EasyNIRLib Programming Guide

版本	1.1.2
日期	2020-12-15
版权	深圳谱研互联科技有限公司

版权所有　　侵权必究

总部
电话：
传真：
深圳谱研互联科技有限公司
深圳市宝安区西乡街道共和工业路 107 号
华丰互联网创意园 A 座 116 室
销售：sales@pynect.com
订购：orders@pynect.com
支持：techsupport@pynect.com

Copyright © 2020Pynect, Inc.
保留所有权利。本文档的任何部分不得复制、存储在检索系统中，未经深圳谱研互联科技有限公司书面许可，不得擅自以任何方式通过电子、机械、复印、录音等活动进行传播。
责任范围
我们已尽一切努力使本手册尽可能完整和准确，但不暗示任何保证或适用性。所提供的资料是在“现状”的基础上提供的。对于因本手册所含信息而造成的任何损失或损害，深圳谱研互联科技有限公司对任何个人或实体不承担任何责任。

修订记录

日期	修订内容	修订者
2020-06-15	初版	王德全
2020-09-17	增加日志功能（部分接口）	王德全
2020-11-10	1. 增加扫描参数设置功能 2. 统一错误代码	王德全
2020-12-04	增加必要步骤说明	王德全
2020-12-13	完善 labview 动态库说明	王德全
2020-12-15	1. 规范版本名称说明 2. 增加通用版本的测试用例	王德全

关于本手册

文档目标和受众

本文档为您提供 EasyNIRLib 动态库（最新版本 1.1.2）的操作说明。

文档新特性

EasyNIRLib 编程指南的版本更新到 1.1.2 版。

文档摘要

章节	描述
第一章:简介	简述 EasyNIRLib 动态库的概述。
第二章:基本操作	描述应用程序为控制光谱仪和获取数据而必须执行的典型操作顺序。
第三章:采集参数	提供所有可用的采集参数的完整列表。
第四章:可选特性	列出和描述一些光谱仪提供的可选功能。
第五章:开发 EasyNIRLib 应用程序	提供有关为特定编程语言设置开发环境以开发 EasyNIR 应用程序的信息。
第六章:示例程序	包含 C，C++，C 语言的示例程序。
第七章:接口封装	提供有关一些接口封装的详细信息功能。
附录 A:常见问题	列出常见问题和答案。
附录 B:通用接口说明	通用版本动态库说明

产品相关文档

您可以访问我们的网站（http://www.pynect.com）查看谱研互联产品的文档。

选择[技术支持]->[技术文档]->[产品手册]，在产品列表中选择合适的文档。

升级

偶尔，你可能会发现你需要谱研互联来改变或升级你的系统。为了方便这些更改，您必须首先联系技术支持或销售并获得退货或在线升级授权。具体操作请与谱研互联联系以获取具体说明。

第一章
简介

概述

EasyNIRLib 是一个功能强大的软件开发工具包（SDK），适用于 Windows 和 Linux 操作系统，允许您快速地为您的近红外光谱仪编写定制软件的解决方案。

EasyNIRLib 提供了一个应用程序编程接口（API），它是我们最好的软件驱动程序包，并允许您在一个单一的跨平台驱动程序中，充分利用高速数据采集的能力。将 EasyNIRLib 集成到您自己的软件应用程序中，以便在多种操作系统环境中能完全自主控制光谱仪和设备。

跨平台

EasyNIRLib 是用 C++语言开发创建的，适用于 windows、Mac 和 linux。使用 EasyNIRLib，您可以开发健壮的应用程序来控制跨这些不同操作系统的多个近红外光谱仪。

多语言支持

您可以用以下语言开发基于 EasyNIRLib 的应用程序。

-C++
-C#

多接口支持

-C++（可与微软 Visual Studio/Windows，GCC / Linux 一起使用）
-COM 对象接口（仅限 Windows）

使用 C++编写的应用程序，几乎可以适应于任何操作系统，可以跨平台工作。EasyNIRLib 不仅与平台无关，而且与光谱仪无关；相同的代码适用于几乎所有的近红外光谱仪设备。

特性

EasyNIRLib 提供以下功能：

特性

EasyNIRLib 提供个两套动态库，分为普通版和通用版，如果您的开发环境无法方便的使用复杂的接口参数，您需要使用通用版，通用版和普通版的区别就是，通用版无复杂的接口参数，全是基本数据类型，但这样的封装增加了操作步骤，具体见附录 B 通用接口说明。

EasyNIRLib 分别提供 X86 版本和 X64 的 Debug 和 Release 版本，您需要根据您的开发环境，选择不同的版本。在 windows 操作系统中，您可以选中动态库->右键->属性->详细信息->产品名称，获取产品信息。Debug 版本的动态库名称均有 d 后缀，方便您知晓。在 linux 系统中，您可以使用 file 命令查看动态库信息。

若无特殊说明，均以普通版为例。

支持的光谱仪设备

-NIR-M-R2
-NIR-S-G1
-NIR-S-F1

支持的操作系统

-Windows:

Windows XP, Windows 7, Windows 10

32-bit/64-bit versions of XP, and Windows 7

-Mac:

OSX 10.5 or later

-Linux:

支持大多数的 x86 发行版

需要内核 2.4.20 和 libusb 0.1.10 或更高版本

GCC 编译器版本最低为 4.8

尽管除了 Windows 之外，EasyNIRLib 还支持 Linux 和 Mac OS/X 平台，但本编程指南是从 Windows 开发人员的角度编写的。通常，平台之间的差异只是表面的（例如，Linux 有“共享对象”（*.so）文件，Mac 有“*.dylib”文件，而 Windows 有“动态链接库”（*.dll））。

架构

您可以通过 DLL 文件访问 EasyNIRLib 功能：

-EasyNIRLib32.dll（或 64 位系统上的 EasyNIRLib64.dll）包含允许你要控制所有的光谱仪设置　并获取光谱的函数。例如，您可以设置曝光时间、扫描次数等。

编写语言

EasyNIRLib 的大部分功能都是用 C、C++语言实现的。这是 EasyNIRLib 在多个平台上运行的关键。上述动态库中的函数只调用相应的 C++函数，传递所需的参数，动态库中的这些函数会转换每个函数的参数，从而达到进入底层接口所期望的形式，然后调用底层接口的方法。您需要有 c++11 编译器的支持，内部实现使用 c++11 特性，以便能正确解析动态库的操作。

USB 访问

大多数近红外光谱仪都是通过 USB 接口与计算机通信。且使用的是 USB-HID 协议，无需安装驱动即可进行交互。

串口访问

此版本暂无说明

蓝牙访问

此版本暂无说明

几乎所有的近红外光谱仪设备都支持 USB、串口、蓝牙三种通讯方式，通讯优先级如下：

USB > 串口> 蓝牙

接口封装

EasyNIRLib 内部封装了代表每个不同近红外光谱仪的类（例如 NIR-M-R2、NIR-S-G1 等）。但是，对光谱仪的所有访问，不管设备型号如何，都应该通过统一的接口对象进行访问。应用程序应该实例化接口对象，然后使用此接口对象的方法连接并控制光谱仪。

第二章

基本操作

概述

本章描述了应用程序控制光谱仪和获取光谱所必须执行的典型操作序列。这里描述的功能对所有的近红外光谱仪都是通用的。此外，有关每个操作的更多详细信息，请参见第 7 章：接口封装。本章为每个方法调用提供精确的语法，包括所有参数和数据类型。

初始化设备

此步骤是必要的。

在控制光谱仪之前，必须枚举近红外光谱仪设备，这是获取光谱仪所有功能的必须操作。您将获得所有连接的光谱仪设备。enumerateDevices 出参为一个 0-N 之间的整数，表示找到的 USB 光谱仪的数量。如果发生错误，此方法将返回 false。在这种情况下，可以调用 getLastException（）函数来了获取更多错误信息。

```
/// 枚举设备数量
Int devNum = 0;
enumerateDevices (devNum );
```

打开设备

此步骤是必要的。

在您进行光谱仪的操作前，您还必须打开指定的光谱仪设备，如果您在[初始化设备]中获取到光谱仪设备，您需要指定打开的光谱仪序号，序号从 0 开始计数，例如，如果您想打开 1 号光谱仪，那么光谱仪序号则为 0。

```
/// 打开设备
const bool ret = openSpectrometer(0);
```

获取扫描参数

此步骤是可选的。

此步骤可以获取到设备的扫描参数的具体配置。

```
/// 获取设备扫描参数配置
std::multimap<std::string, ScanSection> sectionMap;
const bool ret = getScanSection(sectionMap);
```

设置采集参数

此步骤是可选的。

如果您的设备含有多个采集参数设置，那么您可以指定某个采集参数，以便满足您的需求。如果您的设备只包含一个采集参数设置，或者您没有指定采集参数，则采用默认采集设置。由于扫描配置可以使用最多 5 个扫描截面配置，所以您必须一起传入扫描截面配置参数，如果您修改了其中一个截面的扫描配置，您也必须将其他未修改的扫描配置一起传入。

```
/// 设置设备扫描参数配置
//std::vector<ScanSection> cfgSections;
//ScanSection tmpCfg1;
//tmpCfg.m_sectionScanType = "Column";
//tmpCfg.m_wavelengthStart = 900;
//tmpCfg.m_wavelengthEnd = 1700;
//tmpCfg.m_widthPixel = 8.20;
//tmpCfg.m_patternsNum = 30;
//tmpCfg.m_exposureTime = 30.480;
/// 采样次数
unsigned short averNum = 6;
const bool ret = setTargetLists("Column 1", cfgSections, averNum );
```

获取光谱

此步骤是必要的。
现在您已经做好获取光谱数据的准备工作了。获取光谱仪的光谱数据包含两步操作
首先获取光谱的数据长度，然后获取光谱数据。

```
/// 光谱数据的数据长度
int fileSize = 0;
getFormattedSpectrumLength(&fileSize);
/// 申请数据内存-波长
double* pWavelengthData = (double*)malloc(fileSize * sizeof(double));
/// 申请数据内存-数据
unsigned int* pData = (unsigned int*)malloc(fileSize * sizeof(int));
/// 获取光谱数据
getFormattedSpectrum(pWavelengthData, pData);
```

第三章
采集参数

概述

本章提供了所有可用的采集参数的完整列表。采集参数是控制光谱采集条件的某些方面的设置。所有的光谱仪都支持这些采集参数。

像素列表

获取像素宽度。

函数调用:

```
getWidthPixels(list)
```

曝光时间列表

获取曝光时间列表,曝光时间为固定的数值,分别为
0.635ms, 1.270ms, 2.54ms, 5.08ms, 15.24ms, 30.48ms, 60.96ms。

函数调用:

```
getExposureTimes(list)
```

模式列表

获取模式范围，NIR 设备最多运行 624 种模式。

函数调用:

```
getPatternsNumRange (list)
```

版本信息

获取 Tiva 版本，DLP 版本，flash 版本，光谱仪版本，校准参数版本，参考参数版本，配置参数版本。

函数调用:

```
getVersion(pTivaSwVersion, pDlpcswVersion, pDlpcFlashBuildVersion, pSpecLibVer, pCalDataVer, pRefCalDataVer, pCfgDataVer)
```

第四章

可选特性

概述

一些特性被定义为一些光谱仪提供的可选功能，但不是所有的光谱仪都有。

开关光源

只有当设备带有光源时，您才可以设置光源的开启或者关闭。

第五章

开发 EasyNIRLib 应用程序

概述

以下部分提供了有关为开发 EasyNIRLib 应用程序的特定编程语言设置开发环境的信息。

使用 C/C++进行开发(所有 IDE)

所需头文件

如果使用 C 或 c++进行开发，则需要几个头文件。
必须通过向 include 列表中添加以下目录来更新 IDE 的项目设置
目录:

```
$(EASYNIR_HOME)\include
```

EASYNIR_HOME 表示动态链接库所在的位置。

在 Microsoft Visual Studio 2017 中添加项目目录

过程如下:

要在 Microsoft Visual Studio 2017 中添加项目目录,
1. 打开项目属性窗口。
2. 展开 C/ C++分支，然后单击常规。
3.将上面列出的目录添加到附加的附件包含目录属性中。

所需库文件

必须更新 IDE 的项目设置，以便能够定位 EasyNIRLib.lib 文件。

过程如下:

在 Microsoft Visual Studio 2017 中,
1. 打开项目属性窗口，展开链接器分支，然后单击常规。
2. 将 EASYNIR_HOME 目录的完整路径添加到附加库目录属性的目录列表中。
3.单击链接器设置的输入项并添加 EasyNIRLib.lib 添加到附加依赖项属性。

第六章
示例程序

概述

本节包含以下编程语言的 EasyNIRLib 示例程序:

C

Labview

C 样例

语言: C 语言
环境: Microsoft Visual Studio 2017
访问方法: 使用 c 接口访问 EasyNIRLib 功能
GUI:无
下面将演示如何操作:
-访问一个光谱仪
-枚举设备数量
-打开设备
-获取采集参数
-获取光谱数据

```
/// 枚举设备数量
int devNum = 0;
if(!enumerateDevices(devNum))
{
	printf("Enumerate NIR device error, exit!!!\r\n");
	system("pause");
	return 0;
}
printf("NIR device number:%d\r\n", devNum);

/// 打开设备
if (!openSpectrometer(devNum - 1)) {
	printf("Failed to open NIR device,error:%s\r\n", getLastException());
	system("pause");
	return 0;
}
printf("Open NIR device success\r\n");
```

```
/// 获取扫描类型
std::map<int, std::string> scanTypeMap;
const auto retType = getScanType(scanTypeMap);

/// 获取扫描类型
int index = 0;
std::vector<double> widthPixelList;
const auto retWidth = getWidthPixels(widthPixelList);
for (auto iter = widthPixelList.begin(); iter != widthPixelList.end(); ++iter) {
    printf("width pixel[%d]:%0.02f\r\n", index++, *iter);
}

std::vector<double> exTimeList;
const auto retExposure = getExposureTimes(exTimeList);
for(auto iter = exTimeList.begin(); iter!=exTimeList.end(); ++iter){
    printf("exposure time:%0.02f\r\n", *iter);
}

int start = 0, end = 0;
const auto retWavlength = getPatternsNumRange(start, end);
printf("start:%d end:%d\r\n", start, end);

/// 获取设备扫描参数配置
std::map<std::string, ScanSection> sectionMap;
const auto retScanSection = getScanSection(sectionMap);

if (!retScanSection ) {
    printf("get dev scan cfg fail,exit!!!\r\n");
    system("pause");
    return 0;
}

for (auto iter = sectionMap.begin(); iter != sectionMap.end(); ++iter)
{
    printf("Name:%s Method:%s Width:%0.02f Start:%d End:%d Dig Resolution:%d Exposure
Time:%0.02f\r\n",
        iter->first.c_str(),
        iter->second.m_sectionScanType.c_str(),
        iter->second.m_widthPixel,
        iter->second.m_wavelengthStart,
        iter->second.m_wavelengthEnd,
        iter->second.m_patternsNum,
        iter->second.m_exposureTime);
}
```

```
int fileSize = 0;
getFormattedSpectrumLength(&fileSize);
double* pWavelengthData = (double*)malloc(fileSize * sizeof(double));
unsigned int* pData = (unsigned int*)malloc(fileSize * sizeof(int));
getFormattedSpectrum(pWavelengthData, pData);
for (int i = 0; i < fileSize; i++)
{
	printf("RawData[%d]:%0.02f %d\n", i, pWavelengthData[i], pData[i]);
}

/// 可选
/// 新增或修改扫描配置
std::vector<ScanSection> cfgSections;
ScanSection tmpCfg1, tmpCfg2, tmpCfg3, tmpCfg4;
tmpCfg1.m_sectionScanType = "Column";
tmpCfg1.m_wavelengthStart = 900;
tmpCfg1.m_wavelengthEnd = 1700;
tmpCfg1.m_widthPixel = 8.20;
tmpCfg1.m_patternsNum = 30;
tmpCfg1.m_exposureTime = 30.480;

tmpCfg2.m_sectionScanType = "Column";
tmpCfg2.m_wavelengthStart = 1000;
tmpCfg2.m_wavelengthEnd = 1200;
tmpCfg2.m_widthPixel = 10.54;
tmpCfg2.m_PatternsNum = 30;
tmpCfg2.m_ExposureTime = 30.480;

tmpCfg3.m_sectionScanType = "Column";
tmpCfg3.m_wavelengthStart = 1200;
tmpCfg3.m_wavelengthEnd = 1400;
tmpCfg3.m_widthPixel = 8.20;
tmpCfg3.m_PatternsNum = 30;
tmpCfg3.m_ExposureTime = 30.480;

tmpCfg4.m_sectionScanType = "Column";
tmpCfg4.m_wavelengthStart = 1400;
tmpCfg4.m_wavelengthEnd = 1700;
tmpCfg4.m_widthPixel = 10.54;
tmpCfg4.m_PatternsNum = 30;
tmpCfg4.m_ExposureTime = 30.480;

cfgSections.push_back(tmpCfg1);
```

```
cfgSections.push_back(tmpCfg2);
cfgSections.push_back(tmpCfg3);
cfgSections.push_back(tmpCfg4);

if(!setTargetLists("Column 3", cfgSections,6))
{
	printf("Set scan cfg ,error:%s\r\n", getLastException());
	system("pause");
	return 0;
}

/// 删除扫描配置
removeTargetCfg("Column 3");

/// 获取扫描参数的剩余模式数
std::string cfgName = "Column 1";
unsigned short num = 0;

if(!getRemainPatternsNum(cfgName, 1, num))
{
	printf("Get remain patterns num ,error:%s\r\n", getLastException());
	system("pause");
	return 0;
}

/// 获取活动扫描参数序号
int activeIndex = -1;
if(!getActiveScanIndex(activeIndex))
{
	printf("Get active index ,error:%s\r\n", getLastException());
	system("pause");
	return 0;
}

/// 更改活动扫描参数序号
char tmpActiveIndex = 0;
setActiveScanIndex(tmpActiveIndex);
```

Labview 样例

语言: C 语言
环境: LabVIEW2010 Version10.0(32-bit)
访问方法: 使用 c 接口访问 EasyNIRLib 功能

GUI:labview 操作面板

样例采样直接调用动态库的方式，此方式不需要头文件，只需要对应的动态库即可。由于 LabVIEW2010 64-bit 无法调用 32-bit 的动态库，这是 64 位 Windows 系统的限制，它不支持混合的 64 位/ 32 位进程。如果您的 LabVIEW 版本位数是 64-bit，请您联系我司技术支持，以获取 64-bit 的动态库。

注意

1. 本示例不做异常处理，如未发现光谱仪设备，或者获取光谱仪数据失败等。示例默认光谱仪已经和电脑正确连接，开启设备成功，获取光谱数据成功。
2. 参数若未截图说明，则为默认配置。
3. 如果您在 labview 中使用复杂的接口参数感觉困难，您可以联系我司，获取简单接口的通用版动态库，您可以在动态库版本信息中查看此动态库是否是通用版本。
4. 接口说明详见附录 B:通用版接口说明

操作日志：

如果您想获取操作日志，步骤如下

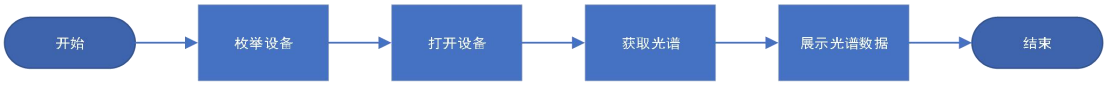
1. 开启日志功能，setLogSwitch（1）；
2. 如果您的操作系统为 32 位，日志完整路径为：C:\Windows\system\2020-11-11-10.log
3. 如果您的操作系统为 64 位，日志完整路径为：C:\Windows\system\2020-11-11-10.log
4. 管理员身份运行 labview 应用程序
5. 日志截图如下图

```
--------------------------------------------------------------------------------
[DEBUG] <2020-09-17 21:40:05.> [enumerateDevices:65]
device num:1
--------------------------------------------------------------------------------
[DEBUG] <2020-09-17 21:40:05.> [openSpectrometer:81]
Open NIR device success,index:0
--------------------------------------------------------------------------------
[DEBUG] <2020-09-17 21:40:05.> [getFormattedSpectrumLength:447]
spectrum pixel num:228
--------------------------------------------------------------------------------
[INFO] <2020-09-17 21:40:07.> [getFormattedSpectrum:644]
Scan success
--------------------------------------------------------------------------------
[ERROR] <2020-09-17 21:40:20.> [enumerateDevices:60]
Not found NIR device,please check connect
--------------------------------------------------------------------------------
[DEBUG] <2020-09-17 21:40:20.> [getFormattedSpectrumLength:447]
spectrum pixel num:228
--------------------------------------------------------------------------------
[ERROR] <2020-09-17 21:40:20.> [performScanReadData:584]
Reading device status for scan completion failed
--------------------------------------------------------------------------------
[ERROR] <2020-09-17 21:40:20.> [getFormattedSpectrum:628]
Scan Failed!
```

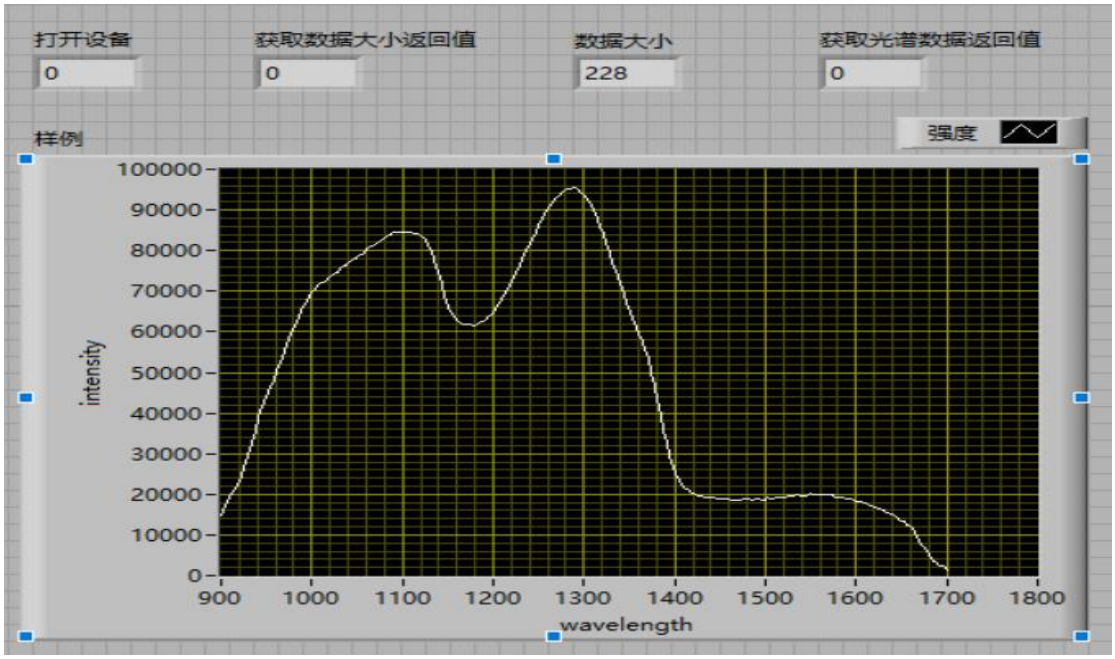
正常

异常

简要流程图

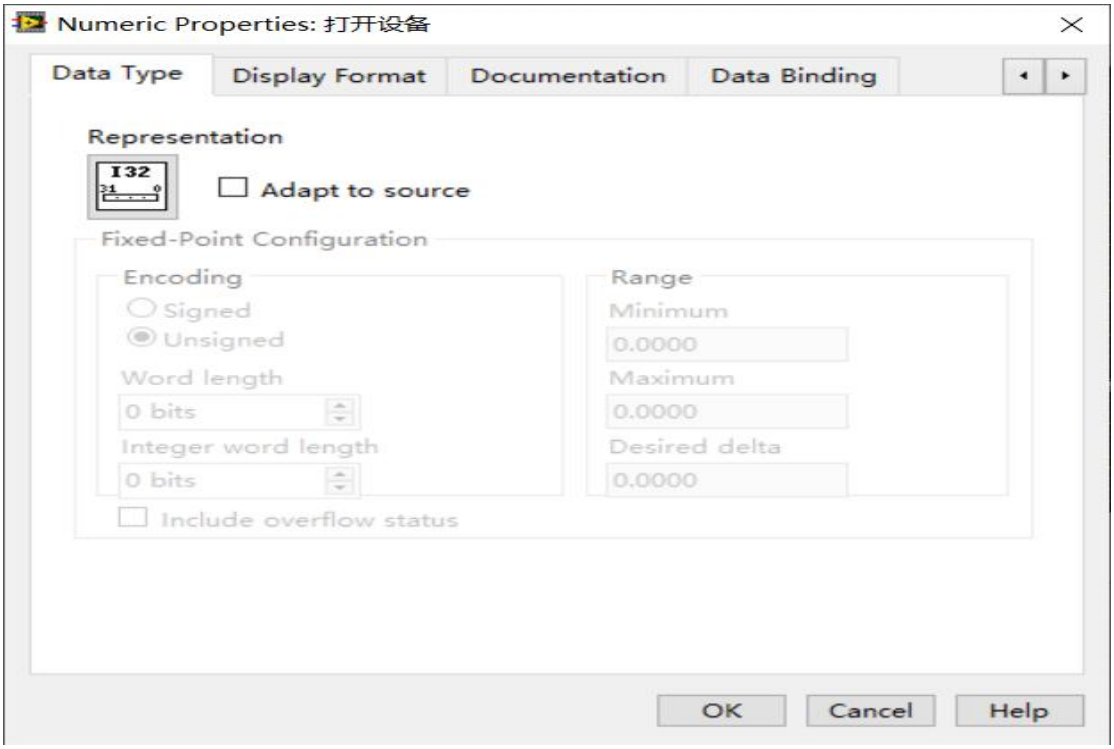


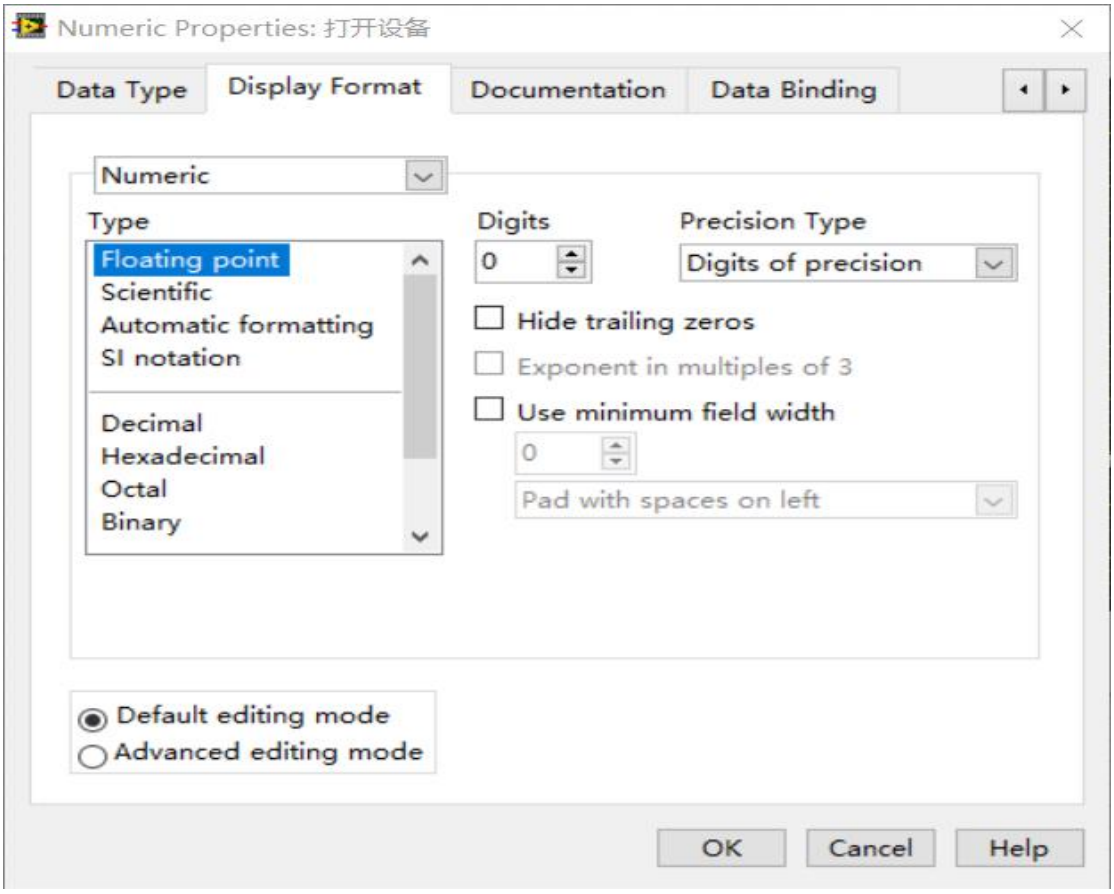
前面板界面



开启设备：光谱仪设备是否正确打开，值为 0：打开成功，否则打开失败。

配置如下：

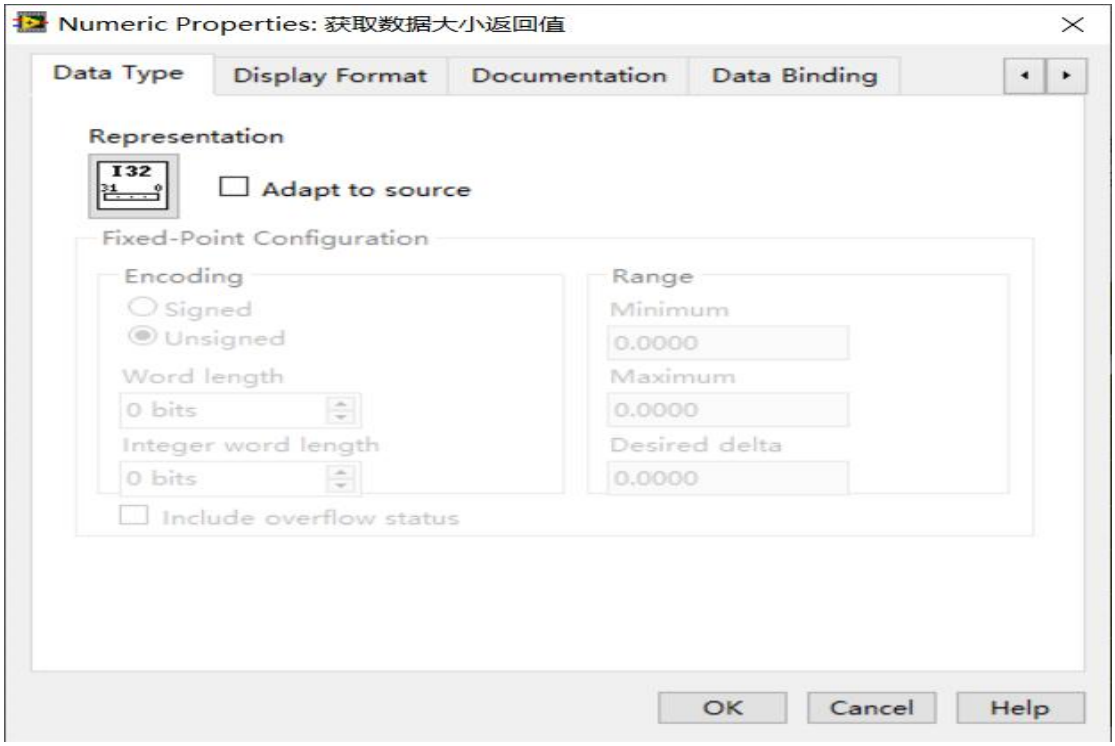


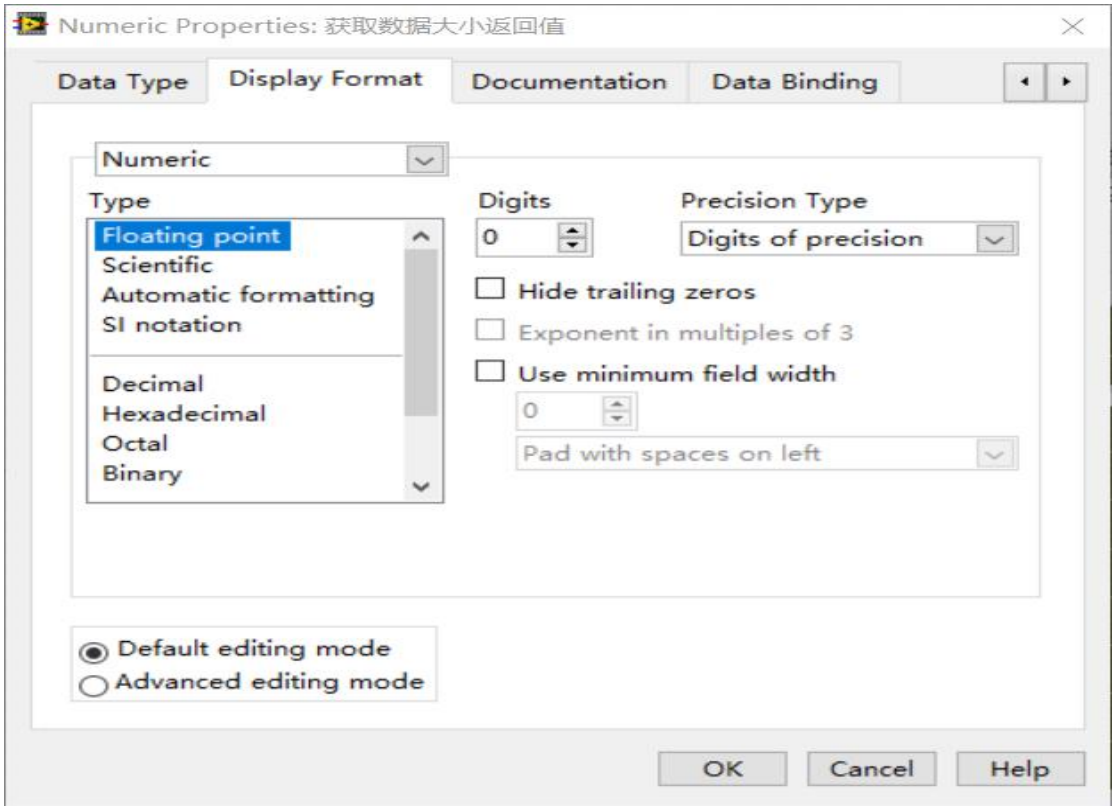


获取数据大小返回值：获取光谱仪前，需要获取光谱仪文件的大小，值为 0：获取成功，否

则获取失败。

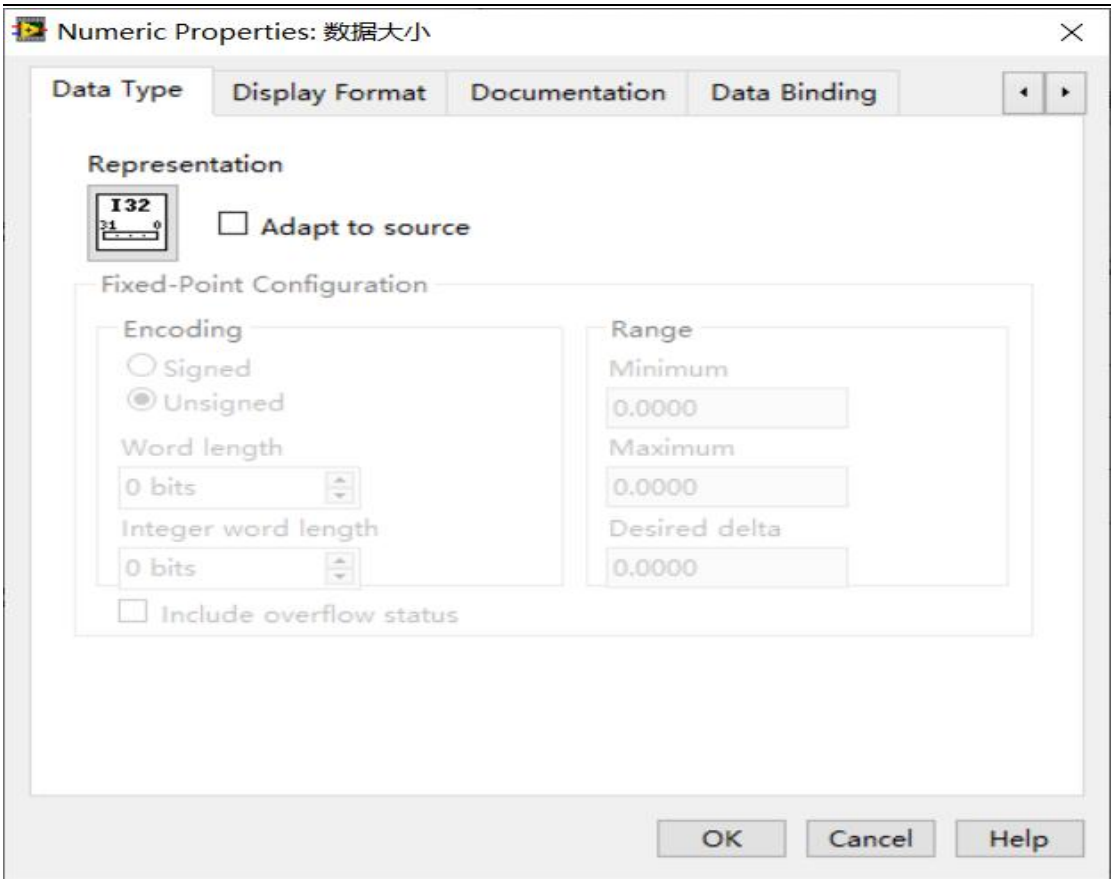
配置如下：

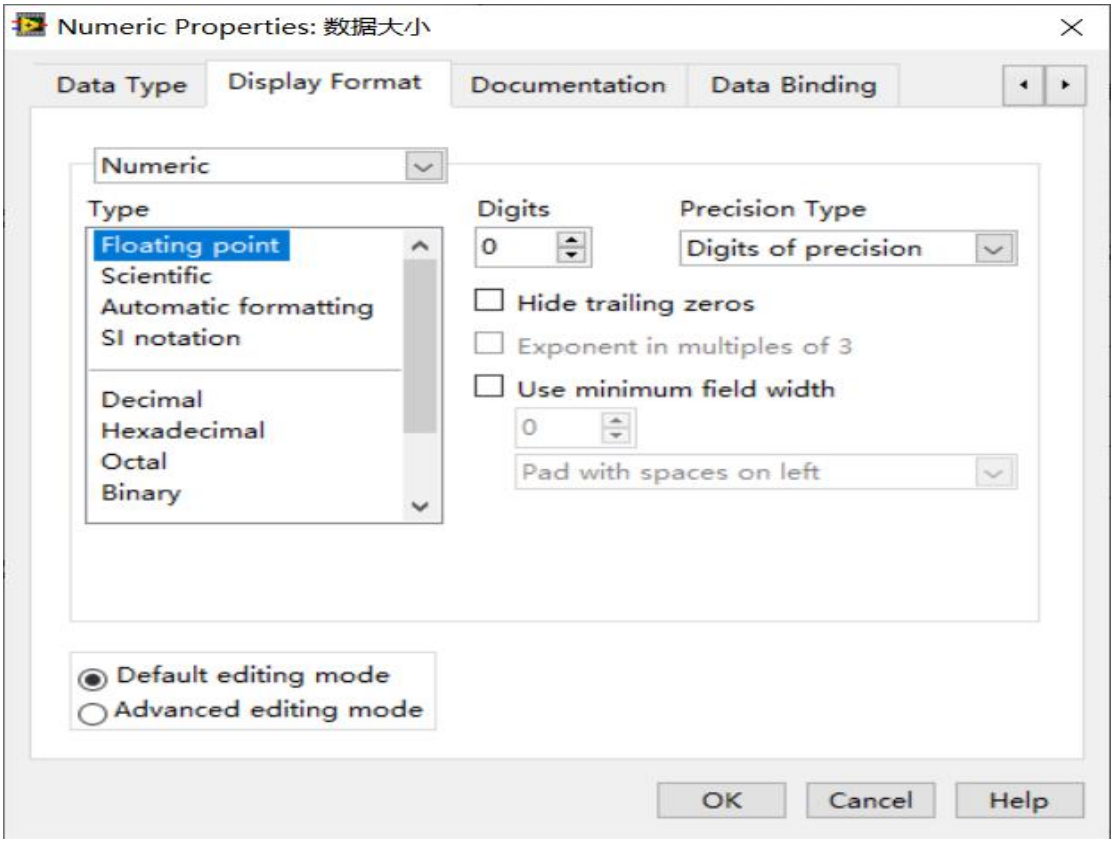




数据大小：光谱数据大小。

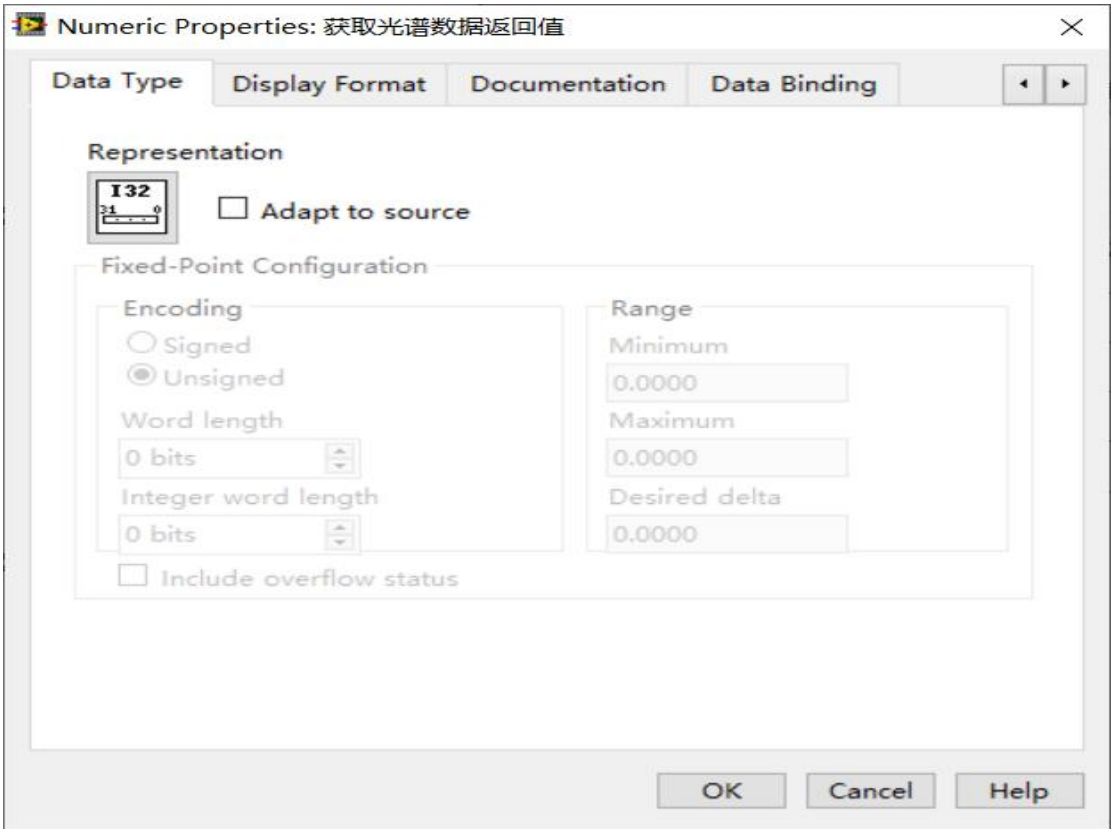
配置如下：

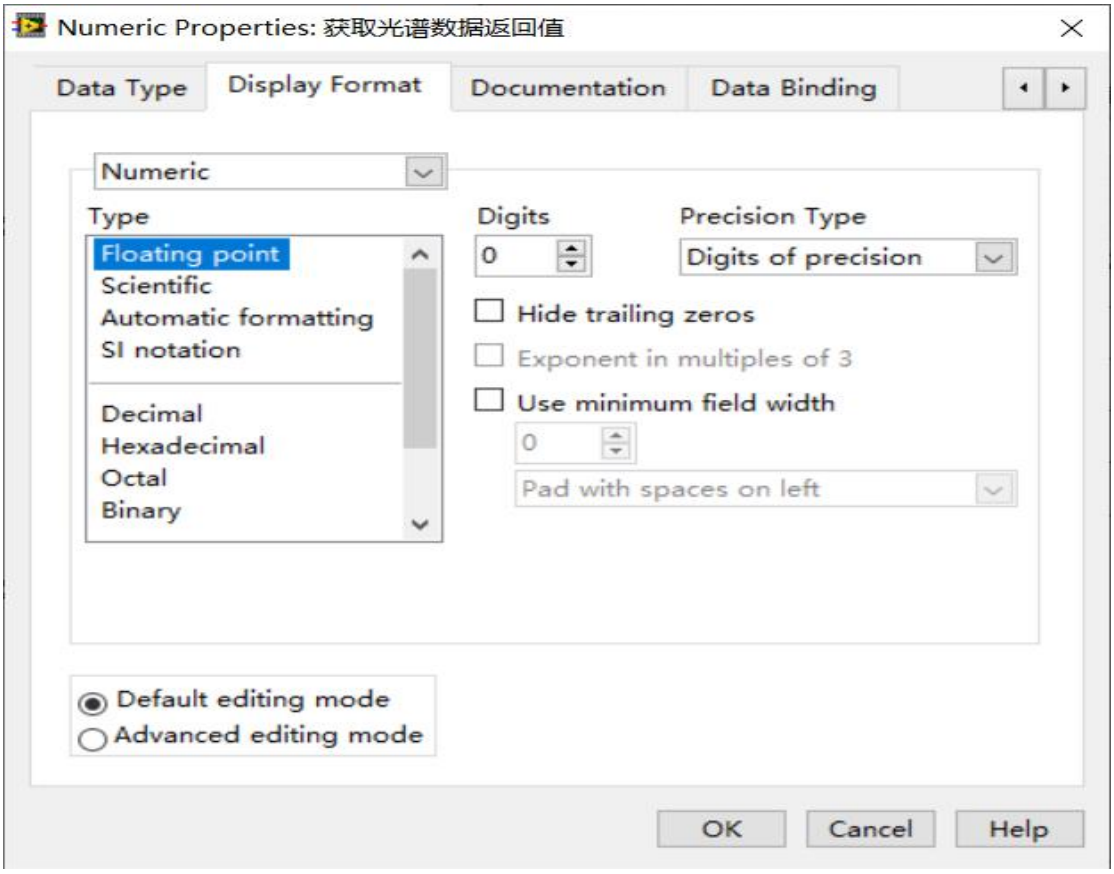




获取光谱数据返回值：获取光谱仪采集到的数据的返回值，值为 0，或者成功，否则获取失败。

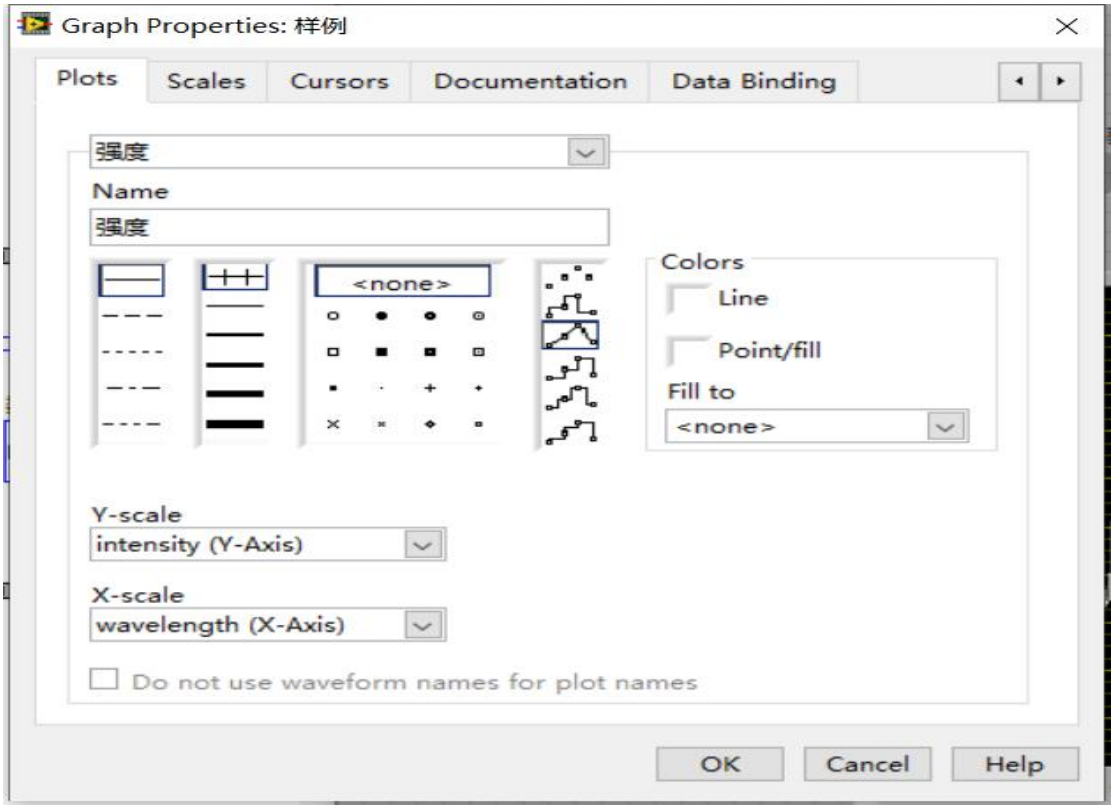
配置如下：

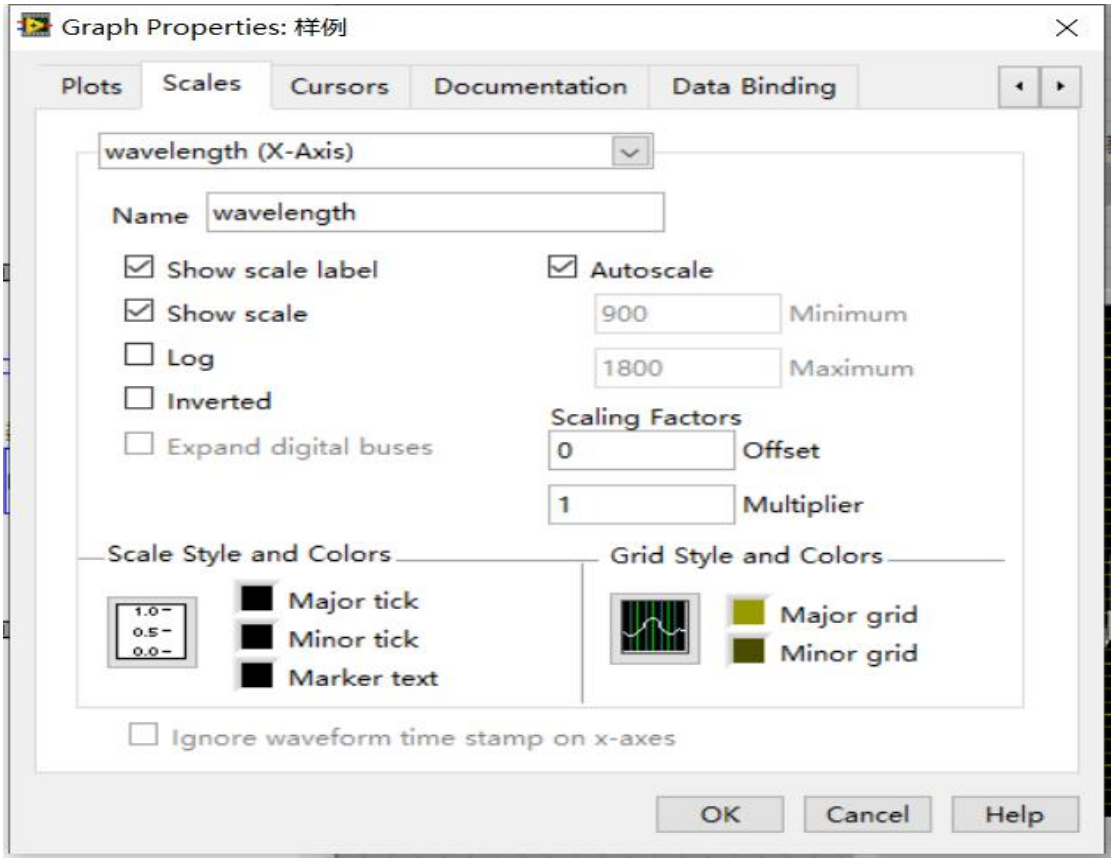


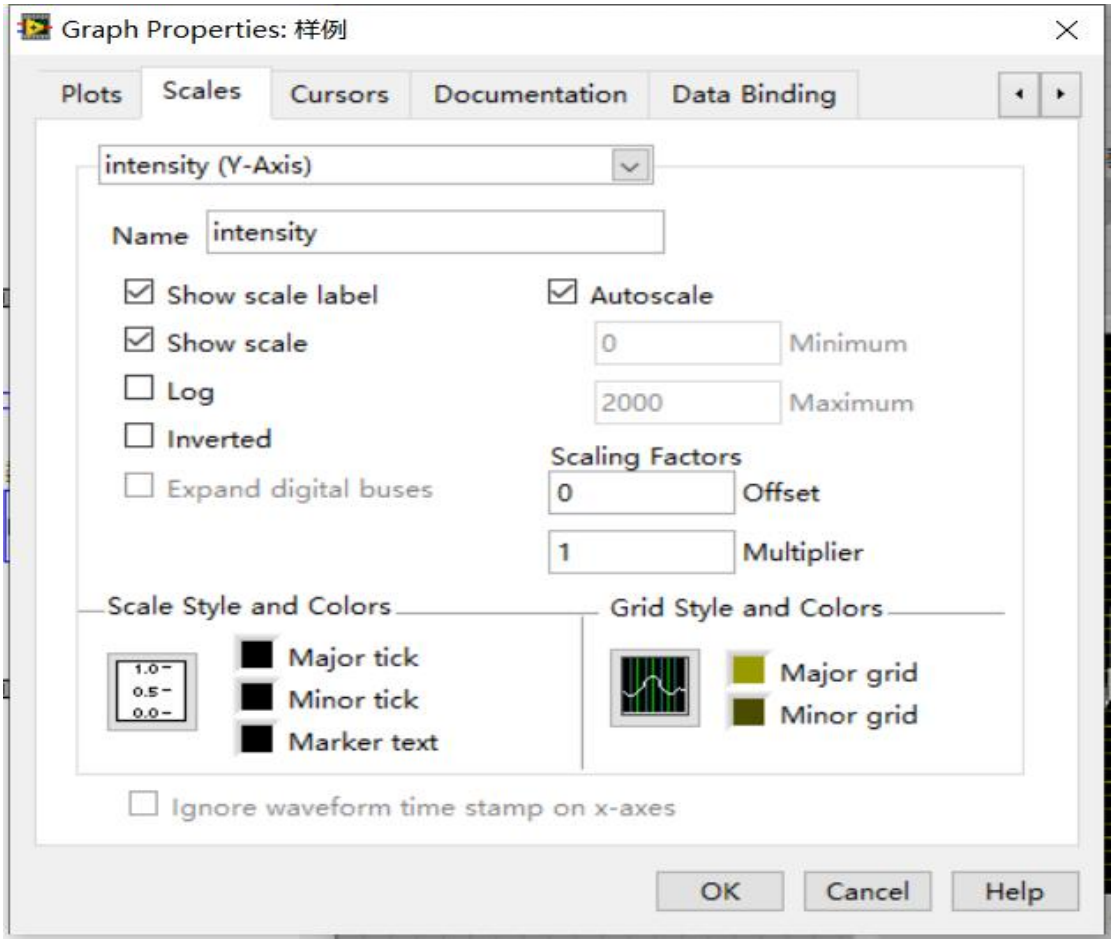


样例：展示获取到的光谱数据。

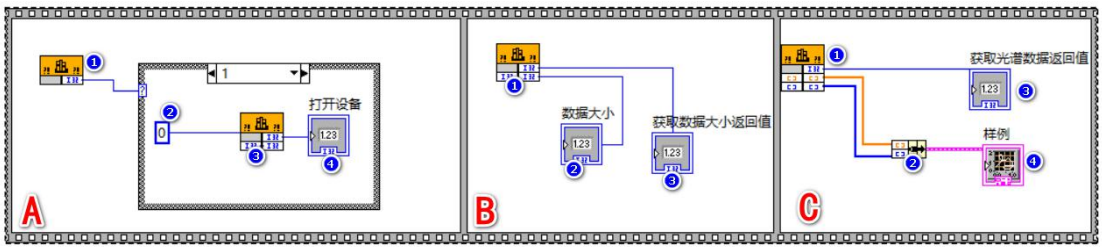
配置如下：


Graph Properties: 样例
Plots
Scales
Cursors
Documentation
Data Binding
强度
Name
强度
<none>
Colors
Line
Point/fill
Fill to
<none>
Y-scale
intensity (Y-Axis)
X-scale
wavelength (X-Axis)
Do not use waveform names for plot names
OK
Cancel
Help


Graph Properties: 样例
Plots
Scales
Cursors
Documentation
Data Binding
wavelength (X-Axis)
Name
wavelength
Show scale label
Show scale
Log
Inverted
Expand digital buses
Autoscale
900
Minimum
1800
Maximum
Scaling Factors
0
Offset
1
Multiplier
Scale Style and Colors
Major tick
Minor tick
Marker text
Grid Style and Colors
Major grid
Minor grid
Ignore waveform time stamp on x-axes
OK
Cancel
Help



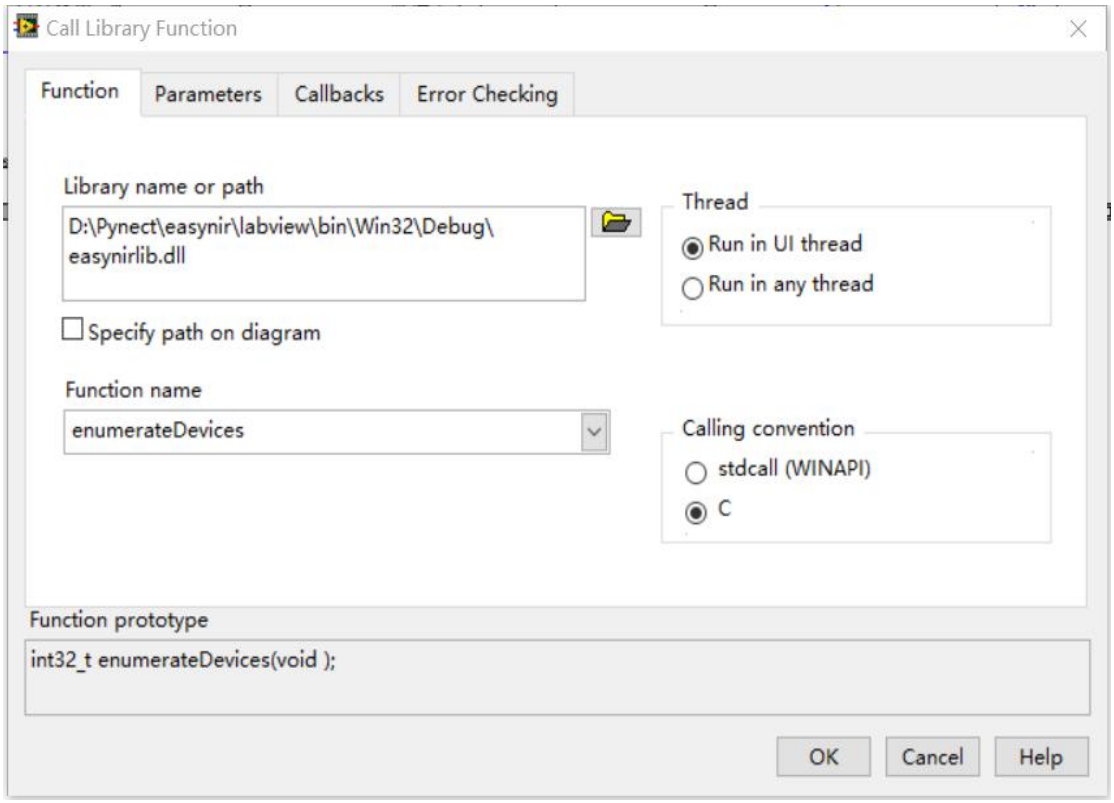
后面板界面

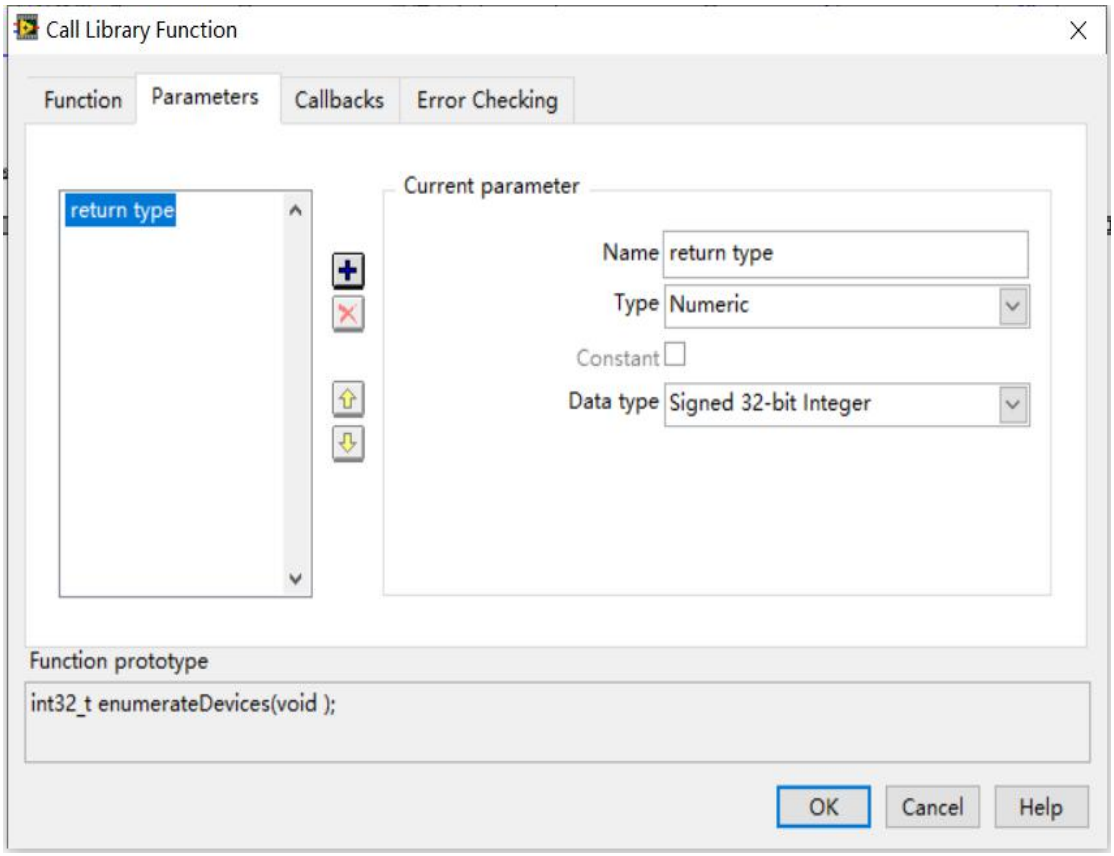


顺序执行 A-B-C

A-1（Connectivity-Libraries&Executables-Call Library Function Node）:枚举光谱仪

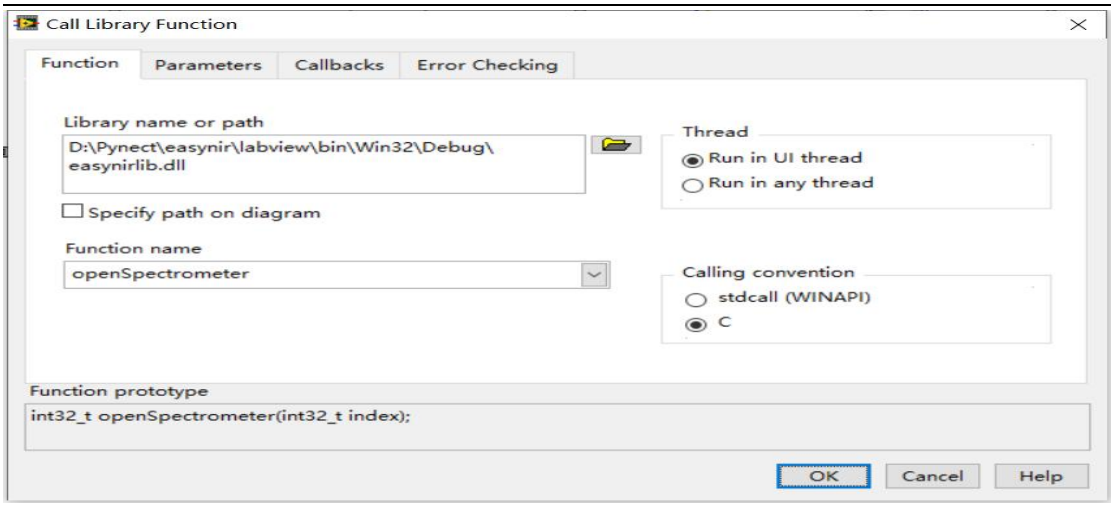
配置如下:

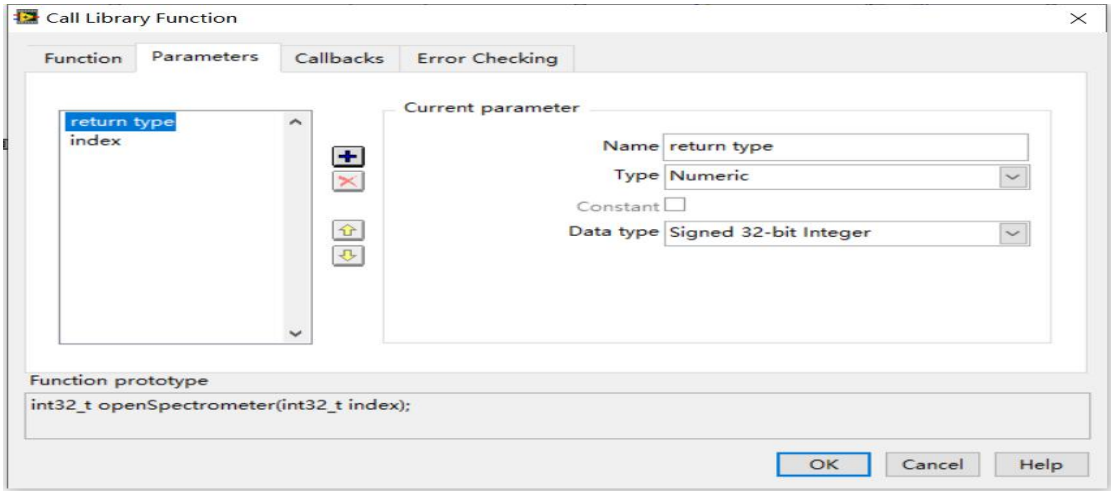


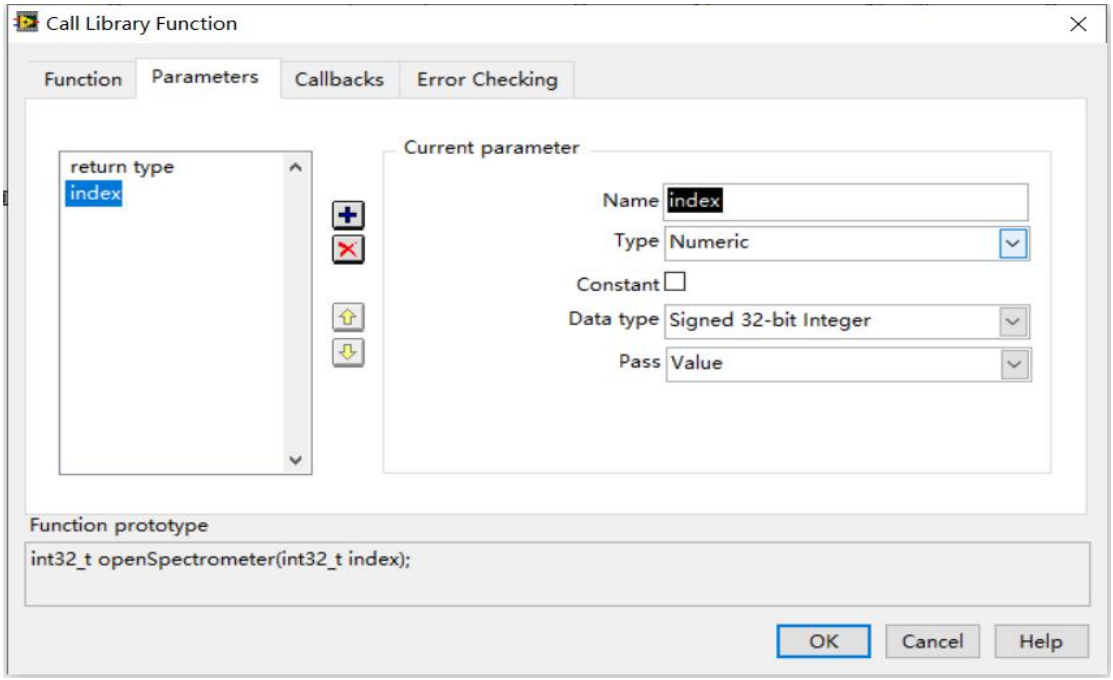


A-2:打开的光谱仪序号，由于只连接了一个光谱仪设备，所以序号设置为 0.

A-3（Connectivity-Libraries&Executables-Call Library Function Node）:打开光谱仪配置如下:

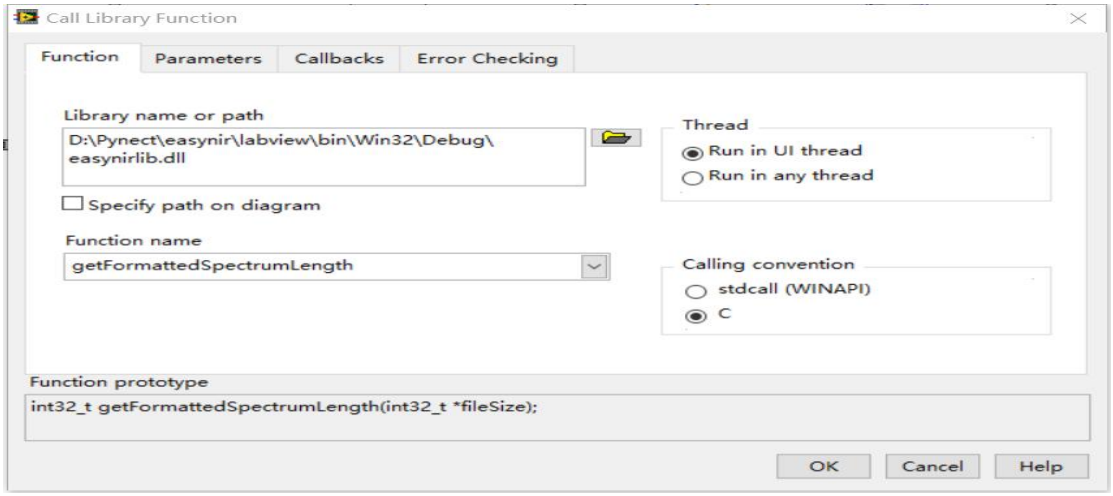


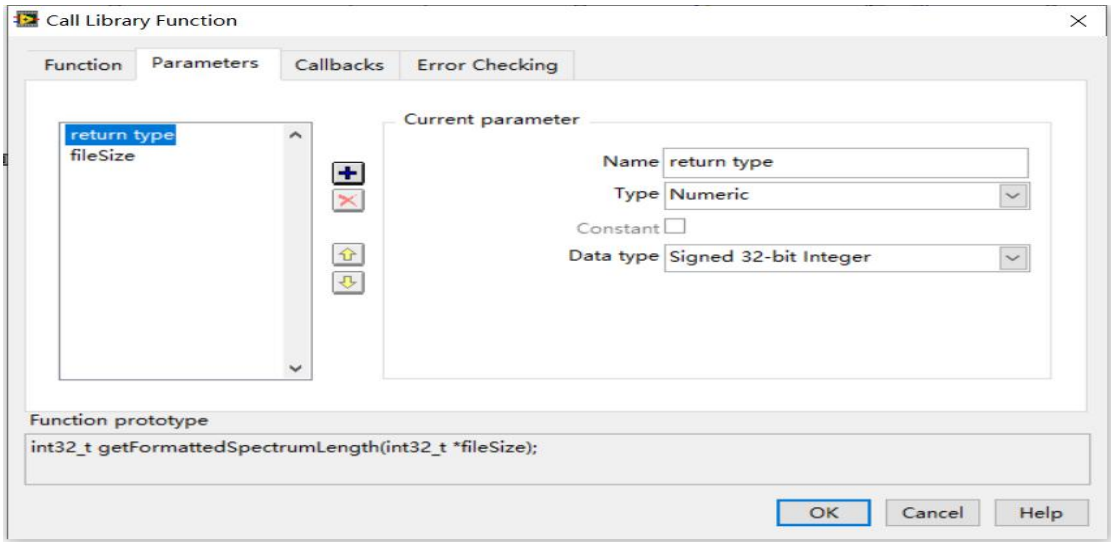


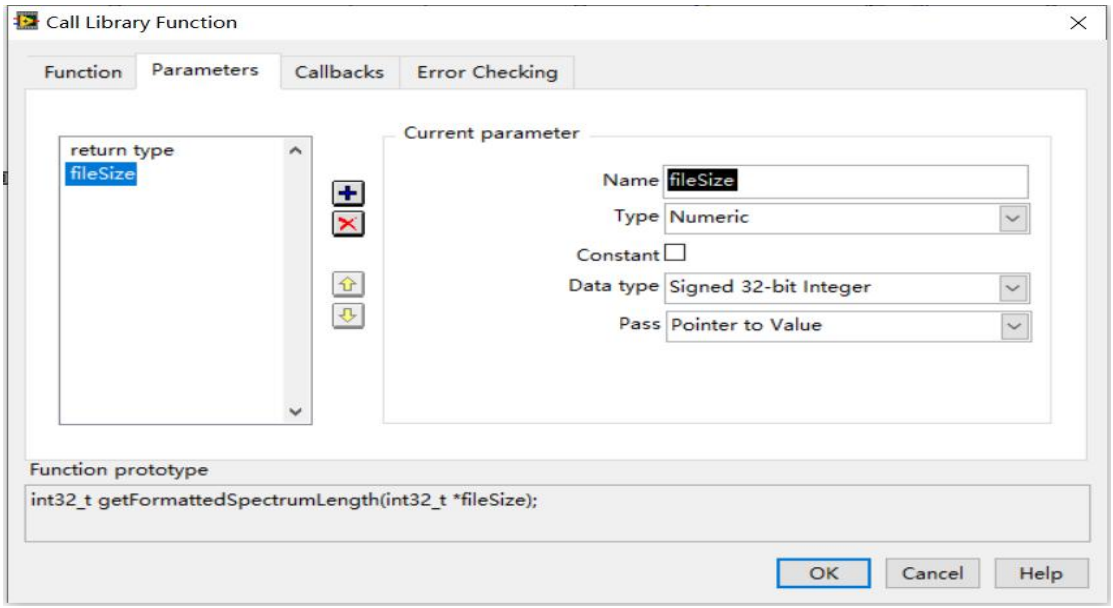


B-1（Connectivity-Libraries&Executables-Call Library Function Node）:光谱仪获取光谱仪数据长度。

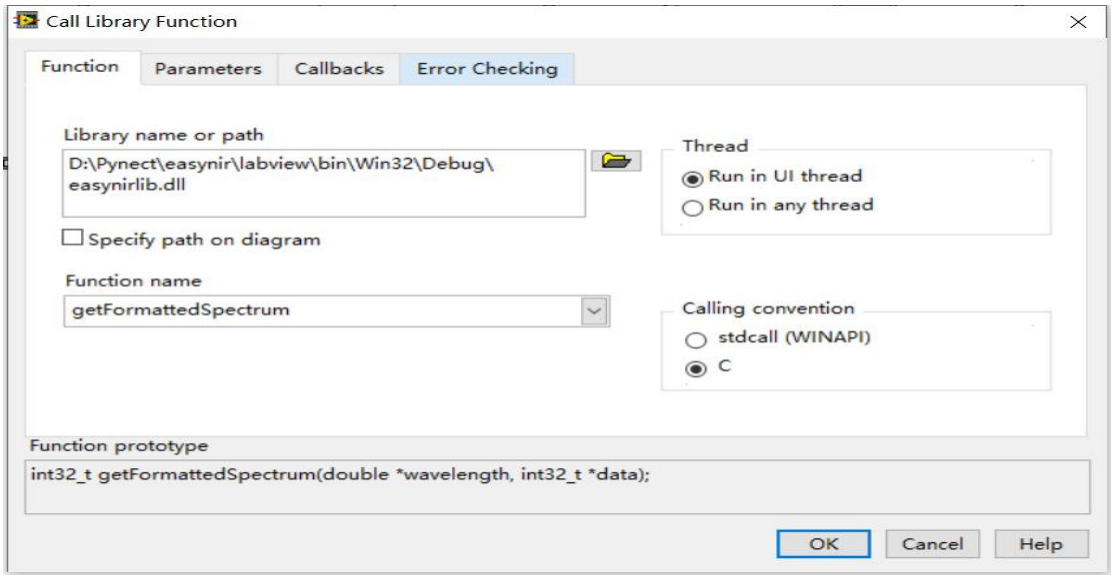
配置如下:

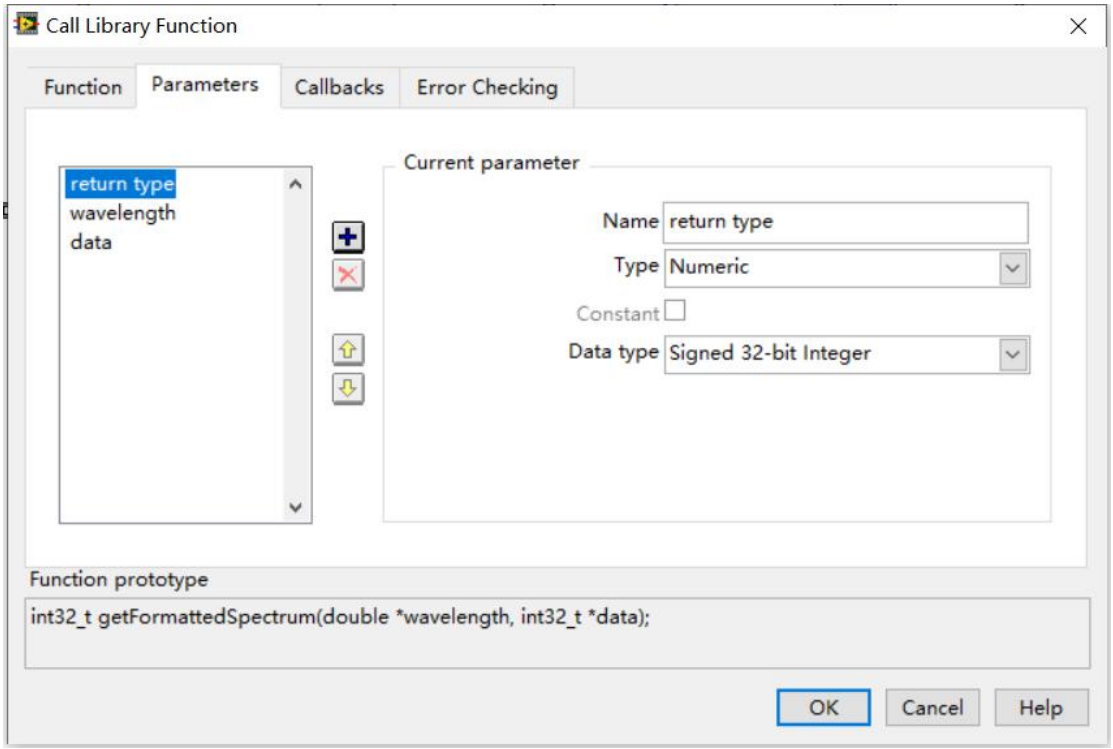


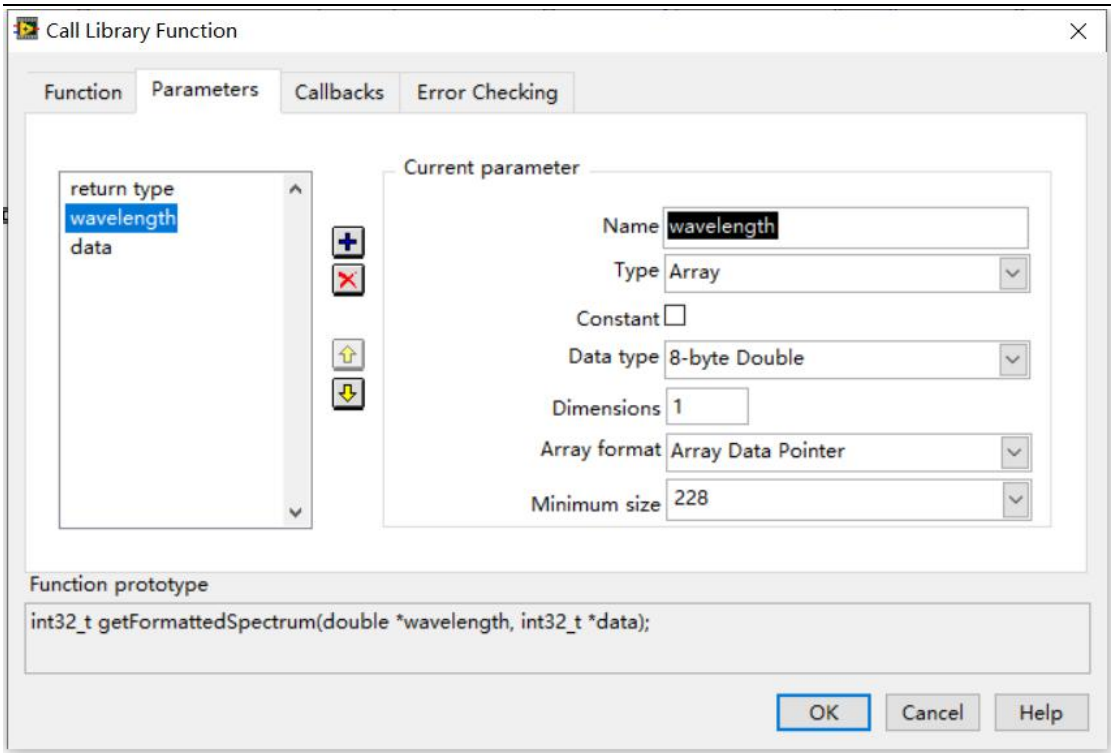


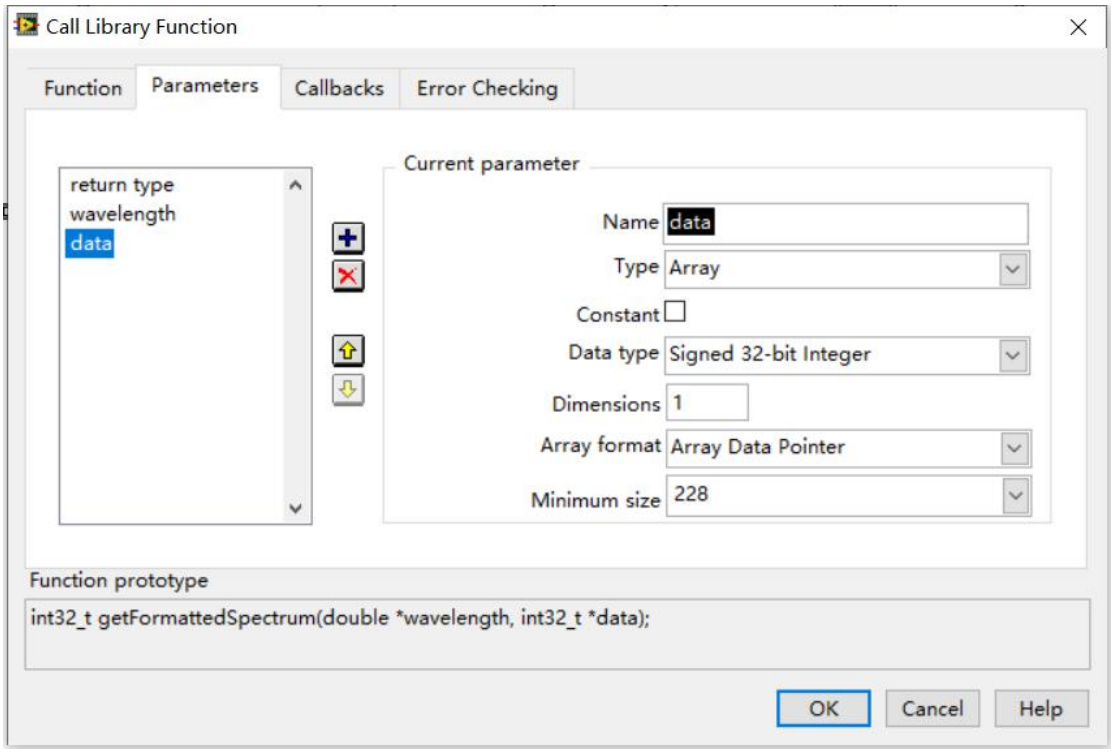


C-1（Connectivity-Libraries&Executables-Call Library Function Node）:获取光谱仪数据。配置如下:


Call Library Function
Function
Parameters
Callbacks
Error Checking
Library name or path
D:\Pynect\easynir\labview\bin\Win32\Debug\
easynirlib.dll
Specify path on diagram
Function name
getFormattedSpectrum
Thread
Run in UI thread
Run in any thread
Calling convention
stdcall (WINAPI)
C
Function prototype
int32_t getFormattedSpectrum(double *wavelength, int32_t *data);
OK
Cancel
Help


Call Library Function
Function
Parameters
Callbacks
Error Checking
return type
wavelength
data
Current parameter
Name return type
Type Numeric
Constant
Data type Signed 32-bit Integer
Function prototype
int32_t getFormattedSpectrum(double *wavelength, int32_t *data);
OK
Cancel
Help





C-2（Programming- Cluster, Class, & Variant - Bundle）:组装集群。

第七章
接口封装

概述

接口 API 是应用程序用于控制光谱仪并从中获取数据的对象和方法的集合。本节将详细介绍其中一些函数。

enumerateDevices()

此方法返回获取到的光谱仪设备的数量，如果未发现光谱仪设备，将返回 false。这是所有方法中最重要的方法，是操作光谱仪设备的起点。

参数

bool enumerateDevices(num)

int& num 枚举到的光谱仪数量。

returns true:枚举成功, false:枚举失败。

openSpectrometer()

打开指定序号的光谱仪设备。

参数

bool openSpectrometer (index)

const int index 指定要访问的光谱仪序号（0-N, N 为 enumerateDevices 枚举到的设备数量-1）。

returns true:打开成功, false:打开失败。

closeSpectrometer()

关闭光谱仪设备。

参数

void closeSpectrometer()

getUsbConnectStatus()

获取设备 USB 连接状态。

参数

bool getUsbConnectStatus(status)

bool * status USB 连接状态。

returns true:获取成功, false:获取失败。

getScanSection()

获取设备扫描配置参数。

参数

bool getScanSection(sectionMap)

std::multimap<std::string, ScanSection>& sectionMap 扫描参数。

returns true:获取成功, false:获取失败。

getScanType()

获取设备扫描参数-扫描类型。

参数

bool getScanType(scanType)

std::map<int, std::string>& scanType 扫描类型。

returns true:获取成功, false:获取失败。

getWidthPixels()

获取像素宽度列表。

参数

bool getWidthPixels(list)

std::vector<double>& list 素宽度列表。

returns true:获取成功, false:获取失败。

getExposureTimes()

获取曝光时间列表。

参数

bool getExposureTimes(list)

std::vector<double>& list 曝光时间列表(毫秒)。

returns true:获取成功, false:获取失败。

getPatternsNumRange()

获取模式范围。

参数

bool getPatternsNumRange(start, end)

int& start 开始。

int& end 结束。

returns true:获取成功, false:获取失败。

getRemainPatternsNum()

获取剩余模式数。

参数

bool getRemainPatternsNum(cfgName,section,num)

std::string & cfgName 配置文件名称。

unsigned char section 当前扫描截面序号。

unsigned short& num 剩余模式数(去除当前界面)。
returns true:获取成功, false:获取失败。

getActiveScanIndex()

获取活动的参数配置序号。
参数
bool getActiveScanIndex(index)
int& index 活动的配置序号。
returns true:获取成功, false:获取失败。

setActiveScanIndex()

设置活动的参数配置序号。
参数
bool setActiveScanIndex (index)
char& index 设置活动的参数配置序号。
returns true:设置成功, false:设置失败。

getEstimatedScanTime()

获取扫描的预期时间（毫秒）。
参数
int getEstimatedScanTime()
returns 预期时间（毫秒）。

setTargetLists()

设置目标板配置参数。
参数
int setTargetLists(name, cfgSections, averNum)
const std::string& name 扫描配置参数名称。
std::vector<ScanSection> cfgSections 扫描参数。
unsigned short averNum 扫描次数。
returns true:设置成功, false:设置失败。

removeTargetCfg()

移除扫描配置。
参数
bool removeTargetCfg(name)
const std::string& name 扫描配置名称。
returns true:移除成功, false:移除失败。

getVersions()

获取版本号。

参数

bool getVersions(pTivaSwVersion,pDlpcswVersion,pDlpcFlashBuildVersion,pSpecLibVer,pCalDataVer,pRefCalDataVer,pCfgDataVer)

unsigned int *pTivaSwVersion Tiva 版本。

unsigned int *pDlpcswVersion DLP 版本。

unsigned int *pDlpcFlashBuildVersion flash 版本。

unsigned int *pSpecLibVer 光谱仪库版本。

unsigned int *pCalDataVer 校准参数版本。

unsigned int *pRefCalDataVer 参考参数版本。

unsigned int *pCfgDataVer 配置参数版本。

returns true:获取正确 false:获取失败。

setLampStatus()

设置光源开关状态。

参数

bool setLampStatus(status)

bool status true:开 false:关。

returns true:设置成功, false:设置失败。

setPGAGain()

设置增益系数。

参数

bool setPGAGain(isFixed, gainVal)

bool isFixed true:固定 false:不固定。

unsigned char gainVal 1,2,4,8,16,32 或 64。

returns true:设置成功, false:设置失败。

getMaxPatterns()

获取最大模式数。

参数

bool getMaxPatterns(cfg_name,start_nm,end_nm,width_index,scan_type,scan_num)

std::string cfg_name 配置名称。

int start_nm 开始波长。

int end_nm 结束波长。

unsigned char width_index 像素宽度。

int scan_type 扫描类型。

unsigned char scan_num 扫描次数。

returns 最大允许模式数。

getFormattedSpectrumLength()

获取光谱数据长度。

参数

bool getFormattedSpectrumLength (fileSize)

int& fileSize 光谱数据长度。

returns true:获取正确 false:获取失败。

setSpectrumAverageTime()

设置光谱仪采集的平均次数。

参数

bool setSpectrumAverageTime(time)

unsigned short time 平均次数。

returns true:设置成功, false:设置失败。

getFormattedSpectrum()

获取光谱数据。

参数

bool getFormattedSpectrum (pWavelength, pData)

double* pWavelength 波长数据。

double* pData 波长数据。

returns true:获取正确 false:获取失败。

getLastException()

获取最后一次的错误信息。

参数

bool getLastException(pWavelength, pData)

returns 错误信息[最大长度为 256 字节]。

setLogSwitch()

日志功能。

参数

void setLogSwitch(onOff)

bool onOff true:开启日志 false:关闭日志。

附录 A
常见问题

查找不到光谱仪设备？

确保 USB 正确连接到电脑，且设备开关已开启。

设备在采集的过程中掉线？

设备供电不足，建议您直接连接至电脑 USB 端口，或者通过带有供电的 hub 连接。

如果更换光谱仪型号，是否需要更改我的程序?

不需要。只要您使用动态库中的 API，您的软件就不需要在更换不同类型的近红外光谱仪时进行修改。但是，请记住，某些特性(例如，“开关光源”)只能在特定定的光谱仪上使用。
参考可选特性列表或者咨询技术支持以得到确定的答案。

32 位 EasyNIRLib 应用程序能在 64 位 Windows 7 上运行吗?

可以的。32 位的动态库可以运行在 64 位的机器上，但是 64 位的机器无法在 32 位上运行，这和机器的寻找方式有关。

程序编译正常，运行崩溃？

可能原因在动态库的版本，如果您在 Debug 模式下，需要使用 Debug 版本的动态库，在 Release 模式下，需要使用 Release 版本的动态库。

附录 B
通用版接口说明

概述

如果您在如 labview 或 C#中操作复杂的接口感到困难，您可以使用简单接口的通用版本的开发库接口。此动态库将通用版本动态库中的复杂接口更改为简单接口，但这无疑增加了操作步骤，如您想获取扫描参数类型，在普通版本中，接口为 bool getScanType(std::map<int, std::string>& scanType)但在通用版动态库中，此操作被拆分成两个步骤，分别为获取扫描参数类型的个数，以及分别获取每个扫描参数的名称，int getScanTypesNumber()和 char* getScanTypesName(int index)。所有复杂的接口均按照此方式进行拆分。

enumerateDevices()

枚举连接到系统的所有 dlp 设备。

参数

int enumerateDevices(int* num)

int* num 设备数量。

returns >= 0:打开成功, < 0:打开失败。

openSpectrometer()

关闭连接到系统的设备。

参数

void closeSpectrometer()

getScanSectionNumber()

获取设备扫描配置参数个数。

参数

int getScanSectionNumber()

int* num 设备数量。

returns 扫描配置参数个数。

getScanSectionName()

获取设备扫描配置名称。

参数

int getScanSectionName(int index, char* secName, unsigned int size)

int index 设备扫描配置序号(参考 getScanSectionNumber() index 必须小于 getScanSectionNumber()返回值)。

char* secName 设备扫描配置名称。

unsigned int size 设备扫描配置名称内存大小。

returns 0:获取成功, <0:获取失败。

getScanSectionNameNumber()

获取某一扫描配置含有的子扫描配置个数。

参数

int getScanSectionNameNumber(char* name)

char* name 扫描配置名称。

returns 子扫描配置个数。

getScanSectionNameCfg()

获取子扫描配置。

参数

int getScanSectionNameCfg(char* name, int index, char* scanType, unsigned int scanTypeLen, double* widthPixel,unsigned short* wavelengthStart, unsigned short* wavelengthEnd,unsigned short* patternsNum, double* exposureTime)

char* name 子扫描配置名称。

int index 子扫描配置里的序号。

char* scanType 扫描类型。

unsigned int scanTypeLen 扫描类型内存长度。

double* widthPixel 像素宽度。

unsigned short* wavelengthStart 开始扫描的最小波长，单位为纳米。

unsigned short* wavelengthEnd 结束扫描的最小波长，单位为纳米。

unsigned short* patternsNum 频谱中所期望的点数。

double* exposureTime 曝光时间。

returns 0:获取成功, <0:获取失败。

getScanTypesNumber()

获取设备扫描参数类型数量。

参数

int getScanTypesNumber()

returns 类型数量。

getScanTypesName()

获取设备扫描参数类型名称。

参数

char* getScanTypesName(int index)

int index 参数类型序号(参考 getScanTypesNumber() index 必须小于 getScanTypesNumber() 返回值)。

returns 扫描参数类型名称。

getWidthPixelsNumber()

获取像素宽度类型个数。

参数

int getWidthPixelsNumber()

returns 像素宽度类型个数。

getWidthPixel()

获取像素宽度类型。

参数

double getWidthPixel(int index)

int index 素宽类型序号(参考 getWidthPixelsNumber() index 必须小于 getWidthPixelsNumber()返回值)。

returns 素宽宽度。

getExposureTimesNumber()

获取曝光时间类型个数。

参数

int getExposureTimesNumber()

returns 曝光时间类型个数。

getExposureTime()

获取曝光时间类型。

参数

double getExposureTime(int index)

int index 素宽类型序号(参考 getExposureTimesNumber() index 必须小于 getExposureTimesNumber()返回值)。

returns 曝光时间类型个数。

getRemainPatternNum()

获取剩余模式数。

参数

int getRemainPatternNum(char* cfgName, unsigned char section, unsigned short* num)

char* cfgName 配置文件名称。

unsigned char section 当前扫描截面序号。

unsigned short* num 剩余模式数(去除当前界面)。

returns 0:获取成功, <0:获取失败。

getPatternsRange()

获取模式范围。

参数

int getPatternsRange(int* start, int* end)

int* start 开始。

int* end 结束。

returns 0:获取成功, <0:获取失败。

getActiveScanIndex()

获取活动的参数配置序号。

参数

int getActiveScanIndex(int* index)

int* index 活动的配置序号。

returns 0:获取成功, <0:获取失败。

setActiveScanIndex()

设置活动的参数配置序号。

参数

int setActiveScanIndex(char index)

int index 配置序号。

returns 0:获取成功, <0:获取失败。

getEstimatedScanTime()

获取扫描的预期时间（毫秒）。

参数

int getEstimatedScanTime()

returns 预期时间（毫秒）。

setTgtCfg()

设置目标板配置参数(**需 setTgtCfgs 才能生效**)。

参数

int setTgtCfg(const char* name, int section, const char* scanType, unsigned short wavelengthStart,unsigned short wavelengthEnd, double widthPixel, unsigned short patternsNum,double exposureTime)

const char* name 配置参数名称。

int section 参数所属扫描截面。

const char* scanType 扫描类型。

unsigned short wavelengthStart 起始波长。

unsigned short wavelengthEnd 结束波长。

double widthPixel 像素宽度。

unsigned short patternsNum 模式数。

double exposureTime 曝光时间。

returns 0:获取成功, <0:获取失败。

setTgtCfgs()

设置目标板配置参数。

参数

int setTgtCfgs(const char* cfgName, unsigned short averNum)

const char* cfgName 配置参数名称。

unsigned short averNum 扫描次数。

returns 0:获取成功, <0:获取失败。

removeTgtCfg()

移除扫描配置。

参数

int removeTgtCfg(const char* name)

const char* name 扫描配置名称。

returns 0:移除成功, <0:移除失败。

getVersions()

获取版本号。

参数

int getVersions(unsigned int *pTivaSwVersion, unsigned int *pDlpcswVersion, unsigned int *pDlpcFlashBuildVersion, unsigned int *pSpecLibVer, unsigned int *pCalDataVer, unsigned int *pRefCalDataVer, unsigned int *pCfgDataVer)

unsigned int *pTivaSwVersion Tiva 版本。

unsigned int *pDlpcswVersion DLP 版本。

unsigned int *pDlpcFlashBuildVersion flash 版本。

unsigned int *pSpecLibVer 光谱仪库版本。

unsigned int *pCalDataVer 校准参数版本。

unsigned int *pRefCalDataVer 参考参数版本。

unsigned int *pCfgDataVer 配置参数版本。

returns 0:获取正确 <0:获取失败。

setLampStatus()

设置光源开关状态。

参数

int setLampStatus(int status)

int status 1:开 0:关。

returns 0:设置成功, <0:设置失败。

setPGAGain()

设置增益系数。

参数

int setPGAGain(int isFixed, unsigned char gainVal)

int isFixed 1:固定 0:不固定。

unsigned char gainVal 1,2,4,8,16,32 或 64。

returns 0:设置成功, <0:设置失败。

getMaxPattern()

获取最大模式数。

参数

int getMaxPattern(char* cfg_name, int start_nm, int end_nm, unsigned char width_index, int scan_type, unsigned char scan_num)

char* cfg_name 配置名称。

int start_nm 开始波长。

int end_nm 结束波长。

unsigned char width_index 像素宽度。

int scan_type 扫描类型。

unsigned char scan_num 扫描次数。

returns 最大允许模式数。

getFormattedSpectrumLengths()

获取光谱数据长度。

参数

int getFormattedSpectrumLengths(int* fileSize)

int* fileSize 数据长度。

returns >= 0:获取正确 < 0:获取失败。

setSpectrumAverageTime()

设置光谱仪采集的平均次数。

参数

int setSpectrumAverageTime(unsigned short time)

unsigned short time 平均次数。

returns 0:设置成功, <0:设置失败。

getFormattedSpectrum()

获取光谱数据。

参数

int getFormattedSpectrum(double *pWavelength, unsigned int *pData)

double *pWavelength 波长数据。

unsigned int *pData 光谱数据。

returns >= 0:获取正确 < 0:获取失败。

getLastException()

获取最后一次的错误信息。

参数

int getFormattedSpectrum(double *pWavelength, unsigned int *pData)

returns 错误信息[最大长度为 256 字节]。

setLogSwitch()

日志功能。

参数

void setLogSwitch(int onOff)

int onOff 1:开启日志 0:关闭日志。

样例

```
// 枚举设备数量
int devNum = 0;
if(enumerateDevices(&devNum) < 0)
{
	printf("Enumerate NIR device error, exit!!!\r\n");
	system("pause");
	return 0;
}
/
printf("NIR device number:%d\r\n", devNum);

// 打开设备
if (openSpectrometer(devNum - 1) < 0) {
	printf("Failed to open NIR device,error:%s\r\n", getLastException());
	system("pause");
	return 0;
}

printf("Open NIR device success\r\n");

const int scanSecNum = getScanSectionNumber();

for(int i = 0; i < scanSecNum; i++)
{
	char pSecName[128] = { 0 };
	if(getScanSectionName(i, pSecName,128) < 0)
```

```
    {
        printf("Get scan section ,error:%s\r\n", getLastException());
        continue;
    }

    const int secNameNum = getScanSectionNameNumber(pSecName);

    //printf("secName:%s Num:%d\n", pSecName, secNameNum);

    for(int j = 0; j < secNameNum; j++)
    {
        char pScanType[128] = { 0 };
        double widthPixel = 0.0;
        unsigned short wavelengthStart = 0;
        unsigned short wavelengthEnd = 0;
        unsigned short patternsNum = 0;
        double exposureTime = 0.0;


        if(getScanSectionNameCfg(pSecName,j,pScanType,128,&widthPixel,
&wavelengthStart, &wavelengthEnd, &patternsNum, &exposureTime) < 0)
        {
            printf("Get scan section ,error:%s\r\n", getLastException());
            continue;
        }

        printf("secName:%s scanType:%s,widthPixel:%0.3f, wavelengthStart:%d,
wavelengthEnd:%d, patternsNum:%d, exposureTime:%0.3f\n",pSecName, pScanType,
widthPixel, wavelengthStart, wavelengthEnd, patternsNum, exposureTime);
    }
    }

    //// 修改参数
    //if(setTgtCfg("Column 3", 0, "Column", 900, 1500, 8.20, 100, 5.08) < 0)
    //{
    //    printf("setTgtCfg error\r\n");
    //    system("pause");
    //    return 0;
    //}

    //if (setTgtCfg("Column 3", 1, "Column", 1500, 1700, 8.20, 100, 5.08) < 0)
    //{
    //    printf("setTgtCfg error\r\n");
    //    system("pause");
```

```
//    return 0;
//}


//if(setTgtCfgs("Column 3",6) < 0)
//{
//    printf("setTgtCfgs error\r\n");
//    system("pause");
//    return 0;
//}

//removeTgtCfg("Column 3");

int repeatTime = 1;

do {
    int fileSize = 0;
    getFormattedSpectrumLengths(&fileSize);

    printf("fileSize:%d\n", fileSize);

    double* pWavelengthData = (double*)malloc(fileSize * sizeof(double));
    unsigned int* pData = (unsigned int*)malloc(fileSize * sizeof(int));

    printf("Collect spectral data\r\n");
    if(getFormattedSpectrum(pWavelengthData, pData) != 0)
    {
        printf("Collect spectral data fail,exit!!! \r\n");
        system("pause");
        return 0;
    }

    for (int i = 0; i < fileSize; i++) {
        printf("RawData[%d]:%0.02f %d\n", i, pWavelengthData[i], pData[i]);
    }

} while (--repeatTime);

//获取光谱数据完成
printf("Collect spectral data success, press any key to exit!!!\r\n");
```