光纤光谱仪二次开发包SDK调用说明
CMS24V1.1

文档版本：CMS24V1.1

编写日期：2024年2月28日

编写人员：沈玉杰，深圳谱研互联科技有限公司

技术支持：沈玉杰，深圳谱研互联科技有限公司

💡 本章以VS2017的桌面应用控制台项目为例，介绍库函数的调用流程。其他集开发环境（IDE）调用步骤类同。其中第一章配置库路径相关内容主要面向不熟悉VS中如何引用第三方库的新手小伙伴或学生党，资深码农可直接略过本章。

1 配置库路径

1.1 文件相对位置

1.1.1 三个位置

在配置库路径时，会涉及到头文件或动态库的相对路径选取和放置，我们需要知道三个重要的文件目录位置，包括解决方案位置、项目文件位置和应用程序位置。有了这几个位置的概念后，我们在配置项目属性页的相对路径时，思路就会很清晰。同时可以了解下项目属性页的描述符。

我们可以简单理解为，三个位置分别对应三个专用后缀文件所在的文件目录（文件夹），如下：

1）解决方案：.sln

2）项目文件：.vcxproj

3）应用程序：.exe

对于新手，我们知道了这三个文件的位置，就可以很清晰的定位IDE在查找需要的文件或依赖库时的相对位置。下面以图文的方式介绍这些位置，有个概念即可。

1.1.2 解决方案位置

在VS中，我们新建工程后，会生成一个解决方案。打开文件夹，可以看到一个**.sln**文件，这就是解决方案所在文件目录位置。

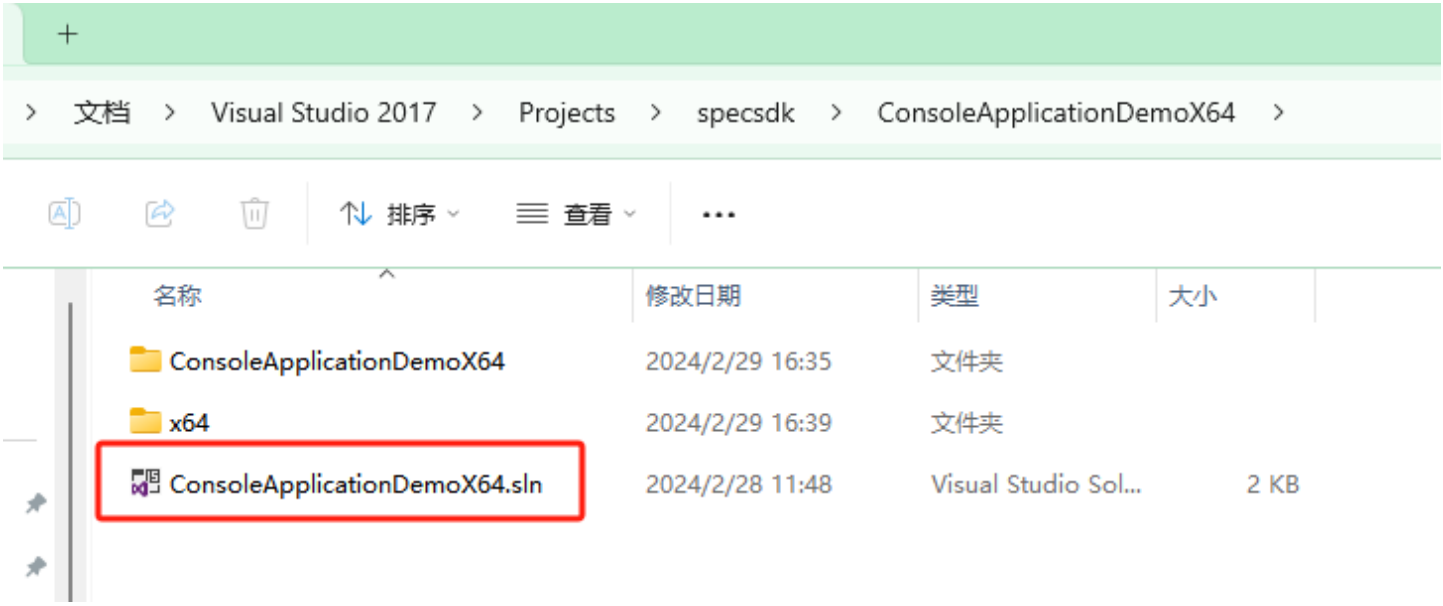


图1 解决方案文件目录位置

1.1.3 应用程序位置

我们可以先了解下项目属性中的描述符。

在“解决方案资源管理器”的项目位置点击“右键”，选择【属性】，在弹出的【配置属性】-【常规】对话框中，有个【输出目录】选项，其默认路径为：$(SolutionDir)$(Platform)\$(Configuration)\

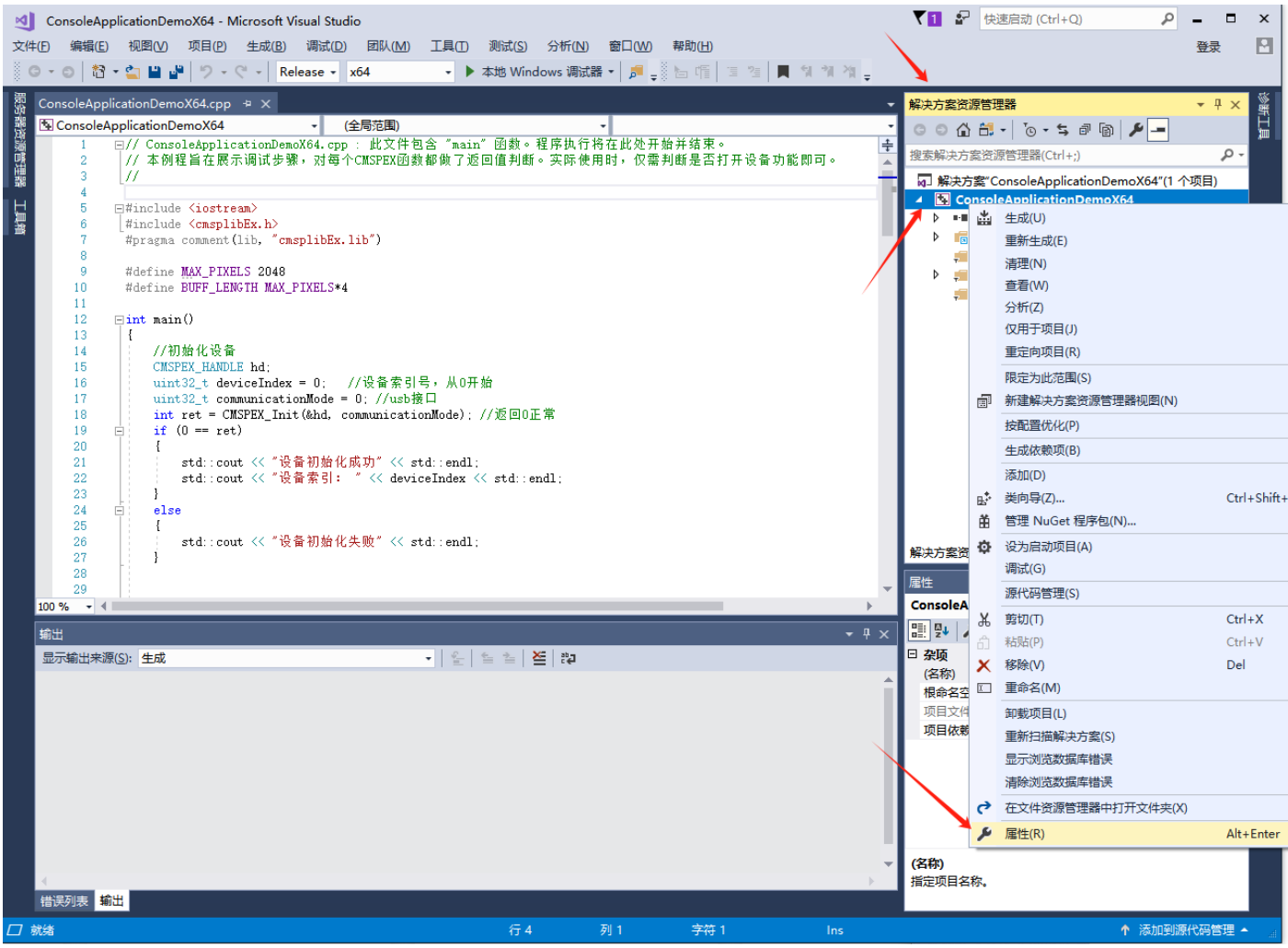


图2 解决方案资源管理器

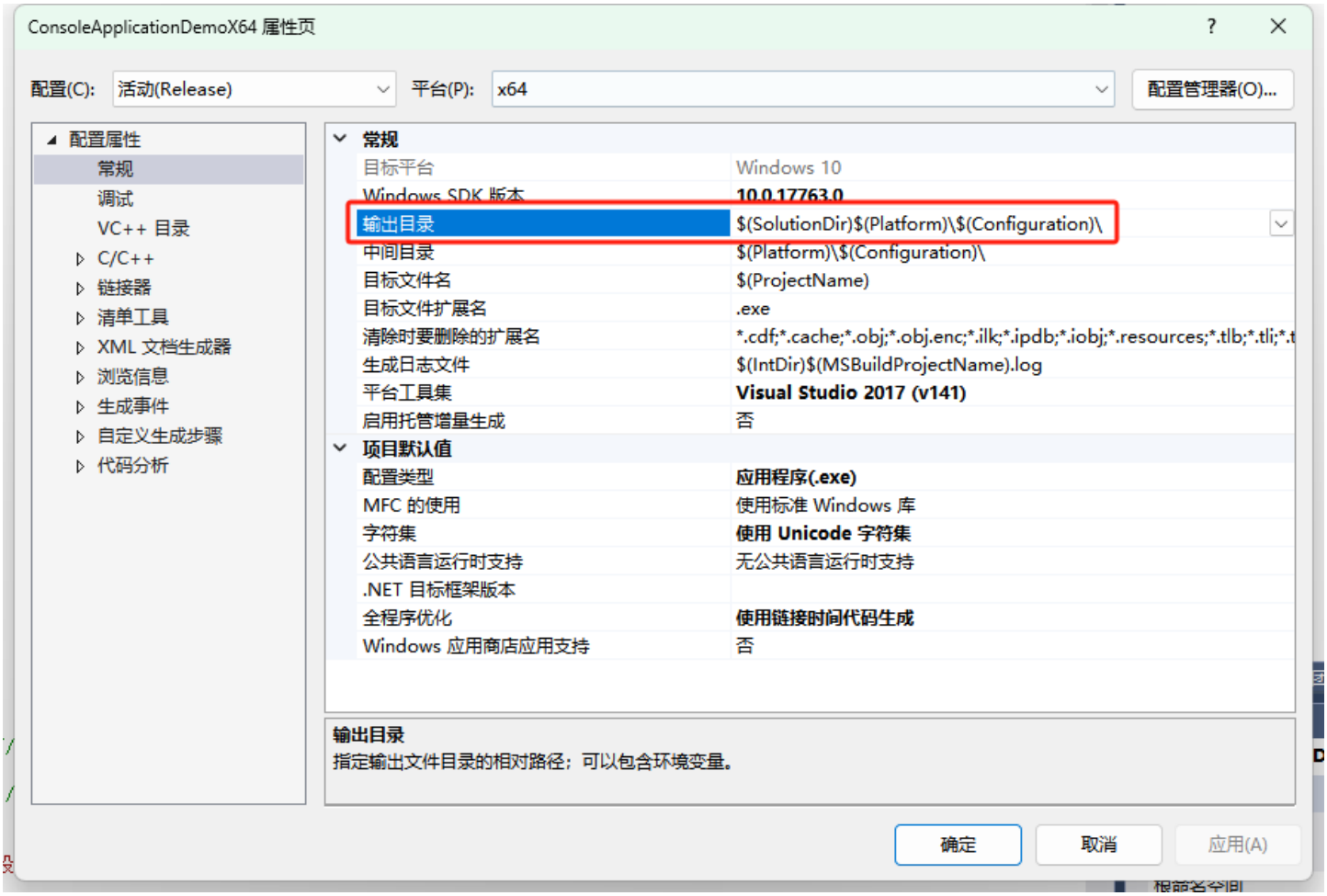


图3 属性页面的输出目录

其中

"$(SolutionDir)"： 表示解决方案文件的目录

"$(Platform)"： 表示目标平台（如 x64或X86/Win32），X86和Win32均是指Windows32位，意思相同

"$(Configuration)"： 表示当前的构建配置（如 Debug 或 Release）。

如新建的控制台项目名称为 ConsoleApplicationDemoX64，目标平台是X64，在Release下编译，则应用程序所在的输出目录为：

ConsoleApplicationDemoX64\x64\Release

这里有

$(SolutionDir)： ConsoleApplicationDemoX64（即.sln所在目录）

$(Platform)： x64

$Configuration： Release

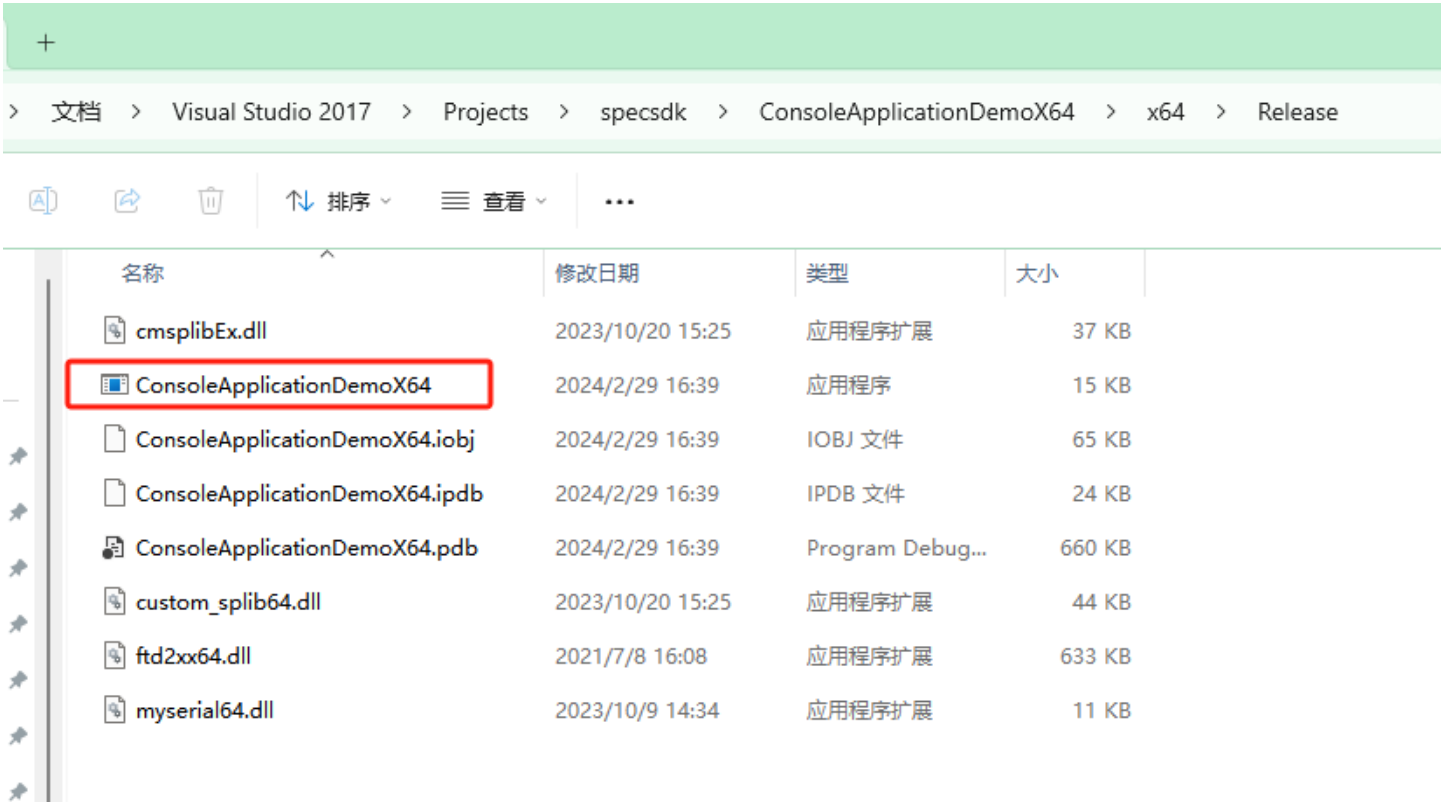


图4 应用程序输出目录

其中图2中的 ConsoleApplicationDemoX64.exe 就是应用程序，其所在的文件目录即应用程序位置。

1.1.4 项目文件位置

我们通常从设备厂家获到的SDK（二次开发包）包括 .h、.lib 和 .dll 文件。要引用这些文件，需要在项目属性中配置附加包含目录和附加库目录。 **.vcxproj** 所在目录位置就是相对路径的起点位置，我们在属性配置页添加相对路径即可。

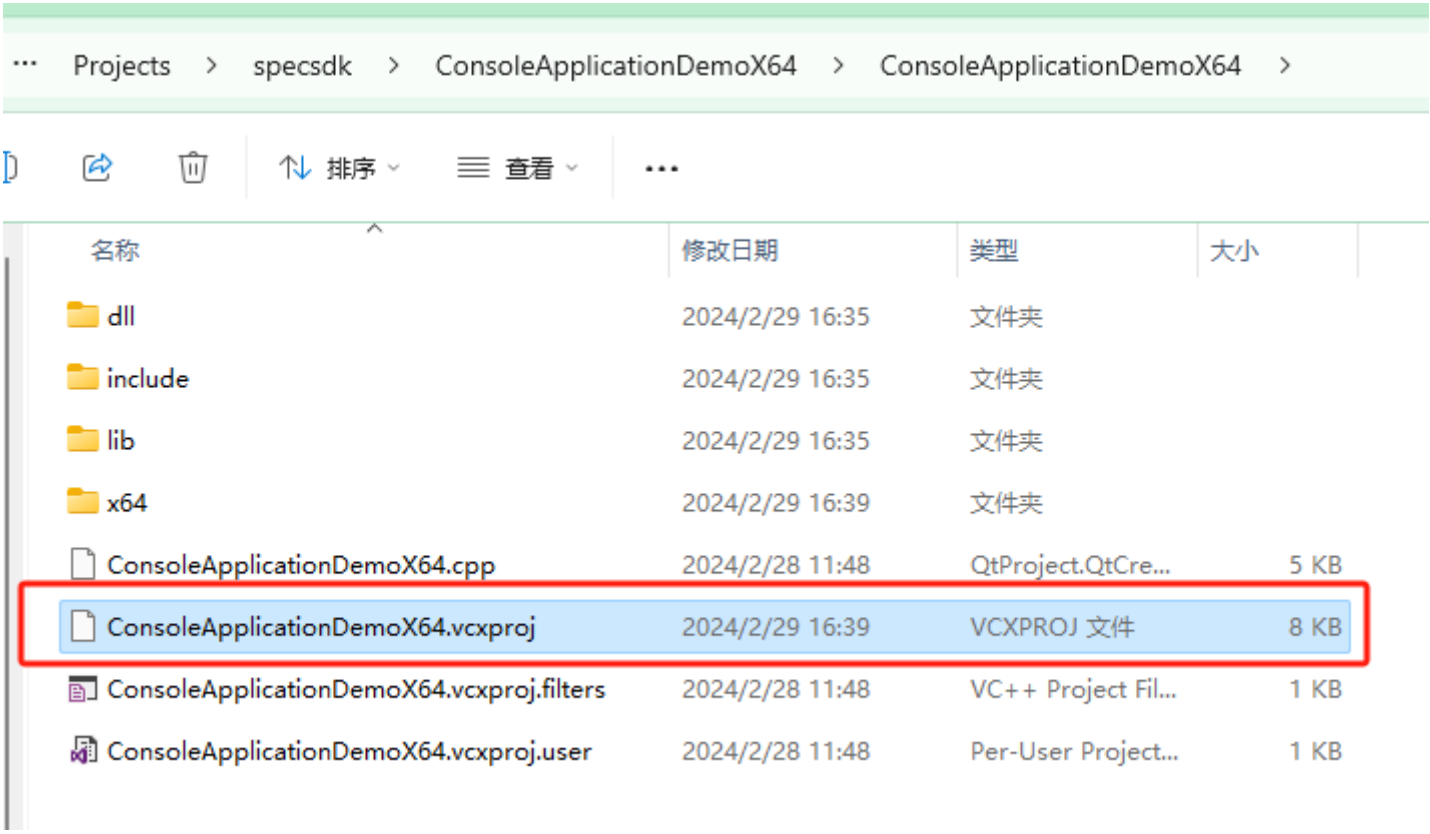


图5 项目文件所在目录

1.2 添加第三方库

谱研互联的光纤光谱仪二次开发（sdk）函数库包括**头文件.h**、**静态库.lib**和**动态库.dll**，分别放在include、lib和dll文件夹中，文件可在谱研互联网站下载中心或找工程师获取。

在添加之前，我们先明确配置的目的，配置目的是要解决以下问题：

1）头文件在哪里找

2）头文件里的函数实现在哪里找

3）应用程序运行时的依赖库在哪里找

4）是否需要宏定义来启用#ifdef语句

新手可能对#ifdef不熟悉，只需知道要激活某些宏定义（一般在.h文件中），需要做一些预处理器配置即可，感兴趣可以深入了解下。

我们可以先把sdk文件的文件夹复制到**项目文件目录**（即.vcxproj文件同级目录），在配置属性时均采用**相对路径**。

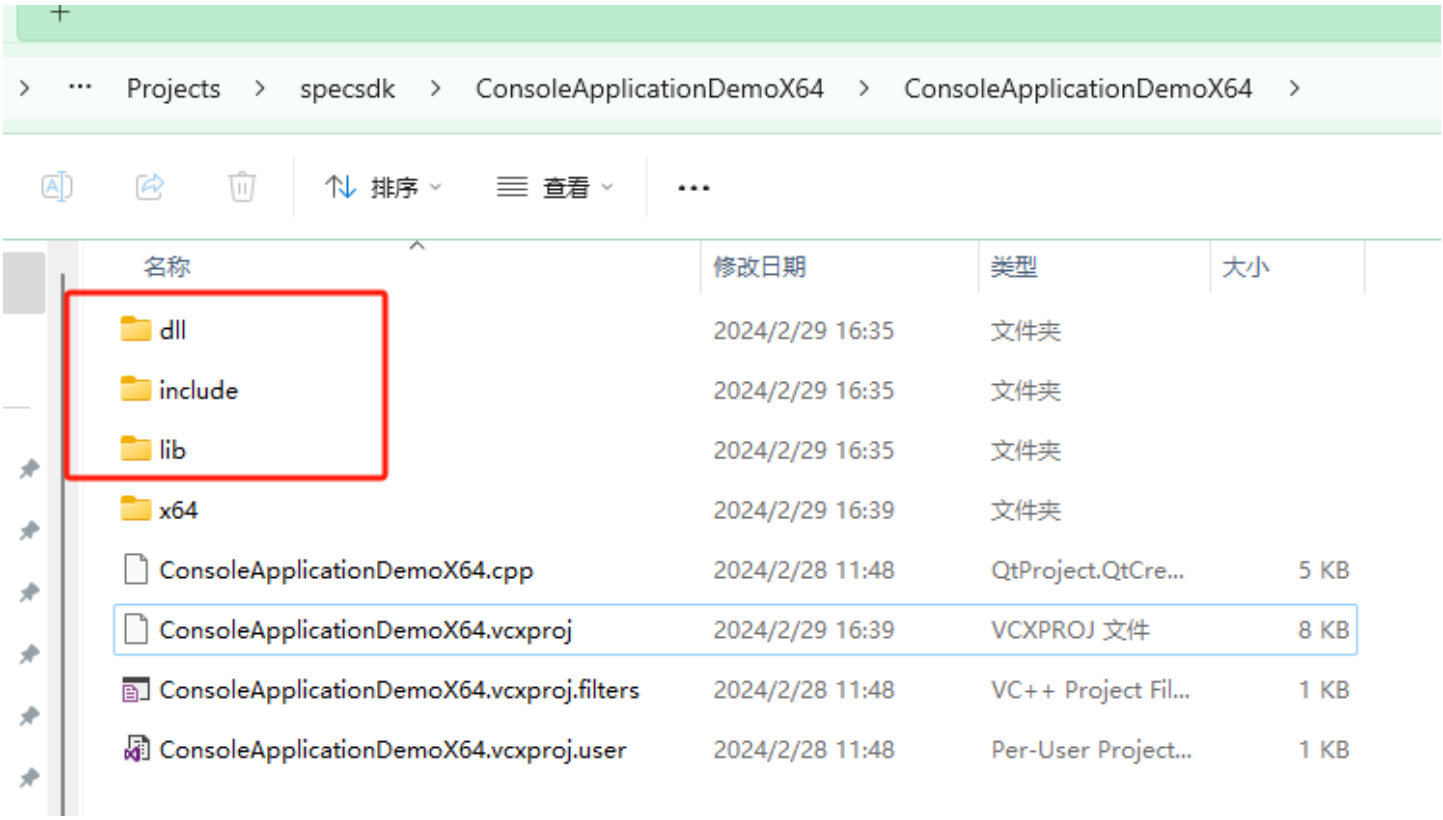


图6 光谱仪sdk文件夹

在项目属性中，有三处需要配置，分别为**附加包含目录**、**预处理器定义**和**附加库目录**。另外一处需要拷贝，即把dll文件夹中的.dll文件拷贝到应用程序所在目录。打开项目属性步骤可参见图2。

特别提醒：每个配置（Release或Debug）和平台（X64或X86/Win32）的组合均要在属性页配置一次才可以生效。如有的小伙伴在属性页配置的是Debug和X64组合，但在编译的时候用的是Release和X64组合，导致编译时提示找不到头文件或依赖库的错误提示。

1）添加附加包含目录

在代码中#include头文件时，预编译器会自动在附加包含目录查找对应的头文件。把光谱仪sdk中的include文件夹复制到项目目录（可参见图5），在【C/C++】-【常规】-【附加包含目录】中，添加include即可。这里的相对路径位置即为项目目录。

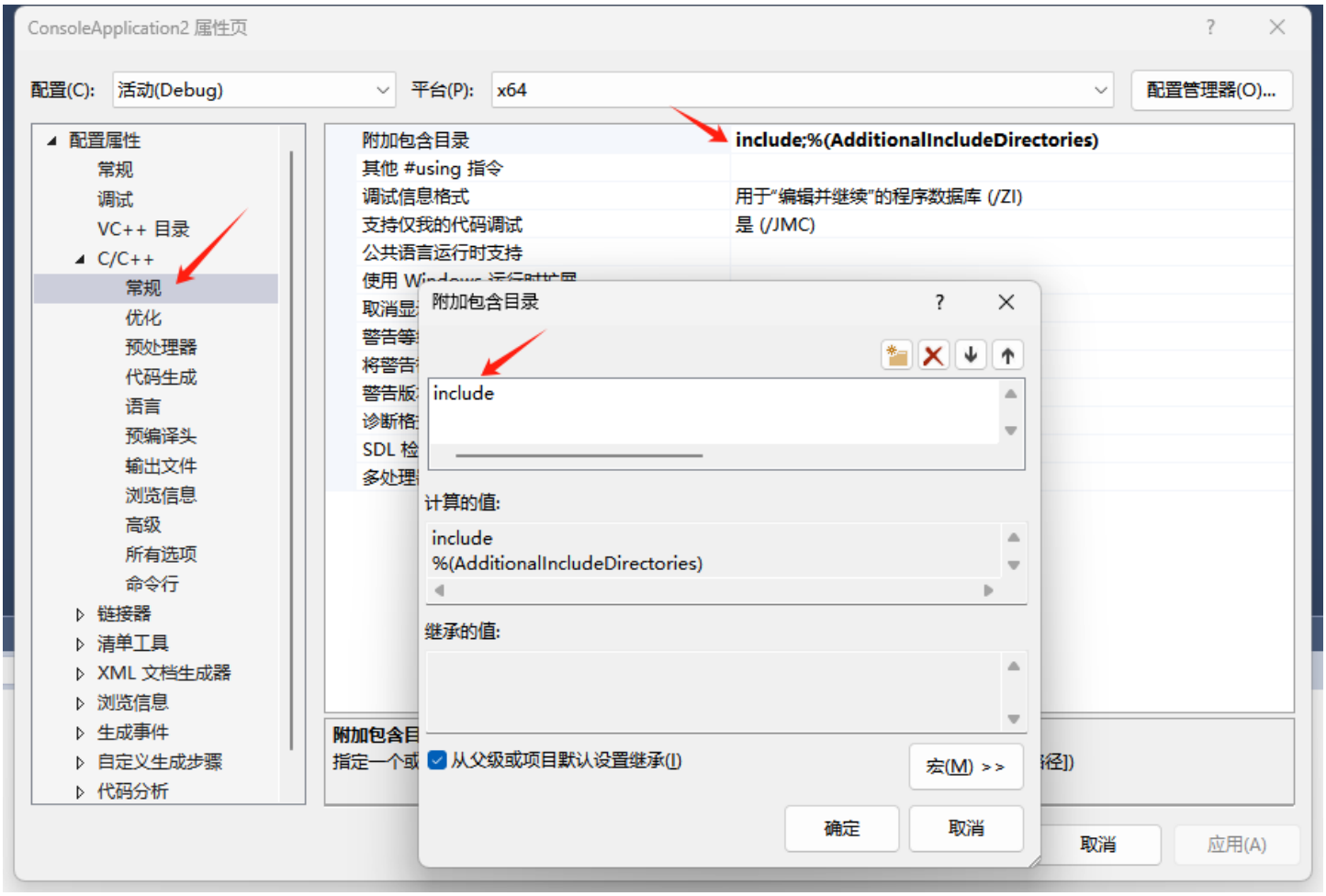


图7 添加附加包含目录

2）添加预处理器定义

调用sdk中的库函数，需要添加两个预处理器，分别为：

EVM_WIN：对应cmsplibEx.h文件中的预编译

_WIN64：编译器识别为64位Windows平台上编译代码（Win32不需要添加）

在【C/C++】-【预处理器】-【预处理器定义】对话中，添加预处理定义EVM_WIN和_WIN64（见图8）。

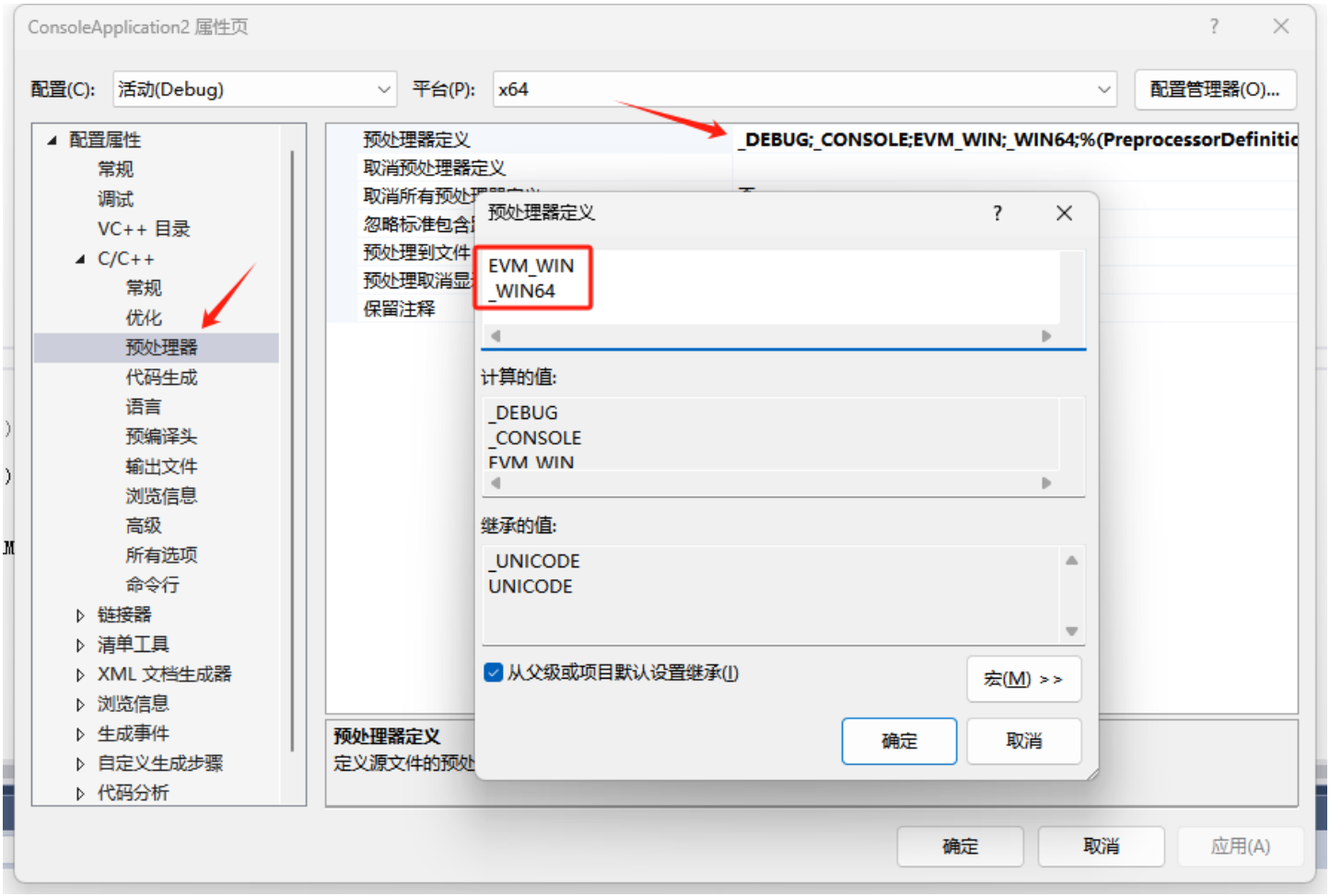


图8 添加预处理器定义

注意：Win32平台只需手动添加EVM_WIN即可，在选择Win32平台时会自动添加WIN32定义。

3）添加附加库目录

在头文件中.h中声明或定义的函数实现，通常封装在静态或动态库中。编译器会在附加库目录查找对应的库并定位函数实现。我们需要把sdk中的lib文件夹复制到项目目录（可参见图5），在项目属性的【链接器】-【常规】-【附加库目录】中添加lib（见图9），点击【确定】后则添加完成。

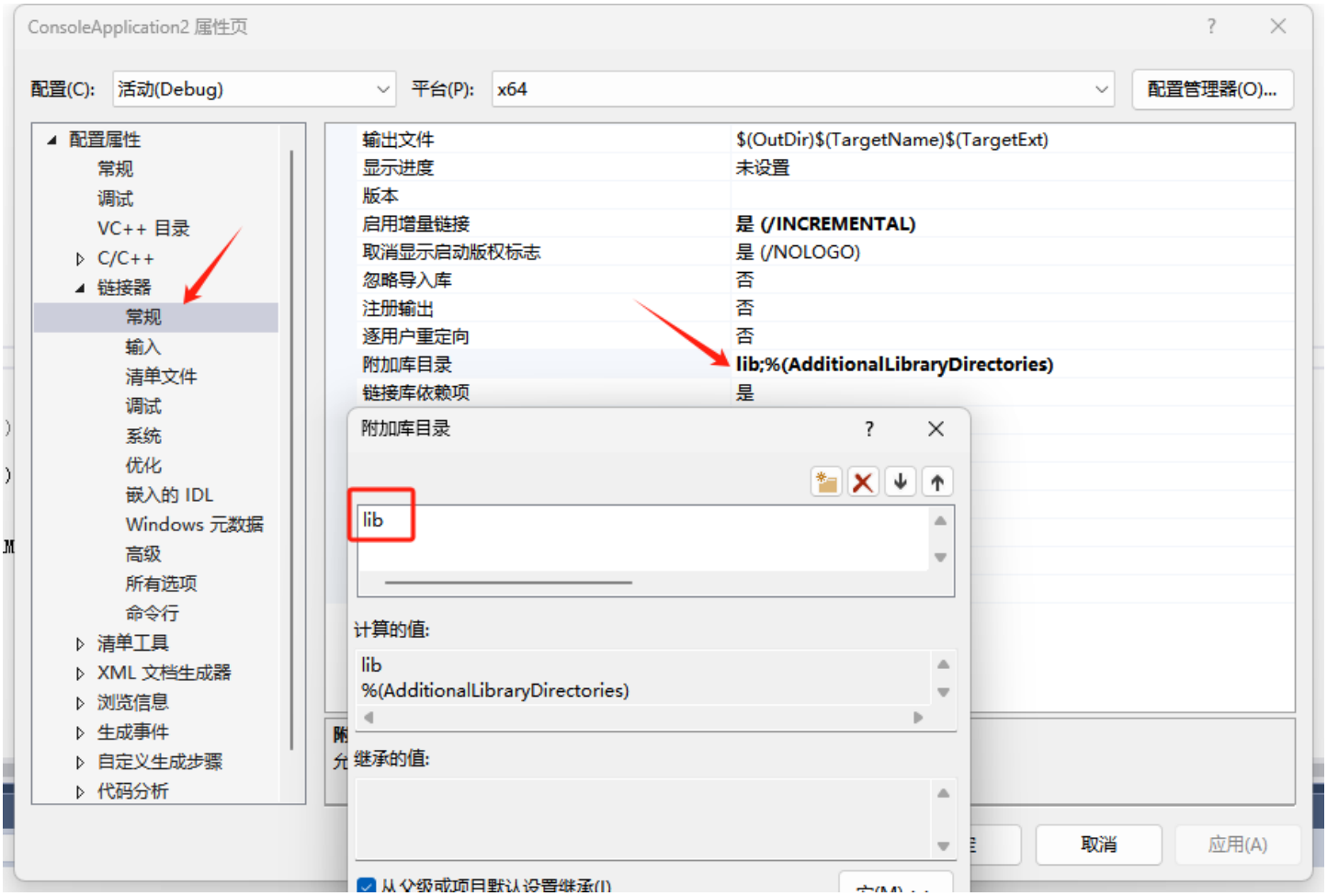


图9 添加附加库目录

4）复制动态库

编译生成的应用程序在运行时，需要依赖动态库。为此，我们需要把sdk中的dll文件夹中的所有.dll文件复制到生成的应用程序目录中（见图10），以保证程序运行后可以在同级目录中找到依赖的动态库。

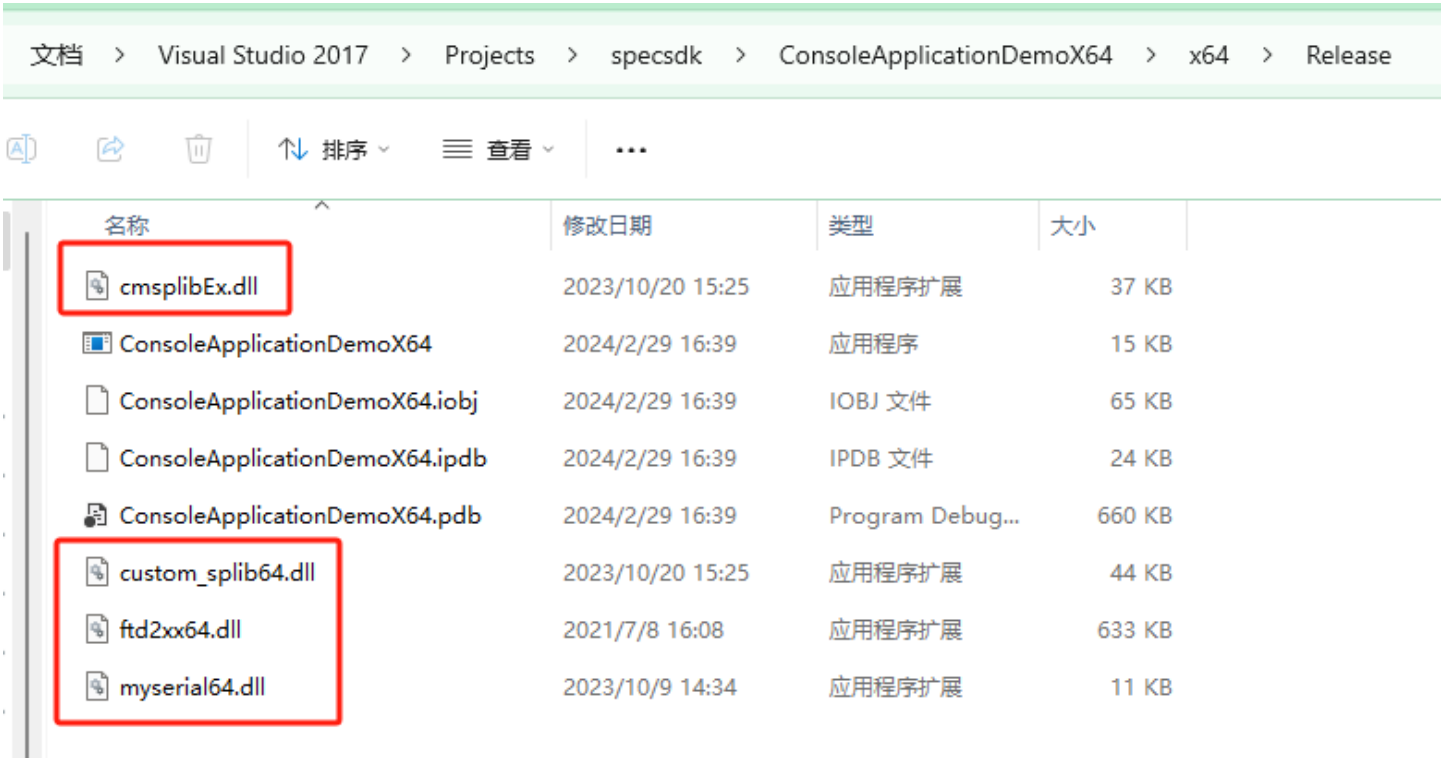


图10 应用程序目录中的动态库

至此，调用sdk中的库函数所需的配置工作已完成。简单总结如下：

1）附加包含目录。编译查找头文件。

2）预处理器定义。编译查找宏定义。

3）附件库目录。编译查找静态库。

4）复制动态库。运行查找依赖动态库。

2 函数调用说明

本例程旨在展示完整的光谱数据读取时所需的设置和调用步骤，且对每个CMSPEX函数都做了返回值判断和打印提示。实际使用时，仅需判断是否打开设备功能即可，不需要每个参数设置步骤都做返回值判断。SDK中的函数非常多，使用时遇到具体需求或问题可查看手册或联系技术支持工程师。

2.1引用头文件和静态库

```
1 #include <cmsplibEx.h>
2 #pragma comment(lib, "cmsplibEx.lib")
```

sdk中的include及lib中有很多文件，我们只需要引用cmsplibEx即可，其他均为cmsplibEx调用所需的依赖文件。

2.2 宏定义

```
#define MAX_PIXELS 2048
#define WL_BUFF_LEN MAX_PIXELS*4
```

光纤光谱仪CMS及CDS系列光谱仪采用的均是2048像素探测器，这里的宏定义MAX_PIXELS是全部像素，WL_BUFF_LEN 是读取全部像素值对应波长值的缓存字节长度，一个波长值用4个字节表示。

2.3 初始化设备

```
//初始化设备
CMSPEX_HANDLE hd;                   //定义设备句柄
uint32_t deviceIndex = 0;           //设备索引号，从0开始
uint32_t commMode = 0;              //使用usb作为通信接口

int ret = CMSPEX_Init(&hd, commMode); //返回0正常

if (0 == ret)
{
        std::cout << "设备初始化成功" << std::endl;
        std::cout << "设备索引:  " << deviceIndex << std::endl;
}
else
{
        std::cout << "设备初始化失败" << std::endl;
}
```

1. 通信方式支持usb和串口通信，0为usb通信，1为串口通信。
2. 定义了设备索引号，从0开始。当连接多台设备时，可根据索引操作对应设备。本例仅展示连接一台光谱仪设备的情形。
3. 这里定义了ret作为返回值变量，用于判断CMSPEX函数返回值并打印提示。

2.4 打开光谱仪设备

```
//*******打开光谱仪设备*******
ret = CMSPEX_OpenByIndex(hd, deviceIndex);
```

```
if ( 0 == ret)
{
        std::cout << "设备" << deviceIndex << ": 打开成功" << std::endl;
}
else
{
        std::cout << "设备" << deviceIndex << ": 打开失败" << std::endl;
}
```

打开光谱仪有多种方式，可根据设备序列号、设备索引，如采用串口通信可直接使用串口参数打开。本例仅展示最为常用的设备索引打开方式。

2.5 设置积分时间

```
//*******设置积分时间*******
uint32_t integrationTimeMs = 100;         //单位是ms
uint32_t integrationTimeUs = integrationTimeMs * 1000; //单位是us

ret = CMSPEX_SetIntegrationTime(hd, integrationTimeUs);         //设置积分时间，参数是us，注意与ms作区分
if (0 == ret)
{
        std::cout << "设备" << deviceIndex << ": 积分时间设置成功" << std::endl;
}
else
{
        std::cout << "设备" << deviceIndex << ": 积分时间设置失败" << std::endl;
}
```

参数中的积分时间单位是us，实际使用ms更直观，建议采用如下方式：

```
uint32_t integrationTimeMs = 100;         //单位是ms
ret = CMSPEX_SetIntegrationTime(hd, integrationTimeMs*1000);
```

2.6 设置平均次数

```
//*******设置平均次数*******
uint32_t averageTimes = 2;                  //设置平均次数为2

ret = CMSPEX_SetAccAvg(hd, averageTimes);
```

```
5
6 if (0 == ret)
7 {
8         std::cout << "设备" << deviceIndex << ": 平均次数设置成功" << std::endl;
9 }
10 else
11 {
12         std::cout << "设备" << deviceIndex << ": 平均次数设置失败" << std::endl;
13 }
```

光谱仪设备的采集时间 = 积分时间*平均次数，在设备平均次数时，需要注意结合积分时间，不能导致采集时间过长，如采集时间超过1s时就会让人有比较强烈的卡顿感。

2.7 设置平滑次数

```
1 //*******设置平滑次数*******
2 uint32_t smoothingTimes = 2;         //设置平滑次数为2
3
4 ret = CMSPEX_SetPixSmoothingWidthDown(hd, smoothingTimes);
5
6 if (0 == ret)
7 {
8         std::cout << "设备" << deviceIndex << ": 平滑次数设置成功" << std::endl;
9 }
10 else
11 {
12         std::cout << "设备" << deviceIndex << ": 平滑次数设置失败" << std::endl;
13 }
```

平滑次数越高，光谱曲线观察起来越平滑，看着也更舒适。但是，平滑本质上是每个像素与相邻像素作平均后求得，会有“消峰”效果，对于激光波长、原子谱线等需要观察峰值的场景，平滑次数应设置为0。对于较为平坦的宽谱，如led光谱、钨灯光谱等，可适当设置平滑次数，但不宜设置过大导致影响观察的特征谱。

2.8 读取波长

```
1 //*******获取所有波长值*******
2 uint32_t wl_len;
3
4 //每个像素对应的波长值用4个字节（char型）表示,wl_len用于读取波长时控制获取char型长度
5 wl_len = MAX_PIXELS * 4;
6
```

```
 //存储波长返回值的数组，共需要 MAX_PIXELS*4 个字节
unsigned char buf[WL_BUFF_LEN] = { '0' };

double wavelengths[MAX_PIXELS];

memset(wavelengths, 0, sizeof(double)*MAX_PIXELS); //波长数组清零

//读取波长数据到buf数组,其中第3个参数0是指从第0个像素开始， 第4个参数len是返回的char型字节数量
ret = CMSPEX_GetWavelength(hd, buf, 0, wl_len);

if (ret == 0) {
        // 获取波长成功
        // 进行数据转化，以低位在前，高位在后的方式，每4个字节为一个波长数据

        float fData; //定义单精度浮点型，4个字节，用于临时存储波长值

        for (int i = 0; i < MAX_PIXELS; i++)
        {
                //每4个字节为一组，赋值给 fData，获取一个波长值
                memcpy( &fData, buf + (i * 4), sizeof(char) * 4 );
                wavelengths[i] = fData; //获取的波长值赋值给波长数组对应索引
        }

        //打印第501-505个波长值
        for (int j = 500; j < 505; j++)
        {
                std::cout <<  "像素索引" << j << ": " << wavelengths[j] << std::endl;
        }
}
else
{
        std::cout << "获取波长失败!" << std::endl;
}
```

在读取波长时，CMSPEX_GetWavelength获取的是字节流，每个波长值用4个字节表示，所以总字节长度wl_len值为 MAX_PIXELS * 4，即2048*4=8192。

波长存储定义了2个数组，其中buf数组用于接收字节流，wavelengths数组用于存储波长值。获取的字节流需要进行数据转换后，才能存储到wavelengths数组。不理解该行代码的小伙伴，在数据转换时直接复制例程代码即可。

```
memcpy( &fData, buf + (i * 4), sizeof(char) * 4 );
```

最后打印出第501-505个波长值用于展示。

2.9 获取光谱数据

```
//*******获取光谱数据*******
uint32_t valuesLength = MAX_PIXELS * 2; //2048*2 = 4096
int timeout = 602;
unsigned char valuesBuf[4096] = { '0' };        //光谱返回值字节流缓存
uint16_t values[MAX_PIXELS] = { '0' };                //光谱值数组

//从第0个值开始读取，共4096字节
ret = CMSPEX_Collection(hd, valuesBuf, 0, valuesLength, timeout);

if ( 0 == ret)
{
        //获取光谱数据成功

        uint8_t tBuf[2] = { '0' };
        uint16_t value;        //用于临时存储光谱值

        for (int i = 0; i < MAX_PIXELS; i++)
        {
                tBuf[1] = valuesBuf[2*i];
                tBuf[0] = valuesBuf[2*i + 1];
                memcpy(&value, tBuf, sizeof(char) * 2);
                values[i] = value;
        }

        //打印第501-505个波长值和对应光谱强度值

        std::cout << std::endl;                //打印换行，便于观察

        for (int j = 500; j < 505; j++)
        {
                std::cout << "波长" << ": " << wavelengths[j] << "; " <<
                                                "值" << ": " << values[j] <<
std::endl;
        }
}
std::cout << std::endl;                //打印换行，便于观察
```

3 完整代码和打印提示

3.1 完整代码

以上步骤的完整代码如下，代码后面是程序运行后的打印提示。完整代码可在网站下载中心或联系工程师获取。

```
// ConsoleApplication2.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
// 本例程旨在展示调试步骤，对每个CMSPEX函数都做了返回值判断。实际使用时，仅需判断是否打开设备功能即可。
//

#include <iostream>
#include <cmsplibEx.h>
#pragma comment(lib, "cmsplibEx.lib")

#define MAX_PIXELS 2048
#define WL_BUFF_LEN MAX_PIXELS*4

int main()
{
    //初始化设备
    CMSPEX_HANDLE hd;                           //定义设备句柄
    uint32_t deviceIndex = 0;          //设备索引号，从0开始
    uint32_t commMode = 0;                  //使用usb作为通信接口

    int ret = CMSPEX_Init(&hd, commMode); //返回0正常

    if (0 == ret)
    {
            std::cout << "设备初始化成功" << std::endl;
            std::cout << "设备索引:  " << deviceIndex << std::endl;
    }
    else
    {
            std::cout << "设备初始化失败" << std::endl;
    }
    std::cout << std::endl;                 //打印换行，便于观察

    //打开光谱仪设备
    if (0 == ret)
    {
            //*******打开光谱仪设备*******
            ret = CMSPEX_OpenByIndex(hd, deviceIndex);
            if ( 0 == ret)
            {
```

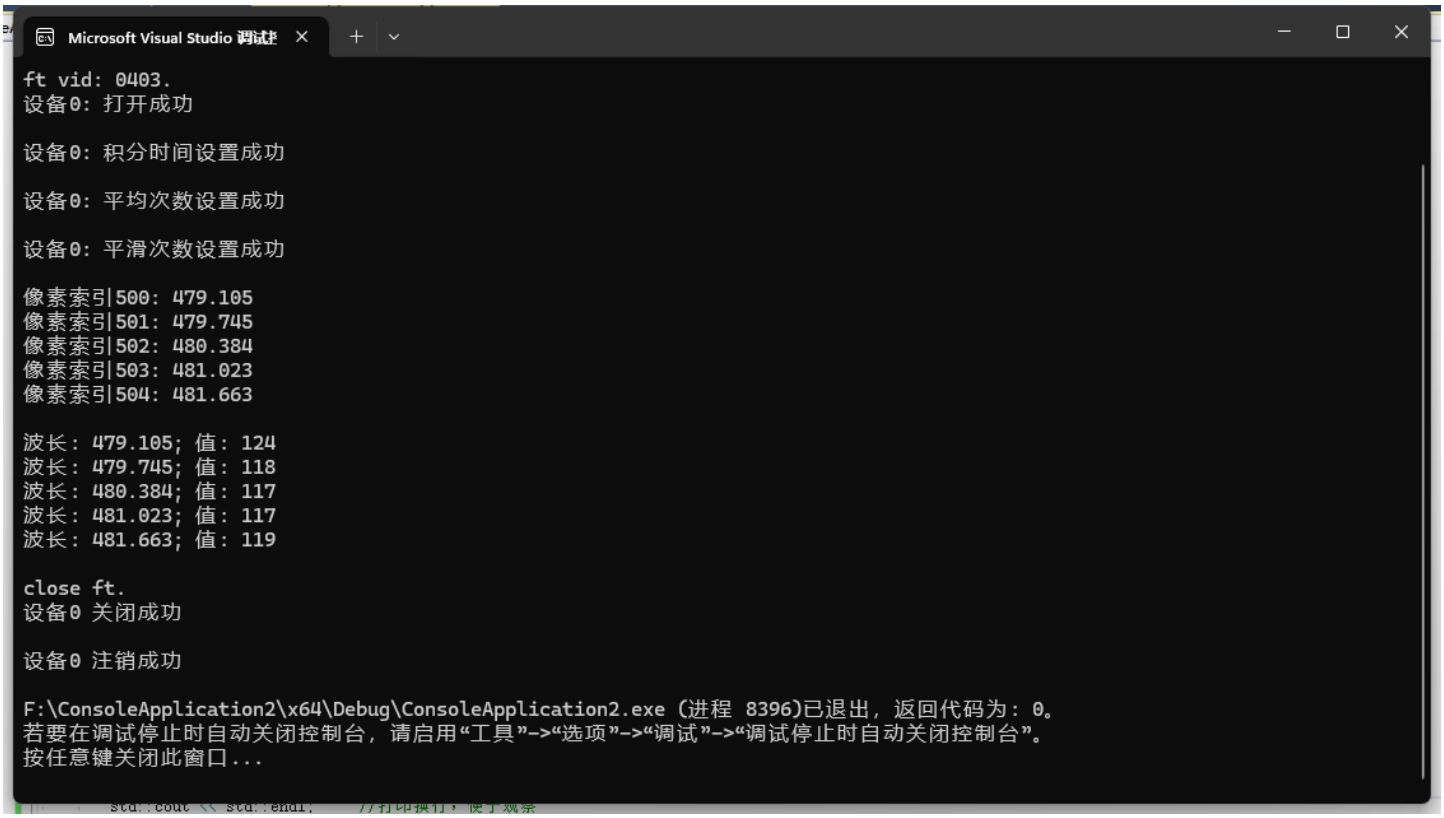
```
39                      std::cout << "设备" << deviceIndex << ": 打开成功" <<
   std::endl;
40              }
41              else
42              {
43                      std::cout << "设备" << deviceIndex << ": 打开失败" <<
   std::endl;
44              }
45              std::cout << std::endl;                 //打印换行，便于观察
46
47
48              //*******设置积分时间*******
49              uint32_t integrationTimeMs = 100;         //单位是ms
50              uint32_t integrationTimeUs = integrationTimeMs * 1000; //单位是us
51
52              //设置积分时间，参数是us，注意与ms作区分
53              ret = CMSPEX_SetIntegrationTime(hd, integrationTimeUs);
54              if (0 == ret)
55              {
56                      std::cout << "设备" << deviceIndex << ": 积分时间设置成功" <<
   std::endl;
57              }
58              else
59              {
60                      std::cout << "设备" << deviceIndex << ": 积分时间设置失败" <<
   std::endl;
61              }
62              std::cout << std::endl;                 //打印换行，便于观察
63
64
65              //*******设置平均次数*******
66              uint32_t averageTimes = 2;                //设置平均次数为2
67
68              ret = CMSPEX_SetAccAvg(hd, averageTimes);
69
70              if (0 == ret)
71              {
72                      std::cout << "设备" << deviceIndex << ": 平均次数设置成功" <<
   std::endl;
73              }
74              else
75              {
76                      std::cout << "设备" << deviceIndex << ": 平均次数设置失败" <<
   std::endl;
77              }
78              std::cout << std::endl;                 //打印换行，便于观察
79
```

```
80
81              //*******设置平滑次数*******
82              uint32_t smoothingTimes = 2;         //设置平滑次数为2
83
84              ret = CMSPEX_SetPixSmoothingWidthDown(hd, smoothingTimes);
85
86              if (0 == ret)
87              {
88                      std::cout << "设备" << deviceIndex << ": 平滑次数设置成功" <<
    std::endl;
89              }
90              else
91              {
92                      std::cout << "设备" << deviceIndex << ": 平滑次数设置失败" <<
    std::endl;
93              }
94              std::cout << std::endl;                  //打印换行，便于观察
95
96
97              //*******获取所有波长值*******
98              uint32_t wl_len;
99
100             //每个像素对应的波长值用4个字节（char型）表示,wl_len用于读取波长时控制获取
    char型长度
101             wl_len = MAX_PIXELS * 4;
102
103             //存储波长返回值的数组，共需要 MAX_PIXELS*4 个字节
104             unsigned char buf[WL_BUFF_LEN] = { '0' };
105
106             double wavelengths[MAX_PIXELS];
107
108             memset(wavelengths, 0, sizeof(double)*MAX_PIXELS); //波长数组清零
109
110             //读取波长数据到buf数组,其中第3个参数0是指从第0个像素开始， 第4个参数len是返
    回的char型字节数量
111             ret = CMSPEX_GetWavelength(hd, buf, 0, wl_len);
112
113             if (ret == 0) {
114                     // 获取波长成功
115                     // 进行数据转化，以低位在前，高位在后的方式，每4个字节为一个波长数
    据
116
117                     float fData; //定义单精度浮点型，4个字节，用于临时存储波长值
118
119                     for (int i = 0; i < MAX_PIXELS; i++)
120                     {
121                             //每4个字节为一组，赋值给 fData，获取一个波长值
```

```
122                             memcpy( &fData, buf + (i * 4), sizeof(char) * 4 );
123
124                             //获取的波长值赋值给波长数组对应索引
125                             wavelengths[i] = fData;
126                     }
127
128                     //打印第501-505个波长值
129                     for (int j = 500; j < 505; j++)
130                     {
131                             std::cout <<  "像素索引" << j << ": " <<
    wavelengths[j] << std::endl;
132                     }
133             }
134             else
135             {
136                     std::cout << "获取波长失败!" << std::endl;
137             }
138
139
140             //*******获取光谱数据*******
141             uint32_t valuesLength = MAX_PIXELS * 2; //2048*2 = 4096
142             int timeout = 602;
143             unsigned char valuesBuf[4096] = { '0' };        //光谱返回值字节流缓存
144             uint16_t values[MAX_PIXELS] = { '0' };                  //光谱值数组
145
146             //从第0个值开始读取，共4096字节
147             ret = CMSPEX_Collection(hd, valuesBuf, 0, valuesLength, timeout);
148
149             if ( 0 == ret)
150             {
151                     //获取光谱数据成功
152
153                     uint8_t tBuf[2] = { '0' };
154                     uint16_t value;         //用于临时存储光谱值
155
156                     for (int i = 0; i < MAX_PIXELS; i++)
157                     {
158                             tBuf[1] = valuesBuf[2*i];
159                             tBuf[0] = valuesBuf[2*i + 1];
160                             memcpy(&value, tBuf, sizeof(char) * 2);
161                             values[i] = value;
162                     }
163
164                     //打印第501-505个波长值和对应光谱强度值
165
166                     std::cout << std::endl;                 //打印换行，便于观察
```

```
167
168                     for (int j = 500; j < 505; j++)
169                     {
170                             std::cout << "波长" << ": " << wavelengths[j] << ";
    " <<
171                                                     "值" << ": " <<
    values[j] << std::endl;
172                     }
173             }
174             std::cout << std::endl;                 //打印换行，便于观察
175
176
177             //*******关闭设备*******
178             ret = CMSPEX_Close(hd);
179
180             if (0 == ret)
181             {
182                     std::cout << "设备" << deviceIndex << " 关闭成功" <<
    std::endl;
183             }
184             else
185             {
186                     std::cout << "设备" << deviceIndex << " 关闭失败" <<
    std::endl;
187             }
188             std::cout << std::endl;                 //打印换行，便于观察
189
190
191             //*******注销设备，非必要操作*******
192
193             ret = CMSPEX_DeInit(hd);
194
195             if (0 == ret)
196             {
197                     std::cout << "设备" << deviceIndex << " 注销成功" <<
    std::endl;
198             }
199             else
200             {
201                     std::cout << "设备" << deviceIndex << " 注销失败" <<
    std::endl;
202             }
203     }
204
205 }
```

3.2 打印提示


Microsoft Visual Studio 调试控
ft vid: 0403.
设备0: 打开成功
设备0: 积分时间设置成功
设备0: 平均次数设置成功
设备0: 平滑次数设置成功
像素索引[500: 479.105
像素索引[501: 479.745
像素索引[502: 480.384
像素索引[503: 481.023
像素索引[504: 481.663
波长: 479.105; 值: 124
波长: 479.745; 值: 118
波长: 480.384; 值: 117
波长: 481.023; 值: 117
波长: 481.663; 值: 119
close ft.
设备0 关闭成功
设备0 注销成功
F:\ConsoleApplication2\x64\Debug\ConsoleApplication2.exe (进程 8396)已退出，返回代码为: 0。
若要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...