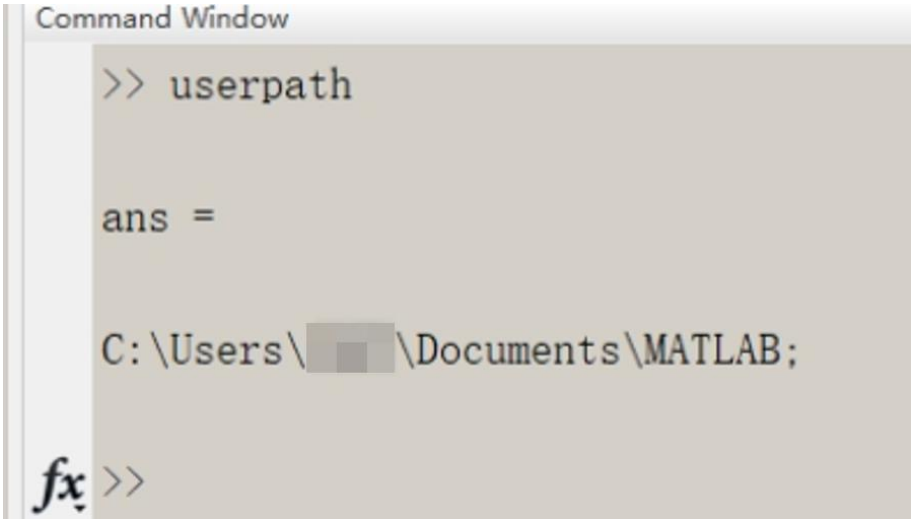
在脚本运行时，运行 setenv('MW_MINGW64_LOC','D:\msys64\ucrt64')和 mex -setup C++命令即可配置编译语言为 C++的编译环境。



注意：D:\msys64\ucrt64 为编译环境所在文件夹，请检查此地址下的 bin 文件夹中是否有 c++.exe、g++.exe 和 gcc.exe 等文件，若有，则此地址即为编译环境地址，若没有，请查找正确的编译环境所在地址。

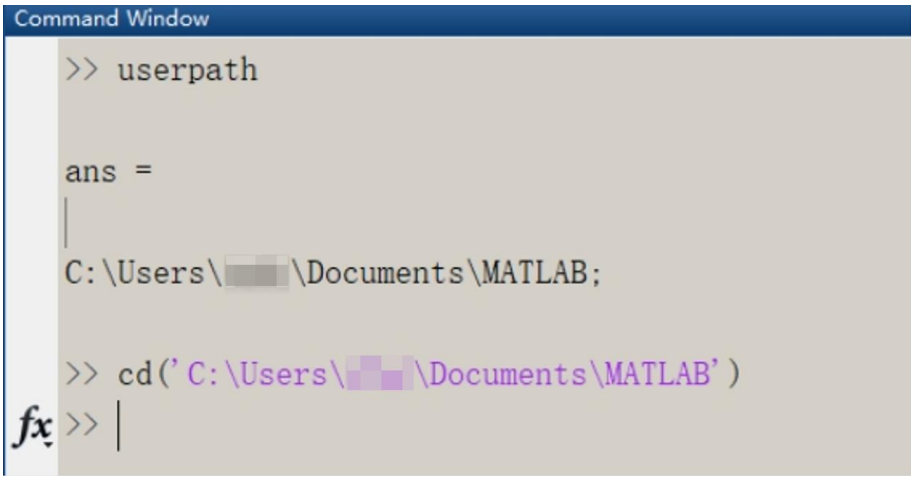
3. 配置编译环境方法二：startup.m 文件修改

（1） Matlab 终端输入：userpath，命令行窗口输出类似结果：C:\Users\YourUsername\Documents\MATLAB

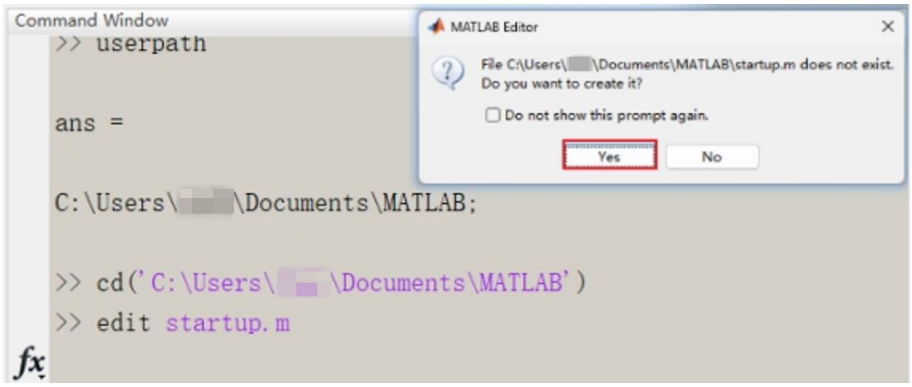


（2） 在 C:\Users\YourUsername\Documents\MATLAB 地址下检查 startup.m 文件是否存在，若不存在，则在此地址下创建一个新的 startup.m 文件。创建步骤如下：

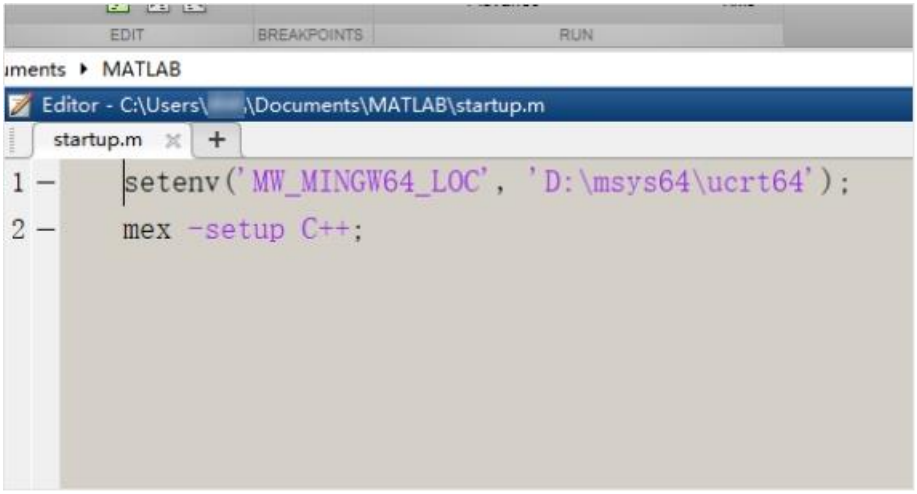
1) Matlab 终端输入：cd('C:\Users\YourUsername\Documents\MATLAB')，切换工作目录至 C:\Users\YourUsername\Documents\MATLAB。



2) Matlab 终端输入：edit startup.m，在弹窗中选择“Yes”，创建 startup.m 文件。



3) 在 startup.m 文件中分别添加命令 setenv('MW_MINGW64_LOC', 'D:\msys64\ucrt64');和 mex -setup C++。



4) startup.m 文件编辑完成后，保存并关闭文件。

5) 重启 Matlab，观察命令行窗口如下图所示即为配置完成。

5.1.3 调用 htra_api.dll 说明

1. loadlibrary

loadlibrary 函数可加载动态链接库。loadlibrary('.\htra_api\htra_api.dll','.\htra_api\htra_api.h');加载 htra_api.dll 和 htra_api.h，请保证.dll 和.h 的文件路径正确。

```
%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');用于查看htra_api.dll中所有可用的函数。

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

3. libpointer

libpointer 允许创建 Matlab 的数据类型指针，并将其传递给外部库函数。

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

4. libstruct

libstruct 用于在 Matlab 中定义结构体类型，并将其传递给外部库函数。

```
%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

get 函数用于获取结构体的属性值。

```
%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 是 Matlab 中用于调用htra_api.dll中的函数的命令。

```
%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用于加载htra_api.m文件中生成的 .mat 结构体文件。

```
% 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 是 Matlab 中用于生成完整文件路径的函数。加载filePath下的BootProfile.mat和BootInfo.mat文件。

```
% 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 是用来卸载一个已经加载的htra_api库，与loadlibrary前后成对出现。

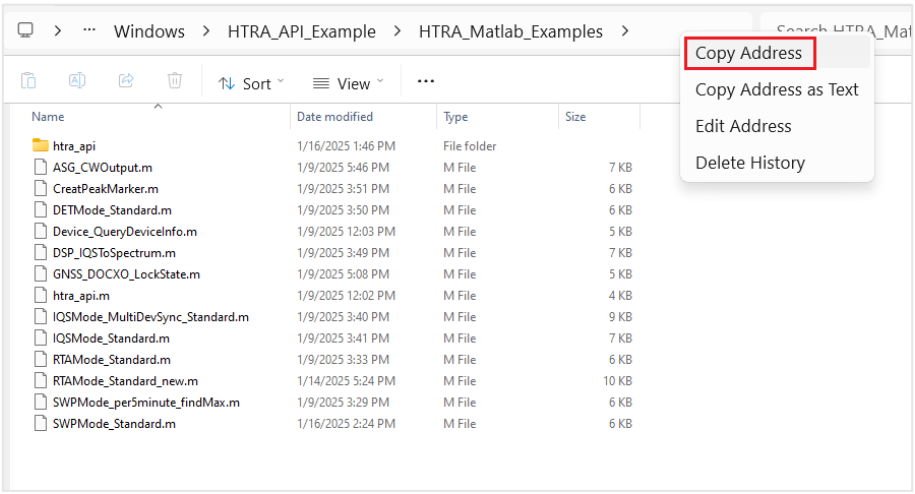
```
%Unload library file
unloadlibrary('htra_api');
disp('Uninstall complete')
```

5.2 Matlab 范例使用流程

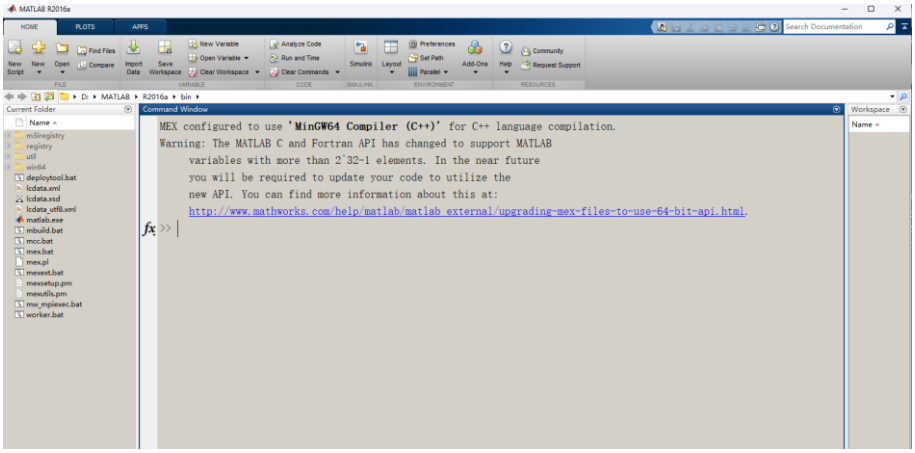
随寄 U 盘中 Matlab 范例使用流程如下：

1. 打开随寄 U 盘 Windows\HTRA_API_Example\HTRA_Matlab_Examples 文件夹，双击任意.m 文件即可打开范例，如何运行范例请直接参考步骤 4。

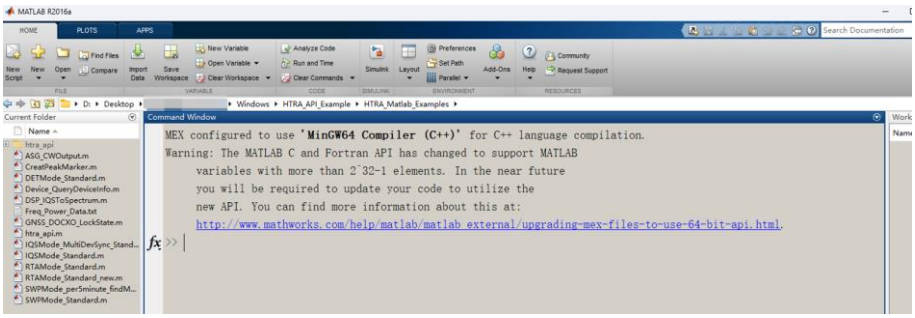
2. 若步骤 1 打不开范例，请继续此步骤，复制 Windows\HTRA_API_Example\HTRA_Matlab_Examples 地址。



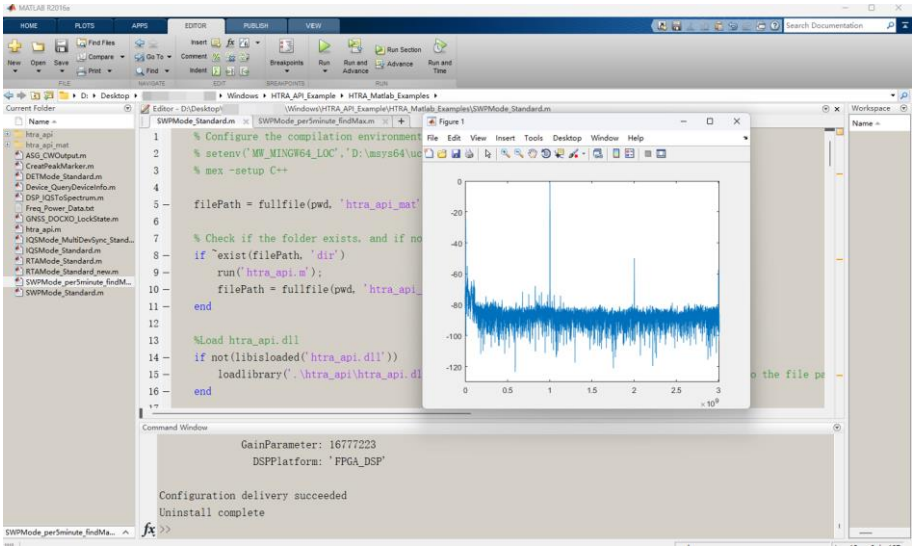
3. 打开系统已安装 Matlab 软件。



4. 将复制的地址粘贴至文件地址框后，按下回车，跳转至随寄资料的\Windows\HTRA_API_Example\HTRA_Matlab_Examples 文件夹中。



5. 按需点击左侧的.m 文件，点击“Run”，待出现 Figure1 窗口表示成功运行范例，各范例的功能说明，请参考 5.2Matlab 范例说明 章节。



5.3 随寄范例简介

5.3.1 获取设备信息

Device_QueryDeviceInfo.m：获取设备信息，包括 API 版本、USB 版本、设备型号、设备 UID、MCU 版本、FPGA 版本和设备温度。

5.3.2 获取标准频谱数据

SWPMode_Standard.m：获取指定频段内完整频谱数据。

5.3.3 创建多个游标，显示游标的频率和功率

CreatPeakMarker.m：获取指定频段内的频谱数据并创建游标并进行寻峰。

5.3.4 每五分钟采集一次频谱的峰值

SWPMode_per5minute_findMax.m：获取指定频段内的频谱数据，每五分钟在全局范围内寻找一次峰值。

5.3.5 获取连续流或固定点数的 IQ 数据

IQSMode_Standard.m：在 IQS 模式的不同触发模式下获取 IQ 数据。

5.3.6 将获取的 IQ 数据转为频谱数据

DSP_IQSToSpectrum.m：获取 IQ 数据后，将获取的 IQ 数据转为频谱数据。

5.3.7 获取连续流或固定点数的检波数据

DET_GetPowerTrace_FixedPoints.m：在 DET 模式的不同触发模式下获取功率检波数据。

5.3.8 获取连续流或固定时长的实时频谱数据

RTAMode_FixedPoints.m：在 RTA 模式的不同触发模式下获取实时频谱数据。

5.3.9 内部信号源输出信号

ASG_CWOutput.m：输出单音信号、频率扫描信号和功率扫描信号。仅适用于有信号源选件的设备使用。

5.3.10 锁定 GNSS 天线和 DOCXO 晶振

GNSS_DOCXO_LockState.m：调用 API 接口锁定 GNSS 天线和 DOCXO 晶振，仅适用于有 IO 拓展板的选件设备使用。

5.3.11 多机同步

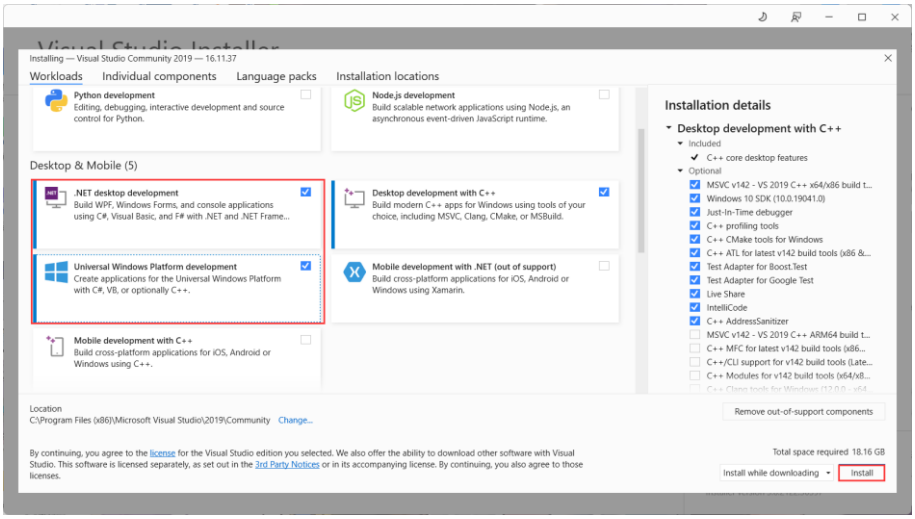
IQSMode_MultiDevSync_Standard.m：在同参考时钟源输入和同触发源输入时，两台设备同时获取 IQ 数据，并可查看其采集数据的同步性。

6. C#

6.1配置开发环境

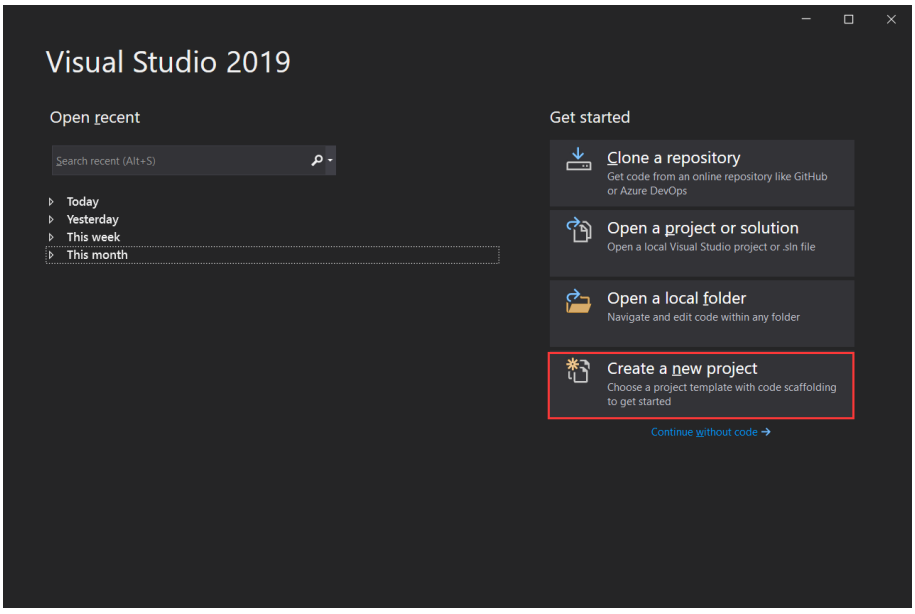
6.1.1 开发环境确认

打开 Visual Studio Installer，勾选.NET 桌面开发组件与通用 Windows 平台开发组件并点击修改，以保证 Visual Studio 2019 具有 C#开发环境。

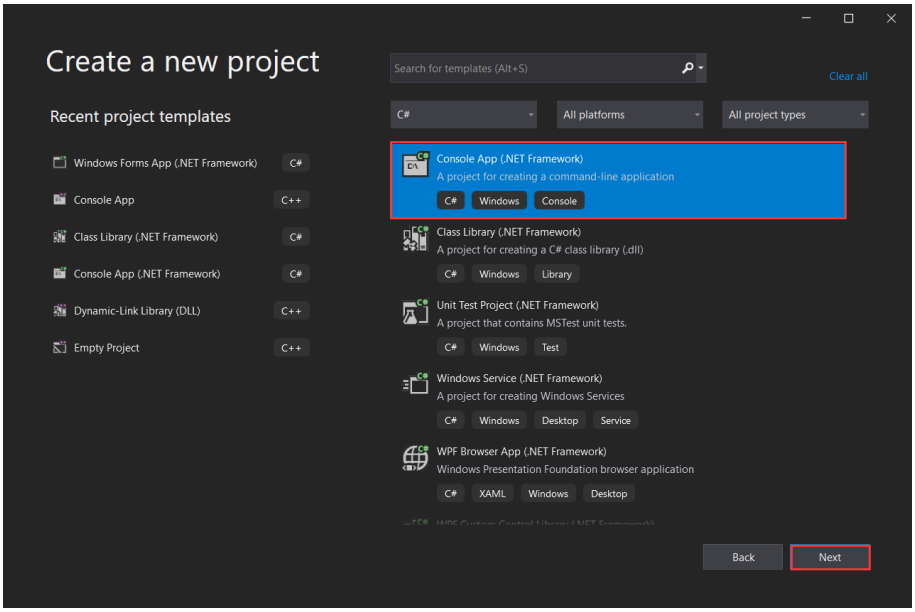


6.1.2 项目搭建

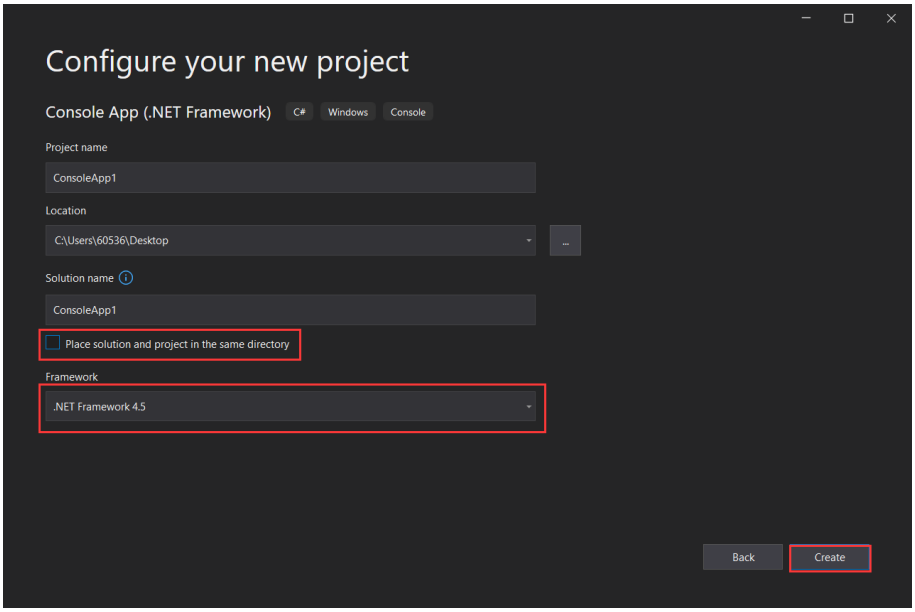
1. 打开 Visual Studio 2019，点击创建新项目。



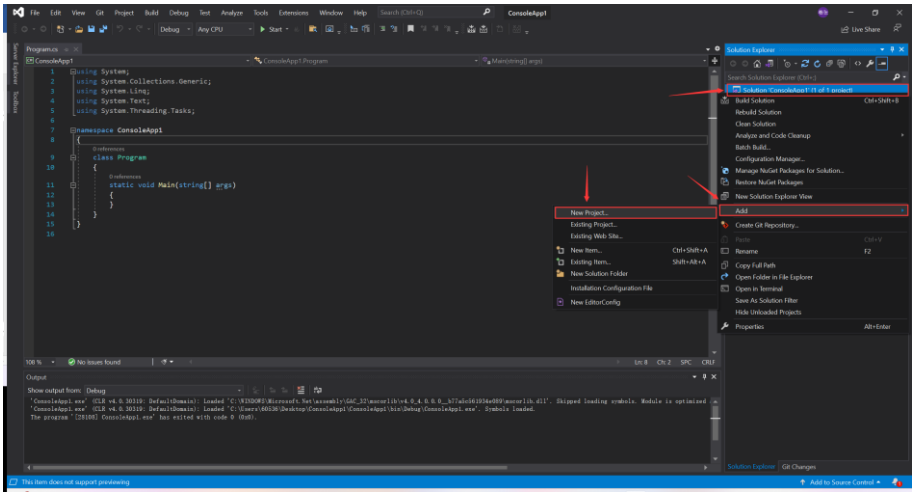
2. 选择 C#控制台应用，点击下一步。



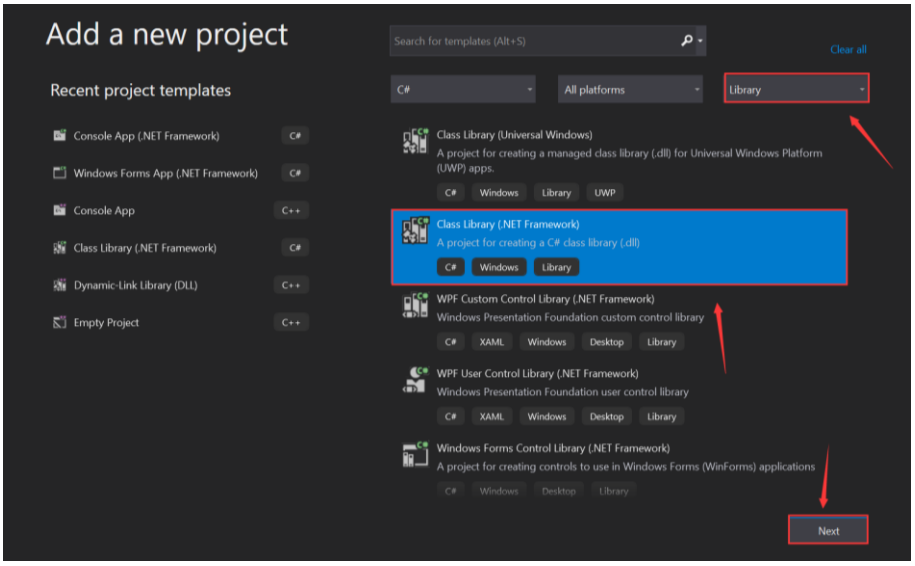
3. 填写项目名称与存放地址，取消勾选将解决方案和项目放在同一目录中。框架选择.NET Framework 4.5，最后点击创建。



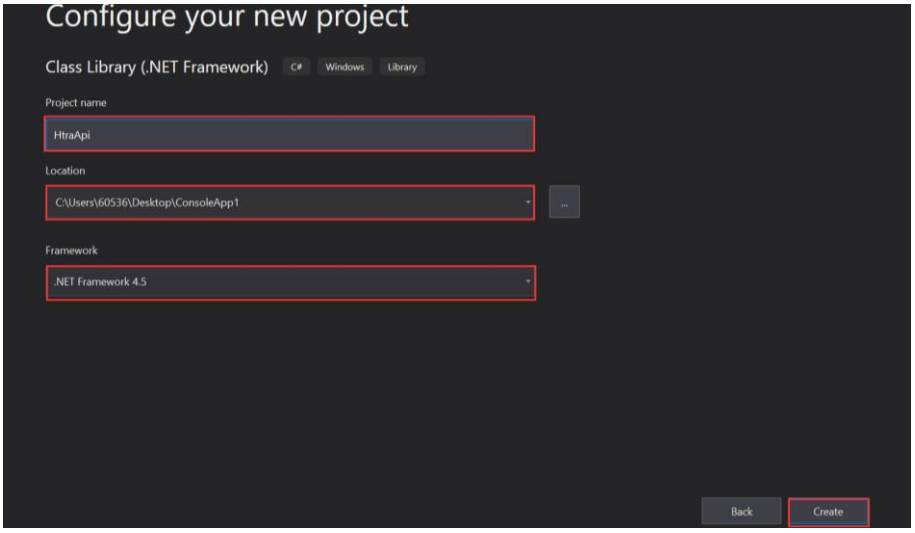
4. 创建完成后，打开项目，右击解决方案，选择新建项目。

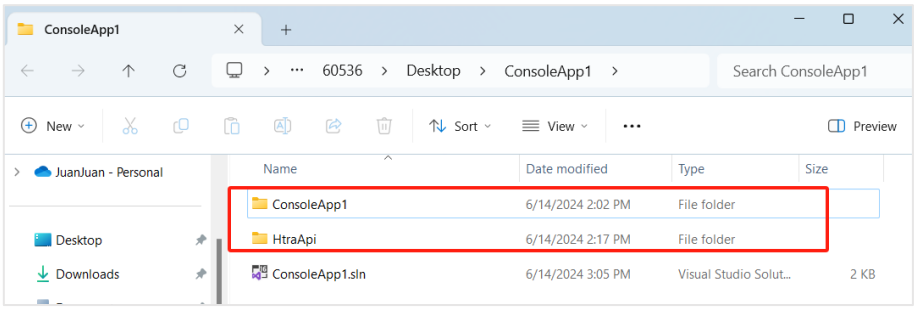


5. 项目类型选择库类型中的类库（.NET Framework），点击下一步。

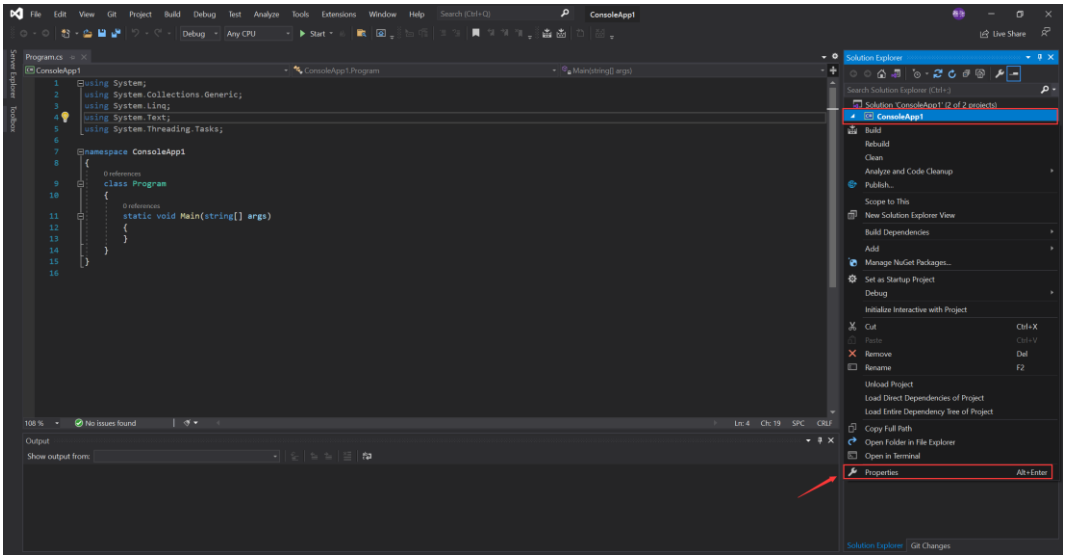


6. 库名称可以按需求修改，如 HtraApi，位置不用修改，与解决方案位于同一级目录，结果如图所示，框架选择.NET Framework 4.5，点击创建。

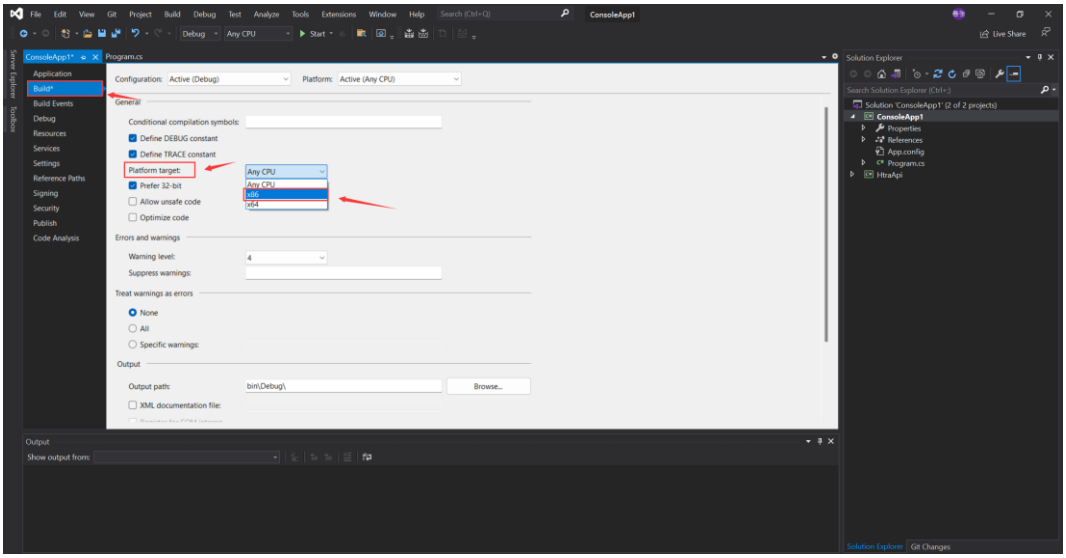


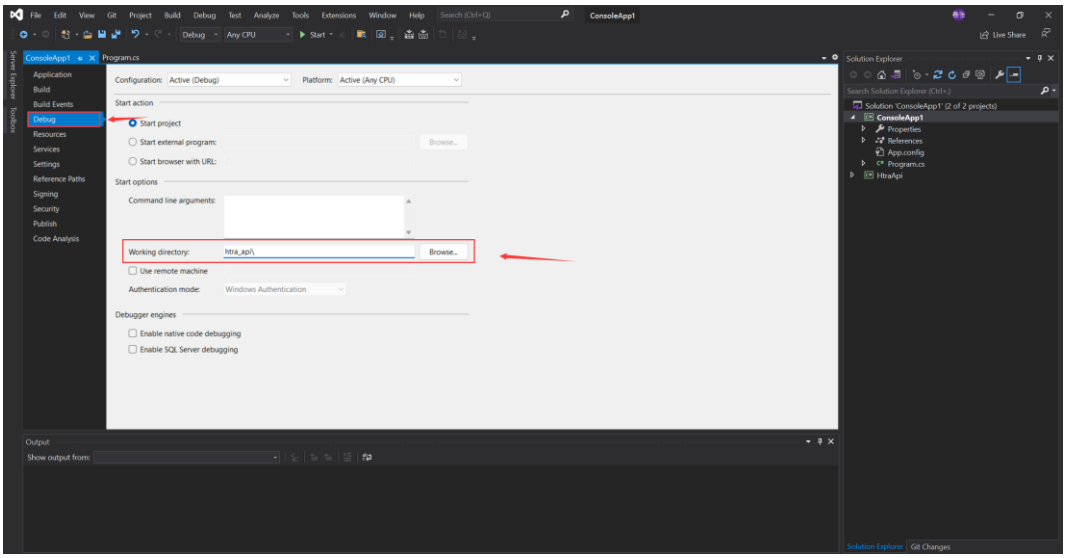


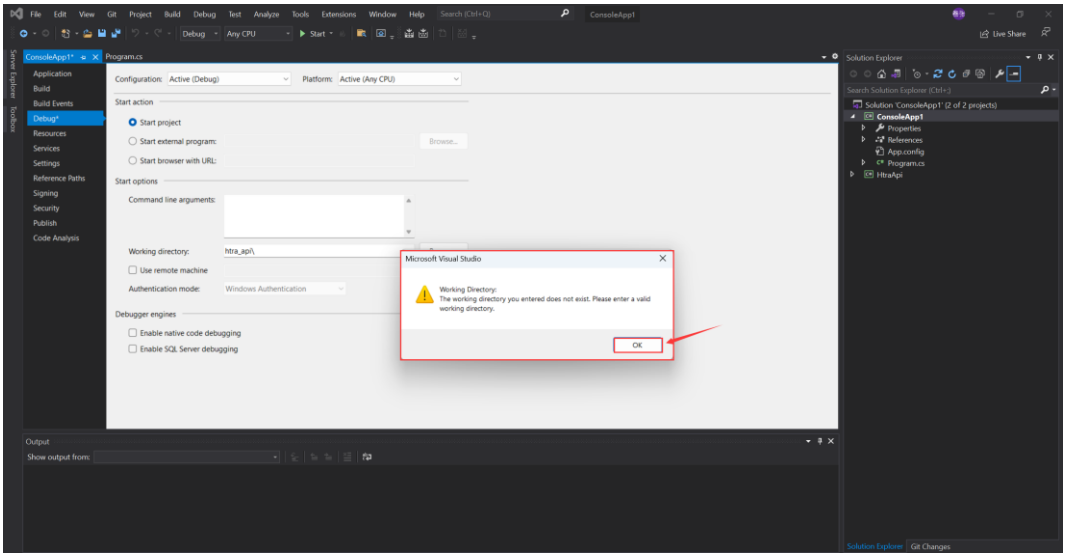
7. 右击 ConsoleApp1，点击属性。



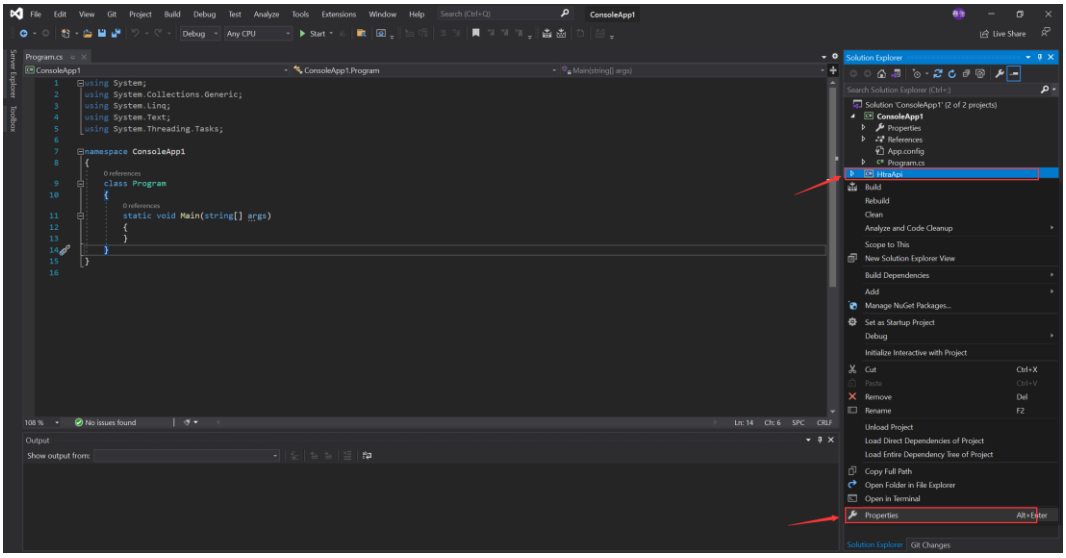
8. 查看项目的生成属性，将目标平台修改为 x86，点击调试，在工作目录中输入 htra_api\并保存，如出现图表 12 的情况，点击确定再保存一遍即可。



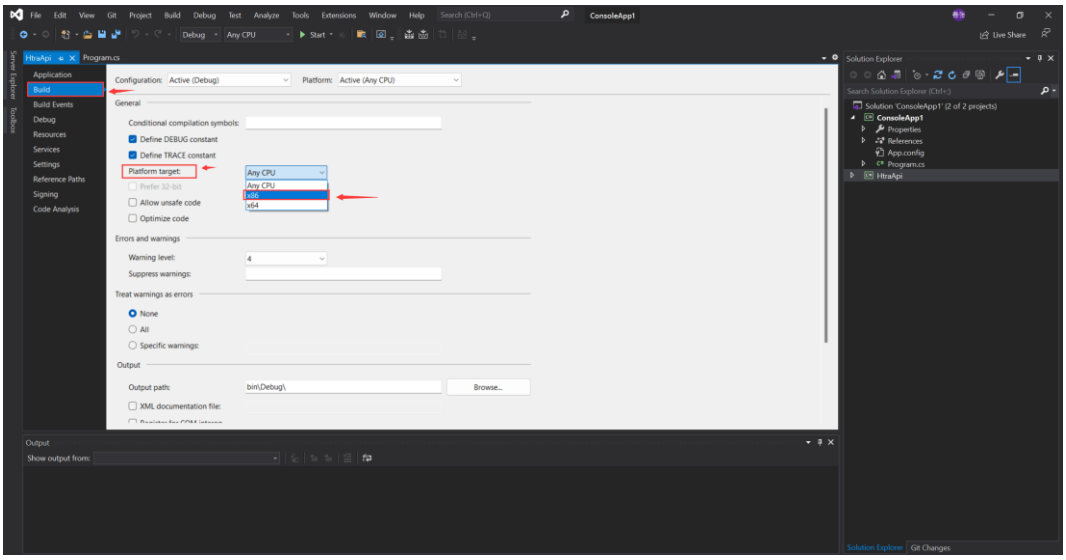




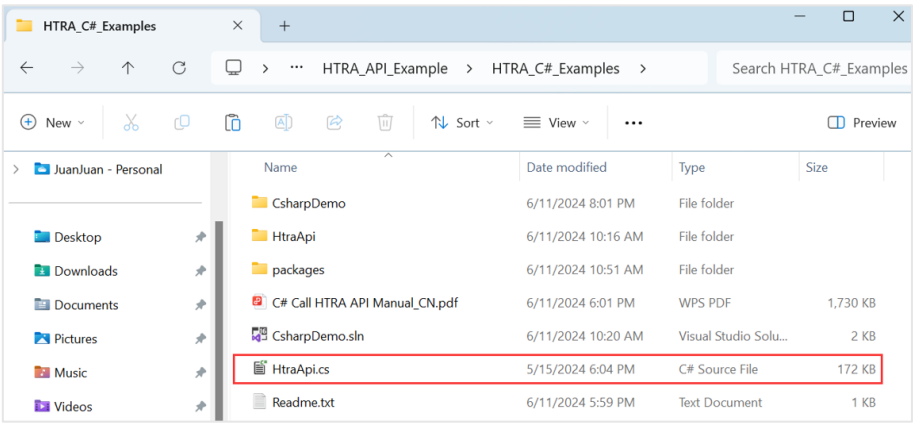
9. 右击类库 HtraApi，点击属性。

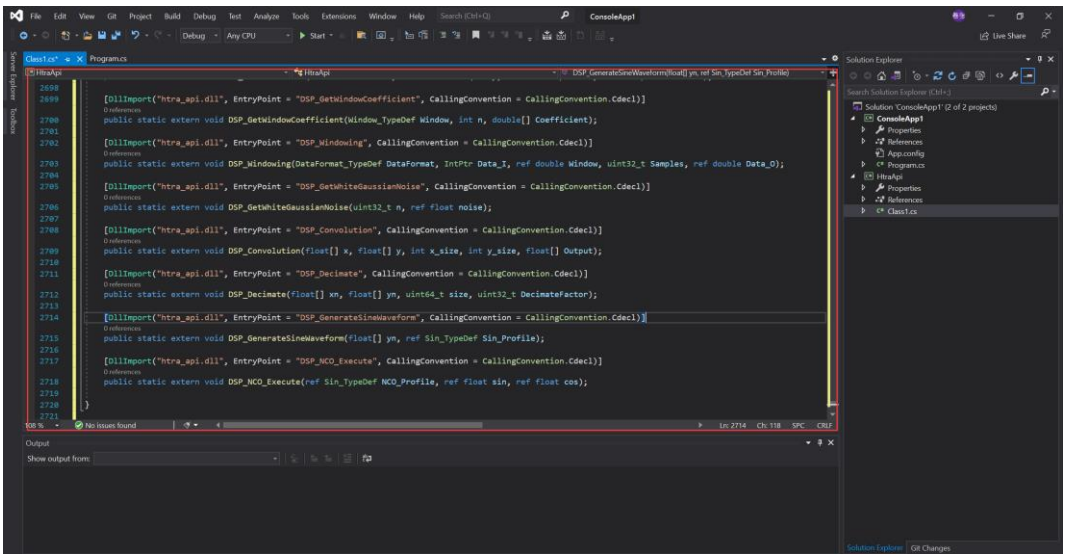


10. 查看类库的生成属性，将目标平台修改为 x86 并保存。

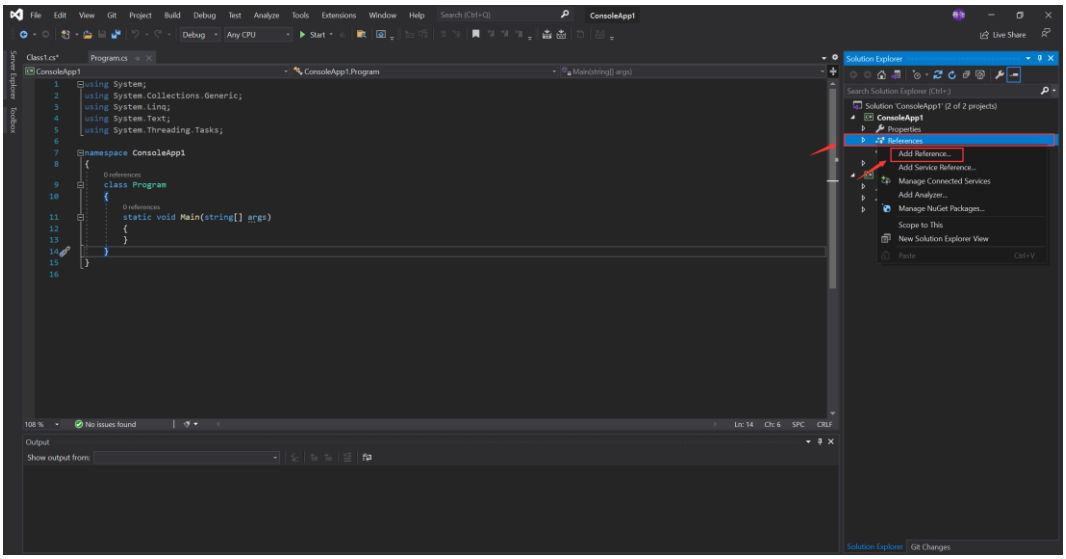


11. 将随寄资料中\Windows\HTRA_API_Example\HTRA_C#_Examples 文件夹下 HtraApi.cs 文件中的内容复制到项目类库中 Class1.cs 文件中并保存。

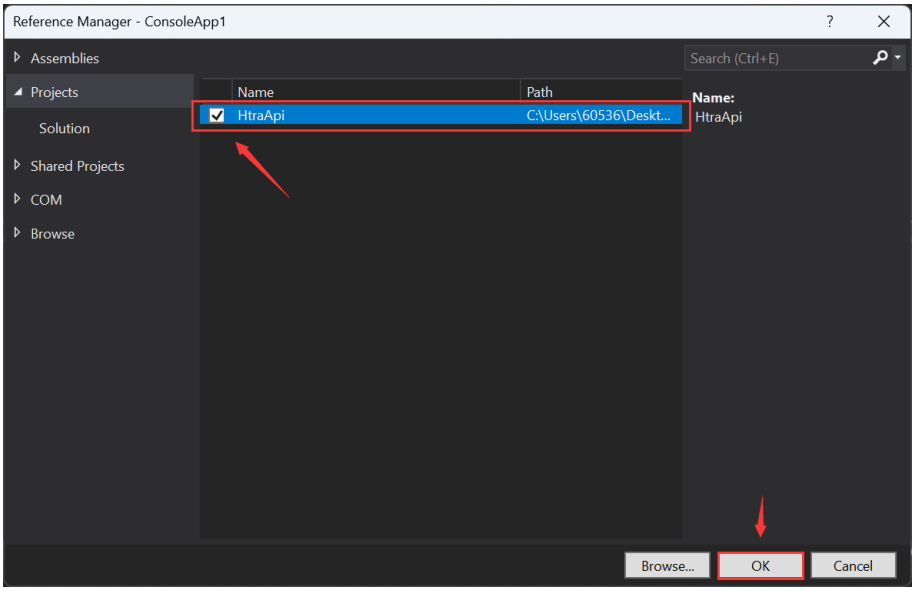




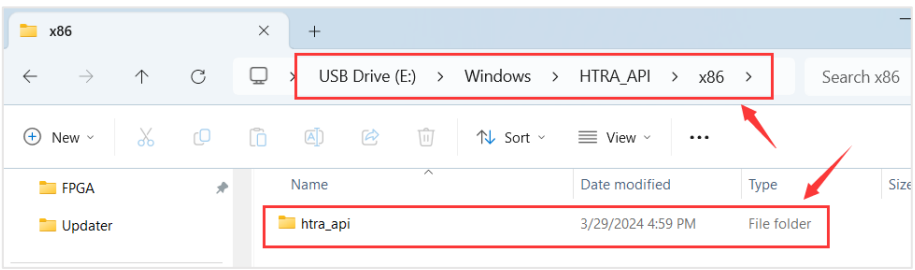
12. 选择 ConsoleApp1 项目，右击引用，选择添加引用。

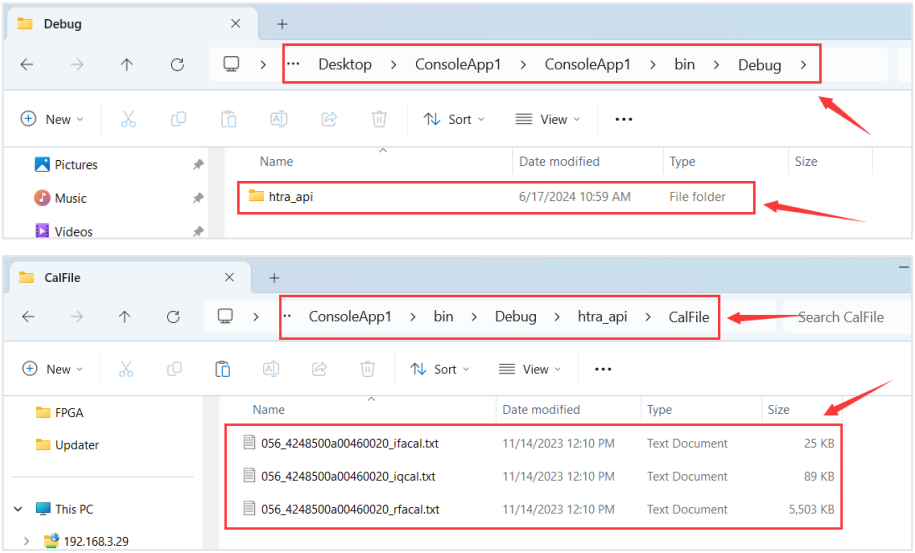


13. 选择 HtraApi 类库，确认添加类库引用。

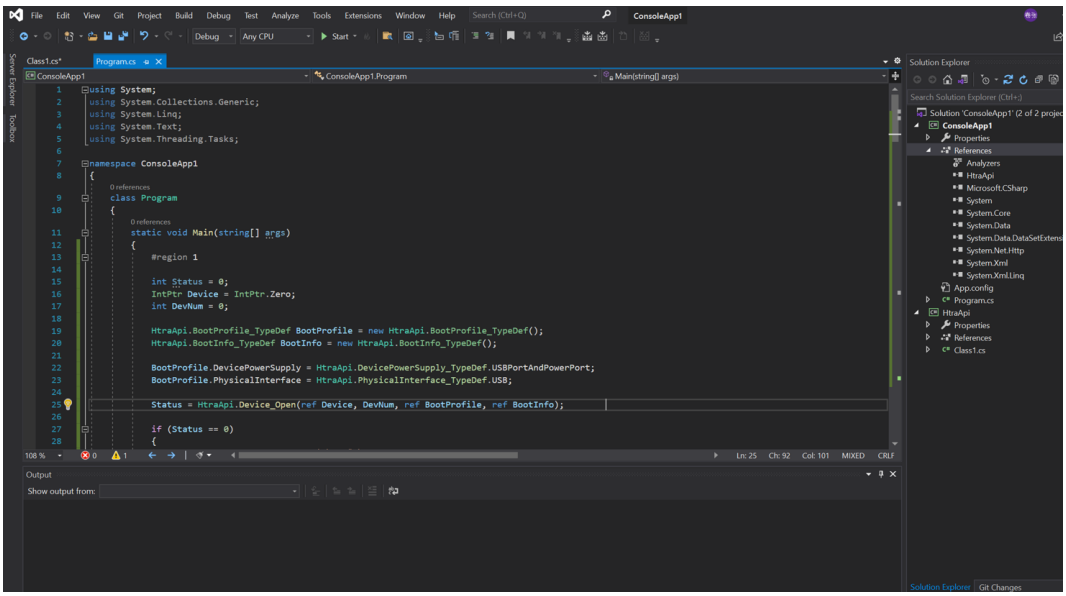


14. 将随寄 U 盘中\Windows\HTRA_API\x86 下 htra_api 文件夹复制至项目文件夹下 bin 中的 Debug 下，并确保 htra_api 文件夹下的 CalFile 文件夹中包含校准文件。





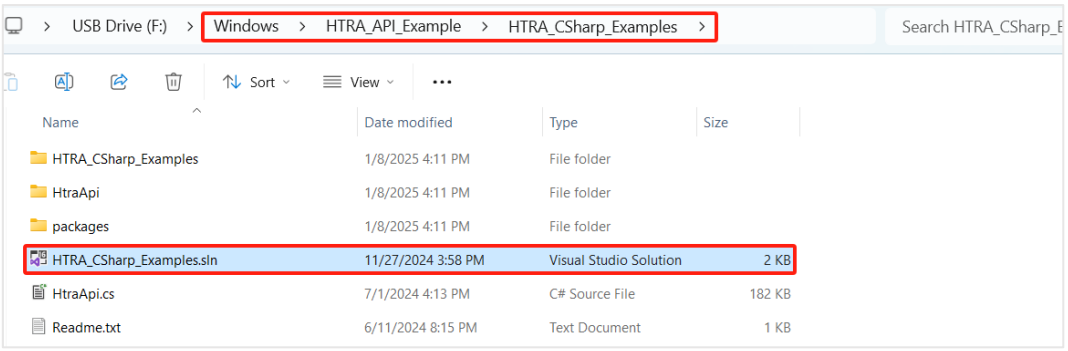
15. 正常编写代码。此处可参照随寄 U 盘中 C# 范例，即 \Windows\HTRA_API_Example\HTRA_C#_Examples 中项目。



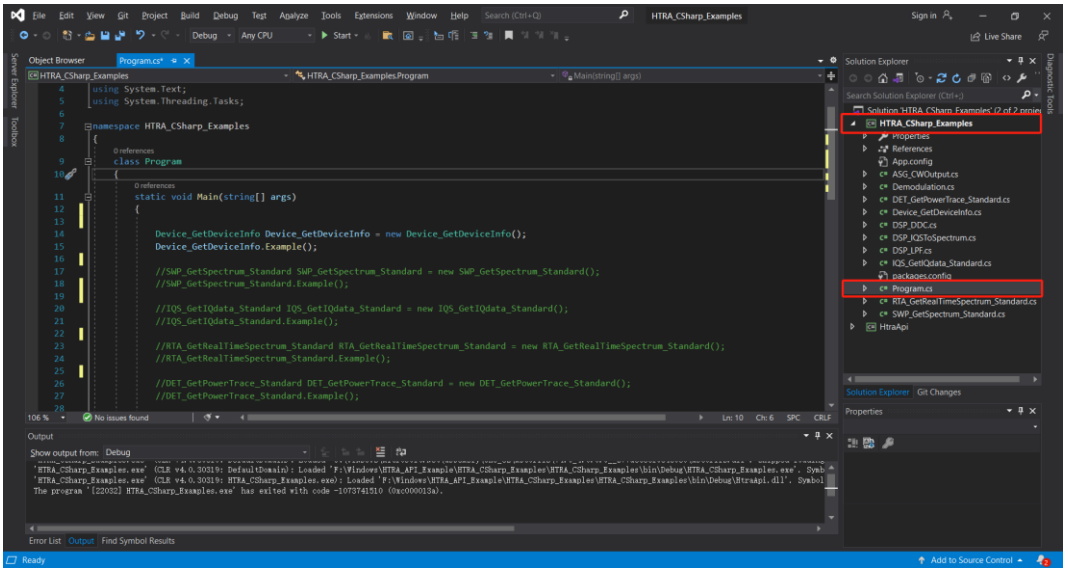
6.2 C#范例使用流程

随寄 U 盘中 C#范例使用流程如下：

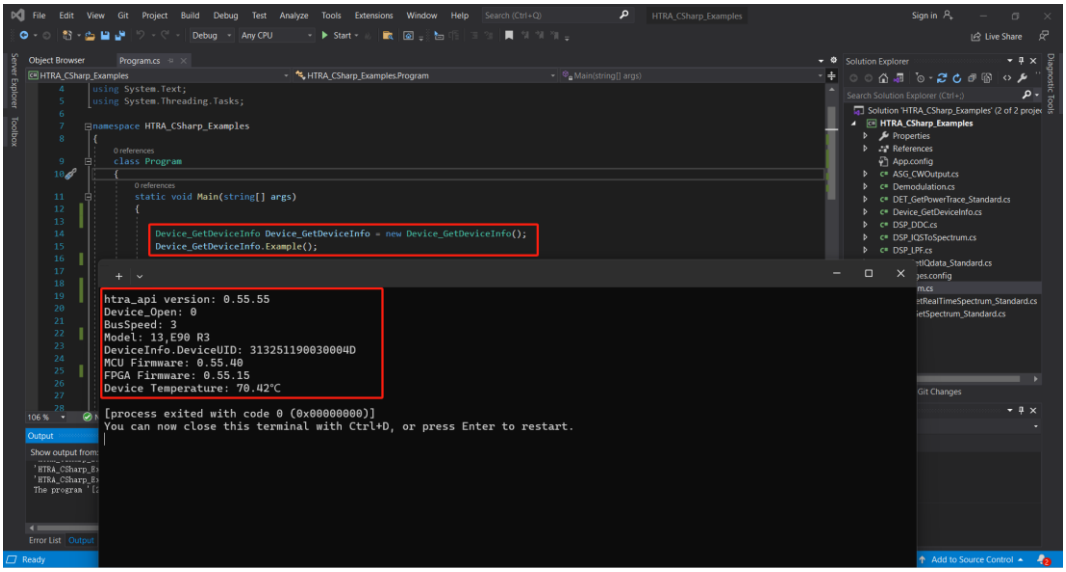
1. 使用 Visual Studio 打开随寄 U 盘 Windows\HTRA_API_Example\HTRA_CSharp_Examples 文件夹下解决方案 HTRA_CSharp_Examples.sln。



2. 点击右侧 HTRA_CSharp_Examples 项目，点击其中的 Programe.cs 文件。



3. 因为 C#随寄范例每个例程都封装在单独的类中，所以使用范例时取消注释即可（不可以同时使用多个范例）。例如测试使用 Device_GetDeviceInfo 例程时，取消注释，点击运行，如图所示为正常运行设备。


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#范例说明

6.3.1 获取设备信息

Device_GetDeviceInfo.cs：获取包括 API 版本、USB 版本、设备型号、设备 UID、MCU 版本、FPGA 版本以及设备温度在内的设备信息。

6.3.2 获取标准频谱数据

SWP_GetSpectrum_Standard.cs：获取指定频段内完整频谱数据。

6.3.3 获取固定点数或时长的 IQ 数据

IQS_GetIQdata_Standard.cs：在 IQS 模式的不同触发模式下获取 IQ 数据。

6.3.4 获取固定点数或时长的检波数据

DET_GetPowerTrace_Standard.cs：在 DET 模式的不同触发模式下获取功率检波数据。

6.3.5 获取固定点数或时长的实时频谱数据

RTA_GetRealTimeSpectrum_Standard.cs：在 RTA 模式的不同触发模式下获取实时频谱数据。

6.3.6 输出单音信号

ASG_CWOutput.cs：含有信号源功能选件的设备通过 ASG 功能输出单音信号、频率扫描信号或功率扫描信号。

6.3.7 AM/FM 解调

Demodulation.cs：DSP_FMDemod 将获取到的 IQ 数据进行 FM 解调以及播放。DSP_AMDemod 将获取到的 IQ 数据进行 AM 解调以及播放。

6.3.8 IQ 转频谱数据

DSP_IQSToSpectrum.cs：将 IQS 模式下获取到的 IQ 数据转换成为频谱数据。

6.3.9 低通滤波

DSP_LPF.cs：对获取到的 IQ 数据进行低通滤波并转频谱。

6.3.10 数字下变频

DSP_DDC.cs：对获取到的 IQ 数据进行数字下变频并转频谱。

6.3.11 相位噪声测试

DSP_TraceAnalysis_PhaseNoise.cs：相位噪声测试功能演示。

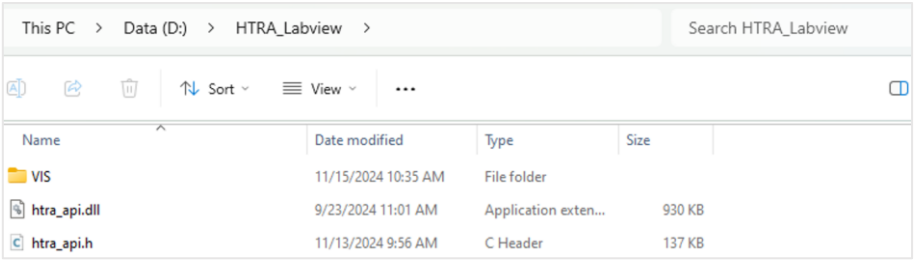
7. Java（待补充）

8. Labview

8.1 配置开发环境

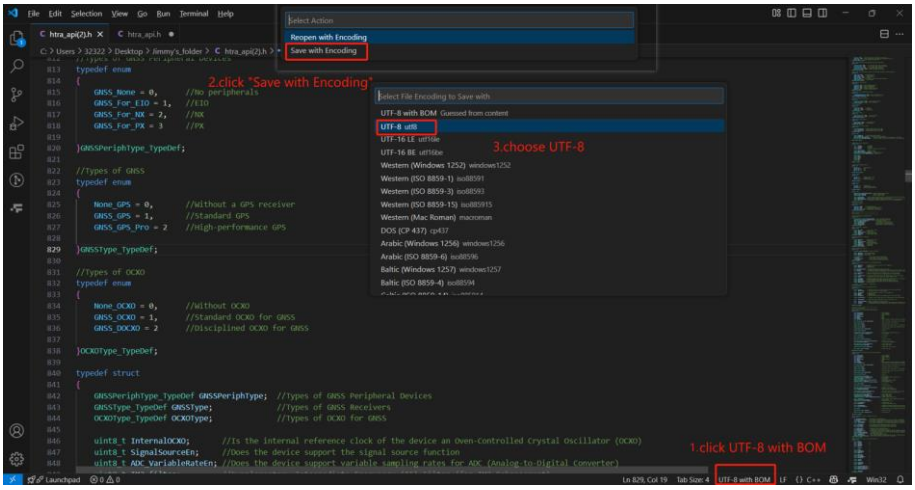
8.1.1 使用 Labview 导出 htra_api.dll 中的库函数

1. 创建一个文件夹（如 HTRA_Labview），将 U 盘中 Windows\HTRA_API\x86 内的 htra_api 文件夹拷贝至该文件夹中。再创建一个文件夹（如 VIS）放置导出后的 vi。

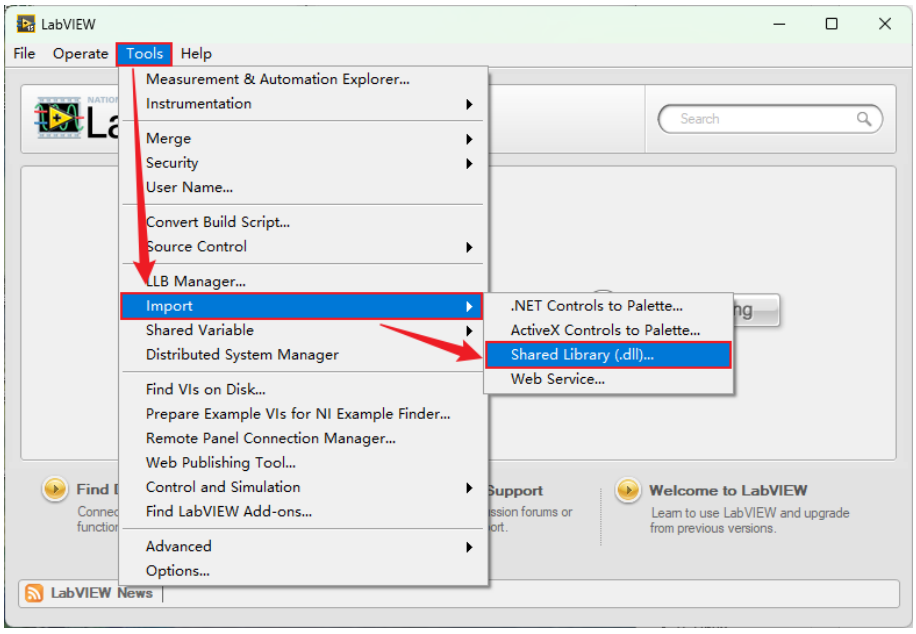


2. Labview 在导入时不识别 uint64_t 和 int64_t 数据类型，所以在导入前需要把 uint64_t 和 int64_t 类型的参数类型全部改为 double，注意导出函数后需在 vi 中将这些改过的参数类型再改回 uint64_t 或 int64_t。

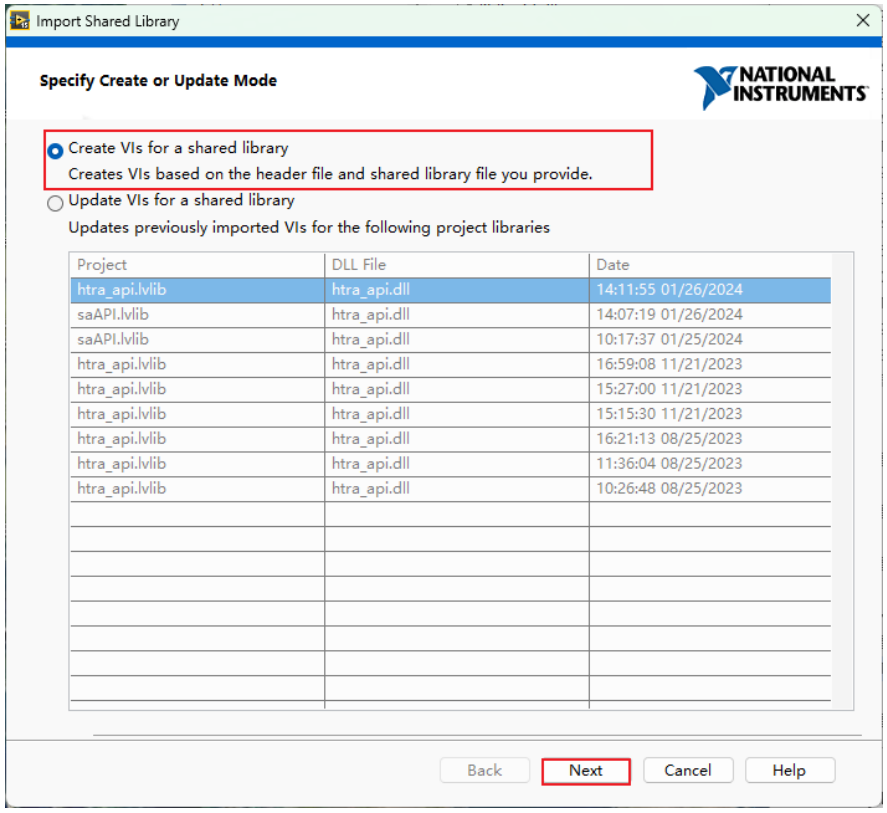
3. 将 htra_api.h 的编码格式改为 UTF-8。



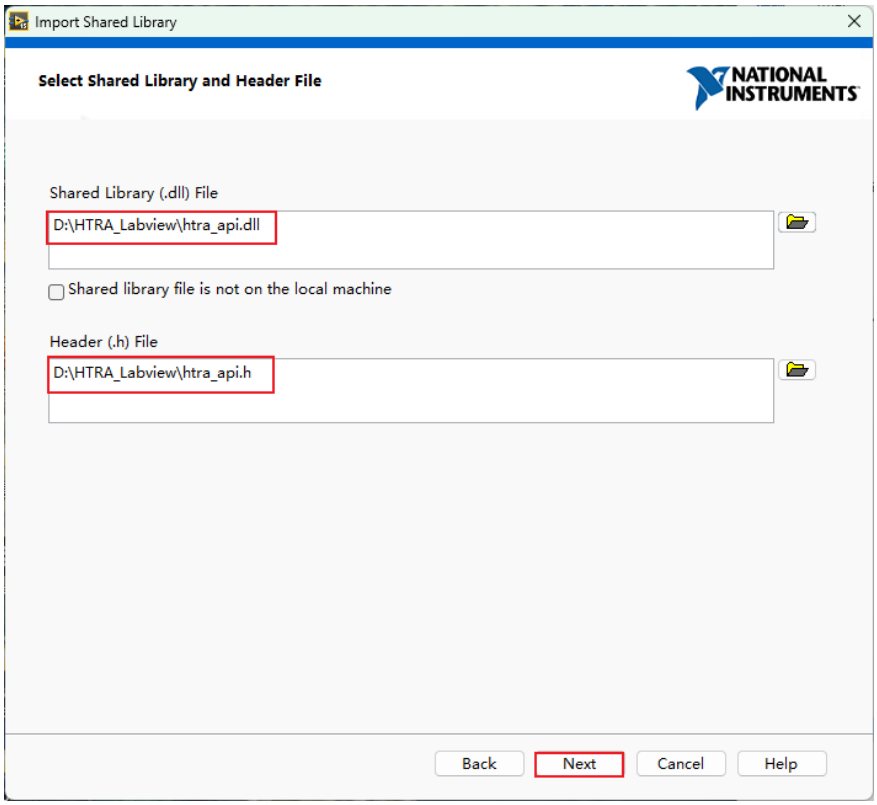
4. 打开 Labview，选择“Tools--->Import--->Share Library”。



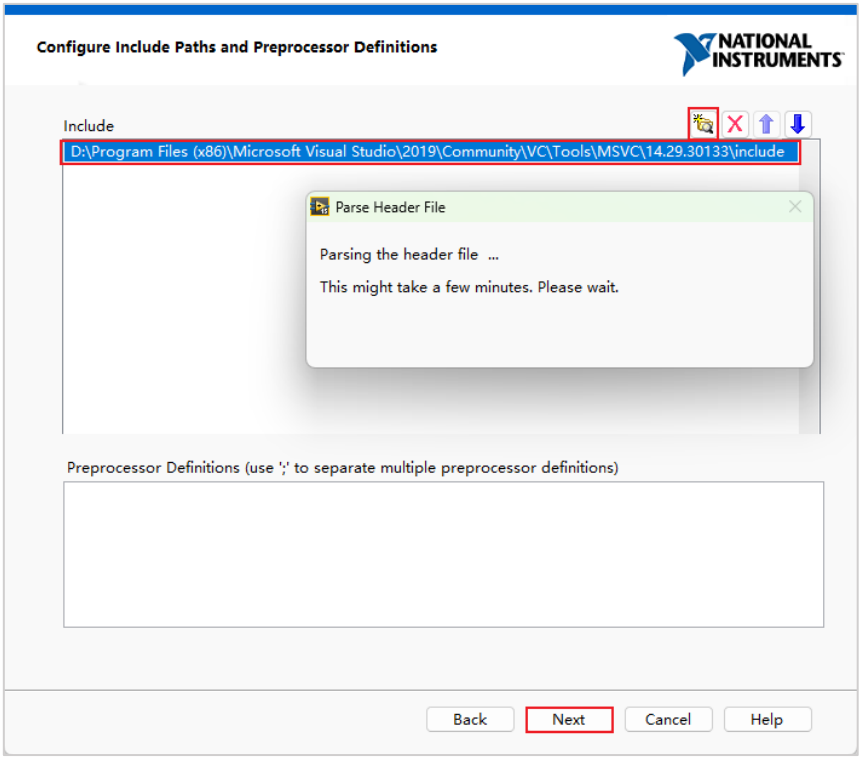
5. 选择“Create VIs for a shared library”，点击“Next”。



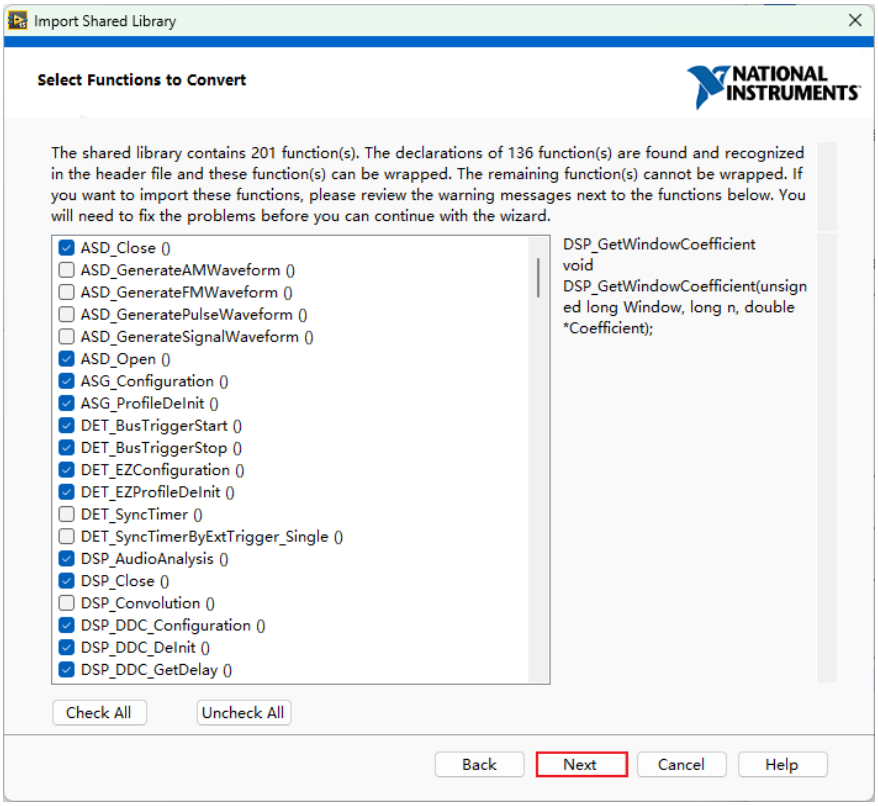
6. “Shared Library(.dll) File”和“Header (.h) File”中选择之前创建的文件夹中对应的库文件。选择好共享库文件路径之后，头文件路径可以自动识别，不需要再选。然后点击“Next”。



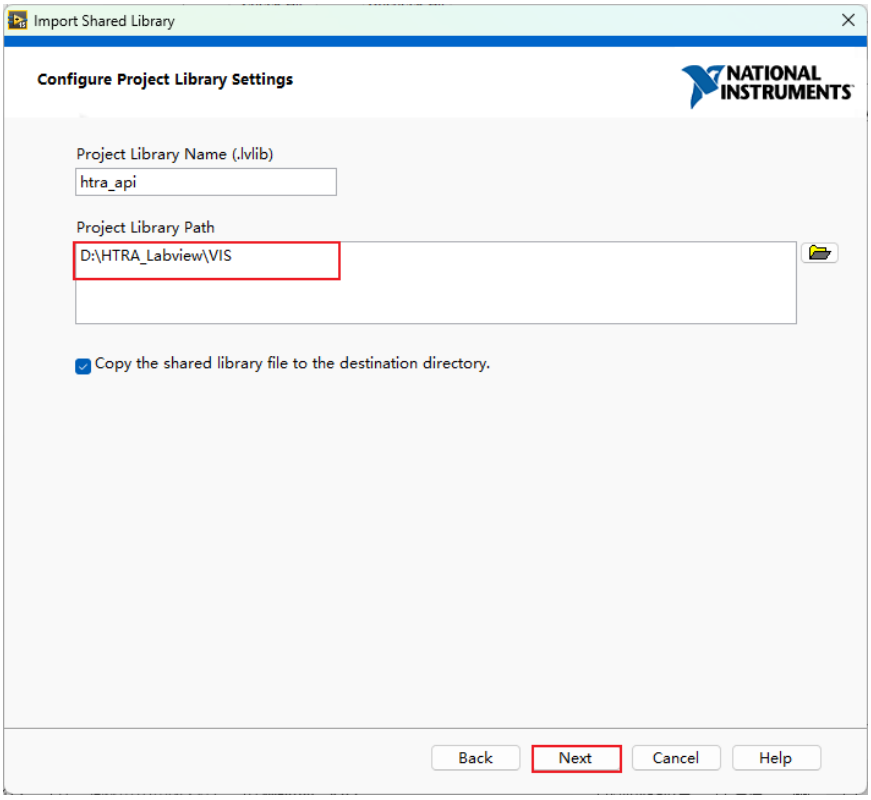
7. “Include”中配置其他依赖文件的路径，一般在安装 VS 的文件夹中，如下图路径所示，然后点击“Next”等待解析。



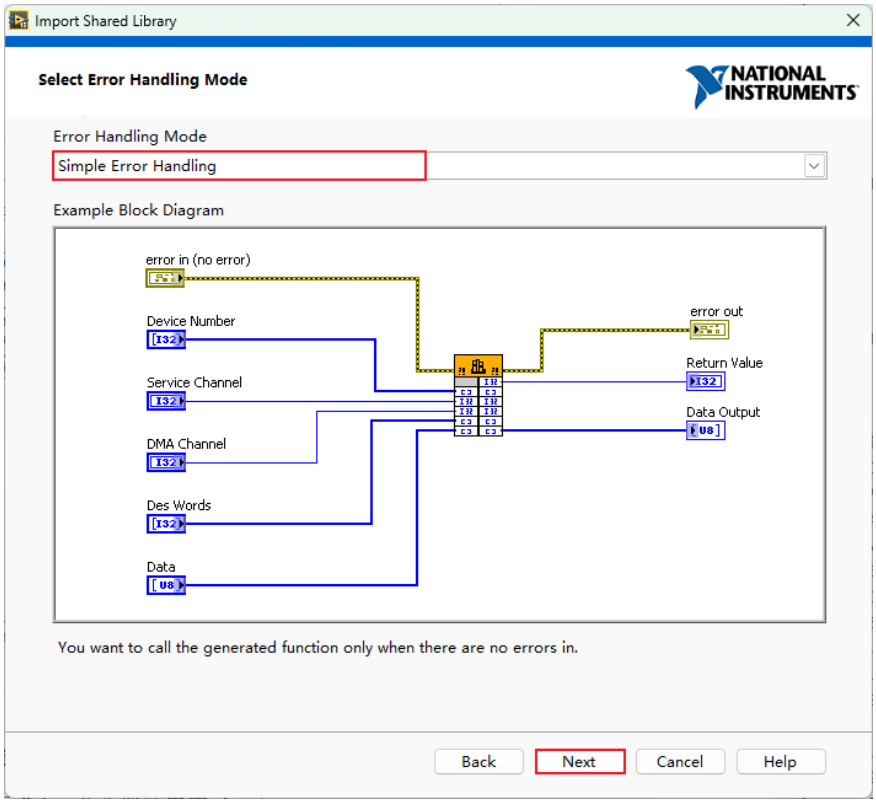
8. 选择需要导出的函数，有一部分函数已弃用或暂未开放，可对照 htra_api.h 进行选择，选好之后点击“Next”。



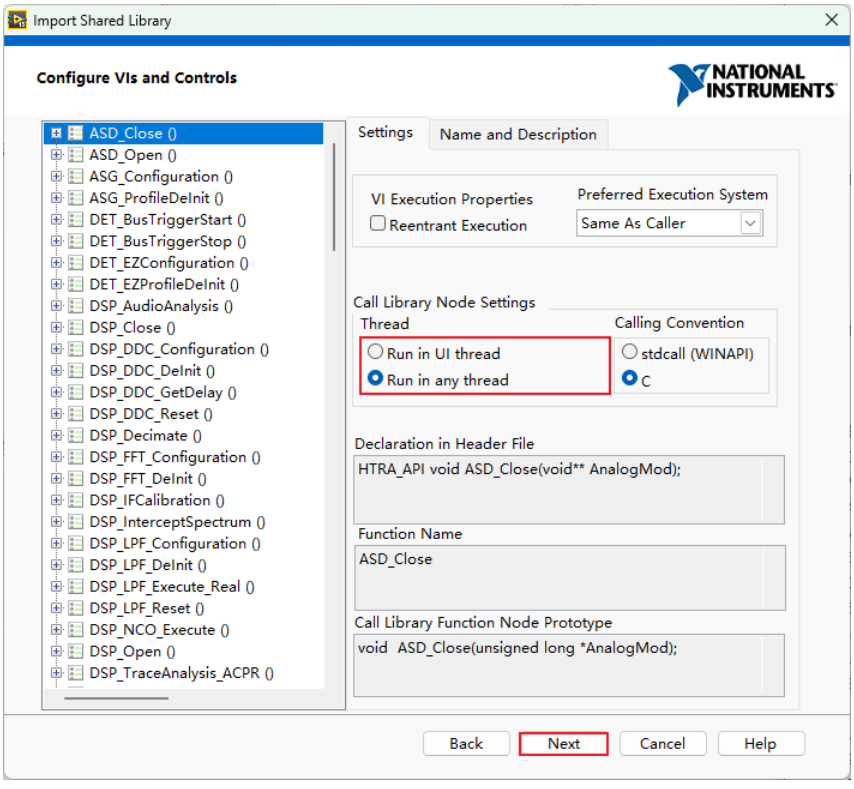
9. “Project Library Path”选择存库和头文件路径中的 VIS 文件夹，然后点击“Next”。



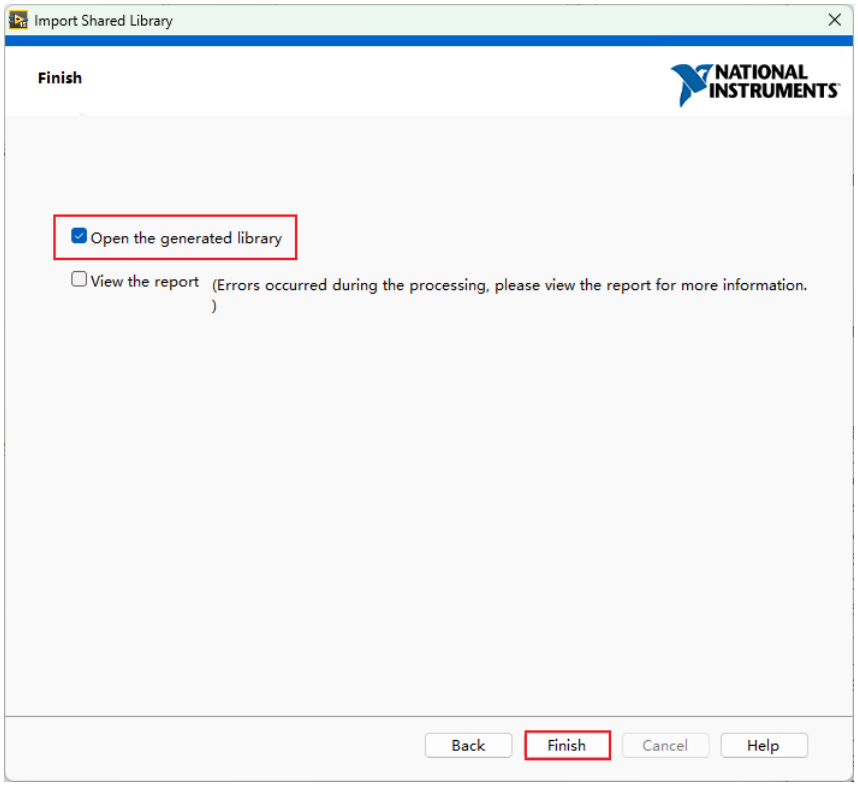
10. “Error Handling Mode”建议选择“Simple Error Handling”，然后点击“Next”。



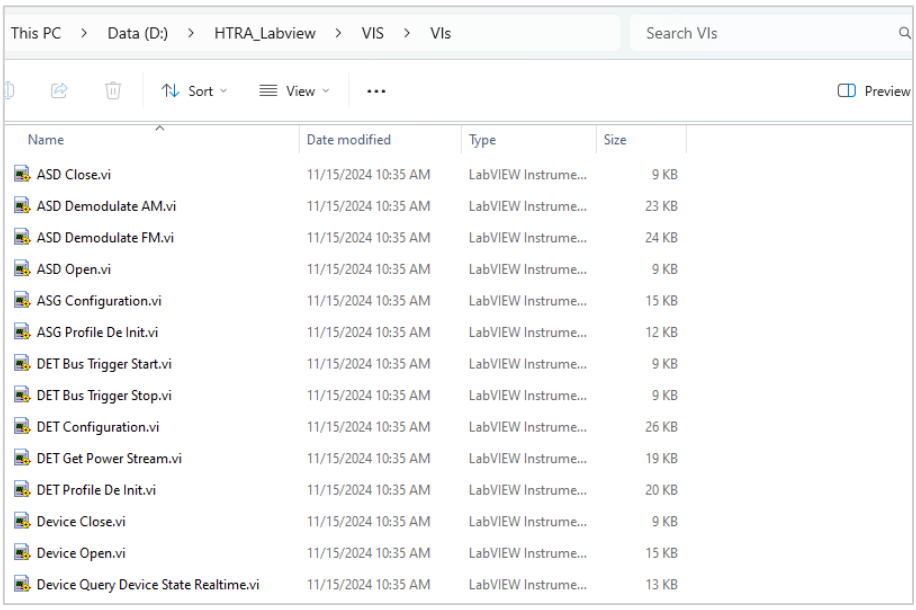
11. 将每一个函数的调用库节点设置选择“在任意线程中运行”。全部设置完之后点击“Next”等待生成 vi。



12. 勾选“Open the generated library”，“View the report”可选可不选，点击“Finish”。

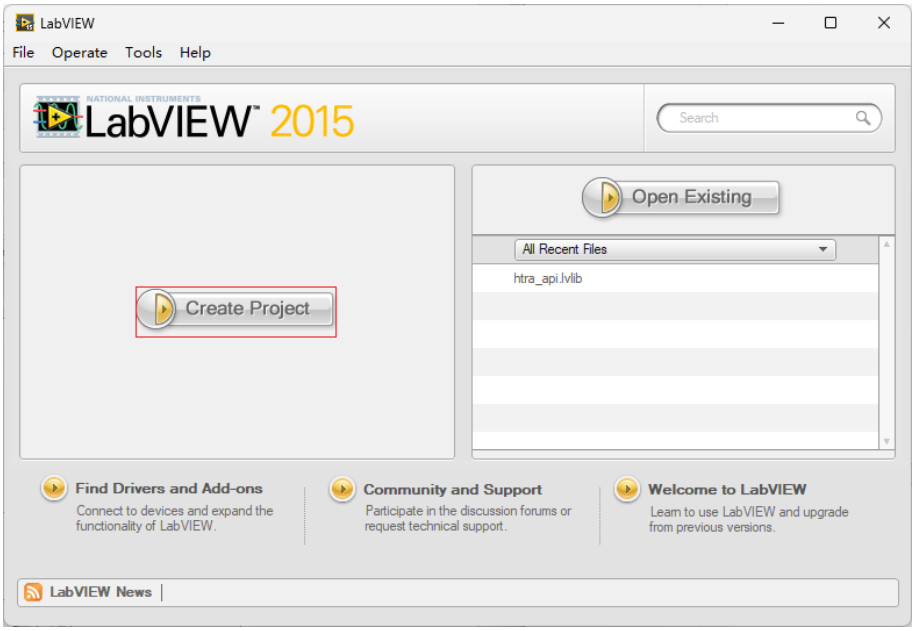


13. VIS 文件夹内的 vi 便是导好的 API 函数。至此，导出 API 函数结束。

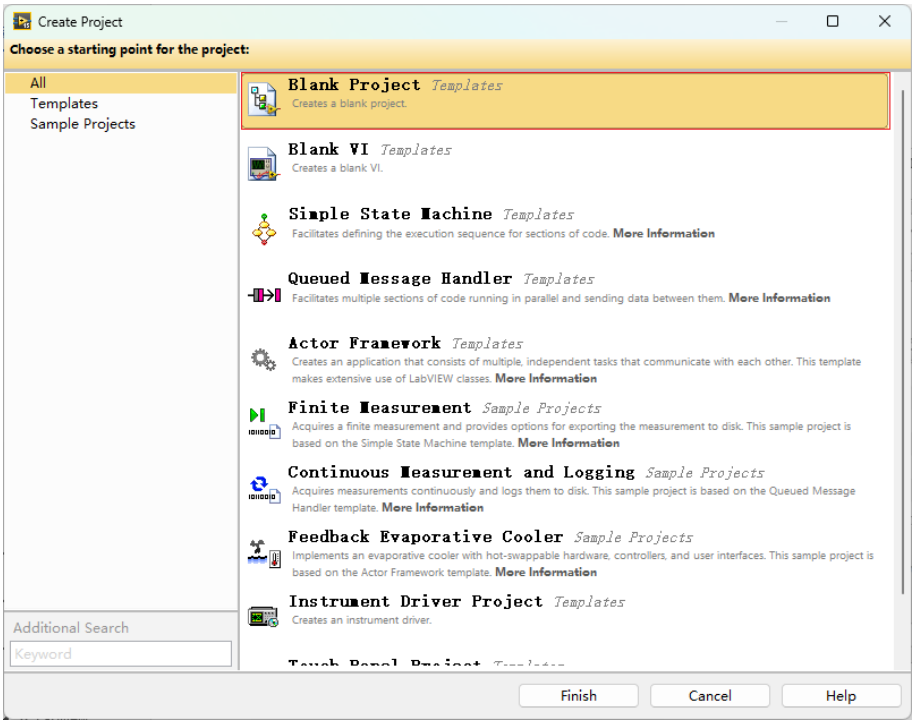


8.1.2 在 Labview 环境中使用 API

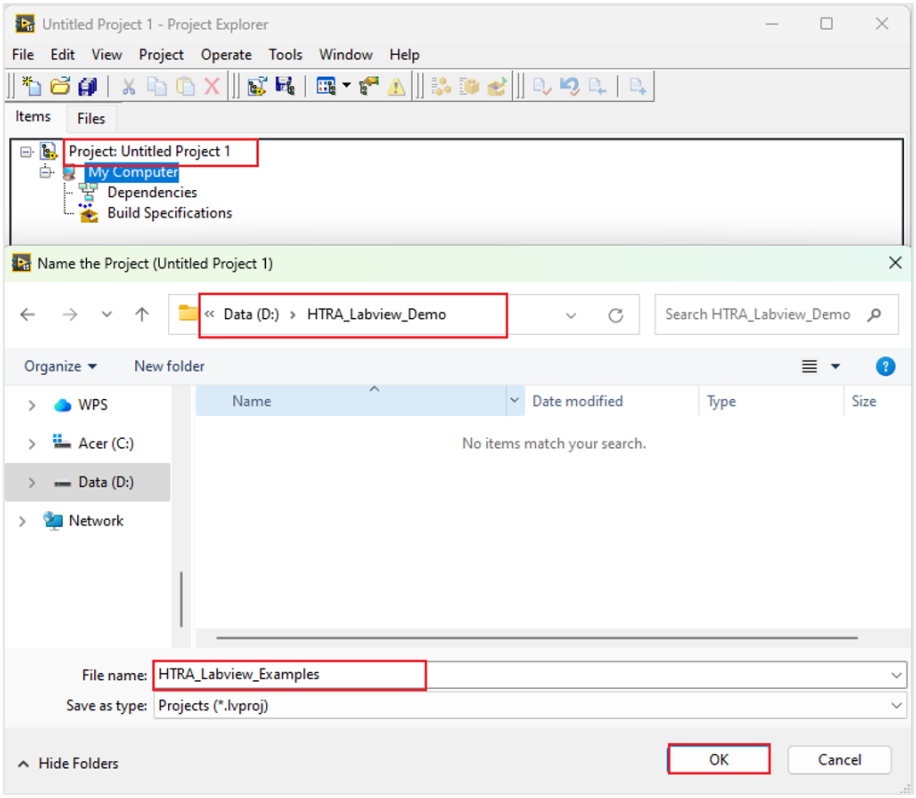
1. 打开 Labview，点击“Create Project”。



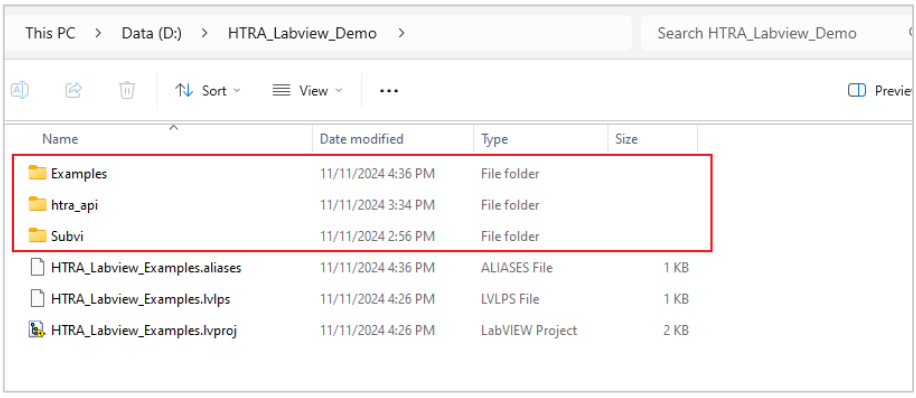
2. 选择“Blank Project”，点击“Finish”。



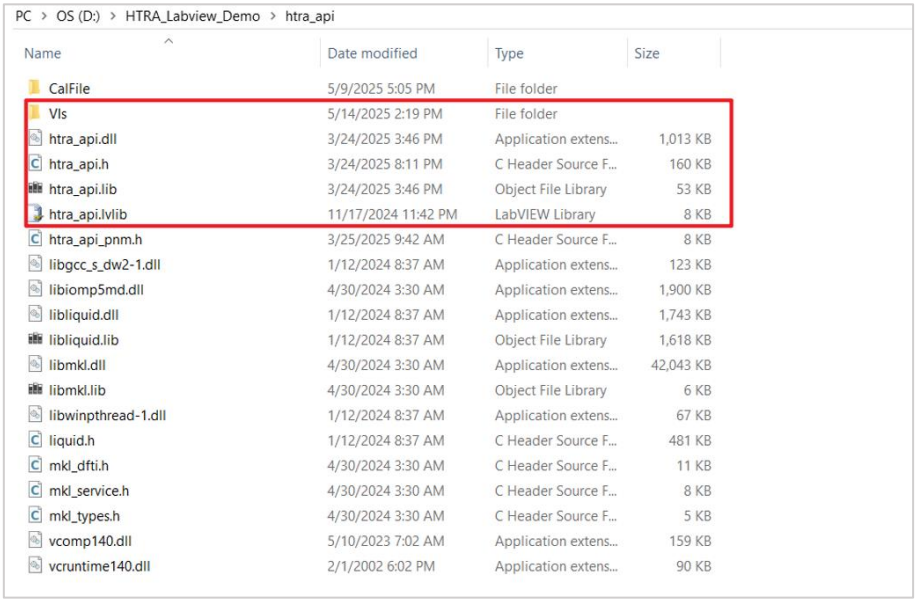
3. 之后会出现一个未命名的空项目，“Ctrl+S”保存项目，选定项目保存路径并给项目命名，然后点击“OK”。



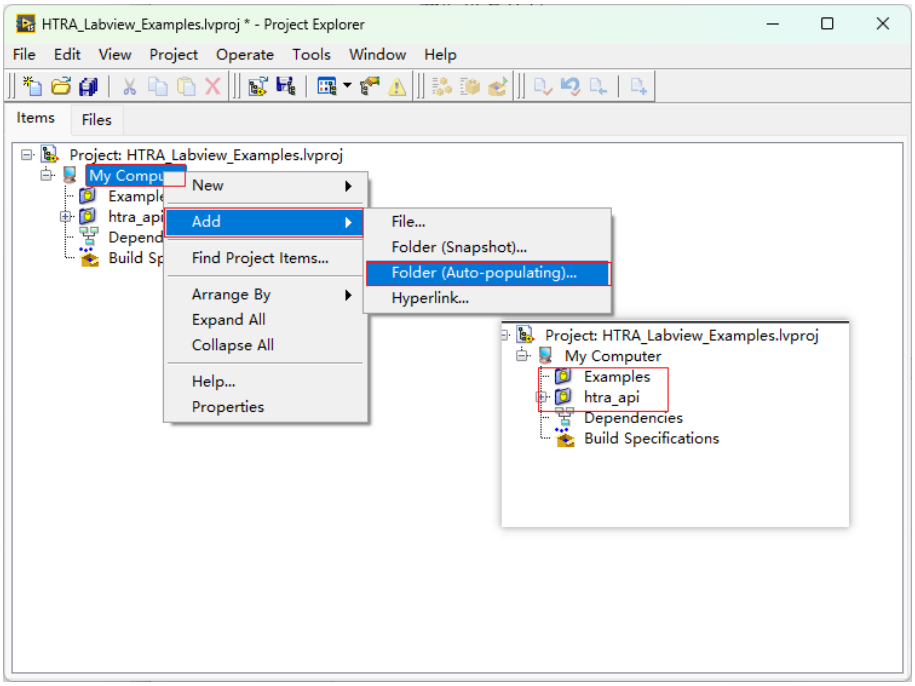
4. 将 U 盘中\Windows\HTRA_API\x86 文件夹中的 htra_api 文件夹拷贝到该项目的同级目录下。除此之外，可以再创建一个 Examples 文件夹，用于存放范例。如果有需要可以再创建一个文件夹如 Subvi，用于存放子 vi。



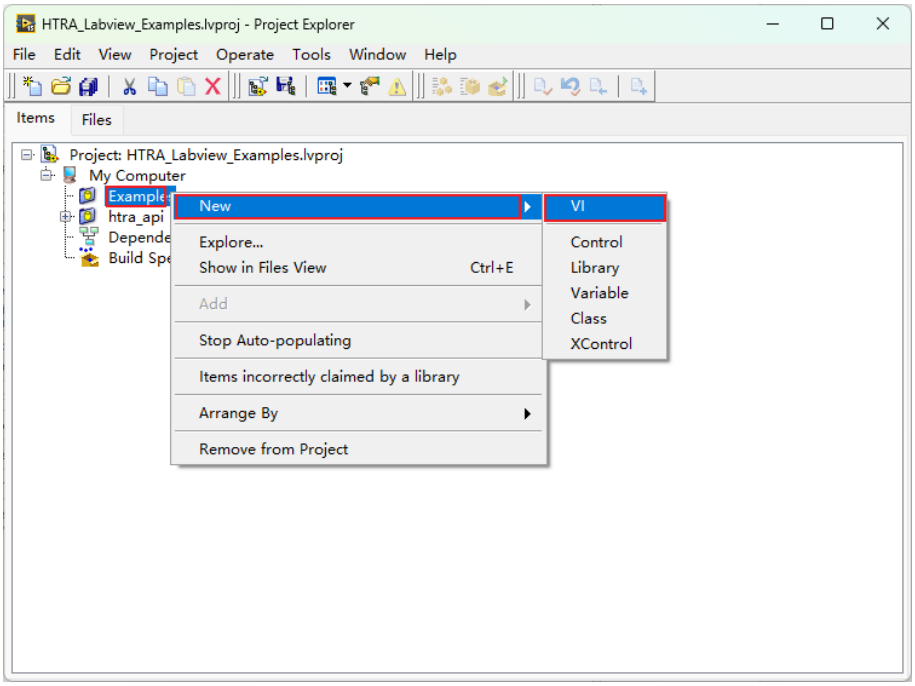
5. 将之前导出好的库函数，即 8.1.1 中创建的 VIS 文件夹中的内容，拷贝至 htra_api 文件夹。



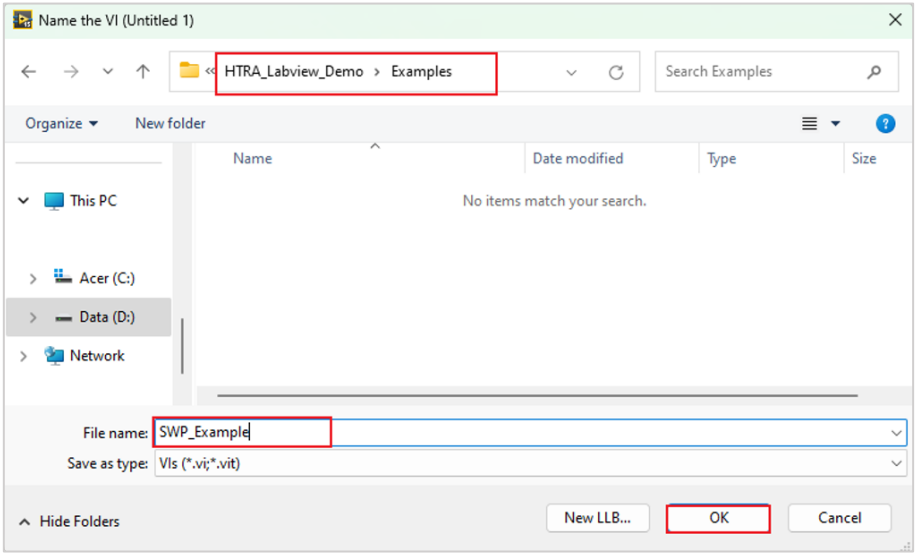
6. 在 Labview 工程中，将新建的 Examples 文件夹和 htra_api 文件夹添加至工程中，然后保存。



7. 在 Examples 文件夹中新创建 VI，在程序框图页内即可调用导出的 Labview API 函数，调用流程与 C 环境中一致。



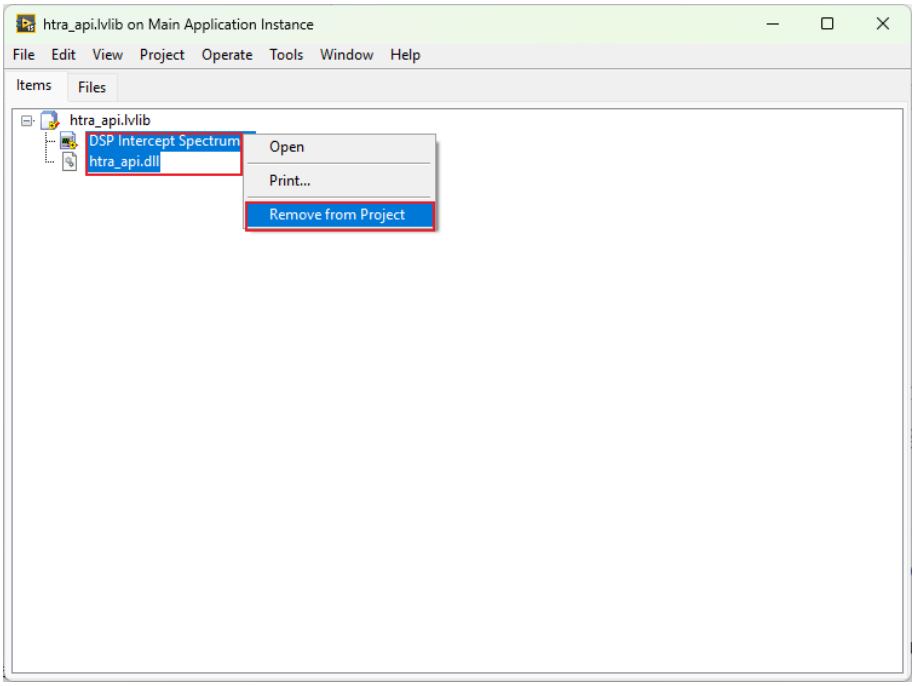
8. 最后将保存该程序到 Examples 文件夹，重新命名即可。



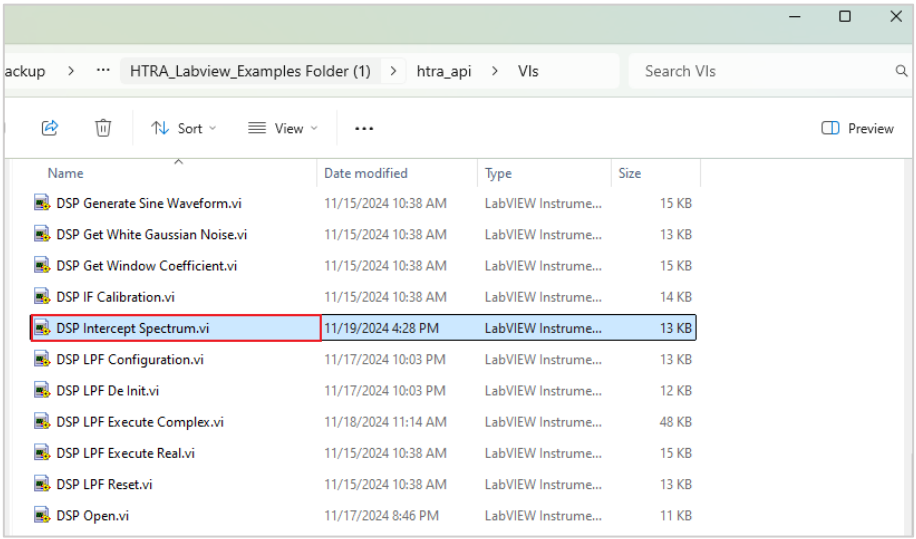
8.1.3 在已有工程中使用新导出的库函数

下面以库函数 DSP_InterceptSpectrum（）为例介绍如何在已有工程下使用新导入的函数。

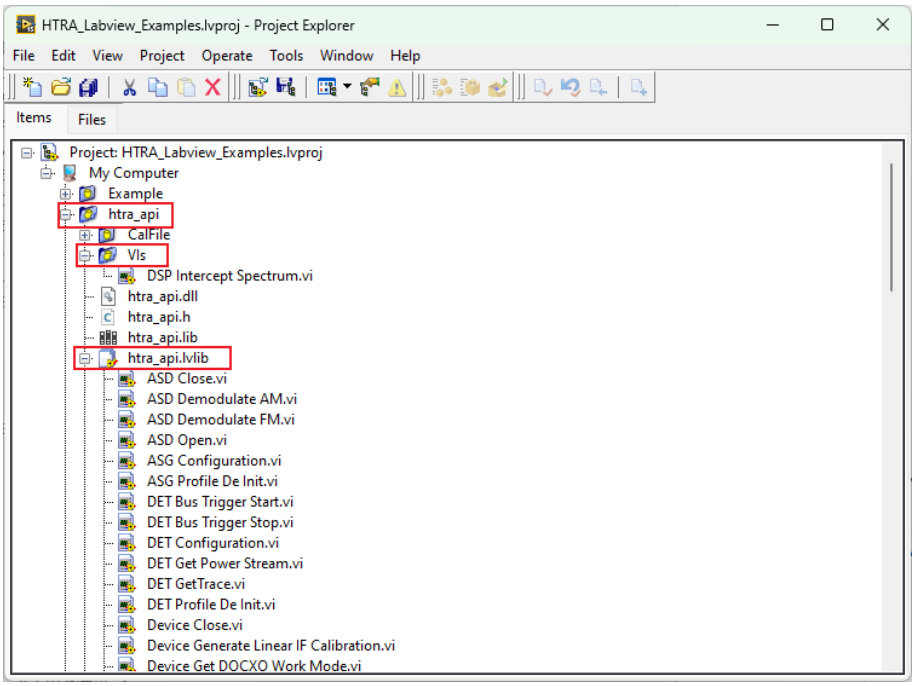
1. 导出库函数的方法请参考 8.1.1 章节，注意最后在弹出的的 htra_api.lvlib 中将导出的函数及 dll 文件从项目中删除，删除后关闭 htra_api.lvlib，如下图所示。



2. 将导出的 DSP_InterceptSpectrum 函数 vi 拷贝到已有工程的 htra_api\VIs 文件夹中。



3. 使用 Labview 打开该工程，然后打开工程下的 htra_api\VIs 文件夹和 htra_api.lvlib。

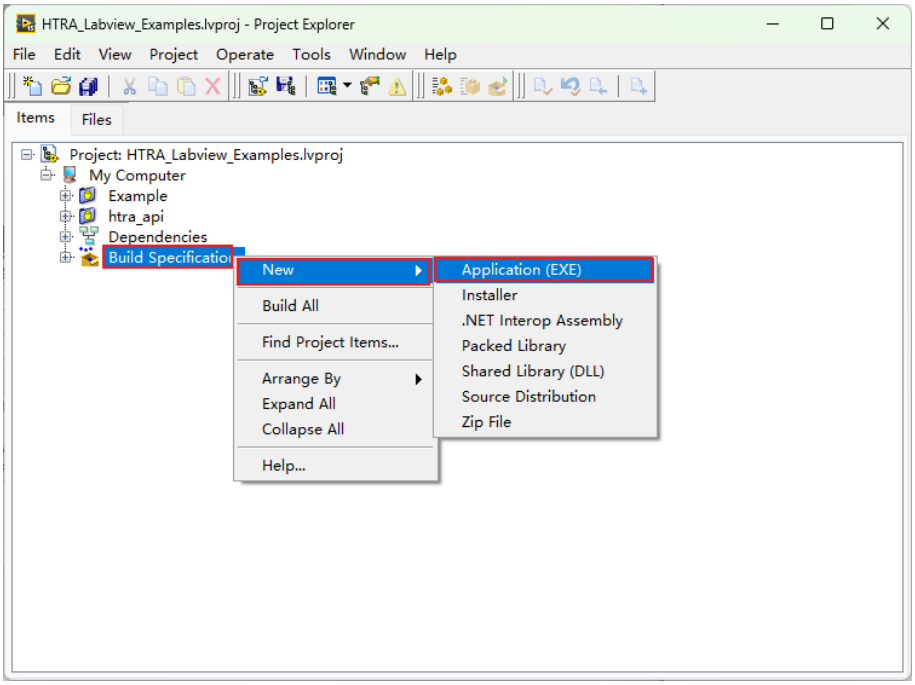


4. 将 Vis 文件夹中的 DSP_InterceptSpectrum 函数 vi 拖拽到 htra_api.lvlib 中。至此，就可以正常使用新加入的 vi 函数了。

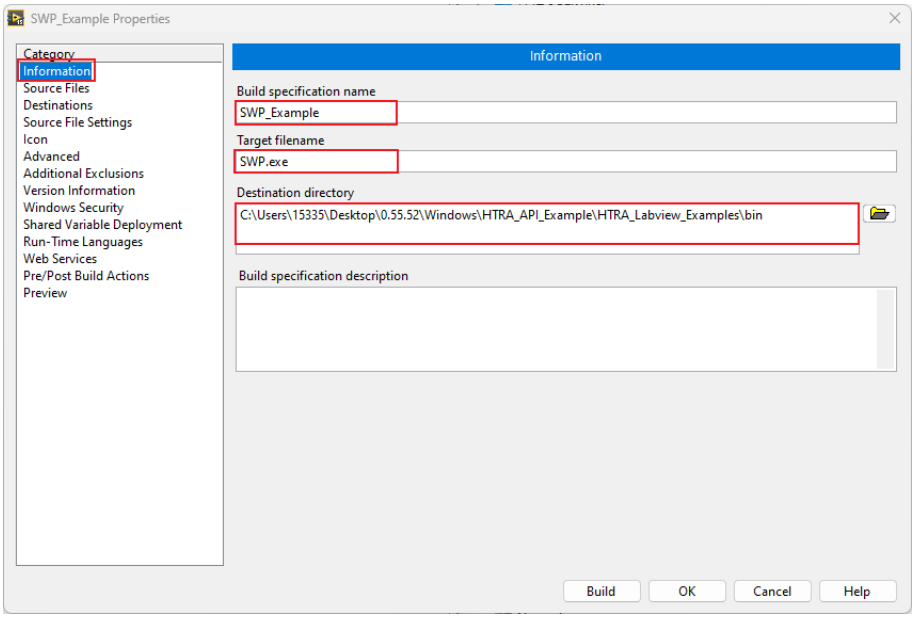
8.1.4 将 Labview 中的 vi 生成 exe

下面以 U 盘中 Windows\HTRA_API_Example 文件夹下的 Labview 工程为例介绍如何将程序 vi 生成 exe。

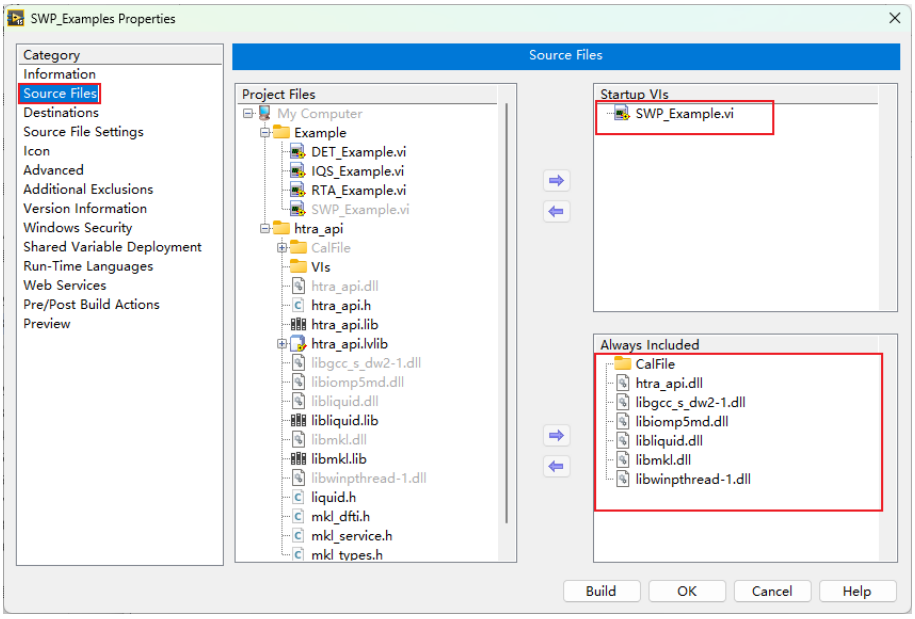
1. 打开 Labview 项目工程，选择“Build Specifications-->New-->Applicatiion(EXE)”。



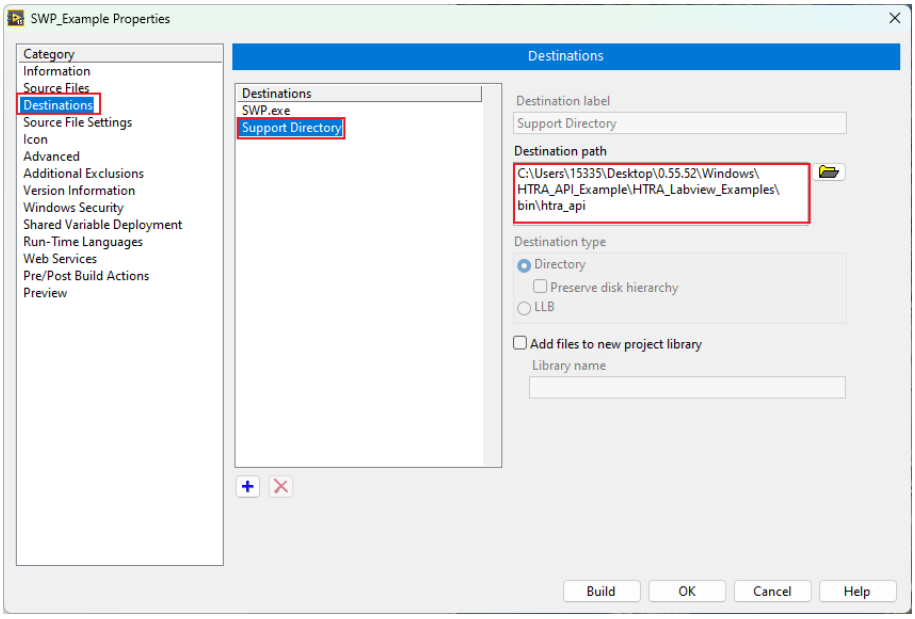
2. 在“Information”中填写生成该程序文件的规范名称、exe 的名称和生成的目标目录路径。



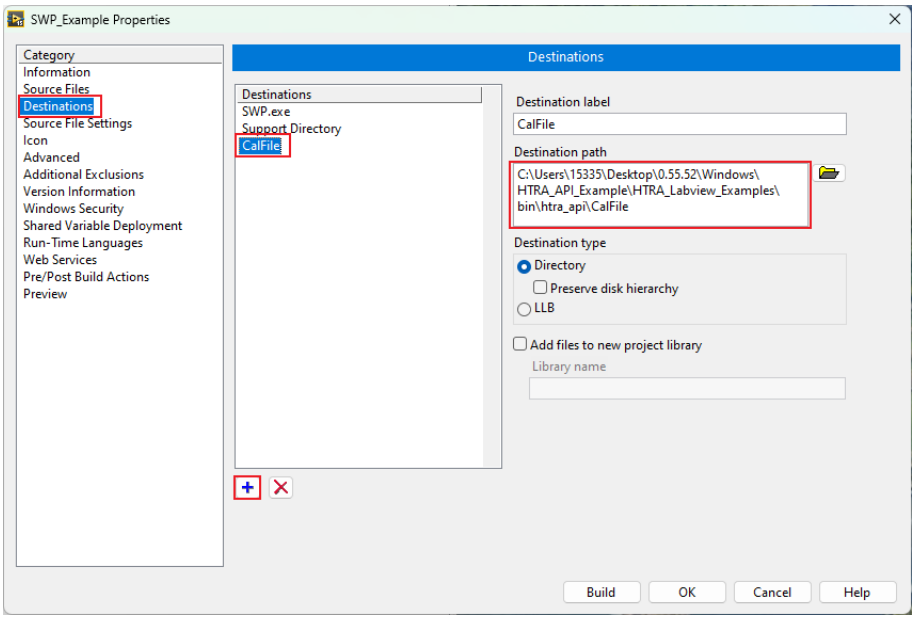
3. 在“Source Files”中选择要导成 exe 的 vi 程序、CalFile 文件夹与程序所依赖的全部库文件。



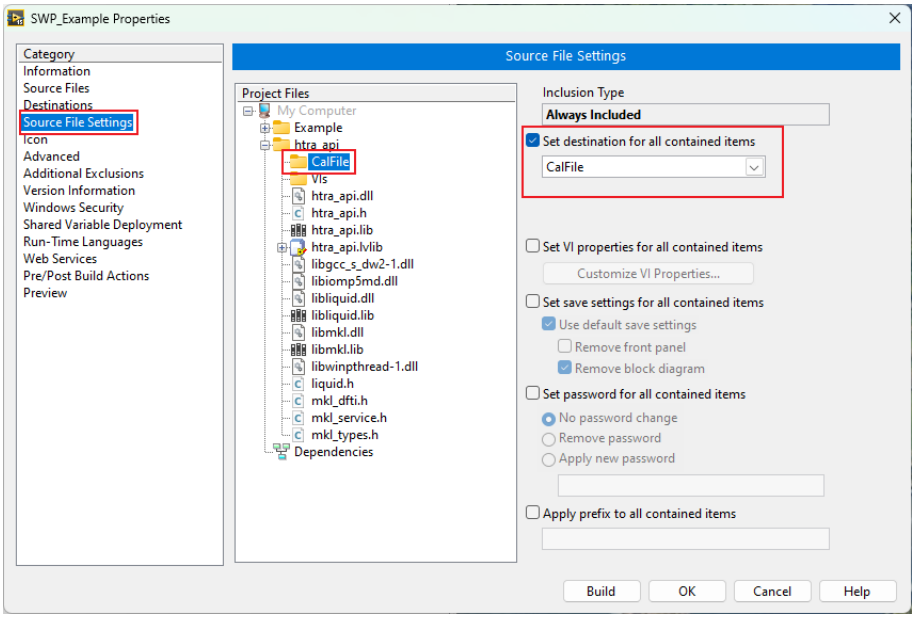
4. 在“Destinations”中将“Support Directory”的目标路径添加一层 htra_api，用来存放库和校准文件夹。



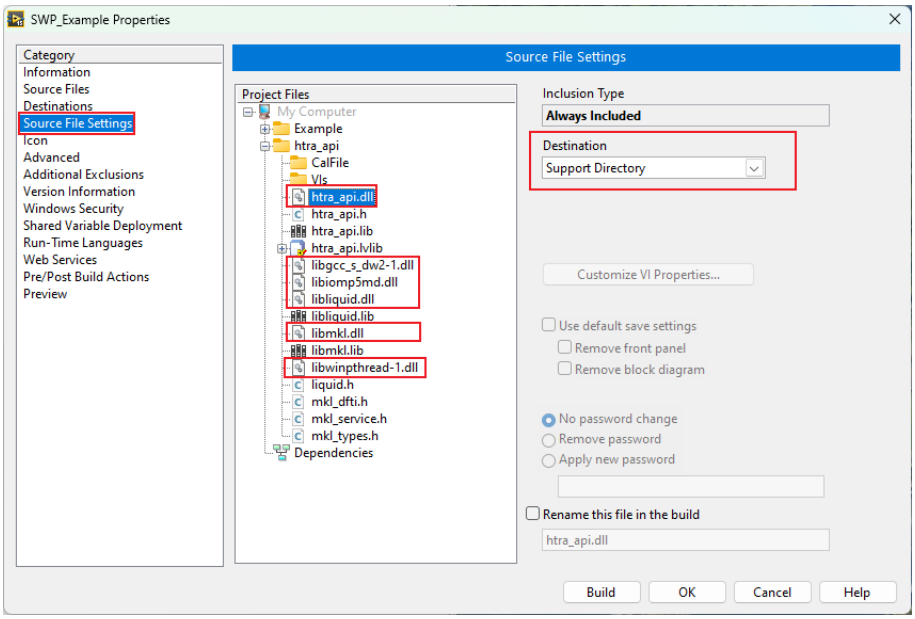
5. 添加一个 CalFile 文件夹用来放设备的校准文件，并将 CalFile 的目标路径添加到“Support Directory”的目录下。



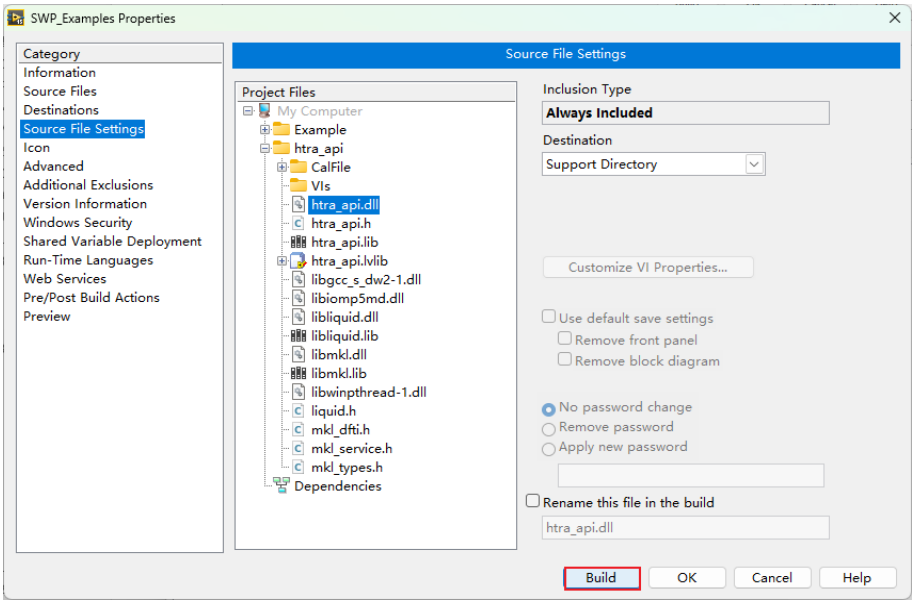
6. 在“Source Files Settings”中将校准文件的目标路径选择到“CalFile”中。



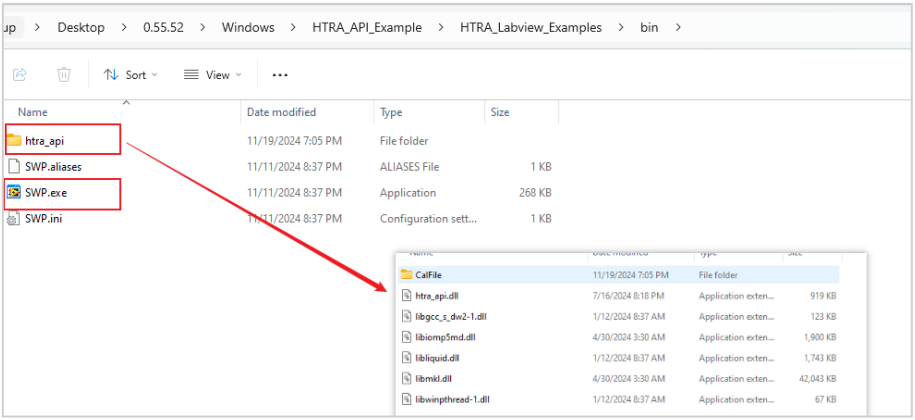
7. 将所有依赖库的目标路径选择到“Support Directory”支持目录中。



8. 点击“Build”。



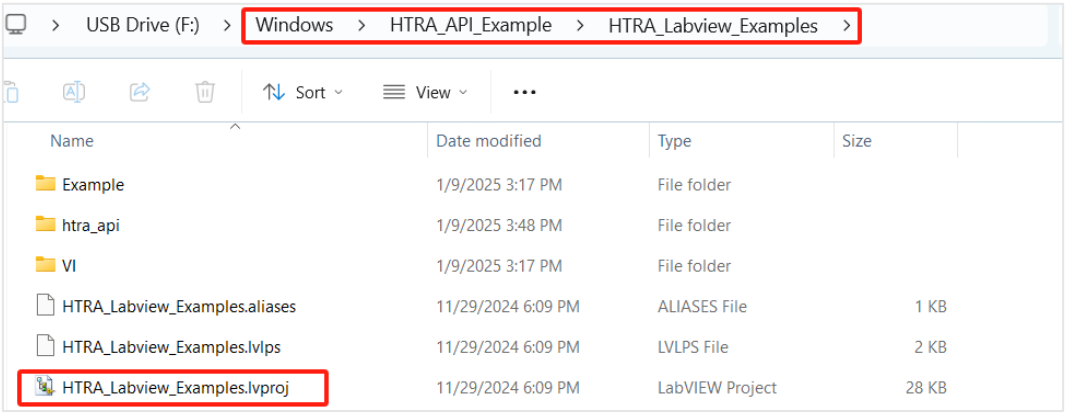
9. 至此，将 Labview 中的 vi 程序生成 exe 程序就完成了。



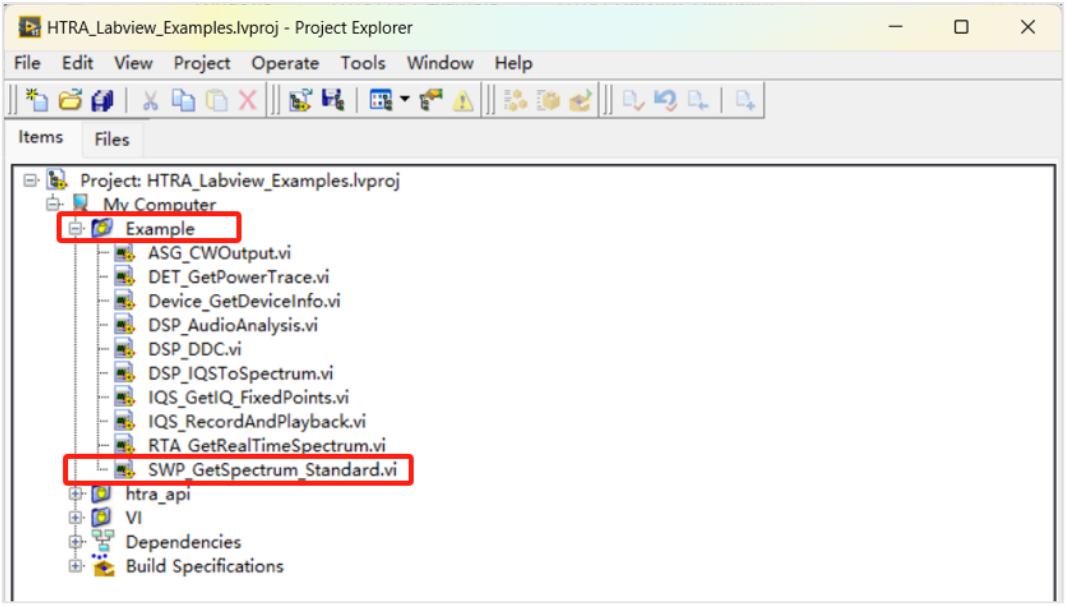
8.2 Labview 范例使用流程

随寄 U 盘中 Labview 范例使用流程如下：

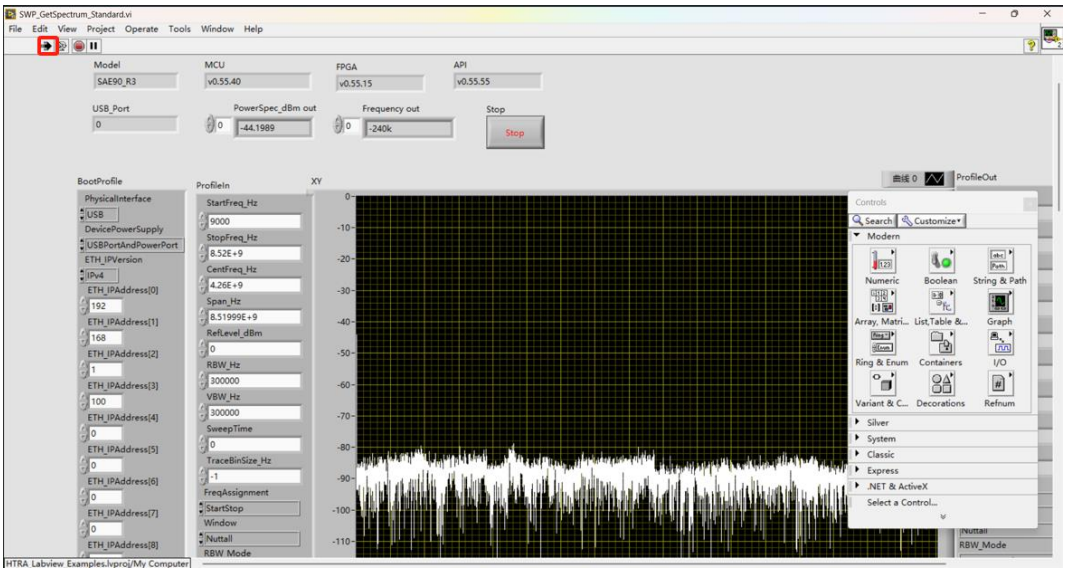
1. 使用 Labview 打开随寄 U 盘 Windows\HTRA_API_Example\HTRA_Labview_Examples 文件夹下工程 HTRA_Labview_Examples.lvproj。



2. 打开工程后，双击打开 Example 文件夹中任意例程 vi。例如此处打开 SWP_GetSpectrum_Standard.vi 使用 SWP 模式例程。



3. 程序打开后，直接点击左上角运行即可，如图所示为程序正常运行。


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
0
PowerSpec_dBm out
-44.1989
Frequency out
-240k
Stop
Stop
BootProfile
PhysicalInterface
USB
DevicePowerSupply
USBPortAndPowerPort
ETH_IPVersion
IPv4
ETH_IPAddress[0]
192
ETH_IPAddress[1]
168
ETH_IPAddress[2]
1
ETH_IPAddress[3]
100
ETH_IPAddress[4]
0
ETH_IPAddress[5]
0
ETH_IPAddress[6]
0
ETH_IPAddress[7]
0
ETH_IPAddress[8]
ProfileIn
StartFreq_Hz
9000
StopFreq_Hz
8.52E+9
CentFreq_Hz
4.26E+9
Span_Hz
8.51999E+9
RefLevel_dBm
0
RBW_Hz
300000
VBW_Hz
300000
SweepTime
0
TraceBinSize_Hz
-1
FreqAssignment
StartStop
Window
Nuttall
RBW Mode
XY
ProfileOut
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 范例说明

8.3.1 获取设备信息

Device_GetDeviceInfo.vi：获取各种设备信息的范例，包括：API 版本、设备型号、设备 UID、MCU 版本、FPGA 版本和设备温度。

8.3.2 标准频谱获取

SWP_GetSpectrum_Standard.vi：获取指定频段内的标准频谱数据并显示图像。

8.3.3 获取固定点数或时长的 IQ 数据

IQS_GetIQ_FixedPoints.vi：获取固定点数的 IQ 数据，当设备接收到 Bus 触发信号后，设备返回固定点数的 IQ 数据。

8.3.4 流盘和读取 IQ 数据

IQS_RecordAndPlayback.vi：获取 IQ 数据、将 IQ 数据记录为 txt 文件以及回放记录的 IQ 数据。

8.3.5 IQ 转频谱数据

DSP_IQSToSpectrum.vi：获取 IQ 数据并将获取的 IQ 数据转为频谱数据。

8.3.6 数字下变频

DSP_DDC.vi：对 IQ 数据流进行数字下变频并重采样生成子 IQ 流进行进一步的频谱分析。

8.3.7 音频分析

DSP_AudioAnalysis.vi：分析音频的音频电压(V)、音频频率(Hz)、信纳德(dB)和总谐波失真(%)。

8.3.8 获取固定点数或时长的检波数据

DET_GetPowerTrace.vi：获取固定点数的 DET 数据。当设备接收到 Bus 触发信号后，设备返回固定点数的 DET 数据。

8.3.9 获取固定点数或时长的实时频谱数据

RTA_GetRealTimeSpectrum.vi：获取固定点数的 RTA 数据。当设备接收到 Bus 触发信号后，设备返回固定点数的 RTA 数据。

8.3.10 ASG 信号源输出信号

ASG_CWOutput.vi：控制设备内部信号发生器输出单音信号、扫频信号和功率扫描信号。

9. Linux

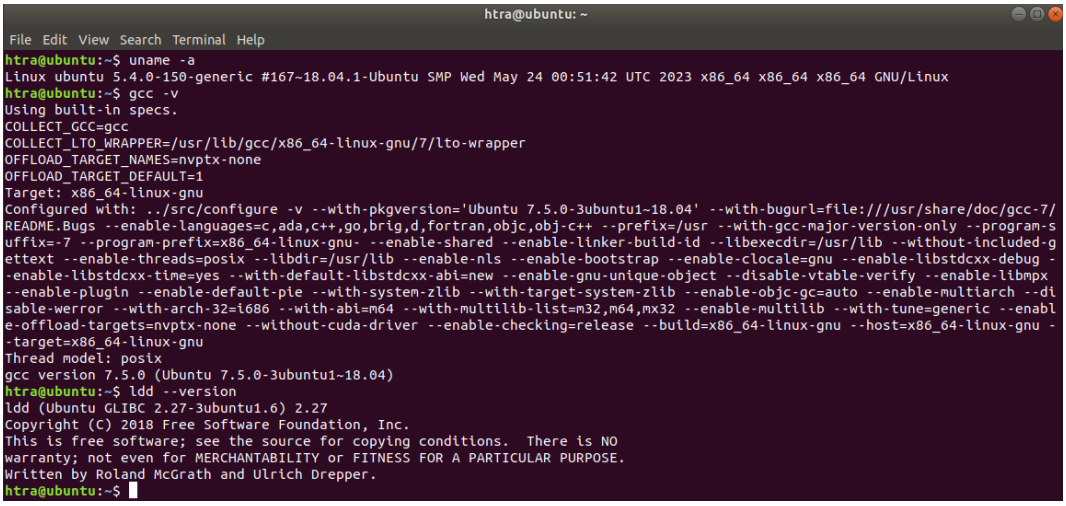
9.1 环境版本适配性自查

在 Linux 系统中使用设备时，首先需要按照下述流程确认当前 Linux 系统的系统架构、gcc 版本以及 GLIBC 版本是否已支持：

1. 打开终端，输入“uname -a”，查看 Linux 系统架构，例如此处 Linux 系统架构为 x86_64。

2. 在终端输入“gcc -v”，查看系统 gcc 版本，例如此处 gcc 版本为 7.5.0。

3. 在终端输入“ldd --version”查看系统 GLIBC 版本，例如此处 GLIBC 版本为 2.27。



4. 根据终端信息对比表格确认当前环境是否已支持，若尚未支持，请联系技术支持人员。

X86 处理器	支持 intel 与 AMD 处理器
ARM 处理器	aarch64（armv8）、armv7 处理器，例如：树莓派 4b、RK3399、RK3568、RK3588、T507、NIVIDA Jeston TX2
编译环境	gcc4.8、glib2.17 及以上
发行版	树莓派 4b 定制系统、ubuntu 18.04 等

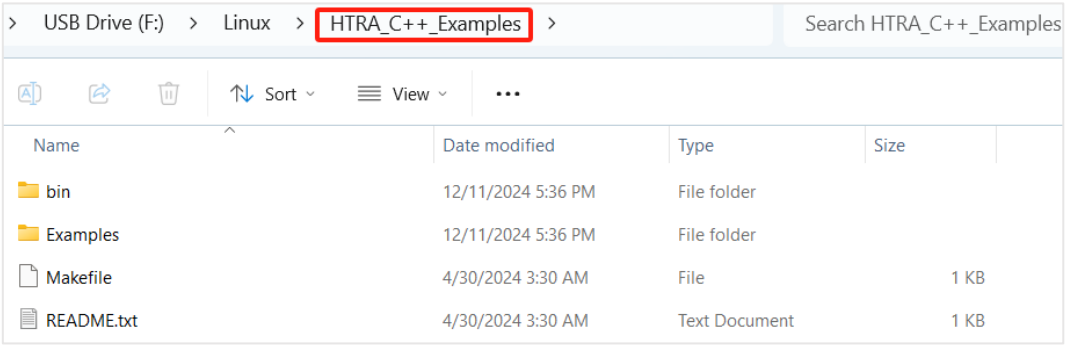
9.2 随寄资料说明

目前随寄 U 盘 Linux 部分包含以下资料：

9.2.1 HTRA_C++_Examples

HTRA_C++_Examples 文件夹包含：

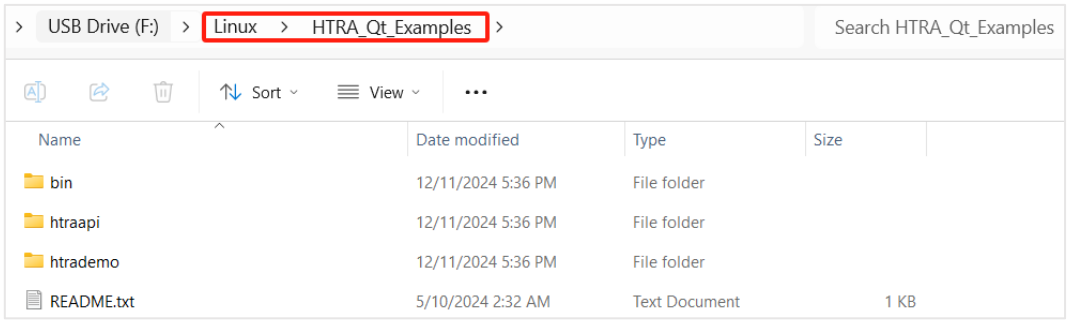
1. Examples 文件夹：C++范例程序（使用方法见章节 9.4）。
2. Makefile 文件：用于将范例程序编译为可执行文件的构建脚本。
3. bin 文件夹：用于存放设备校准文件以及范例程序构建生成的可执行文件。



9.2.2 HTRA_Qt_Examples

HTRA_Qt_Examples 文件夹包含：

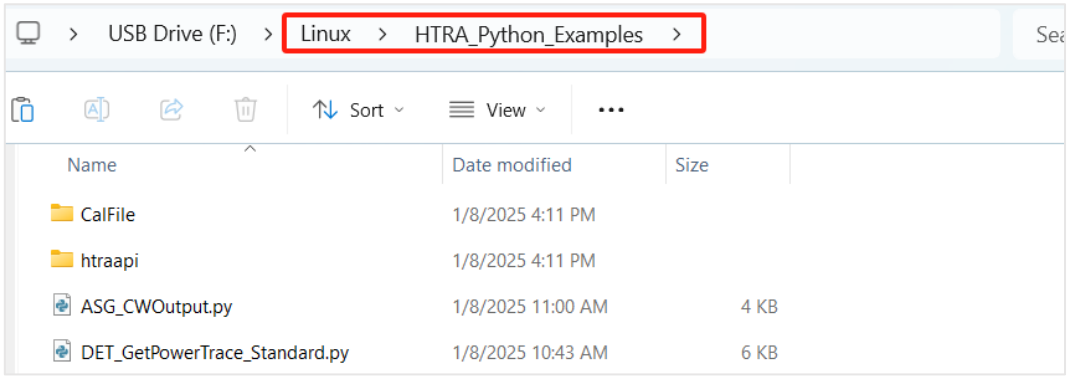
1. htrademo 文件夹：Qt 范例以及 pro 文件（使用方法见章节 9.5）。
2. bin 文件夹：用于存放设备校准文件以及范例程序编译生成的可执行文件。
3. htraapi 文件夹：用于存放动态链接库。



9.2.3 HTRA_Python_Examples

HTRA_Python_Examples 文件夹具体包含：

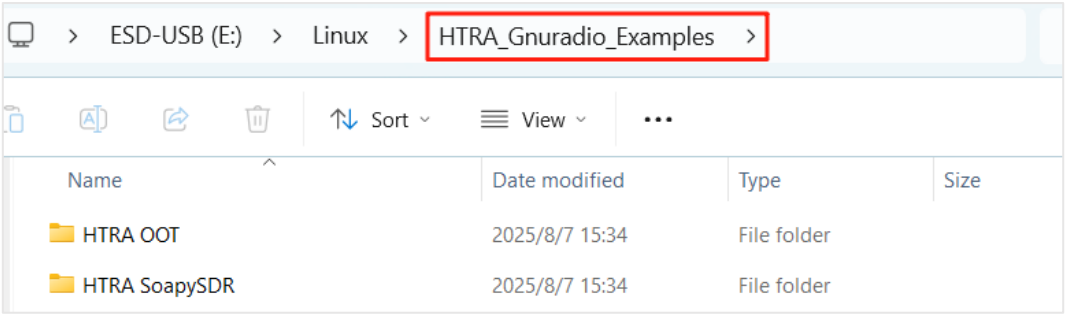
1. Python 范例程序（使用方法见章节 9.6）。
2. CalFile 文件夹：存放设备校准文件。
3. Htraapi 文件夹：用于存放动态链接库。



9.2.4 HTRA_Gnuradio

HTRA_Gnurdaio 文件夹包含两种 Gnuradio 适配方式：

1. HTRA OOT 方式。
2. SoapySDR 适配方式。

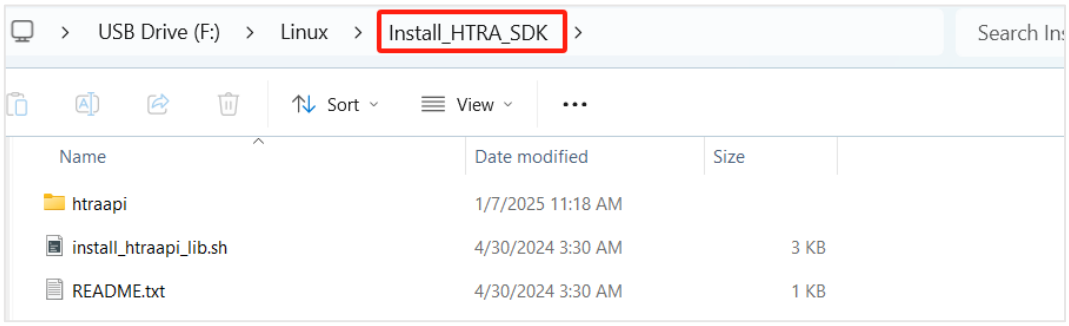


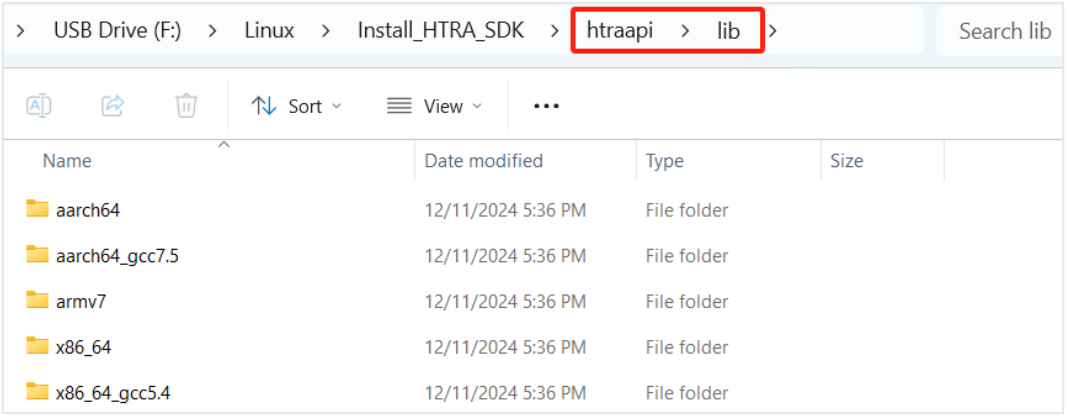
9.2.5 Install_HTRA_SDK

Install_HTRA_SDK 文件夹包含：

1. install_htraapi_lib.sh：驱动配置脚本。
2. Install_HTRA_SDK\htraapi\configs 文件夹：驱动配置文件。
3. Install_HTRA_SDK\htraapi\inc 文件夹：头文件。

4. Install_HTRA_SDK\htraapi\lib\arrch64 文件夹：arrch64 架构动态链接库。
5. Install_HTRA_SDK\htraapi\lib\arrch64_gcc7.5 文件夹：arrch64 架构更高效 FFT 的动态链接库（要求系统 gcc 版本高于 7.5）。
6. Install_HTRA_SDK\htraapi\lib\x86_64 文件夹：x86_64 架构动态链接库。
7. Install_HTRA_SDK\htraapi\lib\x86_64_gcc5.4 文件夹：x86_64 架构更高效 FFT 的动态链接库（要求系统 gcc 版本高于 5.4）。
8. Install_HTRA_SDK\htraapi\lib\armv7 文件夹：armv7 架构动态链接库。

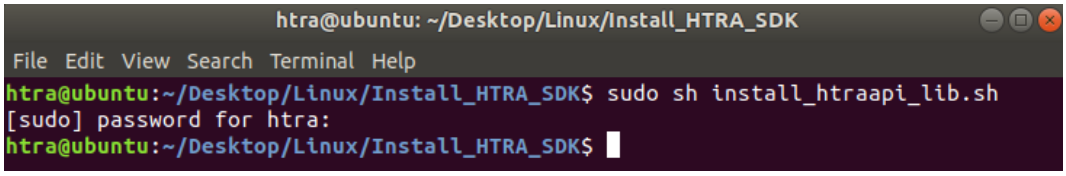


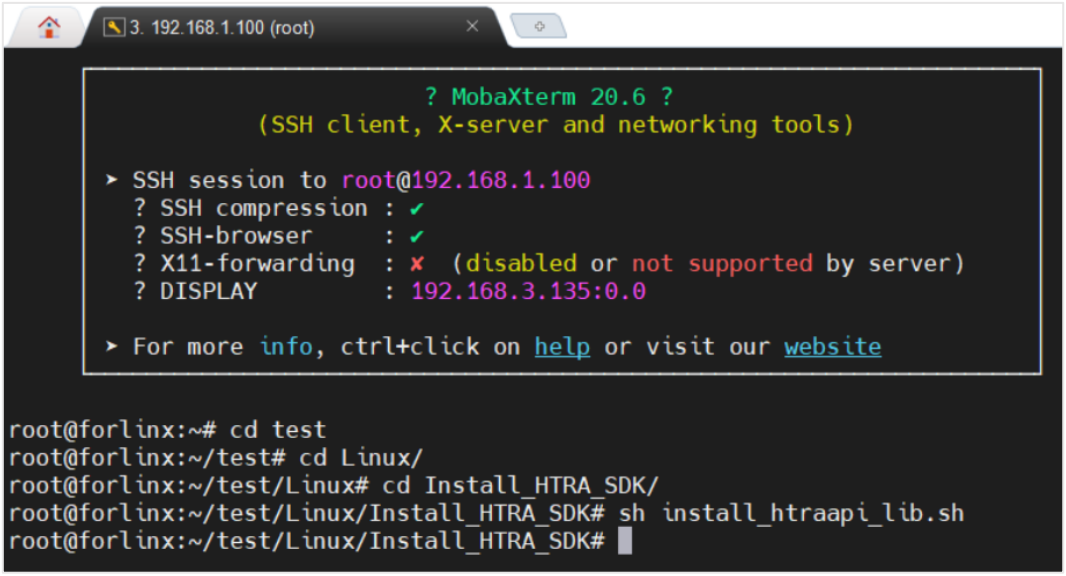


9.3 配置驱动文件

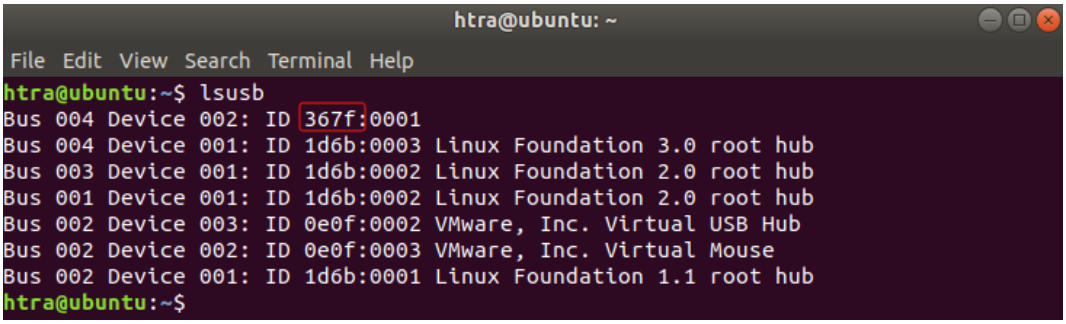
在 Linux 中使用设备必须先进行驱动文件配置，具体流程如下：

1. 配置驱动文件：首先将 Install_HTRA_SDK 文件夹拖入 Linux 上位机，之后在 Install_HTRA_SDK 文件夹下打开终端，输入“sudo sh install_htraapi_lib.sh”配置驱动文件，若特殊开发板无 sudo 命令，输入“sh install_htraapi_lib.sh”即可。





2. 配置情况检查：正确连接设备与上位机（若上位机是虚拟机，需保证设备连接在虚拟机并保证 USB 兼容调整为 USB 3.1）并为设备正常供电，此时如图所示在终端输入“lsusb”，查看本机 USB 设备列表，其中“ID：6430（或 ID：3675 或 ID：04b5）”即为成功接入设备。

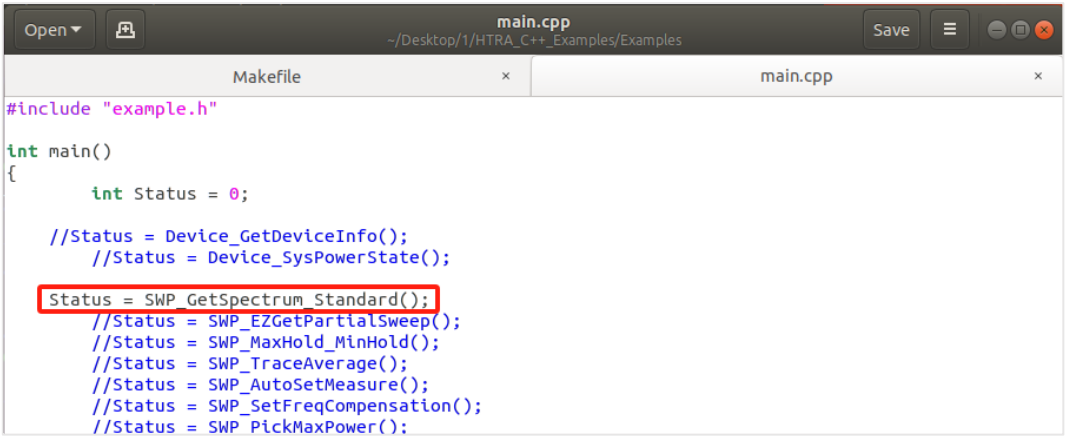


9.4 C++范例使用以及项目创建

9.4.1 C++范例使用

在保证设备正常接入且已按章节 9.3 正确配置驱动文件的前提下，若想要使用随寄 U 盘中 C++范例，可参考以下流程（范例具体作用可直接查看第一章中描述）：

1. 选择需要编译的程序：首先将随寄 U 盘 Linux\HTRA_C++_Examples 文件夹拷贝至上位机。双击打开 Examples 文件夹下 main.cpp，将需要测试使用的范例注释取消，（此处以及后续步骤以 SWP_GetSpectrum_Standard 范例为例）如图所示取消注释。



2. 按章节 9.1 中流程查看系统架构，根据选择测试的范例，在 HTRA_C++_Examples 文件夹下打开终端，按照上位机系统架构

x86_86 系统输入（此处示例例程为 SWP_GetSpectrum_Standard）：

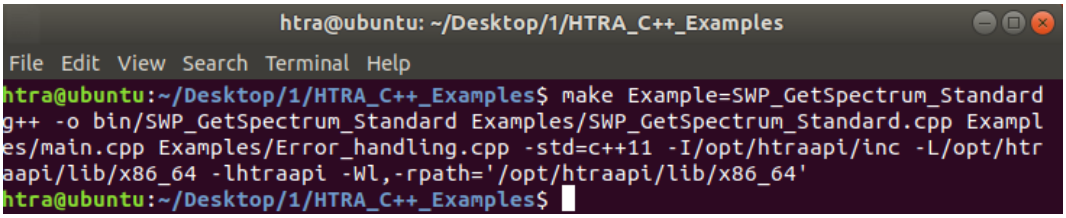
make Example=SWP_GetSpectrum_Standard

aarch64 系统输入：

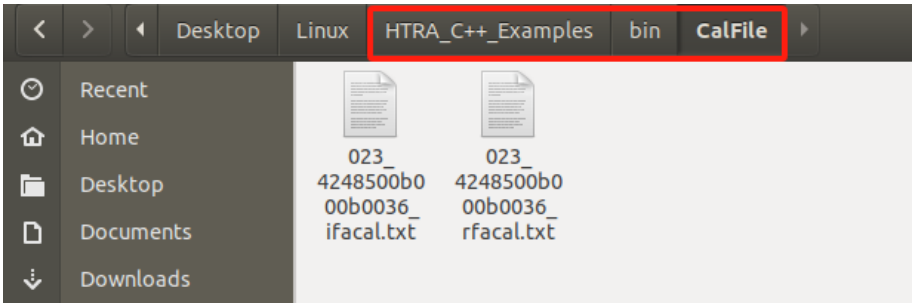
make TARG=aarch64 Example=SWP_GetSpectrum_Standard

armv7 系统输入：

make TARG=armv7 Example=SWP_GetSpectrum_Standard



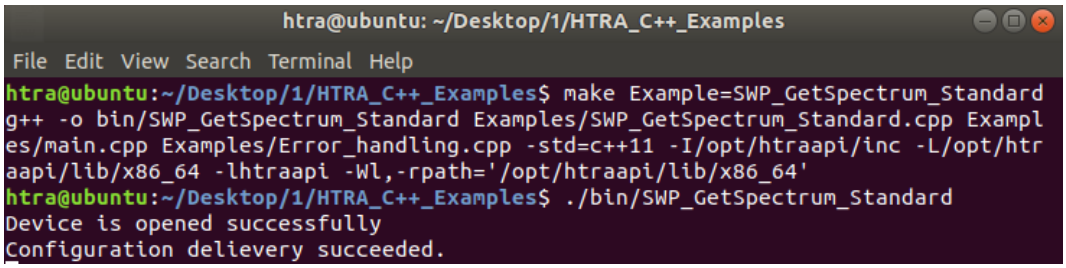
3. 核对校准文件：打开 HTRA_C++_Examples\bin\CalFile 文件夹，如图所示保证文件夹中包含设备校准文件。



4. 运行程序：在程序编译成功并确认校准无误后打开终端输入：

./bin/SWP_GetSpectrum_Standard

如图所示为正常运行范例。



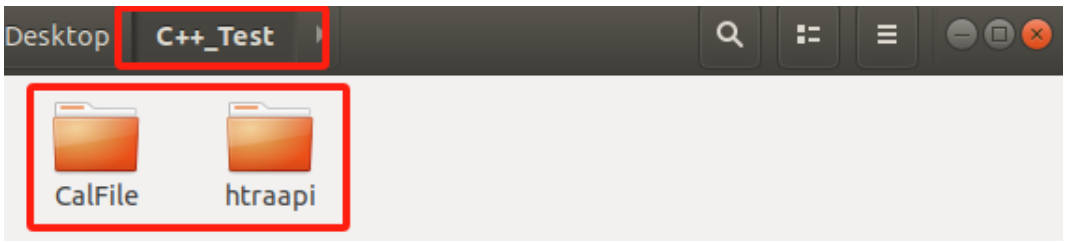
9.4.2 C++项目创建编译

在已按章节 9.3 正确配置驱动文件的前提下，若想要创建 C++项目编译使用，请参考以下流程：

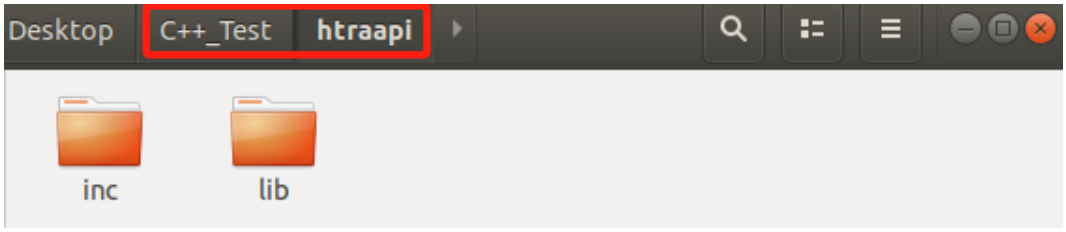
编写代码：因为随寄 U 盘中提供的 Linux 动态链接库与 Windows 中的完全一致，所以代码部分只需符合 API 编程指南即可。

编译运行：

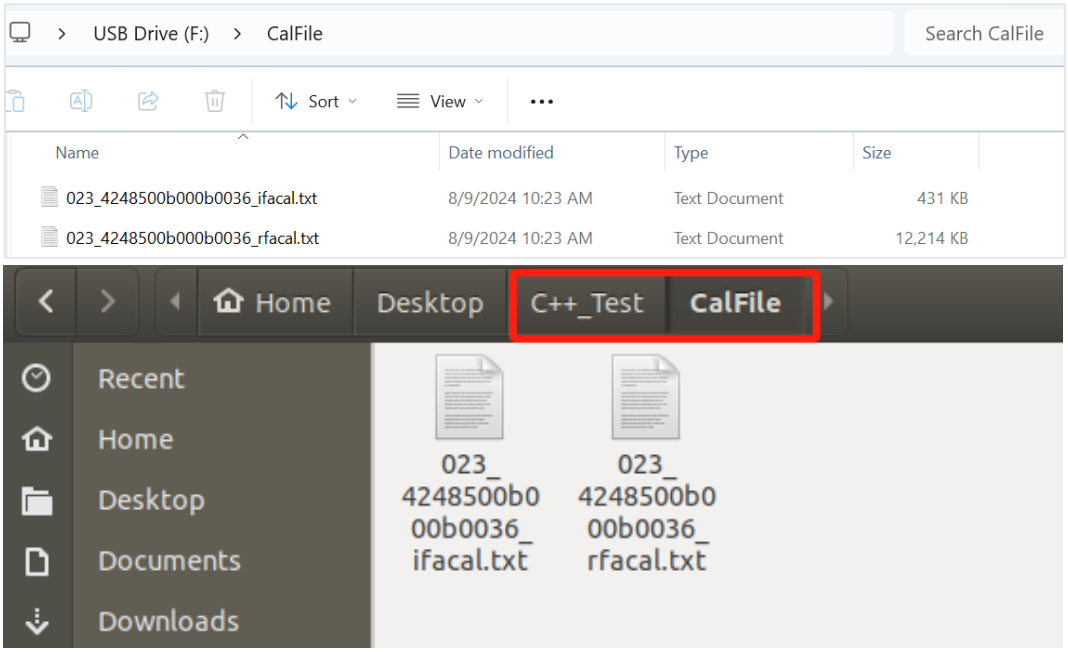
1. 如图所示，首先创建一个新文件夹用于存放整个项目（此处以 C++_Test 为例），然后在文件夹内创建 CalFile 文件夹用于存放校准文件，创建 htraapi 文件夹用于存放头文件以及动态链接库。



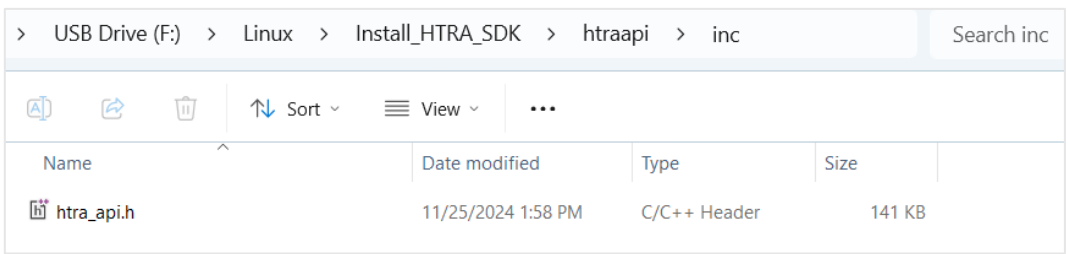
2. 在 htraapi 文件夹下创建 inc 文件夹用于存放头文件，lib 文件夹用于存放动态链接库。

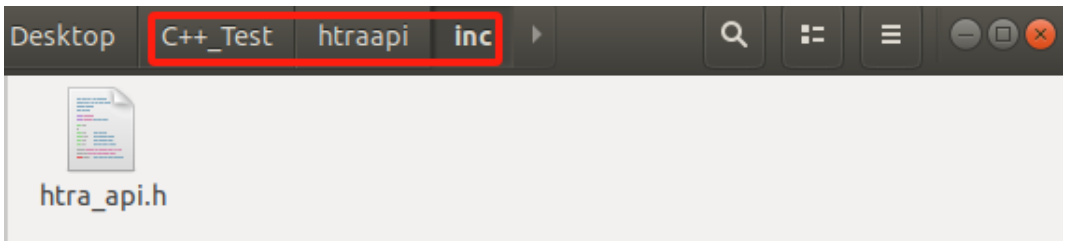


3. 将随寄 U 盘 CalFile 文件夹中的文件复制至刚创建的 C++_Test\CalFile 文件夹中。

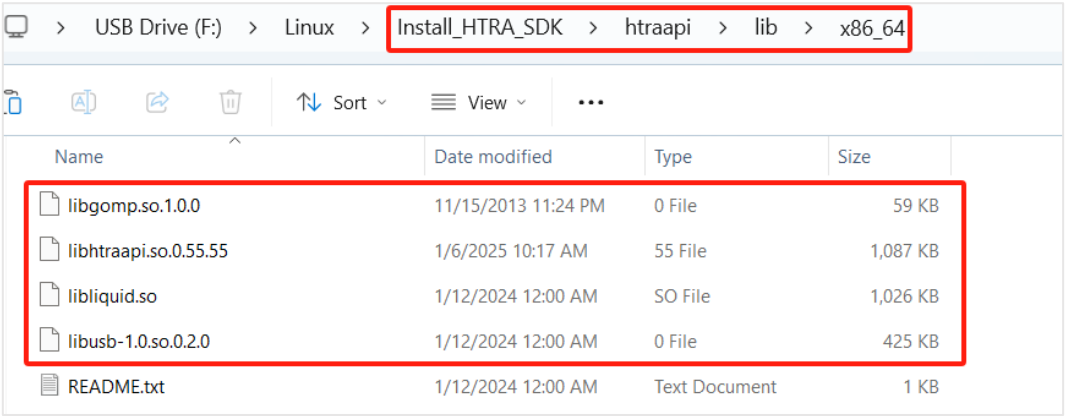


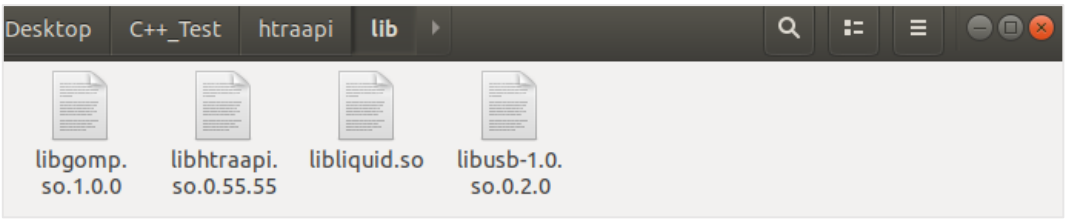
4. 将随寄 U 盘 Linux\Install_HTRA_SDK\htraapi\inc 文件夹中的头文件复制至刚创建的 C++_Test\htraapi\inc 文件夹中。





5. 按照章节 9.1 中流程查看系统架构，然后按照章节 9.2.4 中介绍将随寄 U 盘 Linux\Install_HTRA_SDK\htraapi\lib 文件夹下对应架构的动态链接库复制至刚创建的 C++_Test\htraapi\lib 文件夹中（此处以 x86_64 架构上位机为例）。





6. 在 lib 文件夹位置打开终端输入以下命令，对复制过来的动态链接库进行软链接（三种架构的动态链接库软链接指令无区别）：

ln -sf libhtraapi.so.0.55.55 libhtraapi.so.0（此处 libhtraapi.so 库以 0.55.55 版本为例，其他版本修改版本号即可）

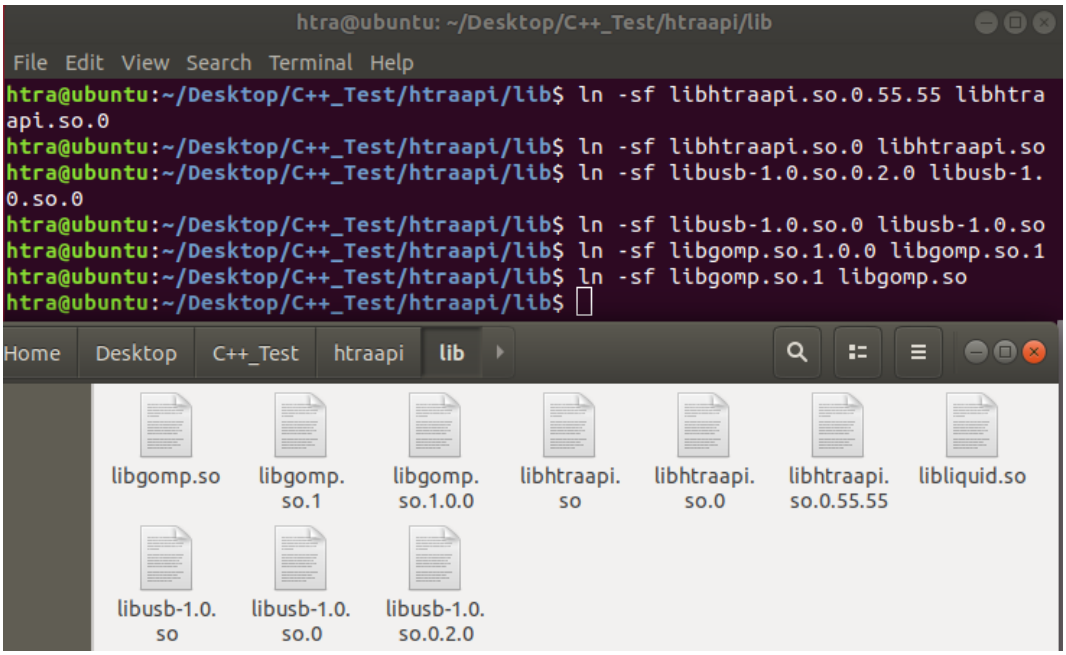
ln -sf libhtraapi.so.0 libhtraapi.so

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

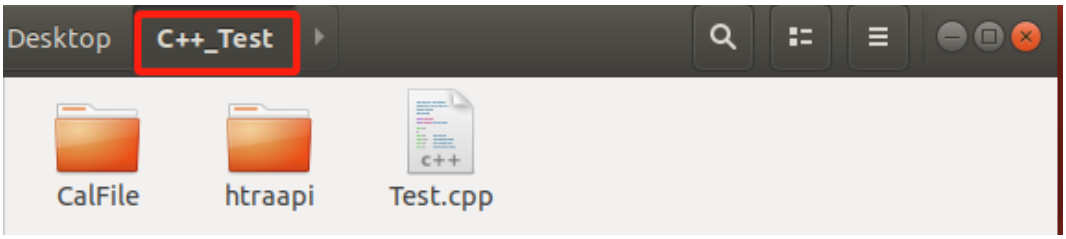
ln -sf libusb-1.0.so.0 libusb-1.0.so

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

ln -sf libgomp.so.1 libgomp.so



7. 将编写好的代码文件存放于 C++_Test 最外层文件夹中。



8. 编译生成可执行文件：首先按照章节 9.1 中流程查看系统架构，然后在 C++_Test 文件夹下打开终端（下图以 x86_64 系统为例），根据上位机系统架构

x86_64 系统输入（此处示例 test.cpp）：

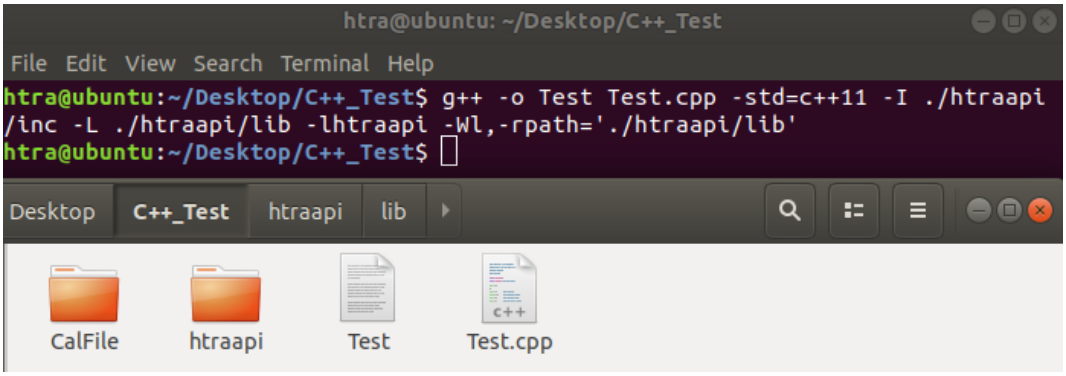
g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L ./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'

arrch64 系统输入：

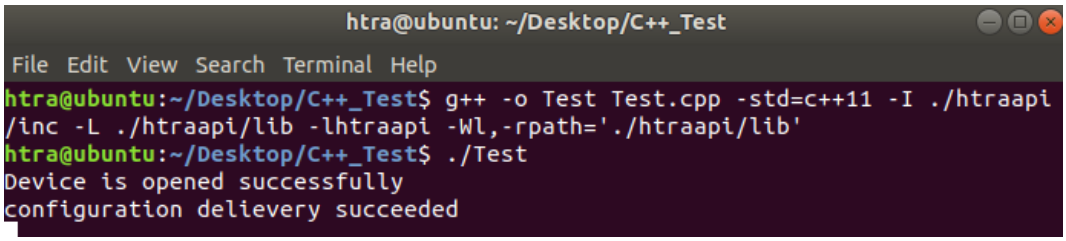
aarch64-linux-gnu-g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L ./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'

armv7 系统输入：

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



9. 运行程序：输入./Test 启动可执行文件。



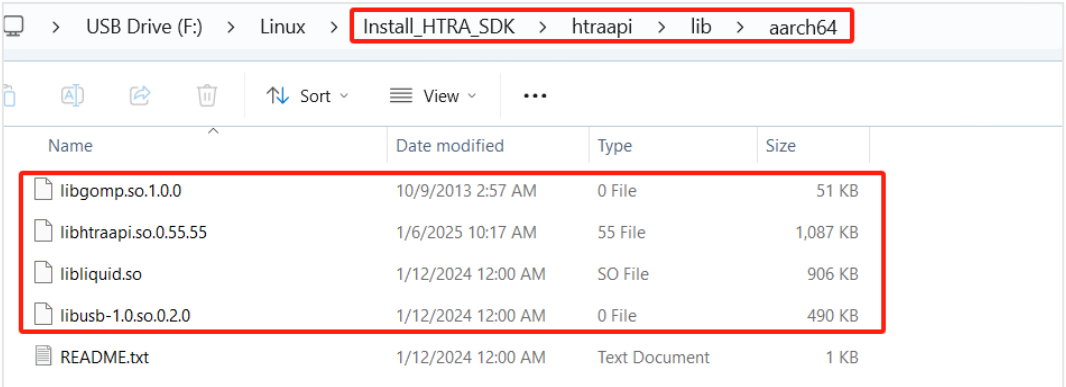
9.4.3 C++项目交叉编译

在上位机有交叉编译链的前提下，若想要交叉编译使用设备，请参考以下流程（此处以在 x86_64 的上位机交叉编译 arrch64 可执行程序为例）：

1. 首先生成目标架构可执行文件：

（1）按照章节 9.4.2 编译运行方法中 1-4 步创建项目并放置校准文件与头文件。

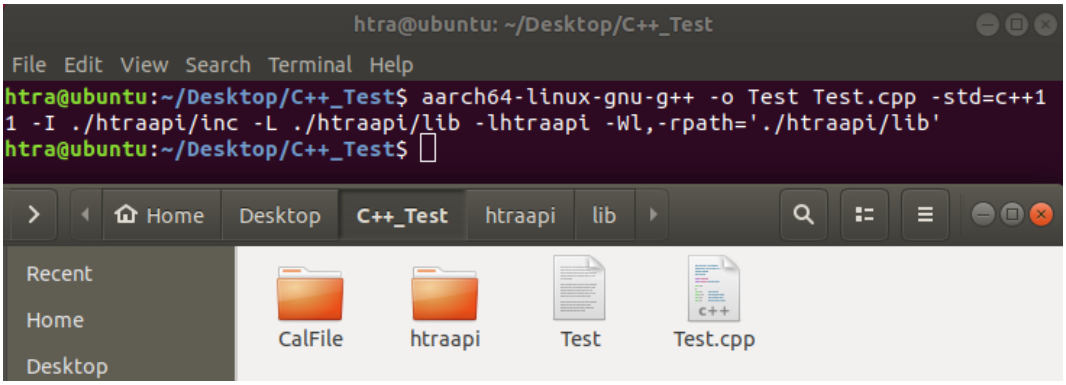
（2）按第 5 步放置交叉编译目标架构库文件。例如预计交叉编译 arrch64 的可执行程序时请放置 arrch64 架构的库文件。



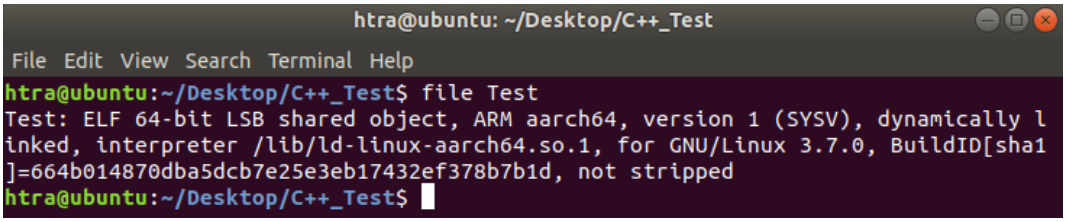
（3）按第 6-7 步进行软链接与程序编写存放。

（4）按第 8 步使用目标架构编译命令编译可执行文件。如下图所示预计编

译 arrch64 架构的可执行文件时使用 arrch64 系统的编译命令。

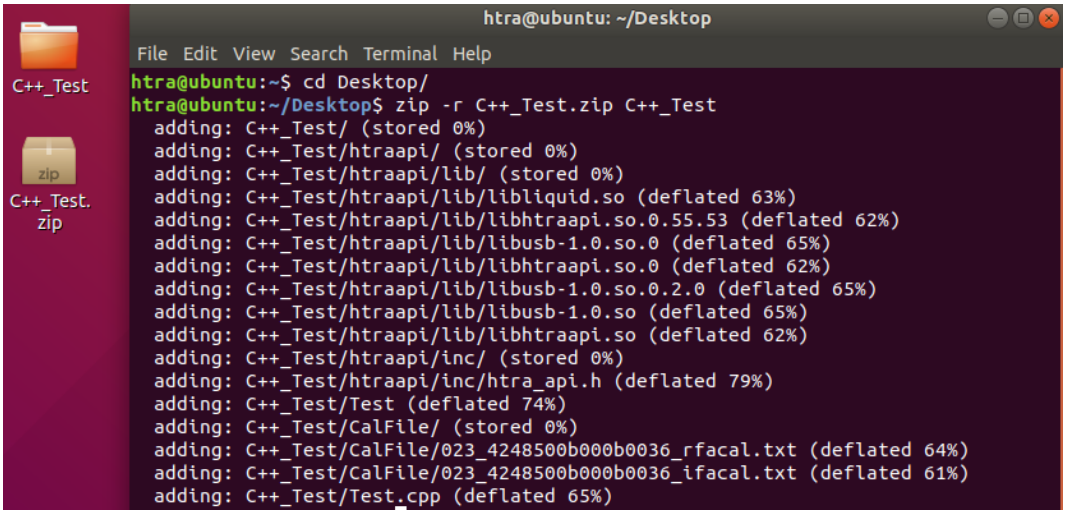


（5）生成可执行文件后，输入 file Test 查看可执行程序的架构，可以看到当前可执行程序架构为 aarch64。

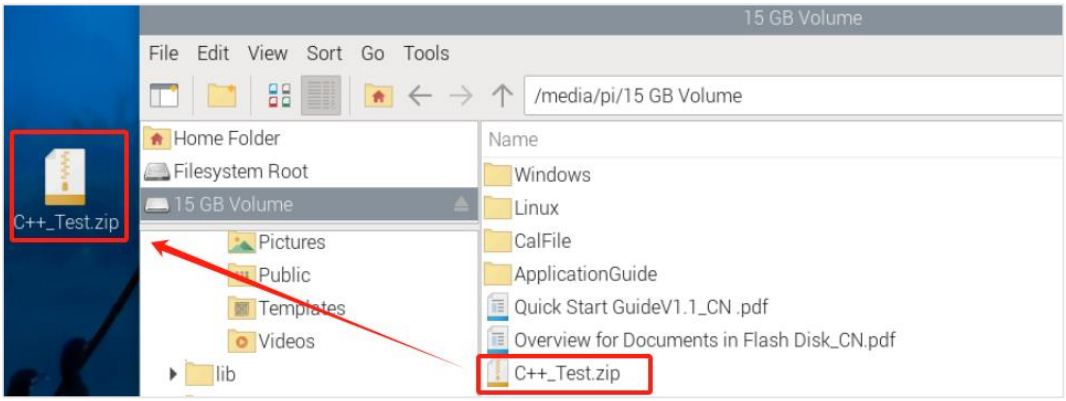


2. 至此可执行程序已成功生成，之后即可在 arrch64 架构上位机运行程序使用设备：

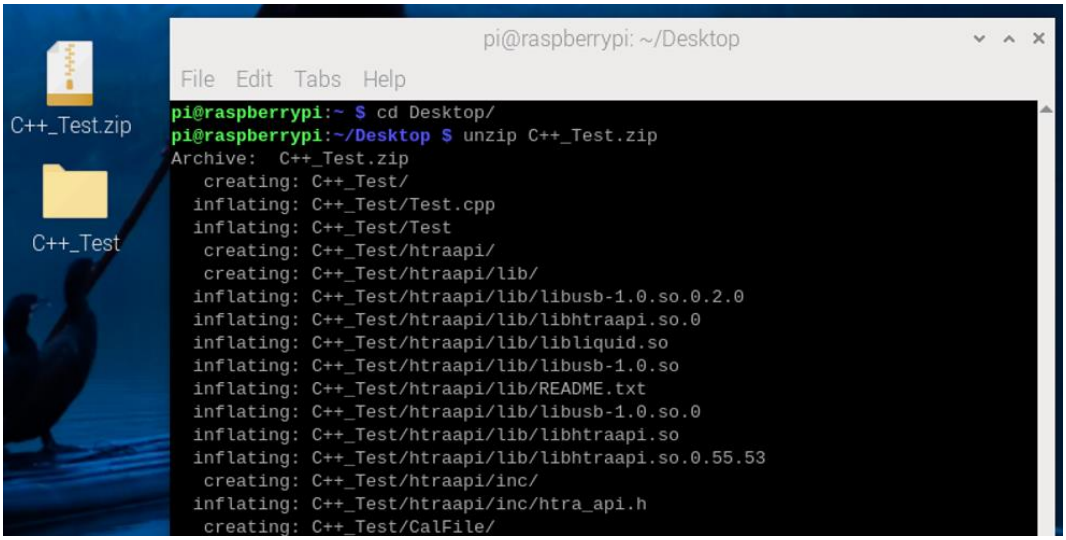
（1）进入项目所在地址（示例项目位于桌面因此输入 cd Desktop/），将整个文件夹压缩为压缩包，例如输入 zip -r C++_Test.zip C++_Test 制作 zip 压缩包。



（2）将压缩包拷贝至 arrch64 上位机中。

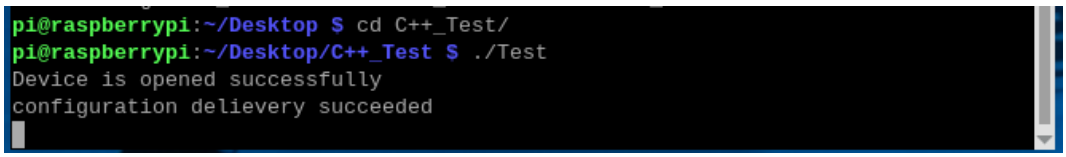


（3）进入压缩包存放位置（此处示例压缩包位于桌面因此输入 cd Desktop/），将项目解压（此处输入 unzip C++_Test）。



（4）按章节 9.3 中步骤在 arrch64 上位机中配置驱动文件。

（5）配置好驱动文件后，输入 cd C++_Test/进入文件夹，之后直接输入./Test 运行程序即可。

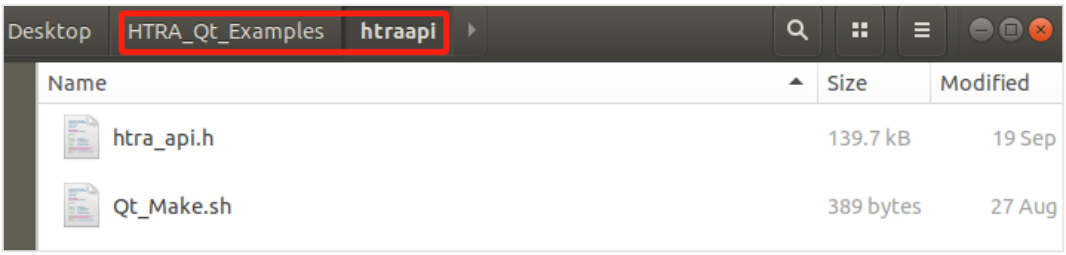


9.5 Qt 范例使用以及项目创建

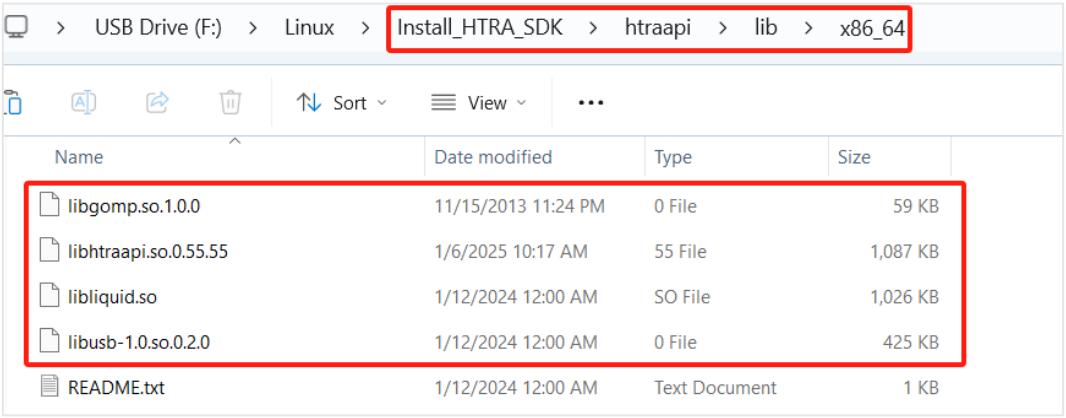
9.5.1 Qt 范例使用

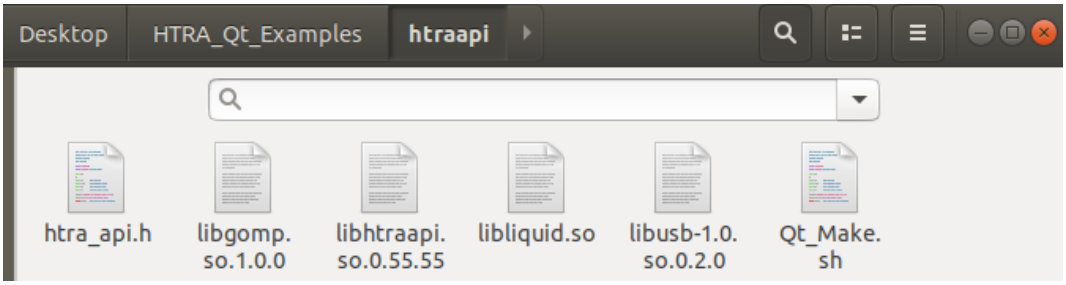
在保证设备正常接入且已按章节 9.3 正确配置驱动文件的前提下，若想要使用随寄 U 盘中 Qt 范例，可参考以下流程（该 Qt 范例的作用为获取指定频段内完整频谱数据）：

1. 将随寄 U 盘 Linux\HTRA_Qt_Examples 文件夹拷贝至上位机，如图所示进入 HTRA_Qt_Examples\htraapi 子文件夹。

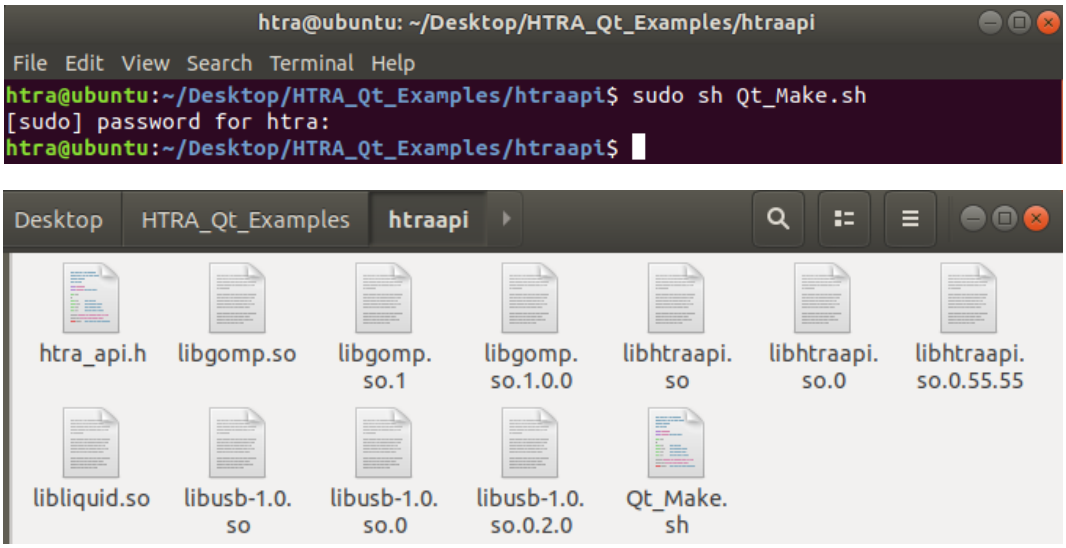


2. 按照章节 9.1 中流程查看系统架构，然后按照章节 9.2.4 中介绍将随寄 U 盘 Linux\Install_HTRA_SDK\htraapi\lib 文件夹下对应系统架构的动态链接库复制至 HTRA_Qt_Examples\htraapi 文件夹中（此处以 x86_64 架构上位机为例）。

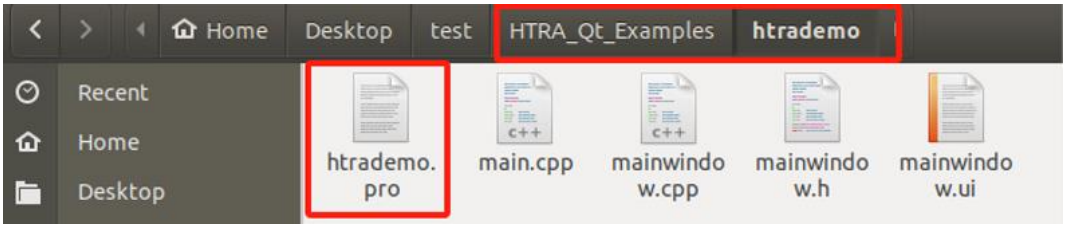




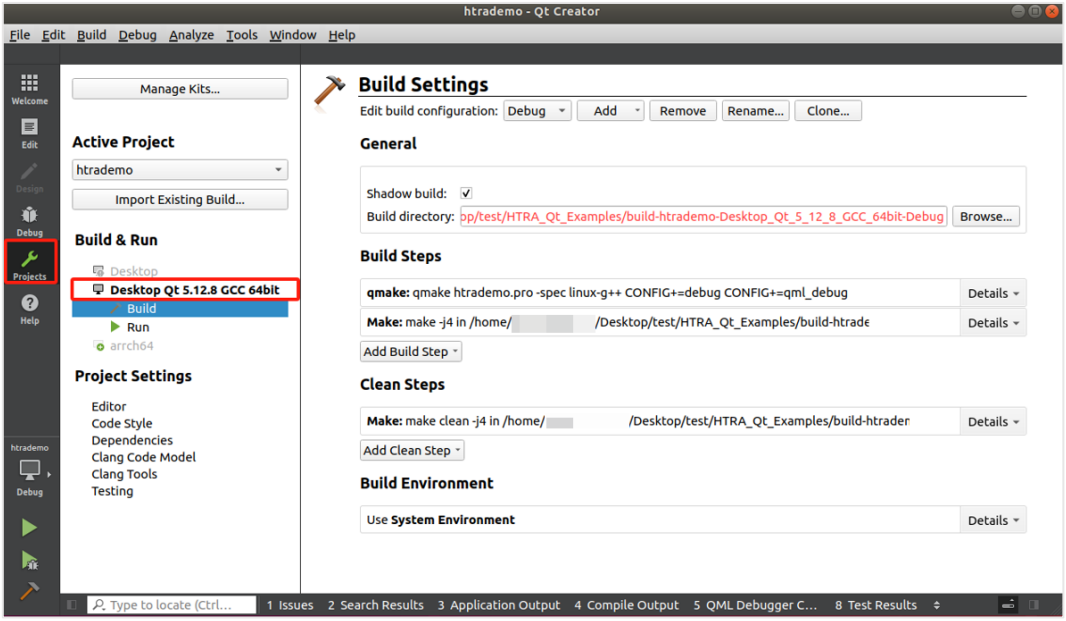
3. 在当前文件夹下打开终端，输入“sudo sh Qt_Make.sh”，之后按提示输入 sudo 密码提供权限对库进行软链接。



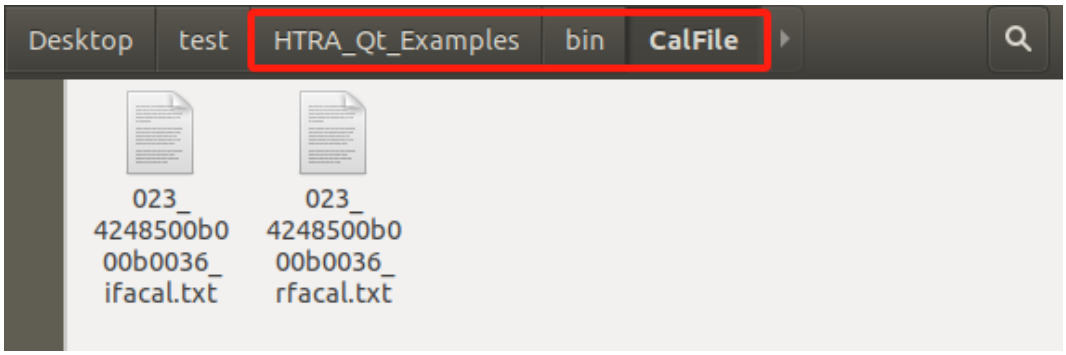
4. 在运行脚本后，使用 qtcreater 打开 htrademo 文件夹中 htrademo.pro 文件。



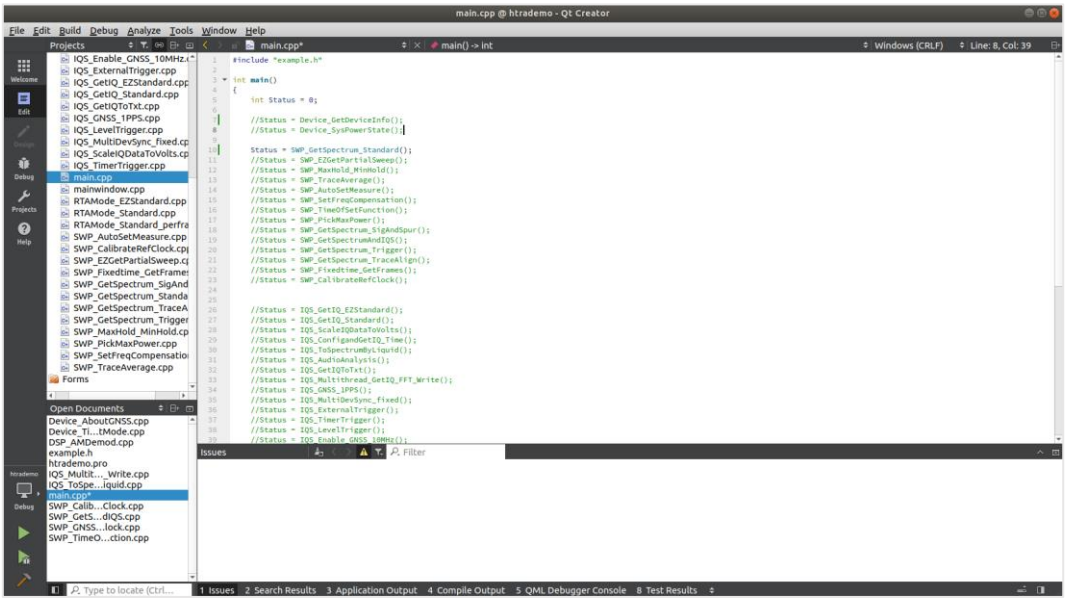
5. 先为项目选择构建环境。



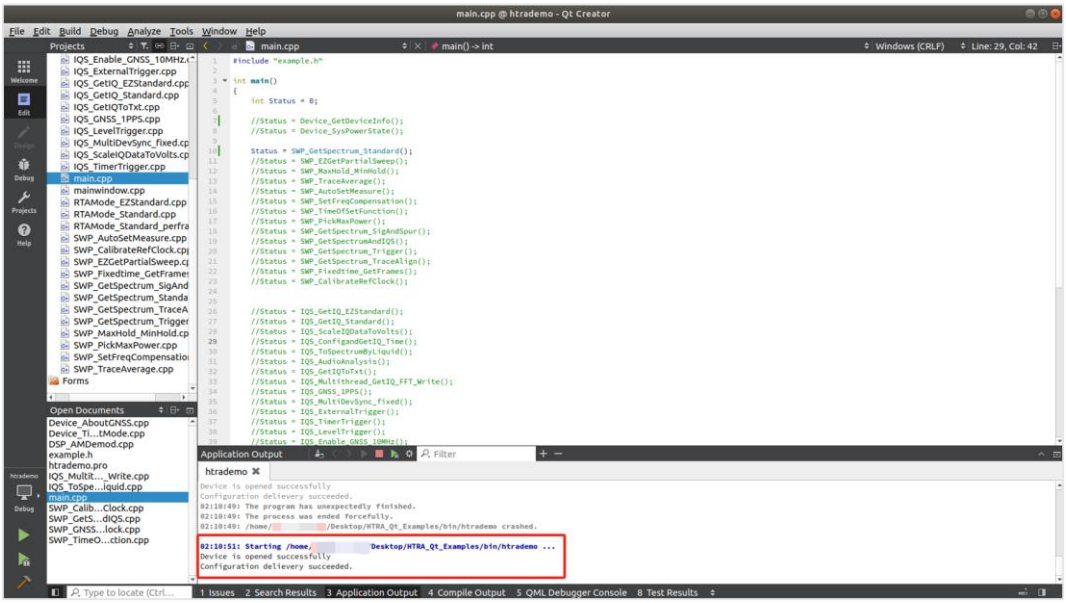
6. 打开 HTRA_Qt_Examples\bin\CalFile 文件夹，确保文件夹中有设备校准文件。



7. 确认有校准文件后，在 main.cpp 中选择任意一个示例。



8. 点击运行，如图所示为设备正常使用。



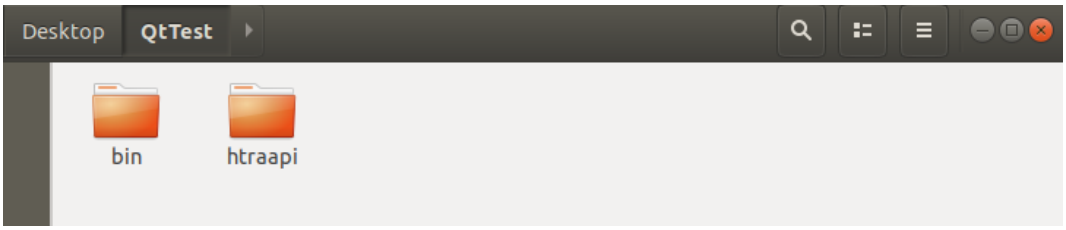
9.5.2 Qt 项目创建编译

在已按章节 9.3 正确配置驱动文件的前提下，若想要创建 Qt 项目编译使用，请参考以下流程：

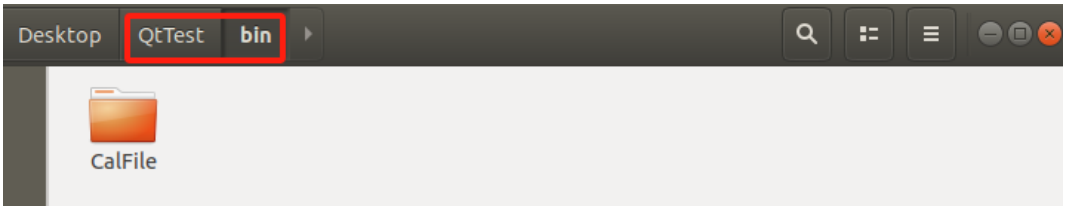
首先在编写代码时，因为随寄 U 盘中提供的 Linux 动态链接库与 Windows 的完全一致，所以代码部分只需符合 API 编程指南即可。

其次在项目创建时，窗体程序创建流程如下：

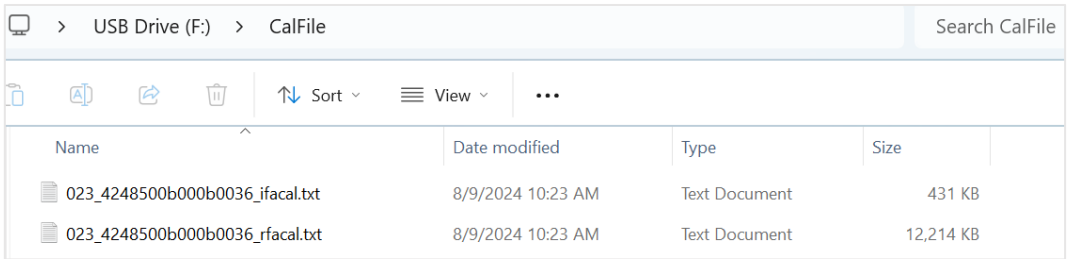
1. 如图所示，首先创建一个新文件夹用于存放整个项目（此处以 QtTest 为例），然后在文件夹内创建 bin 文件夹用于存放校准文件与生成的可执行文件，创建 htraapi 文件夹用于存放头文件以及动态链接库。

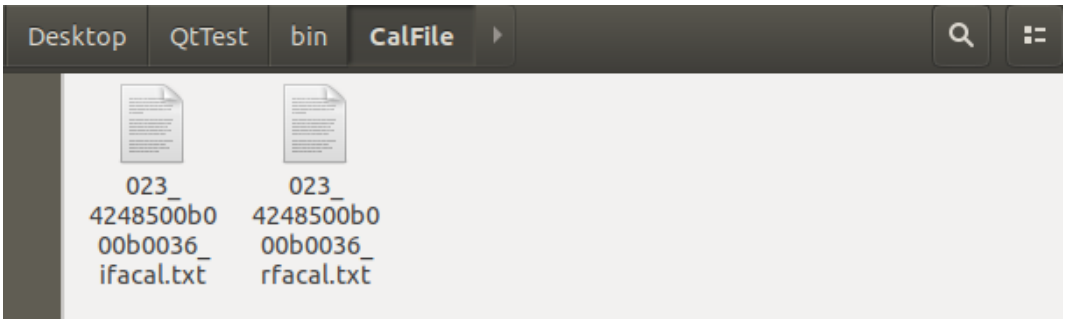


2. 在 bin 文件夹下创建 CalFile 文件夹用于存放设备校准文件。

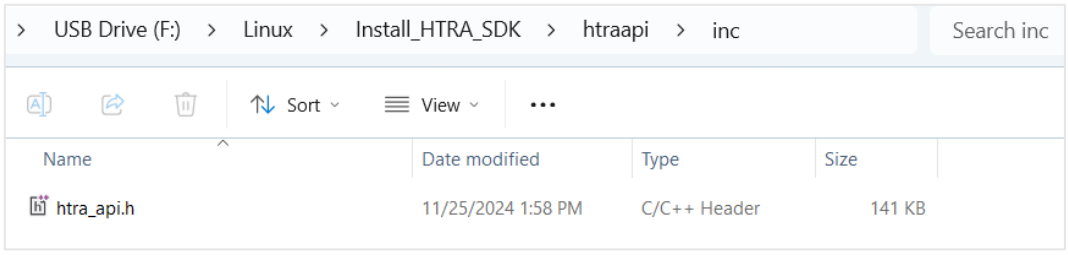


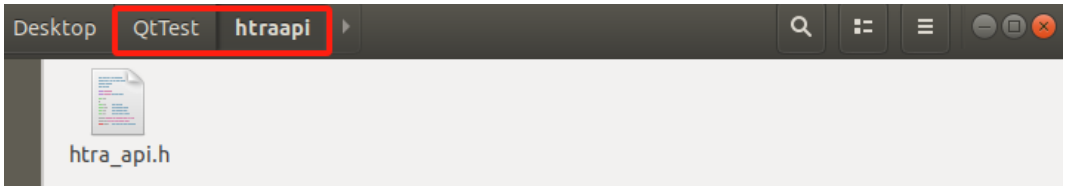
3. 将随寄 U 盘 CalFile 文件夹中的文件复制至刚创建的 QtTest\bin\CalFile 文件夹中。



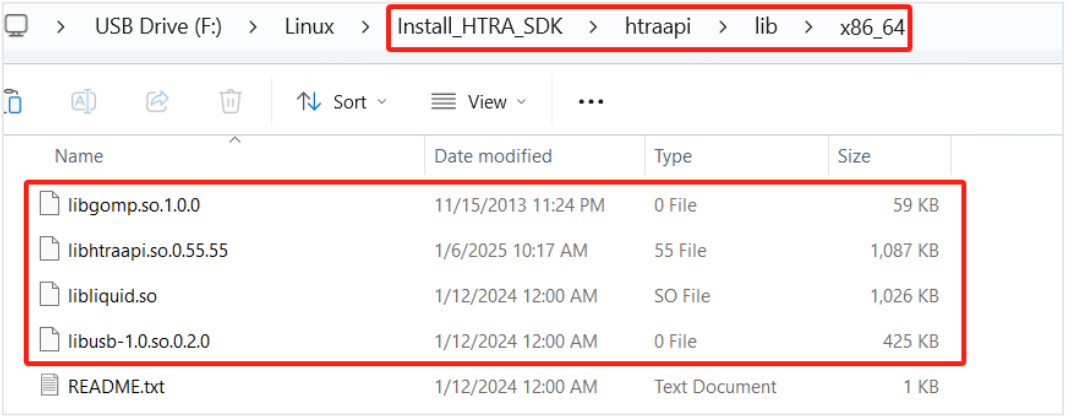


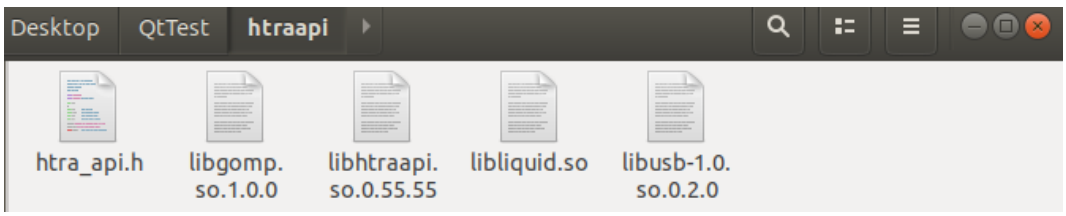
4. 将随寄 U 盘 Linux\Install_HTRA_SDK\htraapi\inc 文件夹中的头文件复制至刚创建的 QtTest\htraapi 文件夹中。





5. 按照章节 9.1 中流程查看系统架构，然后按照章节 9.2.4 中介绍将随寄 U 盘 Linux\Install_HTRA_SDK\htraapi\lib 文件夹下对应系统架构的动态链接库复制至刚创建的 QtTest\htraapi 文件夹中（此处以 x86_64 架构上位机为例）。





6. 在 htraapi 文件夹位置打开终端输入以下命令，对复制过来的动态链接库

进行软链接（不同架构的动态链接库软链接命令无区别）：

ln -sf libhtraapi.so.0.55.55 libhtraapi.so.0（此处 libhtraapi.so 库以 0.55.55 版本为例，其他版本修改版本号即可）

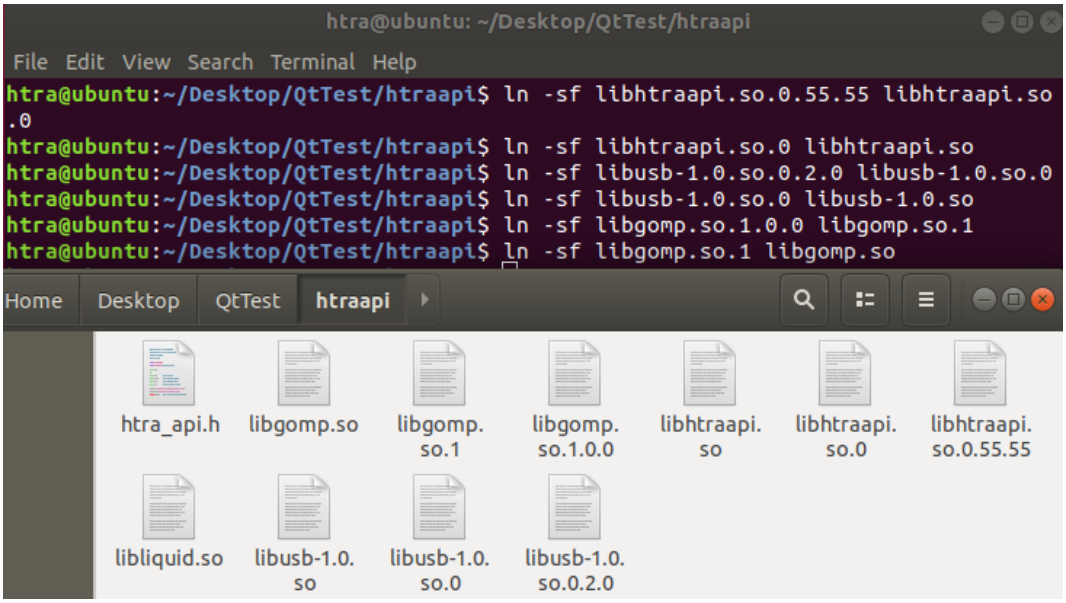
ln -sf libhtraapi.so.0 libhtraapi.so

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

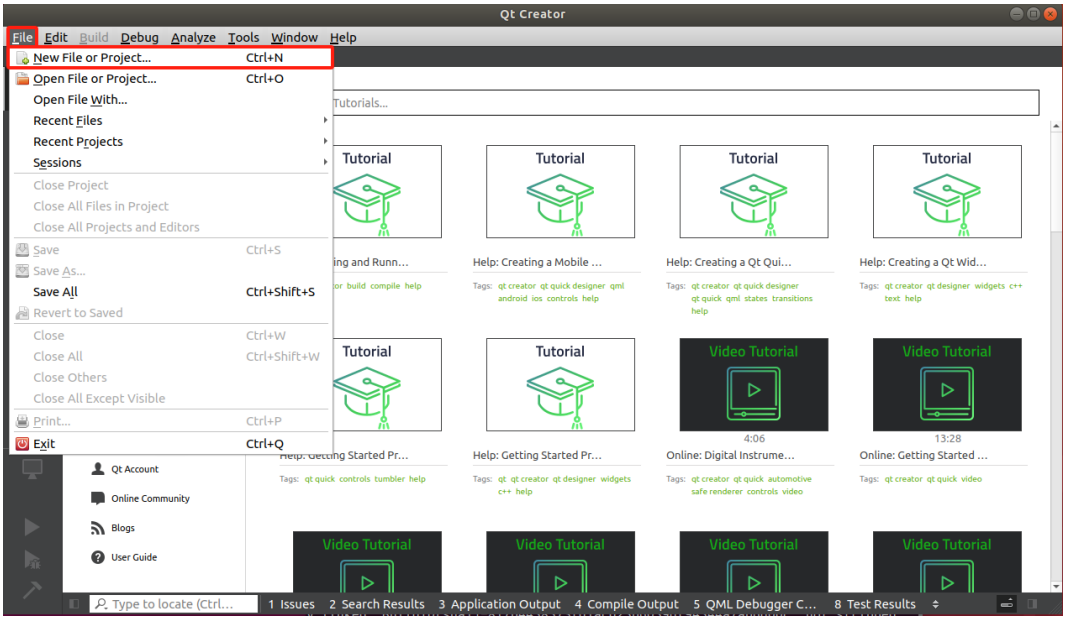
ln -sf libusb-1.0.so.0 libusb-1.0.so

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

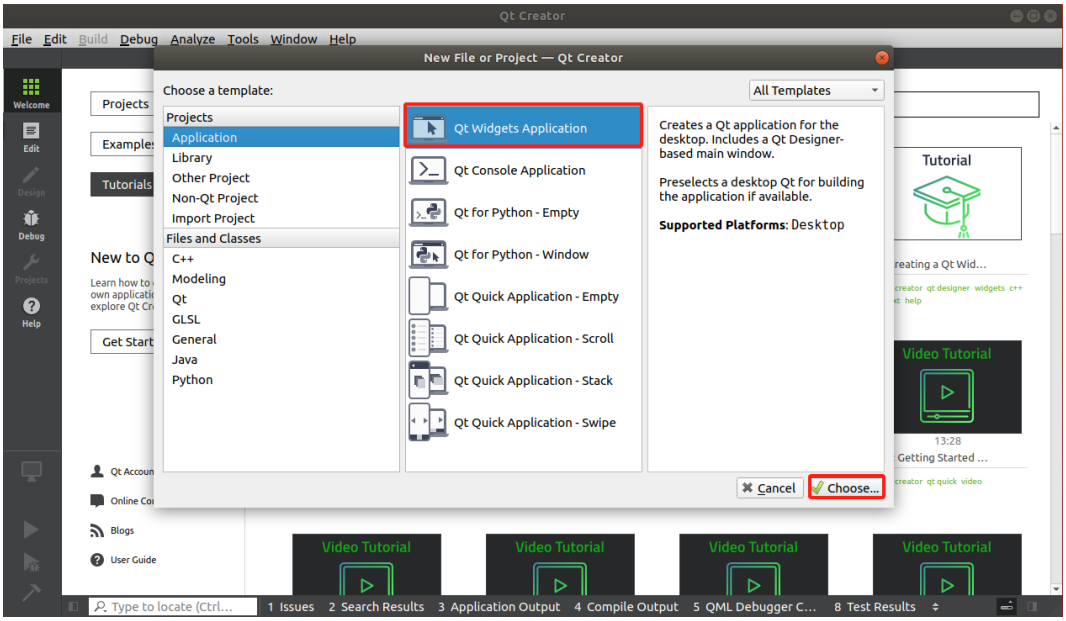
ln -sf libgomp.so.1 libgomp.so



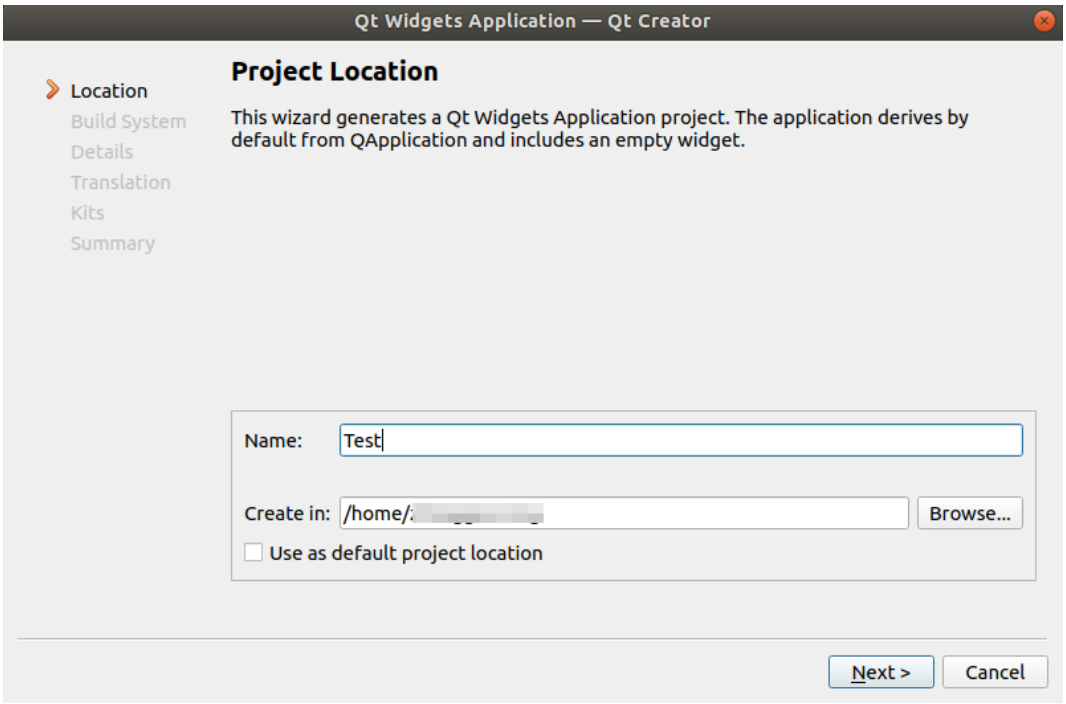
7. 打开 Qtcreater，点击文件，选择新建文件或项目。



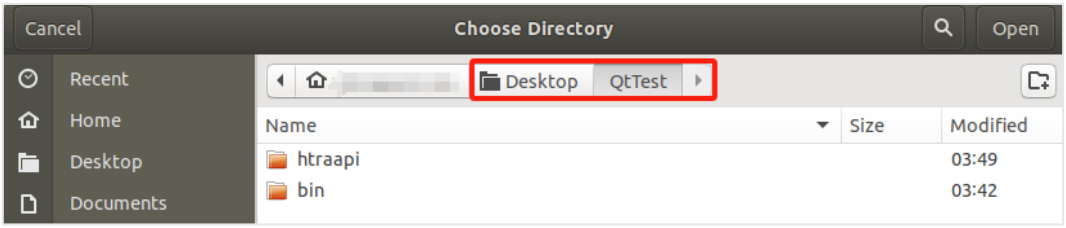
8. 选择创建窗体程序。



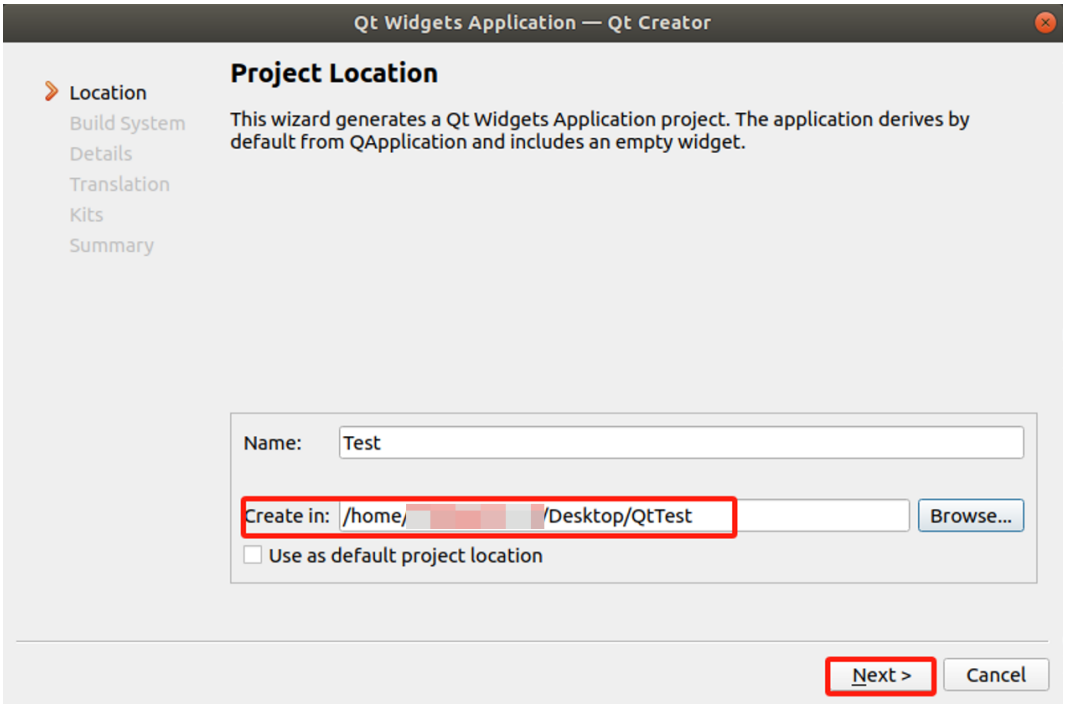
9. 填写项目名称后点击浏览更换项目路径。



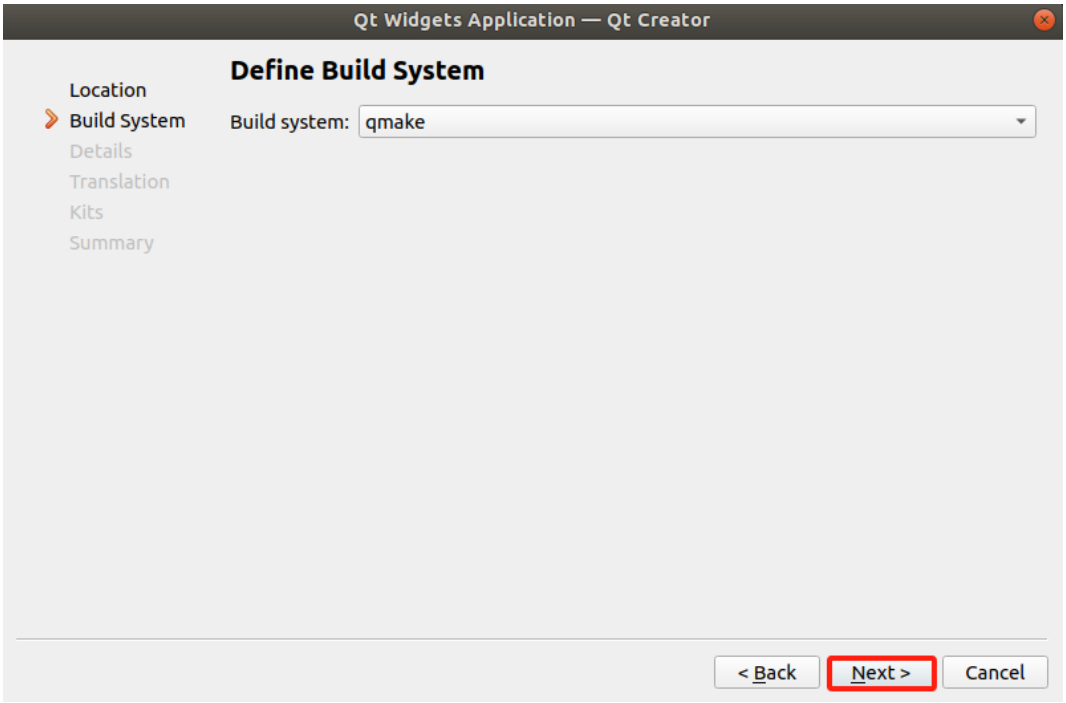
10. 选择目录为前面第一步创建的 QtTest 地址后点击 Open。



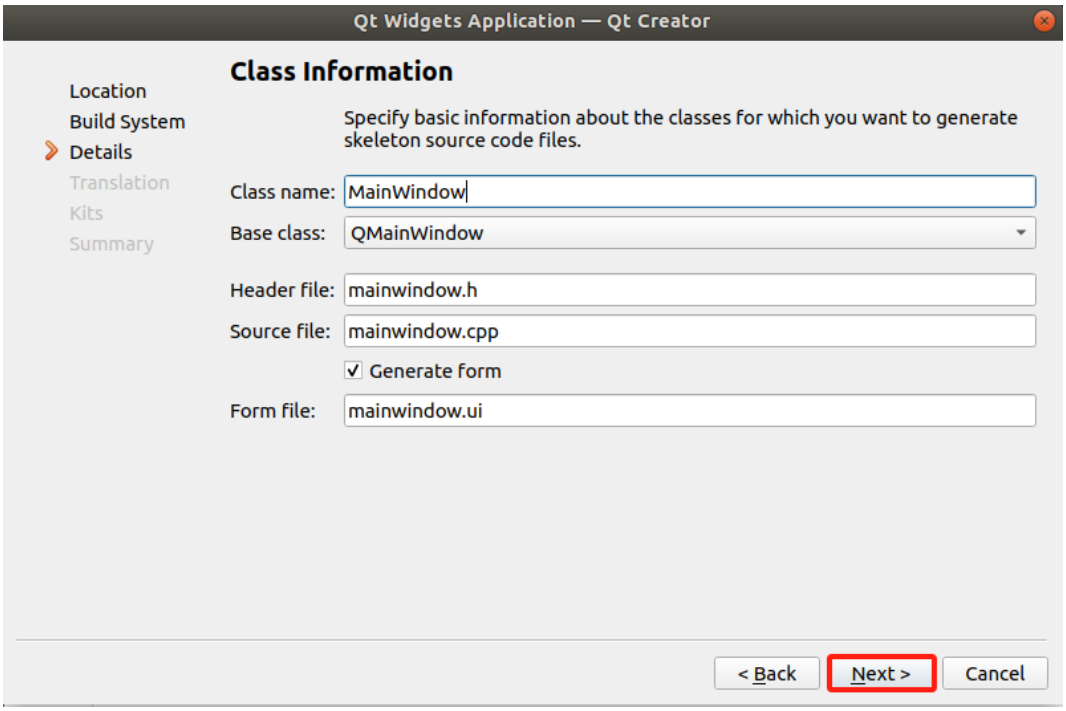
11. 选择好路径后点击下一步。



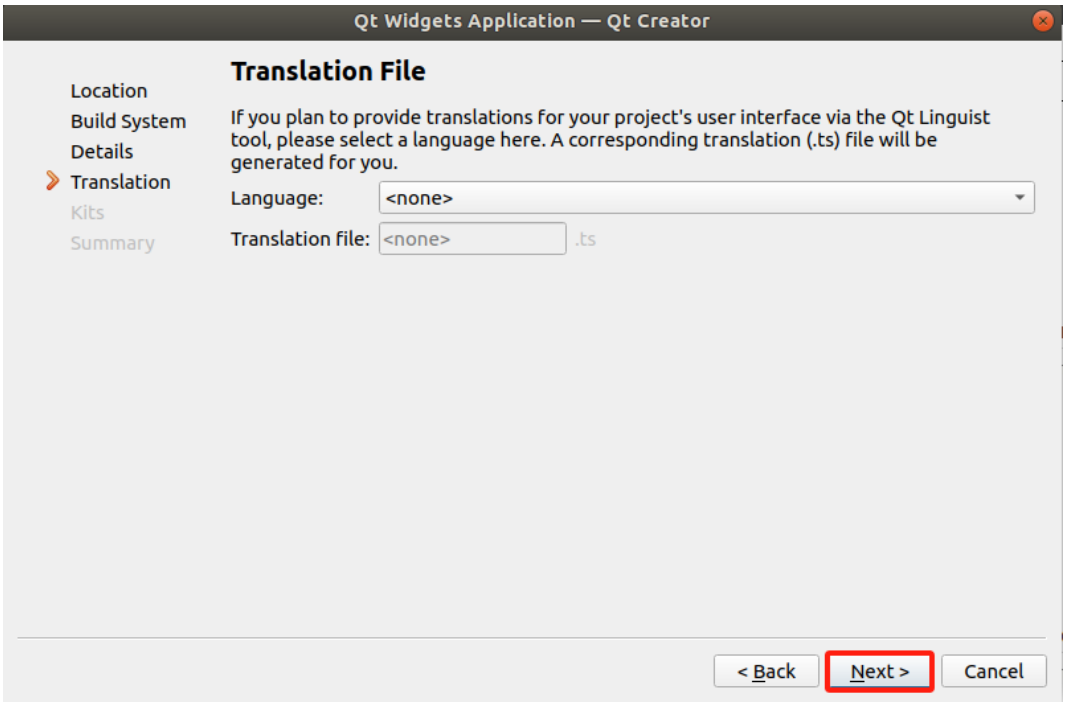
12. 继续点击下一步。



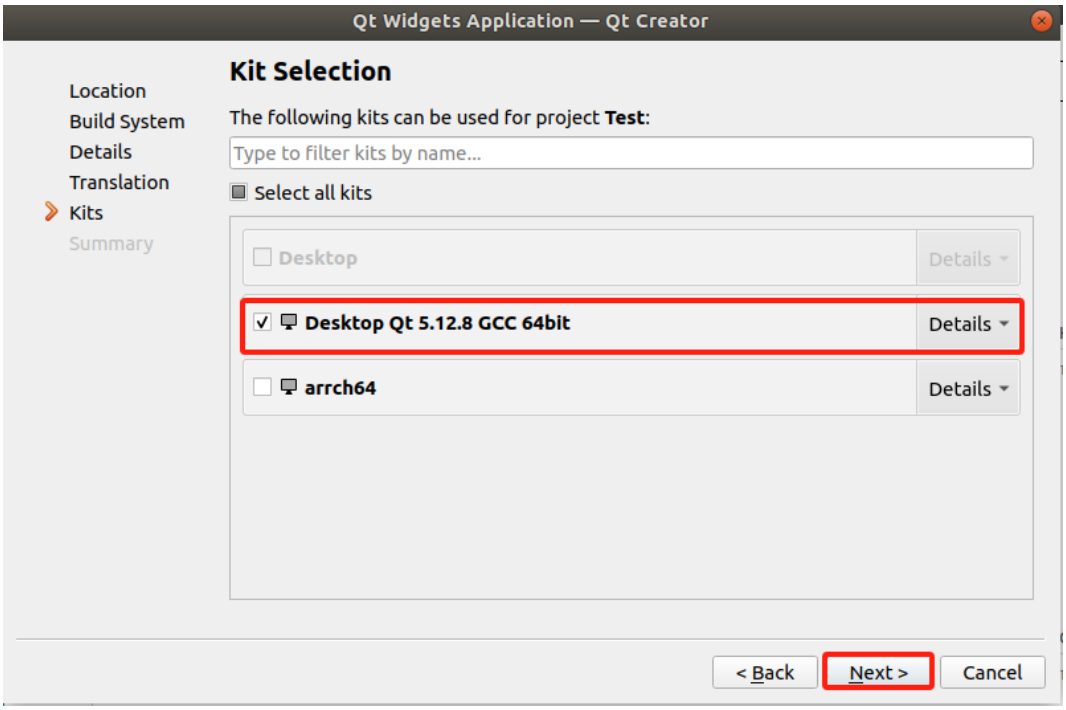
13. 继续点击下一步。



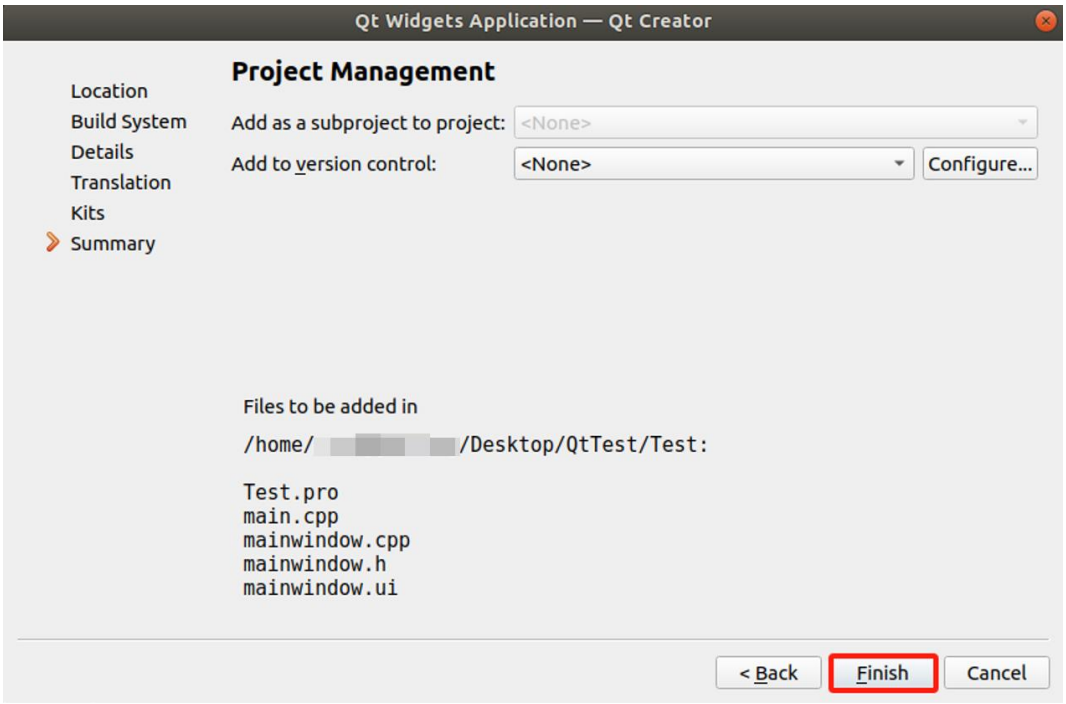
14. 继续点击下一步。



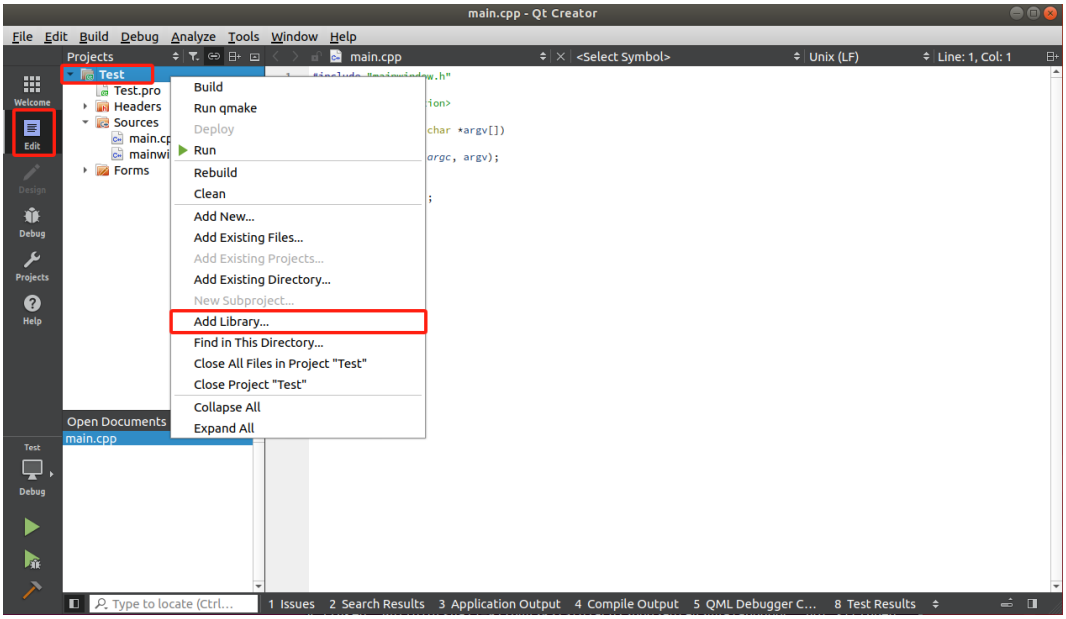
15. 为项目选择 x86_64 构建环境后继续点击下一步。



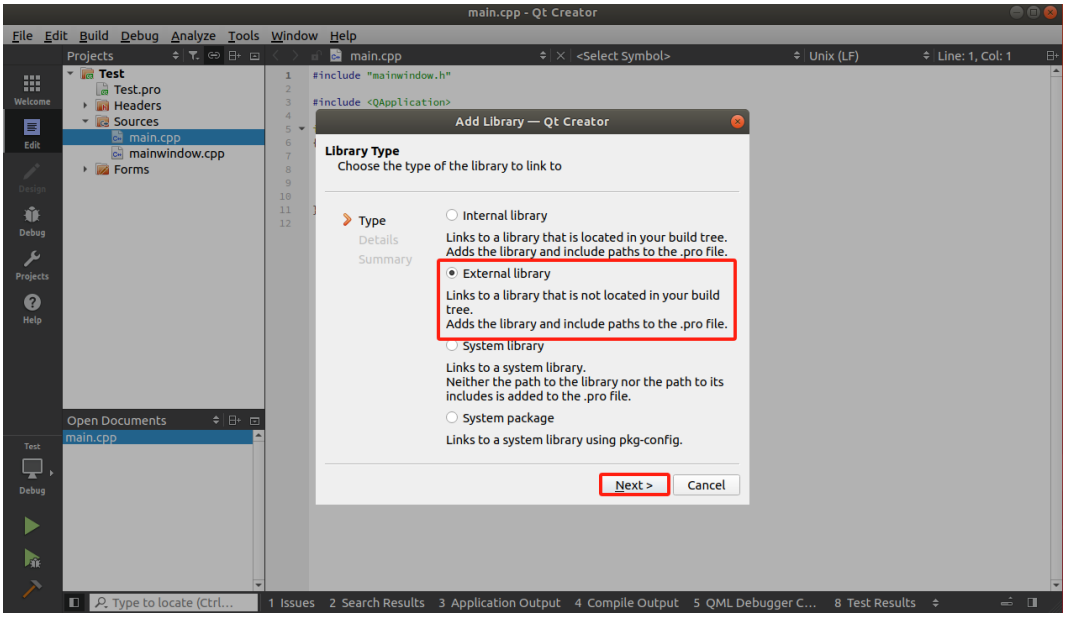
16. 点击完成以创建项目。



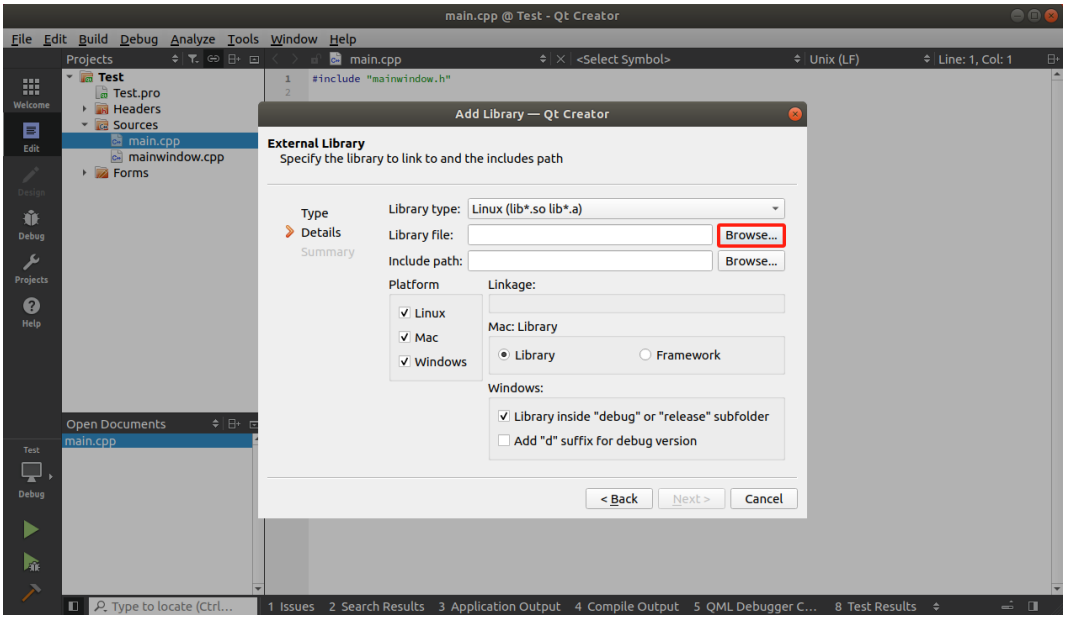
17. 点击编辑，右击 Test 项目，点击添加库。



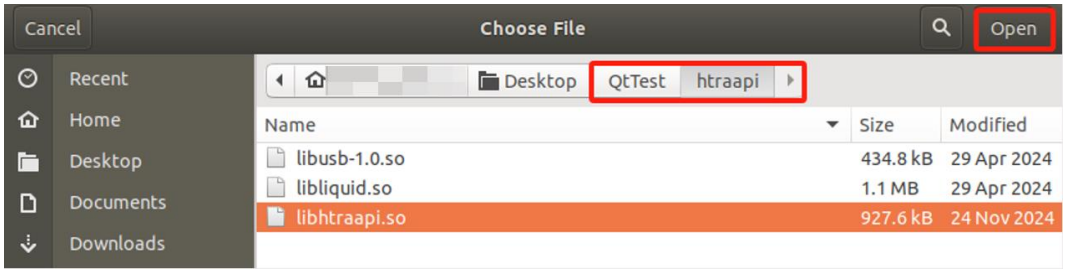
18. 选择外部库，点击下一步。



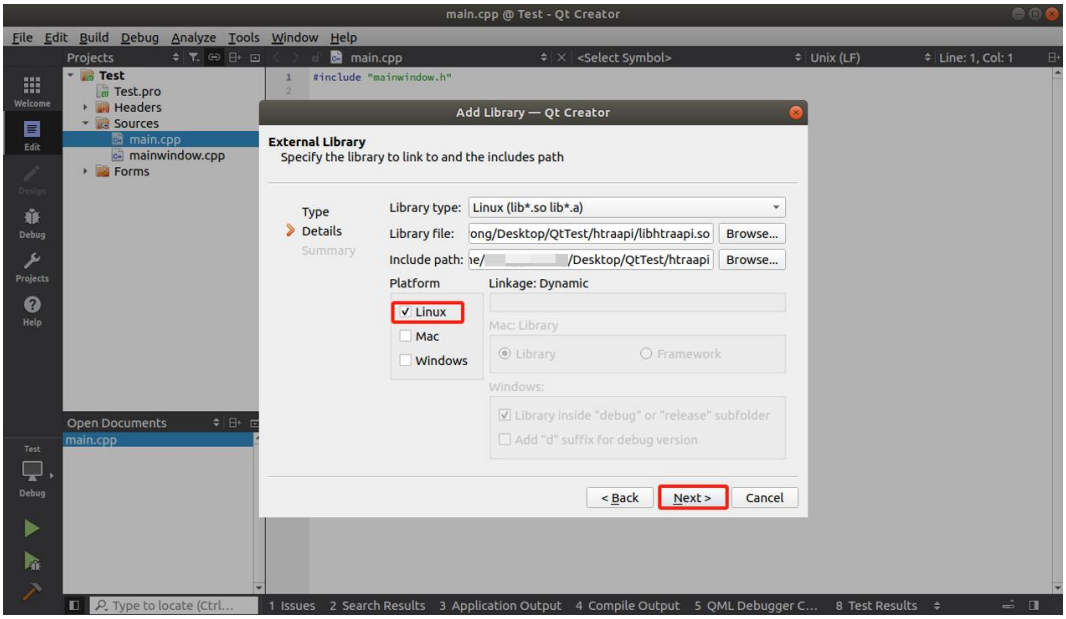
19. 点击浏览库文件。



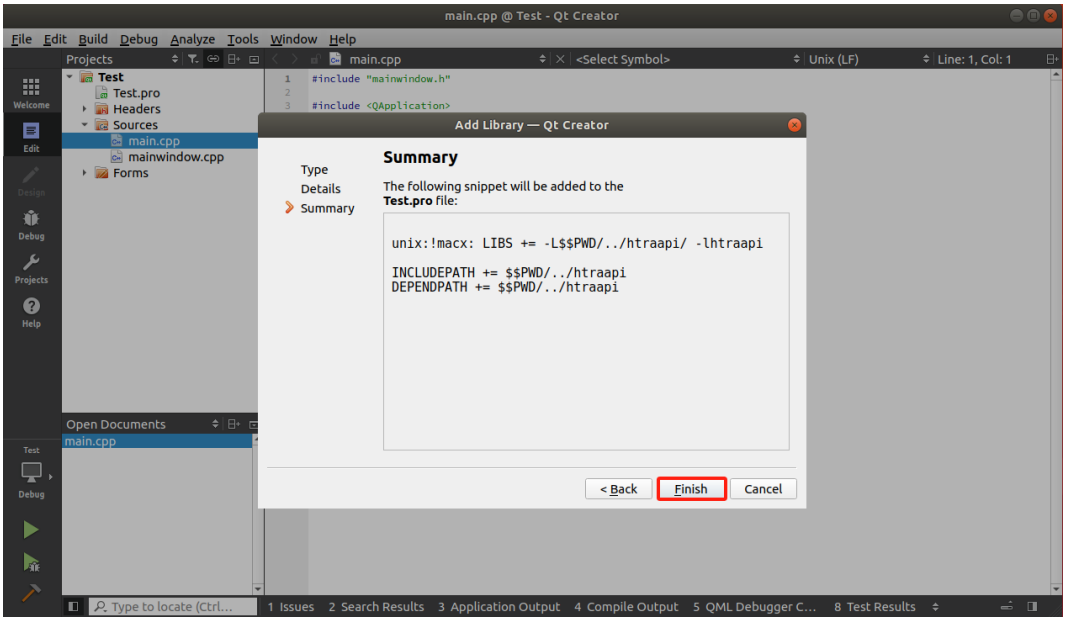
20. 选择 QtTest\htraapi 中 libhtraapi 库，点击 Open。



21. 选择 Linux 平台，点击下一步。



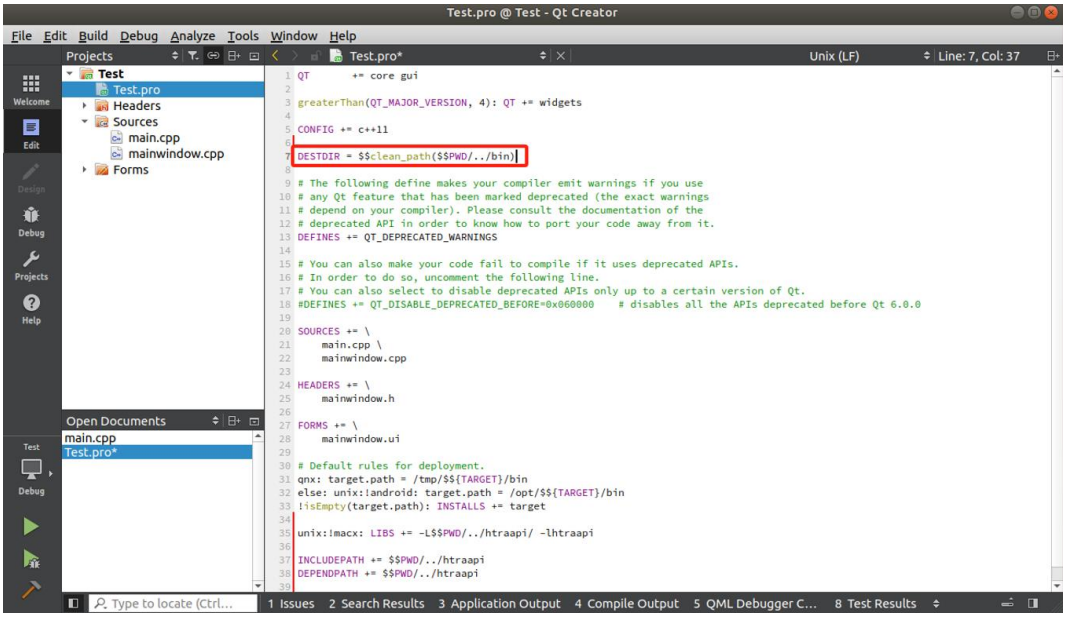
22. 点击完成以添加外部库。



23. 如下图所示，在 CONFIG += c++11 之后添加

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

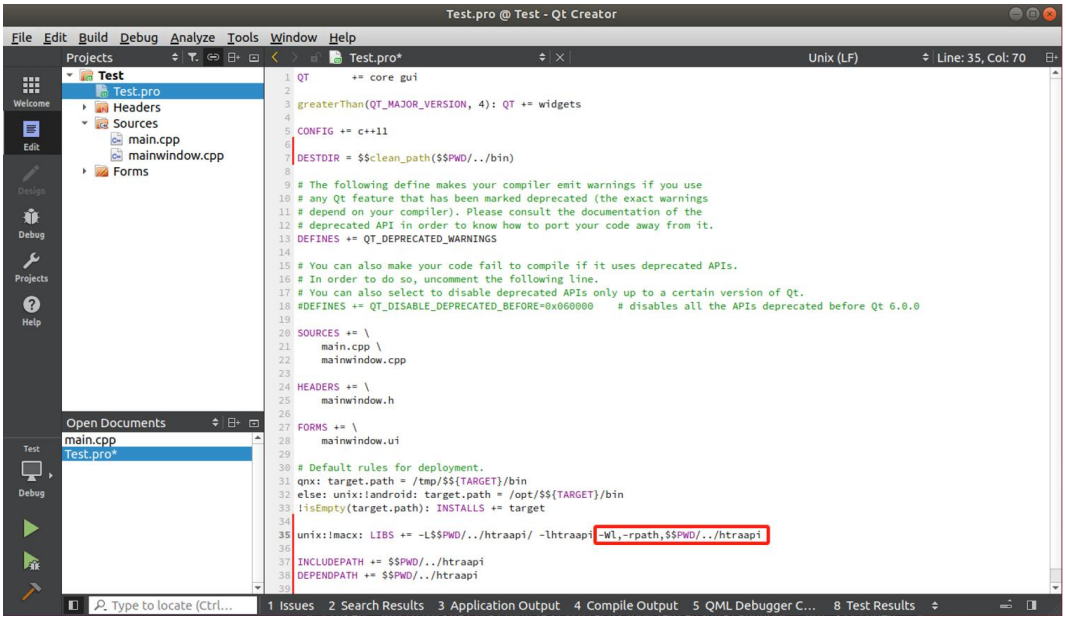
（该步骤为指定可执行程序生成位置）



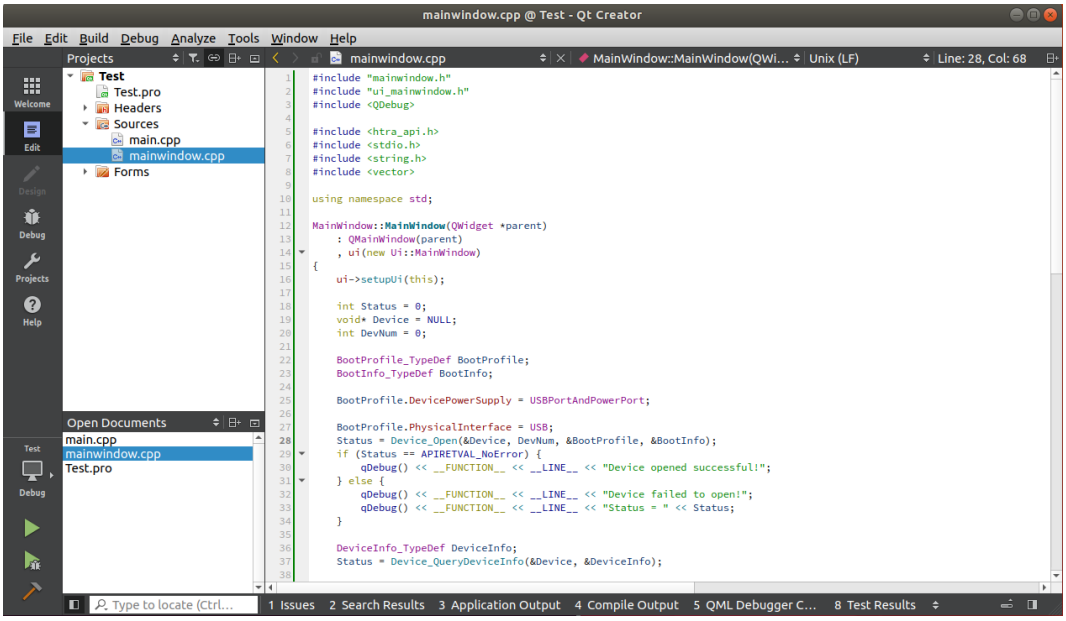
24. 如下图所示，在库文件之后，添加

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

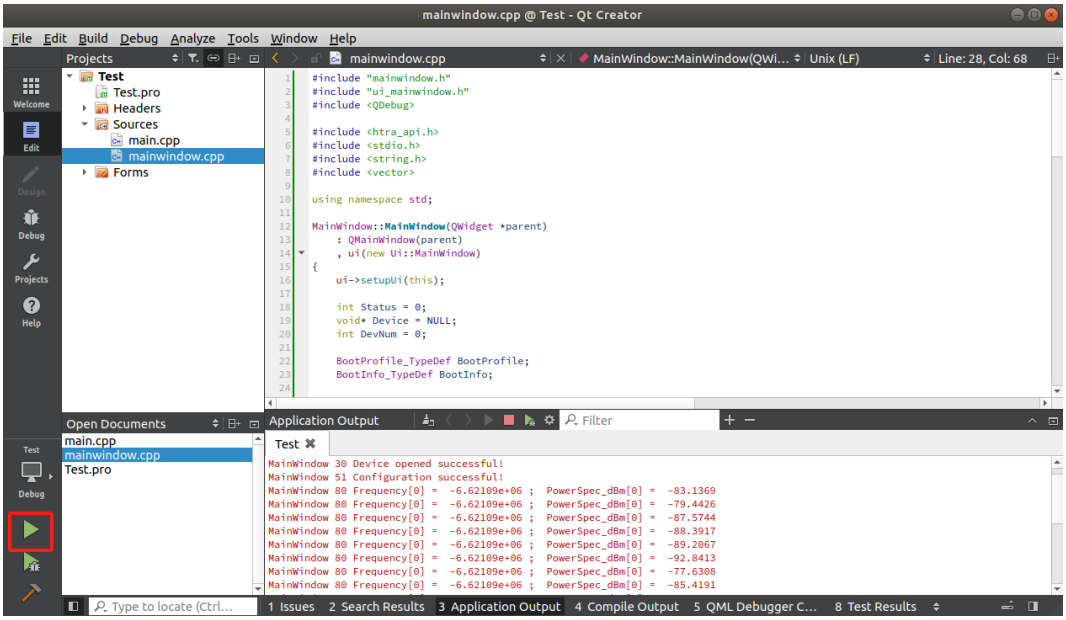
（该步骤为指定可执行程序的链接库路径）



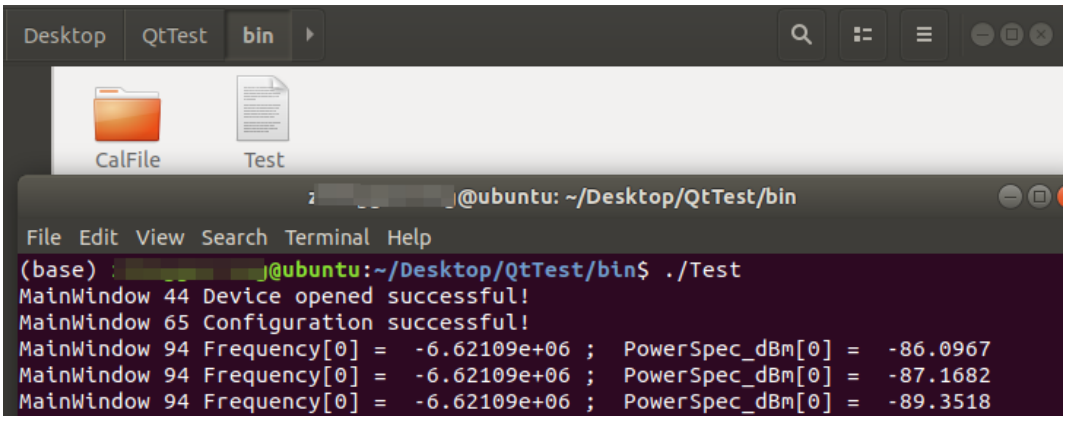
25. 保存 Test.pro 文件，之后在 mainwindow.cpp 中编写代码。



26. 正确编写代码后，点击运行。如图所示在应用程序输出中即可看到程序正常运行。



27. 关闭 Qtcreater，进入 QtTest\bin 文件夹，打开终端，输入./Test 即可运行可执行程序。



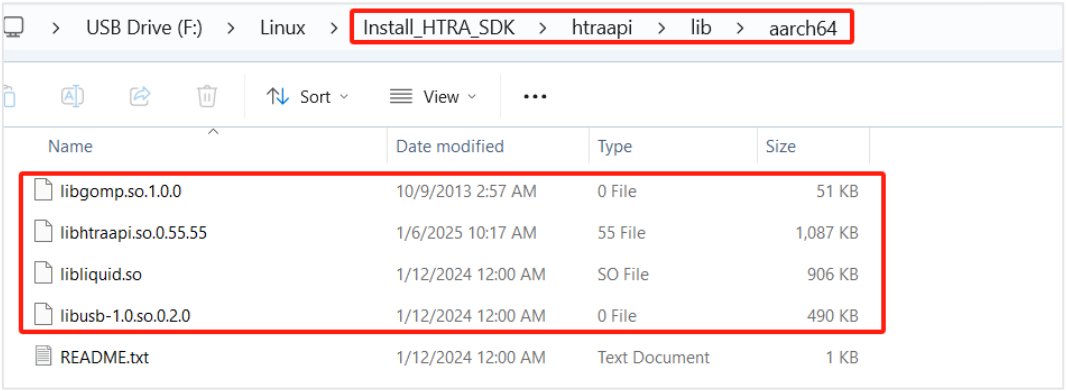
9.5.3 Qt 项目交叉编译

在上位机有交叉编译链的前提下，若想要交叉编译使用设备，请参考以下流程（此处以在 x86_64 的上位机交叉编译 arrch64 可执行程序为例）：

1. 首先生成目标架构可执行文件：

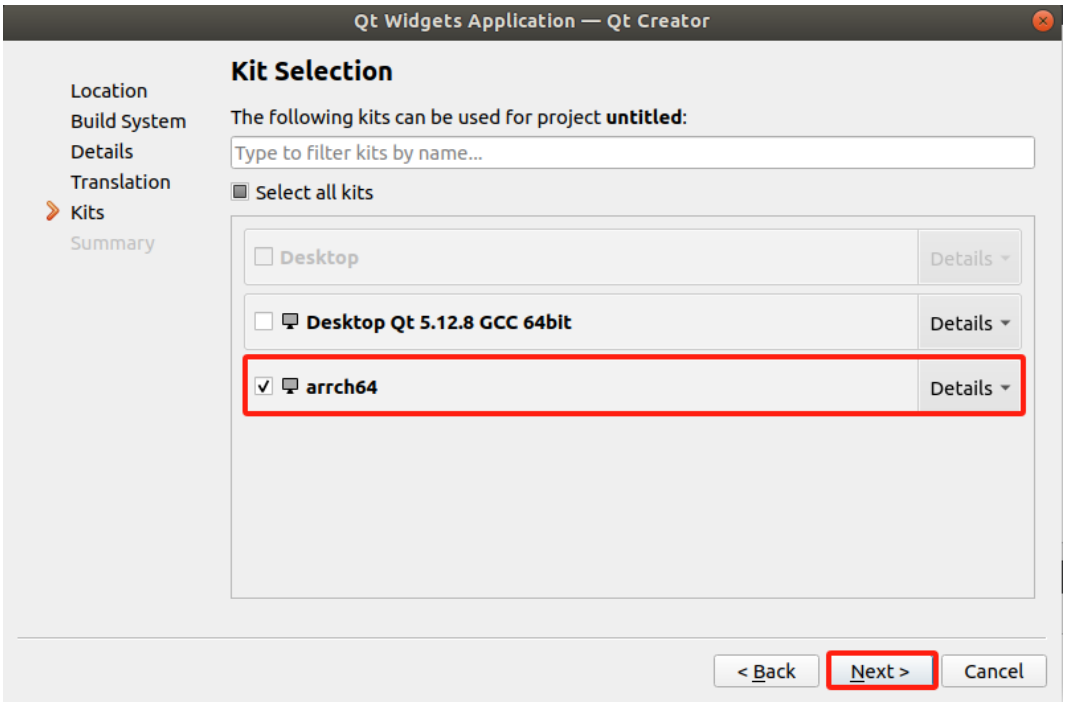
（1）按照 Qt 项目创建编译中窗体程序创建流程的 1-4 步创建项目并放置校准文件与头文件。

（2）按窗体程序创建流程第 5 步放置交叉编译目标架构库文件。例如预计交叉编译 arrch64 的可执行程序时请放置 arrch64 架构的库文件。



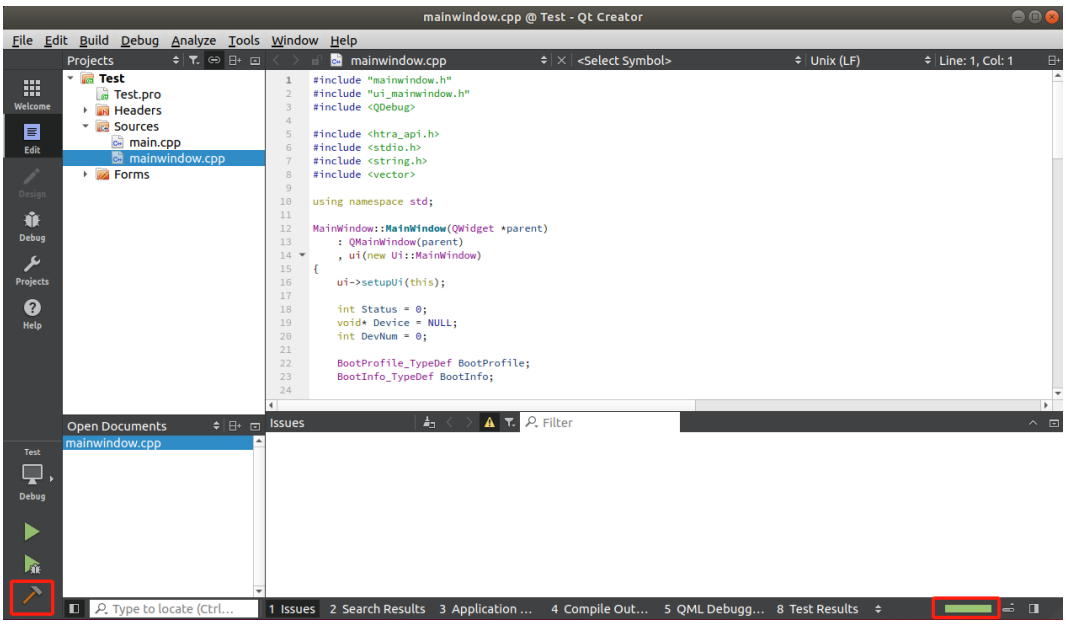
（3）按窗体程序创建流程第 6 步对动态链接库进行软链接。

（4）按窗体程序创建流程第 7-13 步创建程序，在第 14 步选择构建套件时选择交叉编译目标架构的构建套件（例如此次示例选择 arrch64 架构的构建套件）。

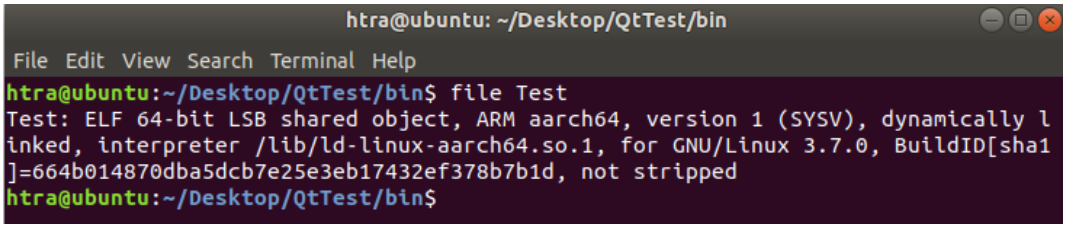


（5）按窗体程序创建流程第 16-26 步进行项目创建、库引用、可执行程序生成位置修改以及项目编写。

（6）点击左下角构建按钮构建可执行程序。

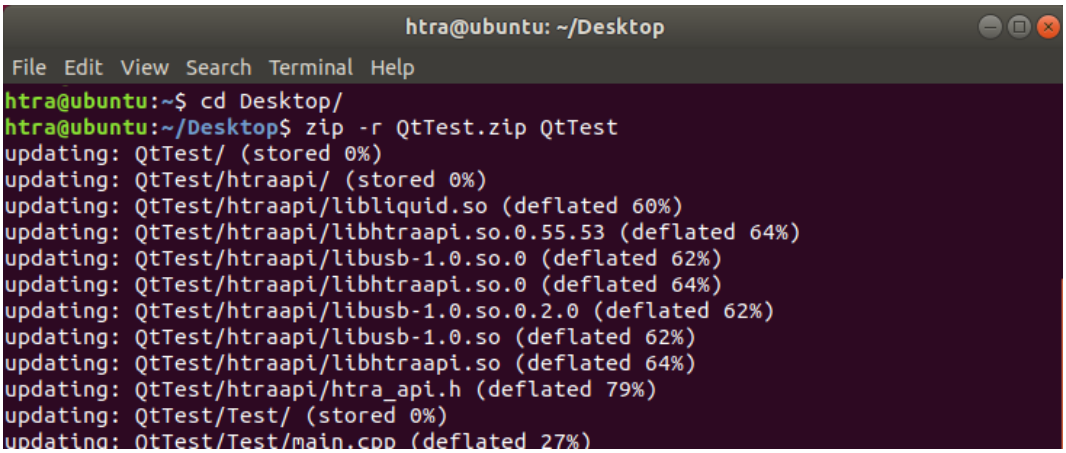


（7）构建好可执行程序后，在 QtTest\bin 文件夹下打开终端输入 file Test 查看可执行程序的架构（此处可执行程序名称为 Test，因此输入 file Test）。

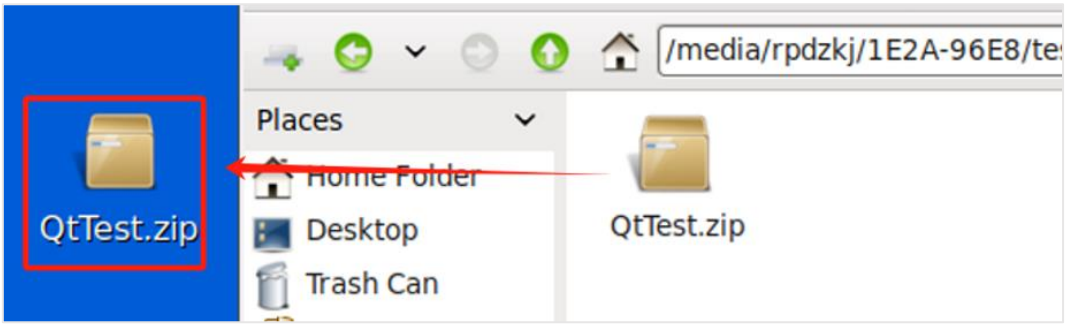


2. 至此可执行程序已成功生成，之后即可在 arrch64 架构上位机运行程序使用设备：

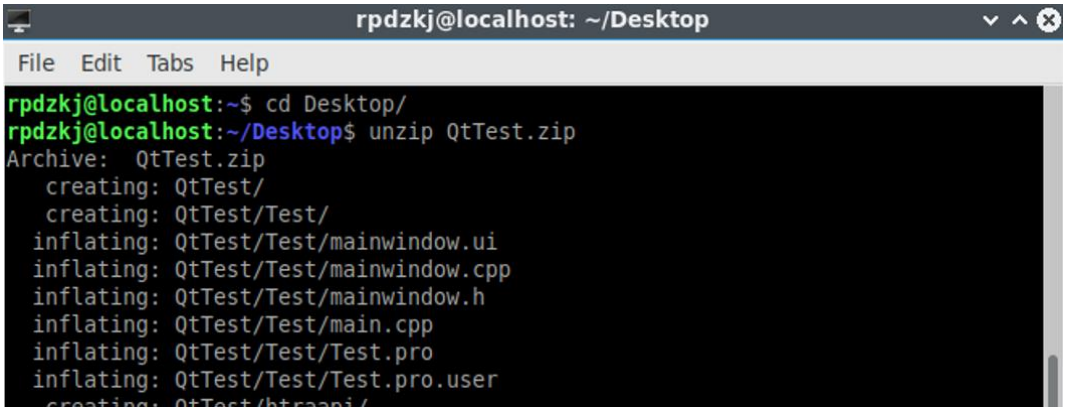
（1）进入项目所在地址（此处示例项目位于桌面因此输入 cd Desktop/），将整个文件夹压缩为压缩包，例如输入 zip -r QtTest.zip QtTest 制作 zip 压缩包。



（2）将压缩包拷贝至 arrch64 上位机中。

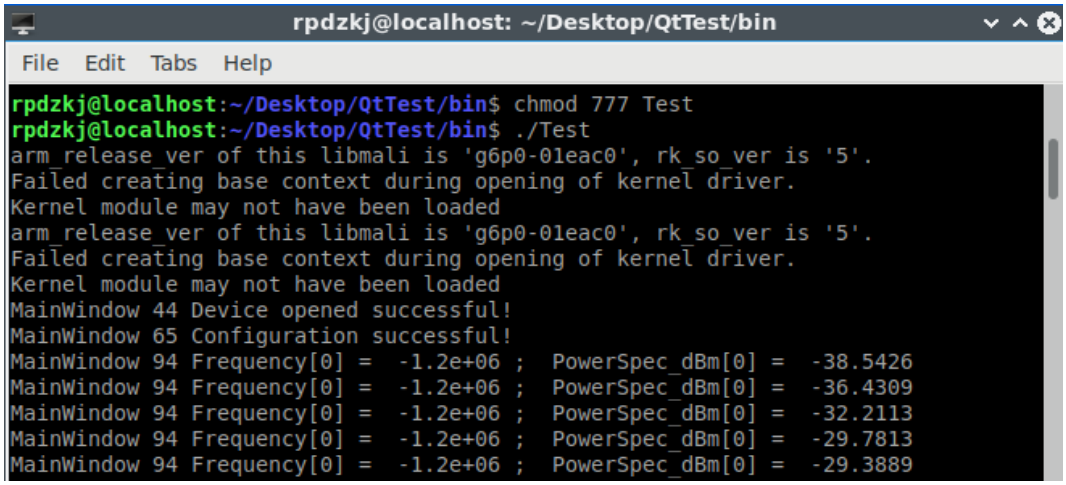


（3）进入压缩包存放位置（此处示例压缩包位于桌面因此输入 cd Desktop/），将项目解压（此处输入 unzip QtTest）。



（4）按章节 9.3 中步骤在 arrch64 上位机中配置驱动文件。

（5）配置好驱动文件后，输入 cd QtTest/进入文件夹，输入 chmod 777 Test 为可执行程序提供权限，之后输入./Test 运行程序即可。

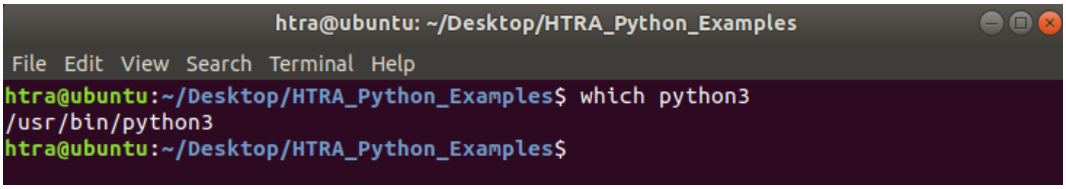


9.6 Python 范例使用以及项目创建

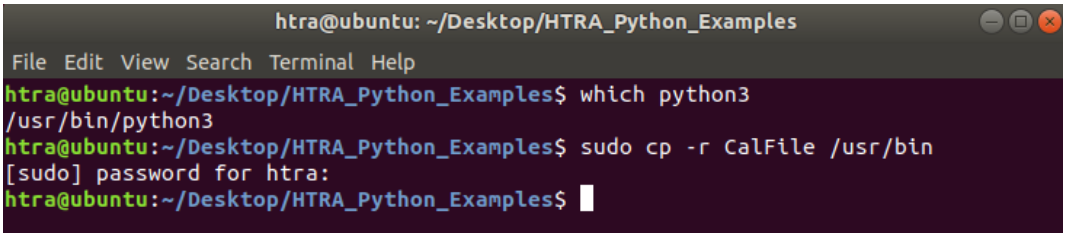
9.6.1 Python 范例使用

在保证设备正常接入且已按章节 9.3 正确配置驱动文件的前提下，若想要使用随寄 U 盘中 Python 范例，可参考以下流程（Python 范例的具体作用可直接查看章节 4.2 中描述）：

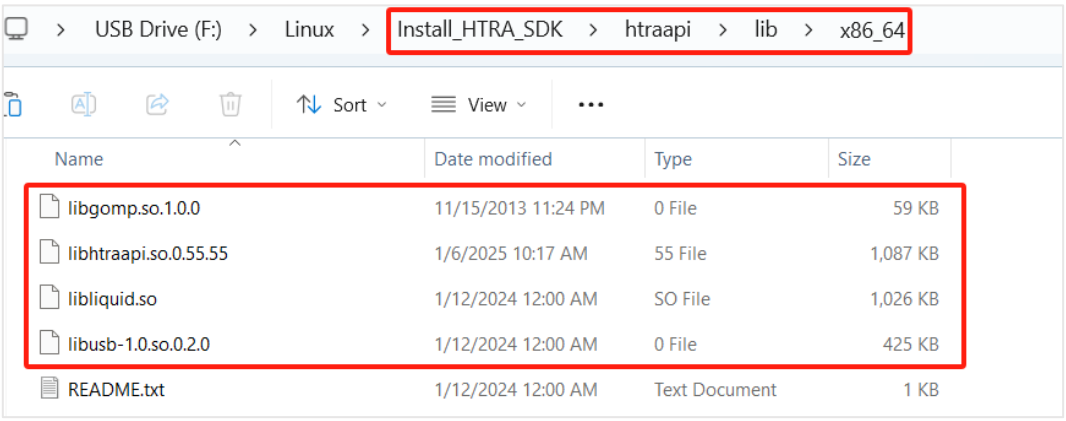
1. 将随寄 U 盘 Linux\HTRA_Python_Examples 文件夹拷贝至上位机，在 HTRA_Python_Examples 文件夹下打开终端输入 which python3 查看 Python 解释器位置（例如此处位置为/usr/bin）。

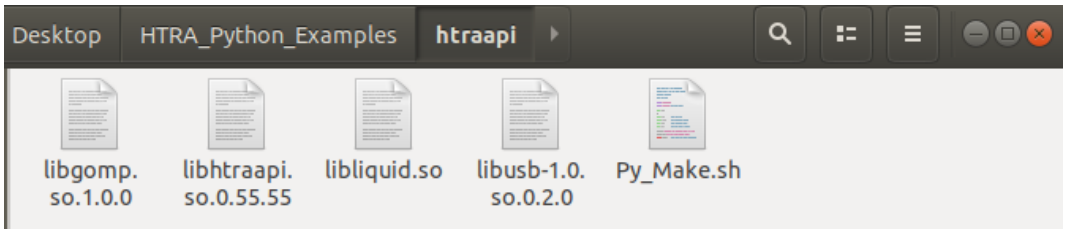


2. 按照刚才获取的解释器地址，输入 sudo cp -r CalFile /usr/bin 将设备校准文件复制至解释器（Python3）同级目录地址（例如本次示例同级目录为/usr/bin，若实际解释器位置不在此处，将该地址修改即可）。

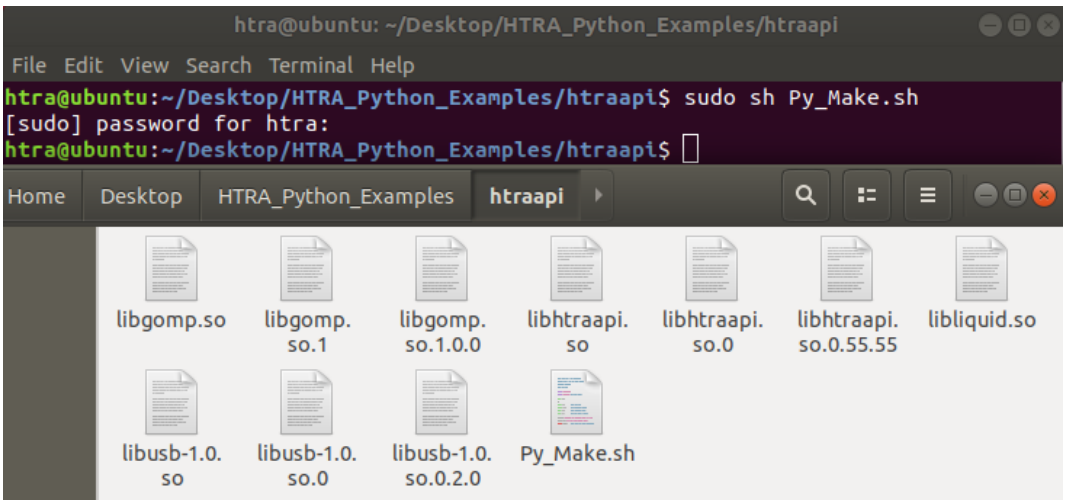


3. 首先按照章节 9.1 中流程查看系统架构，然后按照章节 9.2.4 中介绍将随寄 U 盘 Linux\Install_HTRA_SDK\htraapi\lib 下对应系统架构文件夹中的动态链接库复制至 HTRA_Python_Examples\htraapi 文件夹中（此处以 x86_64 架构上位机为例）。

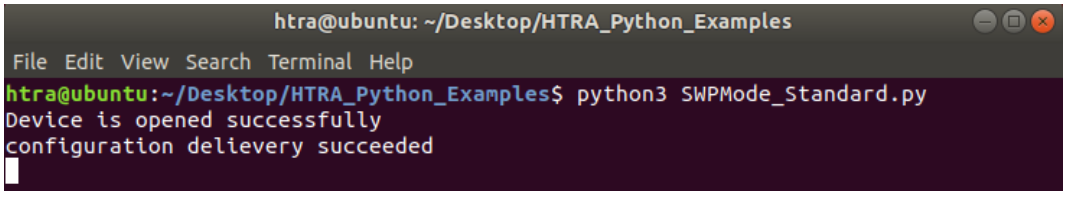




4. 在当前文件夹下打开终端，输入“sudo sh Py_Make.sh”，之后按提示输入 sudo 密码提供权限对库进行软链接。



5. 在 HTRA_Python_Examples 位置打开终端输入 python3 SWPMode_Standard.py 即可运行范例。（此处以 SWPMode_Standard.py 范例为例）



9.6.2 Python 项目创建

在已按章节 9.3 正确配置驱动文件的前提下，若想要创建并编写 Python 项目，请参考以下流程：

首先在编写代码时，因为随寄 U 盘中提供的 Linux 动态链接库与 Windows 中的完全一致，所以代码部分只需符合 API 编程指南即可。

其次在运行程序时，只需将程序存放至范例文件夹下，按照 Python 范例使用章节流程使用即可。

9.7 Gnuradio 模块构建与使用

在设备正常连接并按章节 9.3 完成驱动文件配置后，如需通过 Gnuradio 使用设备，可通过以下两种方式：

9.7.1 HTRA OOT 构建使用

HTRA OOT 模块的获取方式有两种：

（1） 直接从随寄资料 Linux\HTRA_Gnuradio_Examples 文件夹中获取；

（2） 通过 github 拉取，git 地址如下：

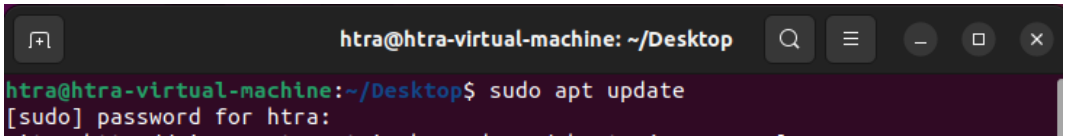
https://github.com/HAROGIC-Technologies/gr-htra

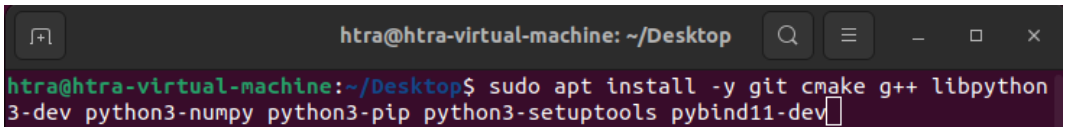
若是通过随寄资料获取，可参考以下流程使用：

1. 配置依赖环境：输入以下指令下载 OOT 模块构建所需依赖环境：

sudo apt update

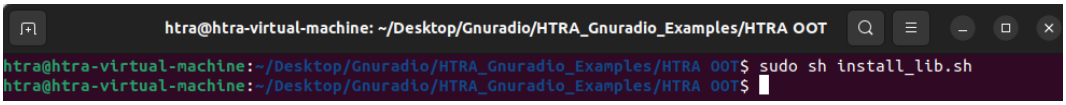
sudo apt install -y git cmake g++ libpython3-dev python3-numpy python3-pip python3-setuptools pybind11-dev





2. 配置构建所需库与头文件：在 HTRA OOT 文件夹下打开终端，输入

sudo sh install_lib.sh



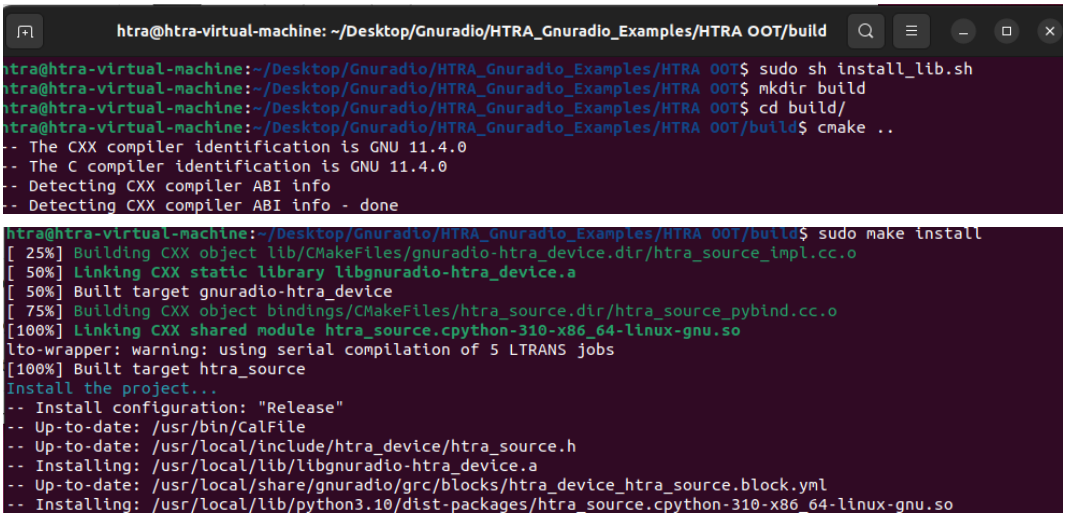
3. OOT 模块构建：依次输入以下指令：

mkdir build

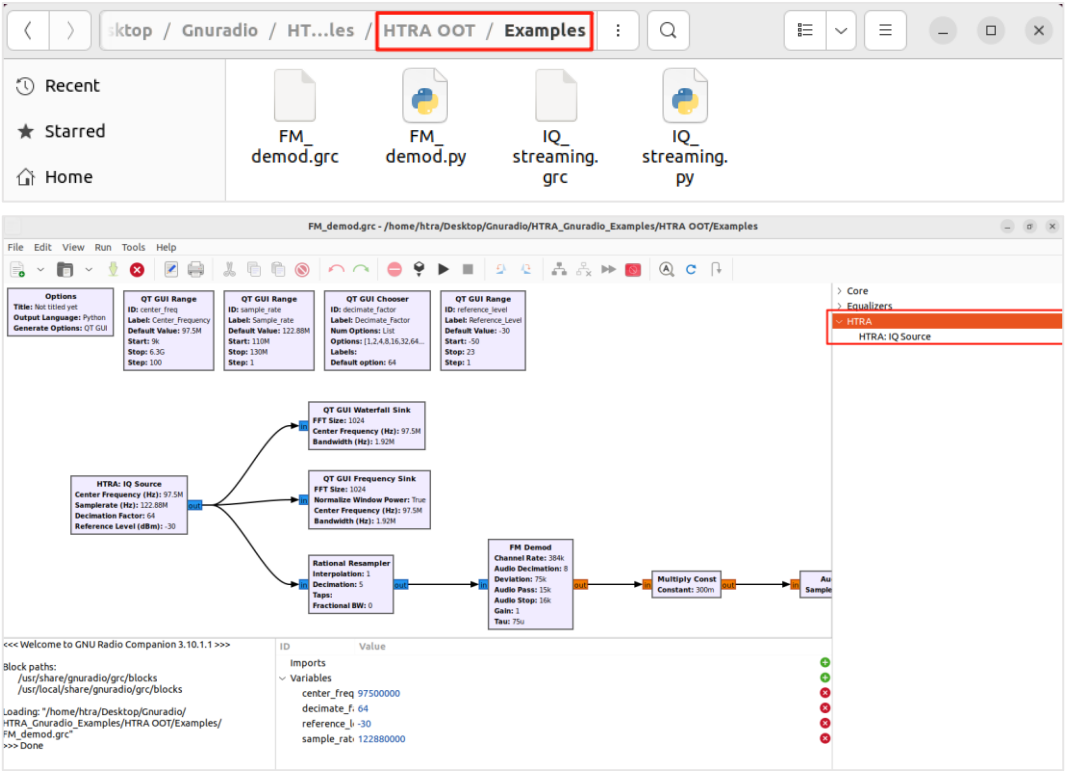
cd build

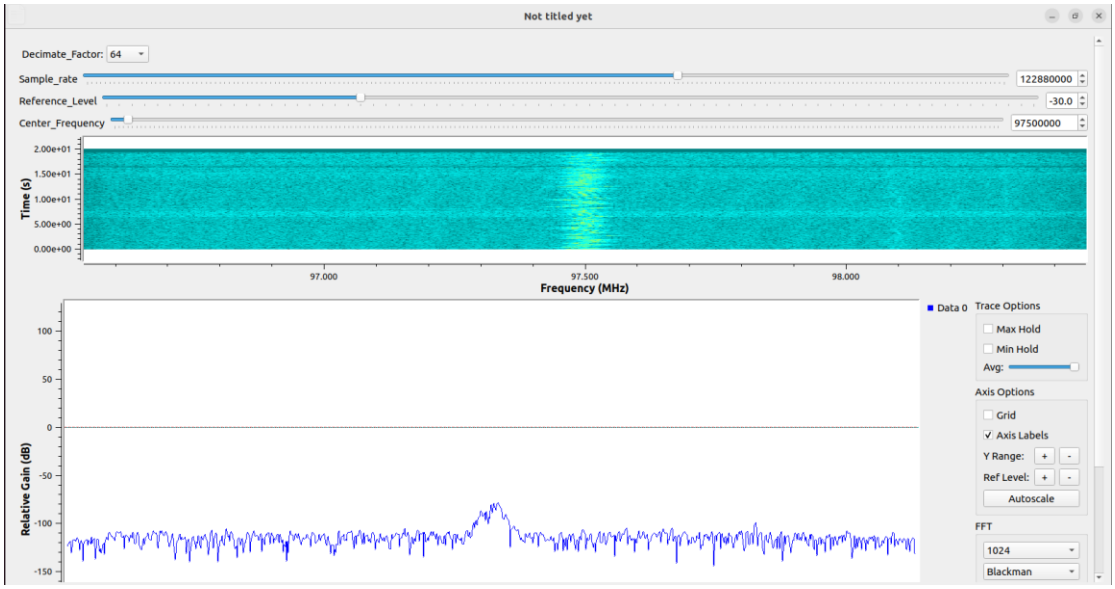
cmake ..

sudo make install



4. 使用设备：构建完毕后可参考 Examples 文件夹中例程在 Gnuradio 中使用设备（此处以 FM 解调为例）：





9.7.2 SoapySDR 构建使用

SoapySDR 适配的获取方式有两种：

（1） 直接从随寄资料 Linux\HTRA_Gnuradio_Examples 文件夹中获取；

（2） 通过 github 拉取（推荐），git 地址如下：

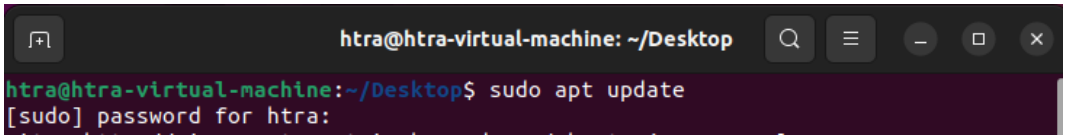
https://github.com/HAROGIC-Technologies/soapy-htra

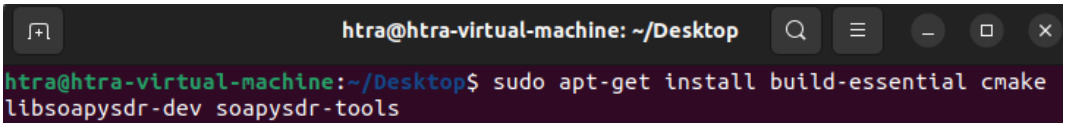
若是通过随寄资料获取，可参考以下流程使用：

1. 配置依赖环境：输入以下指令下载 SoapySDR 模块构建所需依赖环境：

sudo apt-get update

sudo apt-get install build-essential cmake libsoapysdr-dev soapysdr-tools

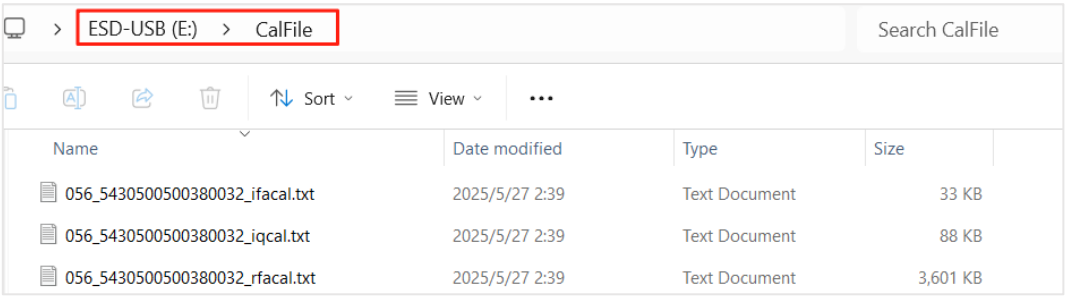


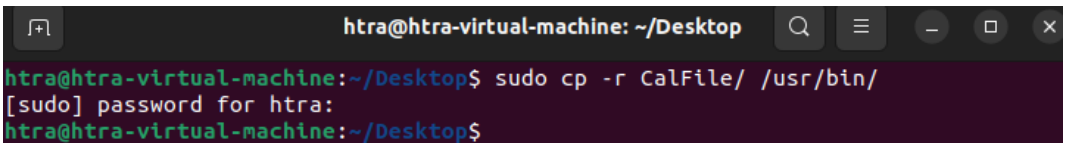


2. 拷贝校准文件：将随寄 U 盘 CalFile 文件夹中文件拷贝至 usr/bin/CalFile 文件夹下

sudo cp -r CalFile/ /usr/bin/

拷贝后可通过 ls /usr/bin/CalFile/查看拷贝情况





3. SoapySDR 模块构建：依次输入以下指令：

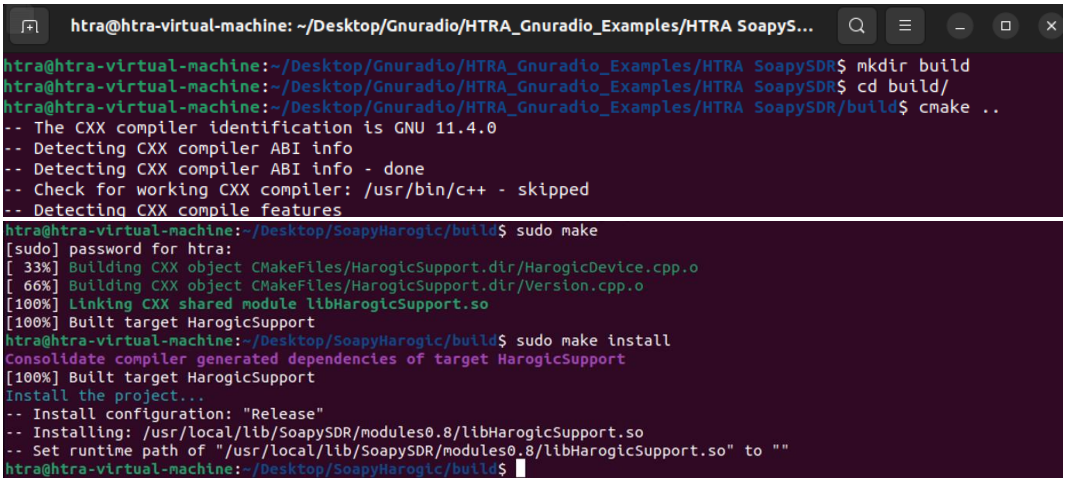
mkdir build

cd build

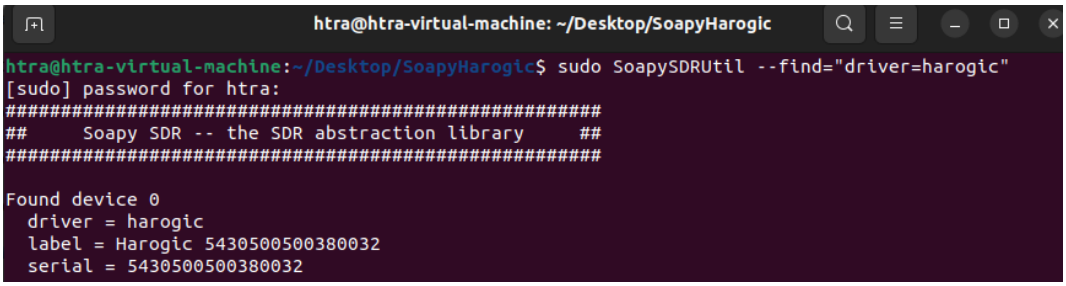
cmake ..

sudo make

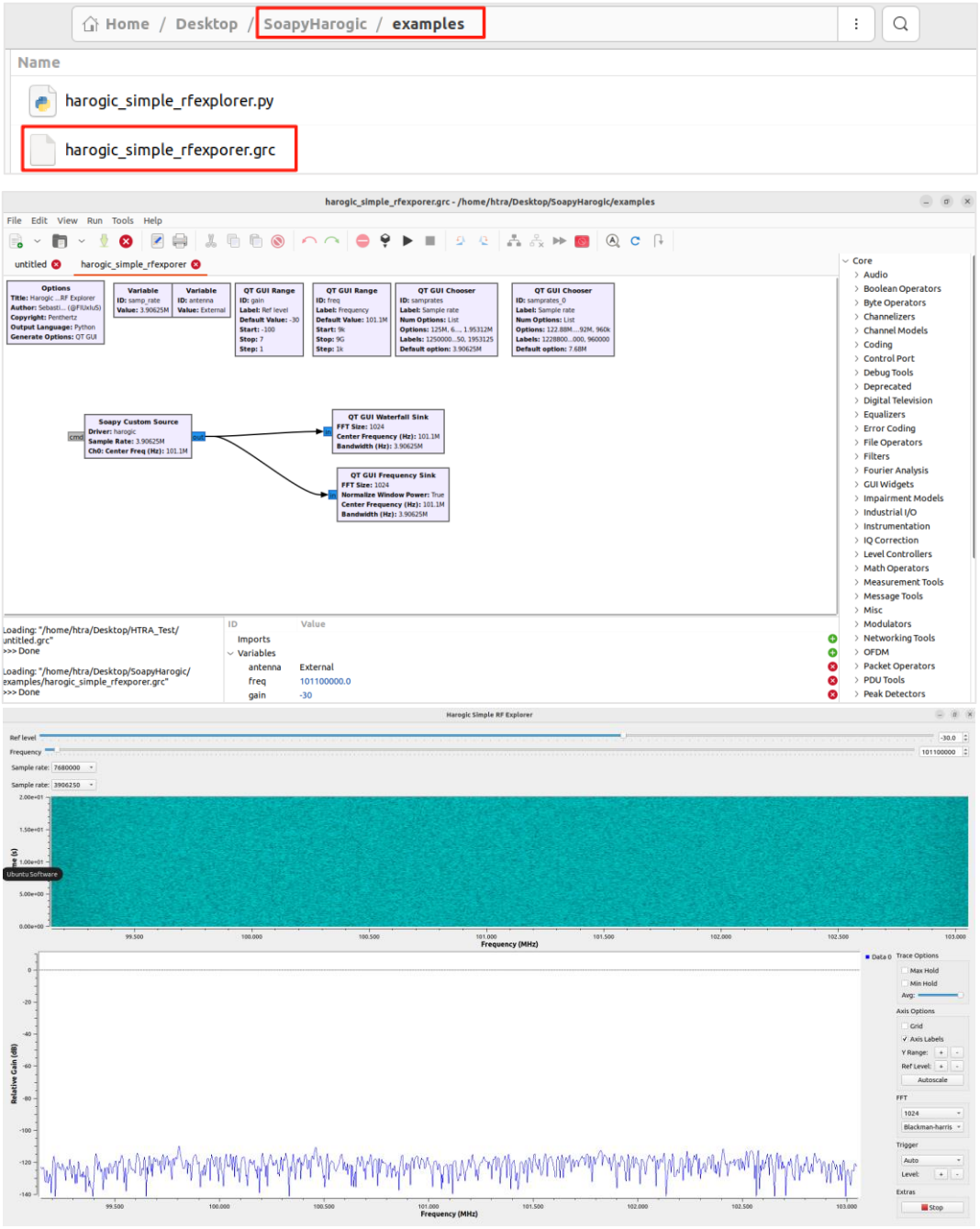
sudo make install



4. 构建完毕后输入 sudo SoapySDRUtil --find=”driver=harogic”查找设备：



5. 之后可参考 Examples 文件夹下文件使用设备：



9.8 Java（待补充）

需要注意一点，Linux 系统中通过 Java 编写程序控制设备时，CalFile 文件夹需要放到 Java 解释器的同级目录。