

无线通信实验在线开放课程

主讲人：吴光 博士

广东省教学质量工程建设项目





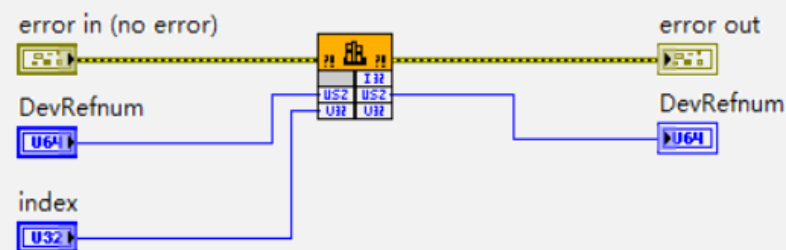
第五章

动态链接库封装和调用



目录

- 动态链接库介绍
- RTL-SDR接口函数封装
- 导入共享库向导
- 动态链接库编译
- LabVIEW调用动态链接库

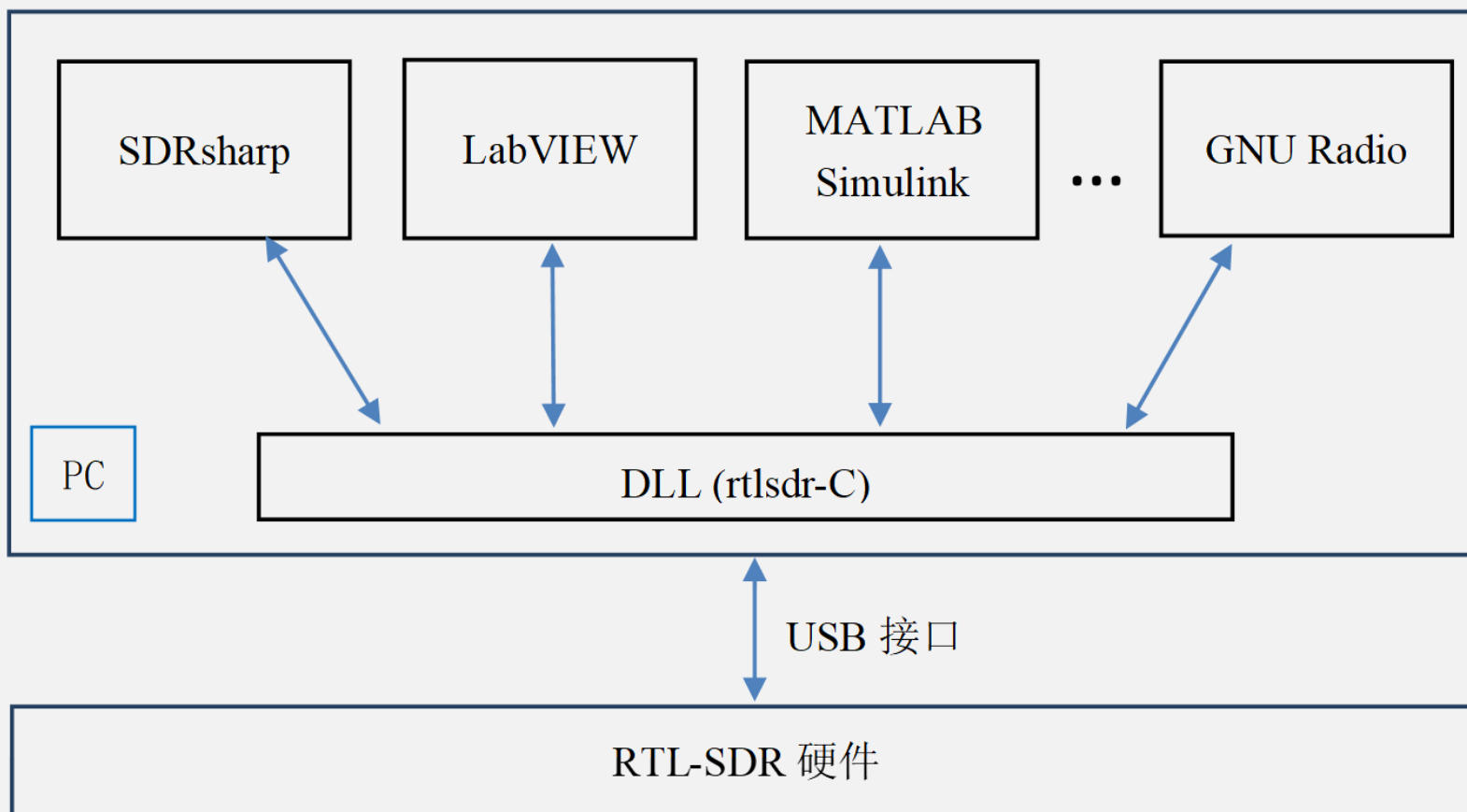




动态链接库介绍

- 动态链接库（DLL）是Windows操作系统实现共享函数库的一种方式。它可以包含许多函数，这些函数可以被其他程序调用，有助于共享函数、节省程序更新时间、提高计算机内存效率。
- 比如RTL-SDR的应用程序接口。将其转化为rtlsdr.dll就可以被上层应用程序直接调用，起到控制硬件的效果。







动态链接库介绍

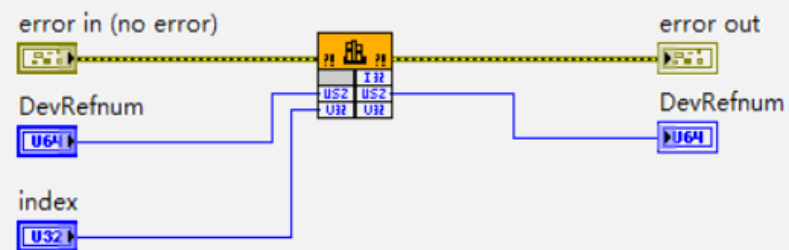
- 为了解动态链接库中可用的接口函数，[打开DLL的头文件rtl-sdr.h](#)，可以[查看接口函数](#)。比如和调谐相关的增益、带宽等接口函数以及[函数声明](#)的参数类型，例如：

```
1  RTLSDR_API enum rtl_sdr_tuner rtl_sdr_get_tuner_type(rtl_sdr_dev_t
    *dev);
2  RTLSDR_API int rtl_sdr_get_tuner_gains(rtl_sdr_dev_t *dev, int *gains);
3  RTLSDR_API int rtl_sdr_set_tuner_gain(rtl_sdr_dev_t *dev, int gain);
4  RTLSDR_API int rtl_sdr_set_tuner_bandwidth(rtl_sdr_dev_t *dev,
    uint32_t bw);
5  RTLSDR_API int rtl_sdr_get_tuner_gain(rtl_sdr_dev_t *dev);
```



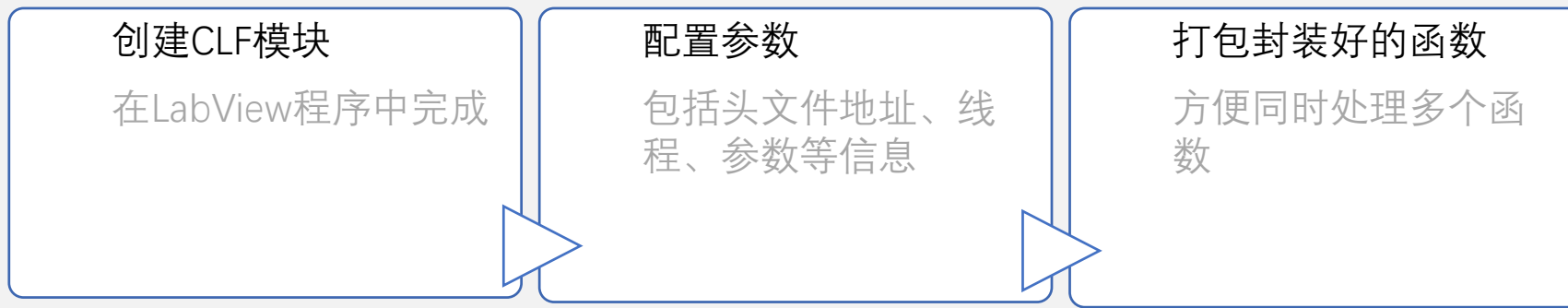
目录

- 动态链接库介绍
- RTL-SDR接口函数封装
- 导入共享库向导
- 动态链接库编译
- LabVIEW调用动态链接库





调用库函数封装流程图

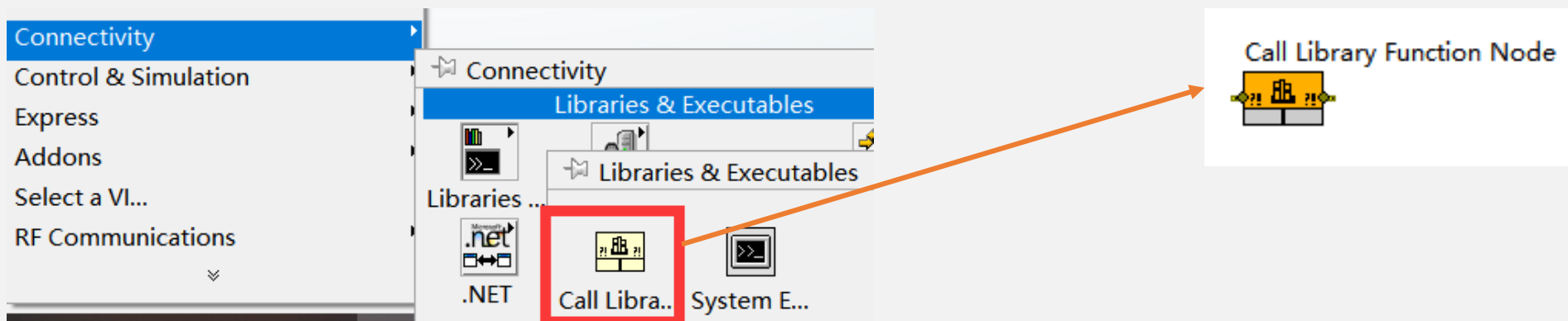




调用库函数封装接口函数

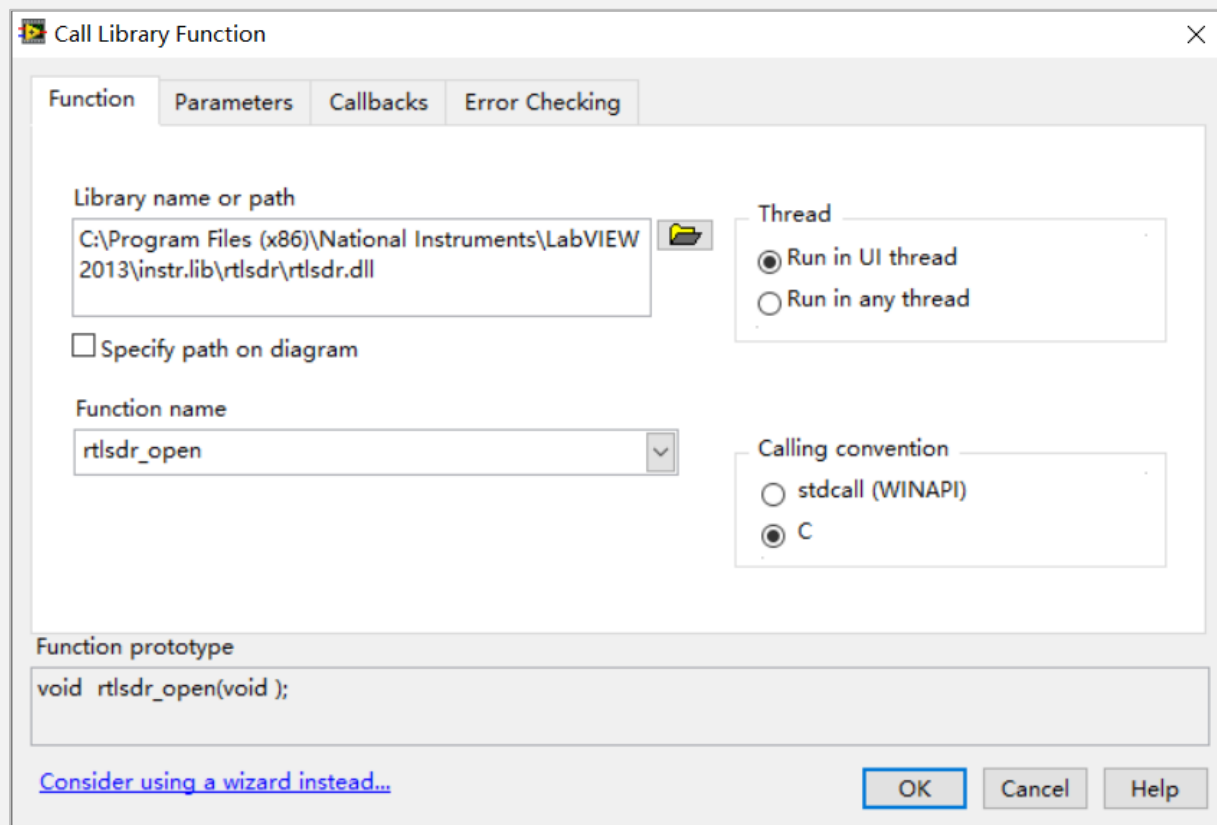
接下来我们可以使用labview中的调用库函数（CLF）将头文件中的接口函数封装成子VI，方便主程序使用。

- (1) 首先准备好库文件.dll和头文件.h， 需要注意的是文件位数要与Labview版本位数一致。
- (2) 在路径“Connectivity”⇒“Libraries & Executables”下找到“Call Library Function Node (CLF)”模块， 导入到程序中。





调用库函数封装接口函数



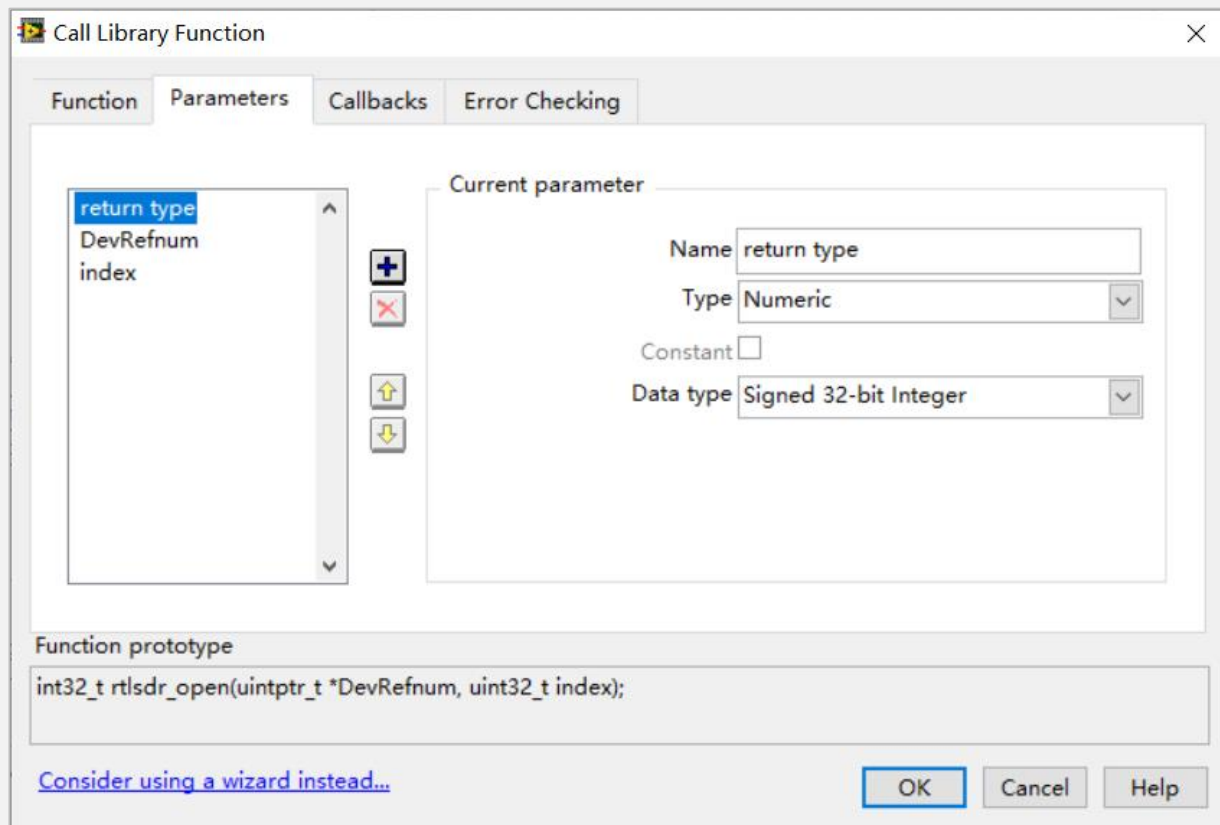
The image shows a 'Call Library Function' dialog box with four tabs: 'Function', 'Parameters', 'Callbacks', and 'Error Checking'. The 'Function' tab is active. It contains the following fields and options:

- Library name or path:** A text box containing 'C:\Program Files (x86)\National Instruments\LabVIEW 2013\instr.lib\rtlsdr\rtlsdr.dll' with a folder icon to its right.
- Specify path on diagram:** An unchecked checkbox.
- Function name:** A dropdown menu showing 'rtlsdr_open'.
- Thread:** Two radio buttons: 'Run in UI thread' (selected) and 'Run in any thread'.
- Calling convention:** Two radio buttons: 'stdcall (WINAPI)' and 'C' (selected).
- Function prototype:** A text box containing 'void rtlsdr_open(void);'.
- Buttons:** 'OK', 'Cancel', and 'Help' at the bottom right.
- Link:** A blue link 'Consider using a wizard instead...' at the bottom left.

(3) 双击CLF模块，在Function弹窗中输入DLL路径、被调函数名、输入和输出参数等信息，thread和calling convention使用默认值。



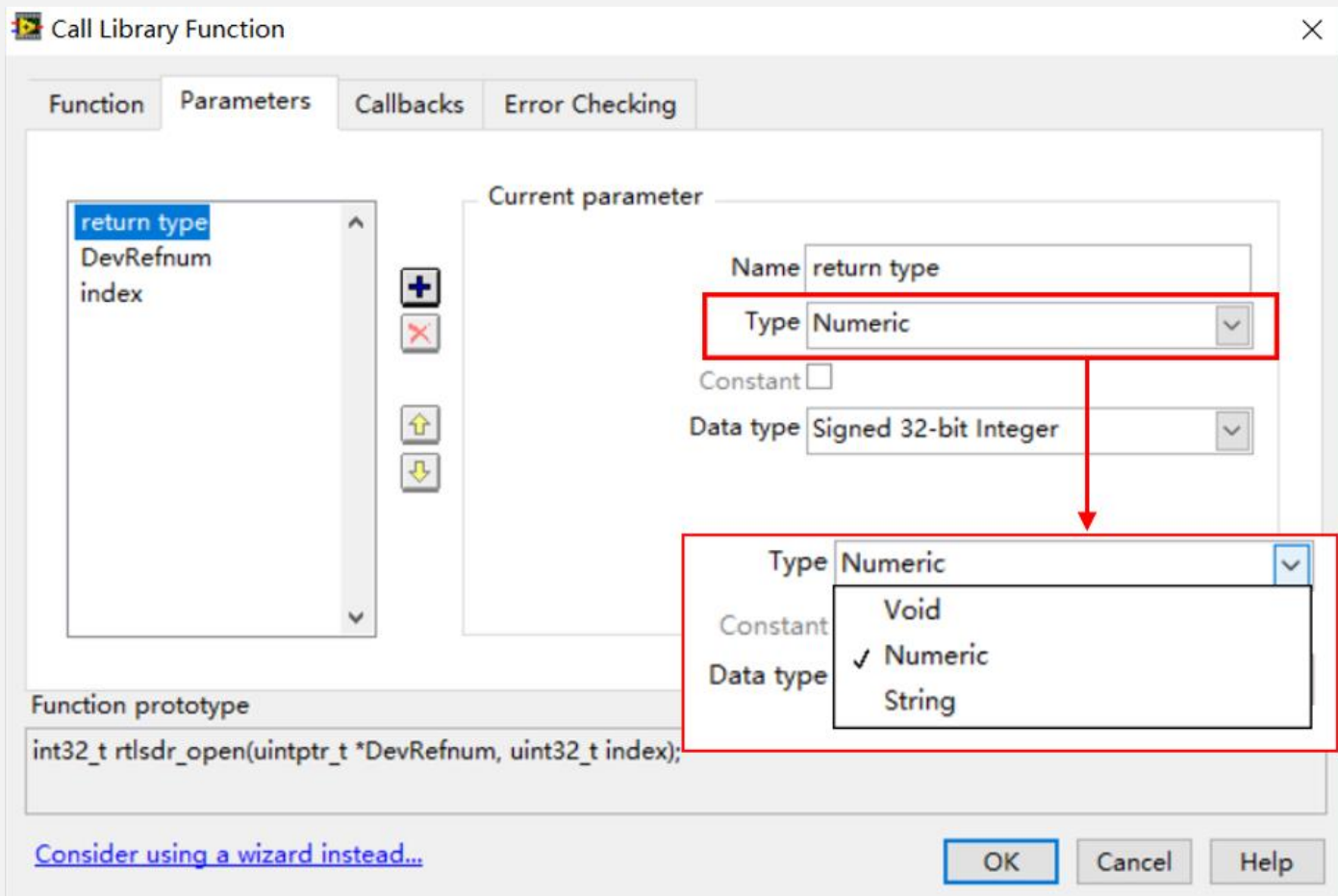
调用库函数封装接口函数



(4) 在parameters页面中。对调用的函数进行参数配置，可以参考头文件中的函数声明。更改数据类型，最终使窗口底部的“Function prototype”与DLL中的函数定义相匹配。



调用库函数封装接口函数

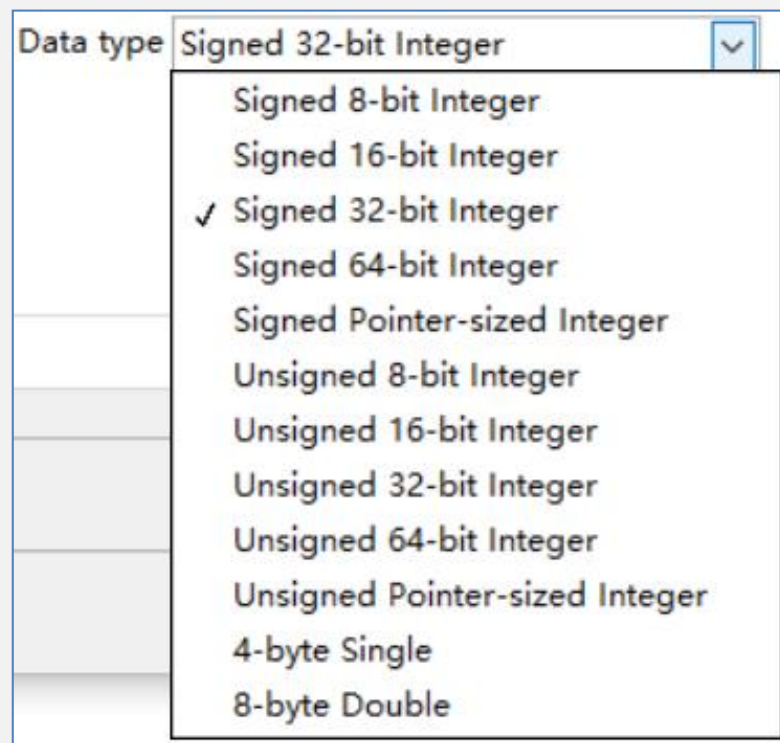


在右侧的“Type”（类型）中选择函数返回值的种类“Void”、“Numeric”或者“String”。



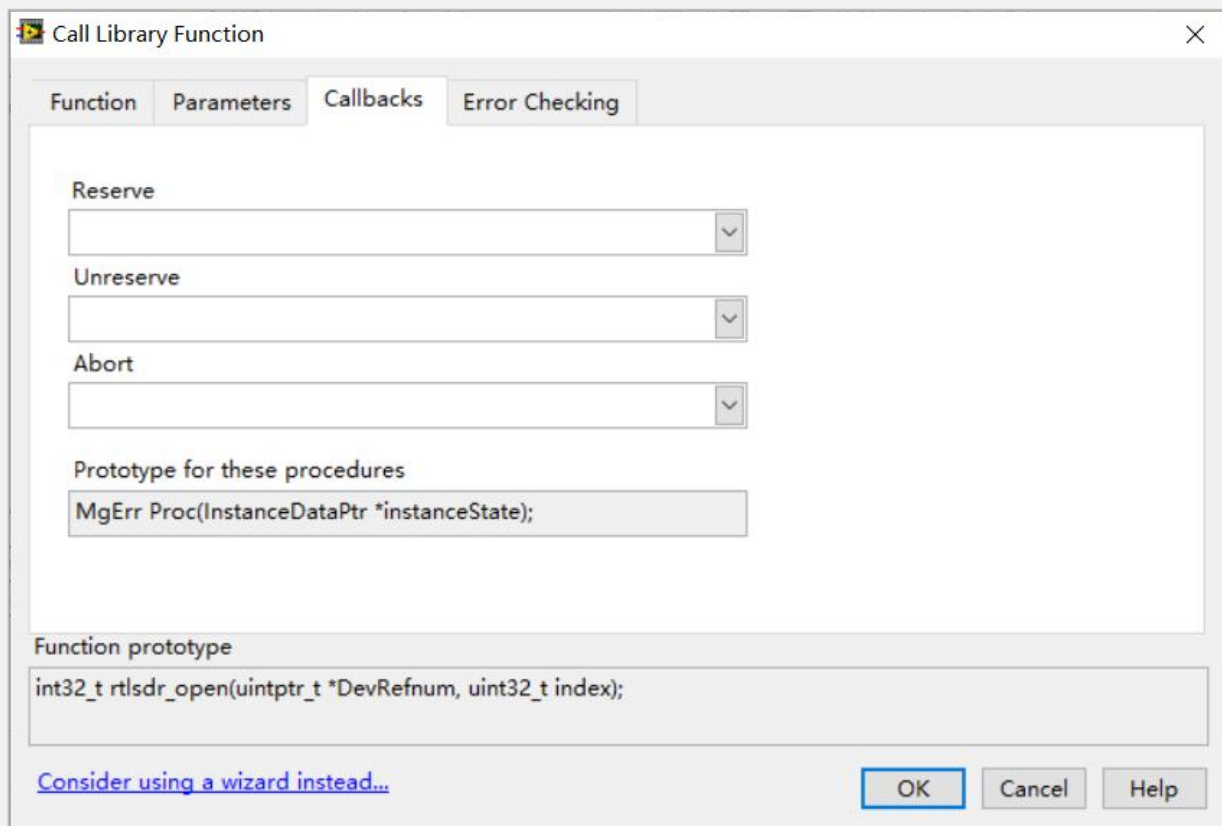
调用库函数封装接口函数

在“Data Type”（数据类型）中选择相应的数据类型。在“Data Type”选项中，我们可以看到多种数据类型，这里需要特别指出的一种数据类型，就是C语言中的指针类型，例如“Signed Pointer-sized Integer”。之后我们将详细介绍指针类型数据在动态链接库中的作用。

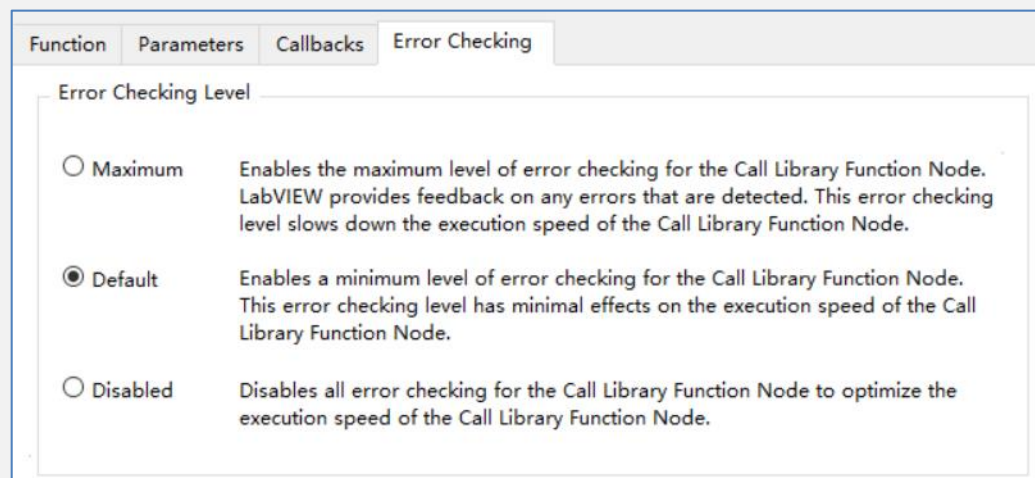




调用库函数封装接口函数



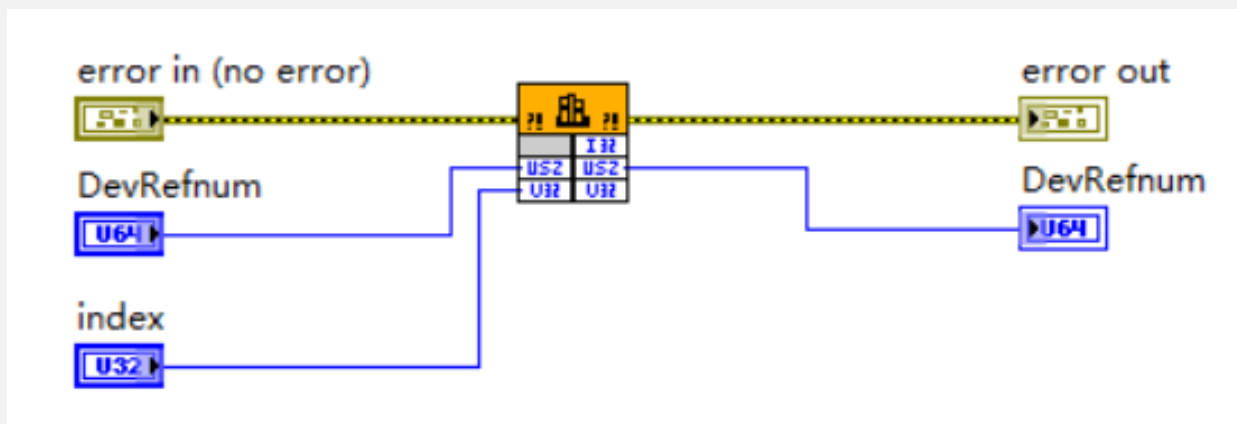
(5) “Callbacks”页面和
“Error Checking”页面使用
默认值，不做更改。





调用库函数封装接口函数

(6) 配置好CLF后，获得了封装好的子VI，和程序接线，就可以实现对接口函数的调用。



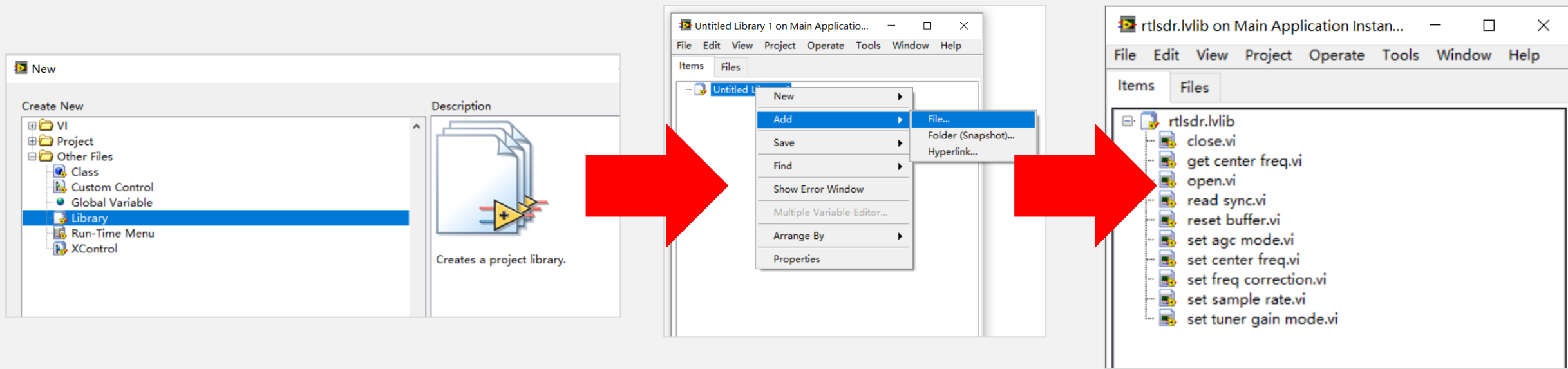
(7) 按照同样的方法，我们可以对头文件rtl-sdr.h中的其他函数进行封装，当我们将需要使用到的函数都封装完成之后，可以将这些封装后的子VI打包成一个库文件，方便维护。



调用库函数封装接口函数

打包封装好的函数：

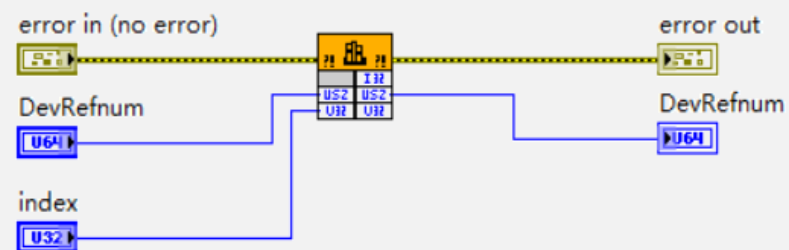
首先在路径“File” ⇒ “New”下新建一个“Library”。然后在库文件名上点击鼠标右键，在弹出的选项中选择“Add”⇒“File”，选择已经封装好的Vi。最后保存，更改库文件名，就完成了库文件的创建，如图。需要注意的是，rtlsdr.dll文件和rtlsdr.lvlib放在同一个文件夹下，方便调用。





目录

- 动态链接库介绍
- RTL-SDR接口函数封装
- **导入共享库向导**
- 动态链接库编译
- LabVIEW调用动态链接库

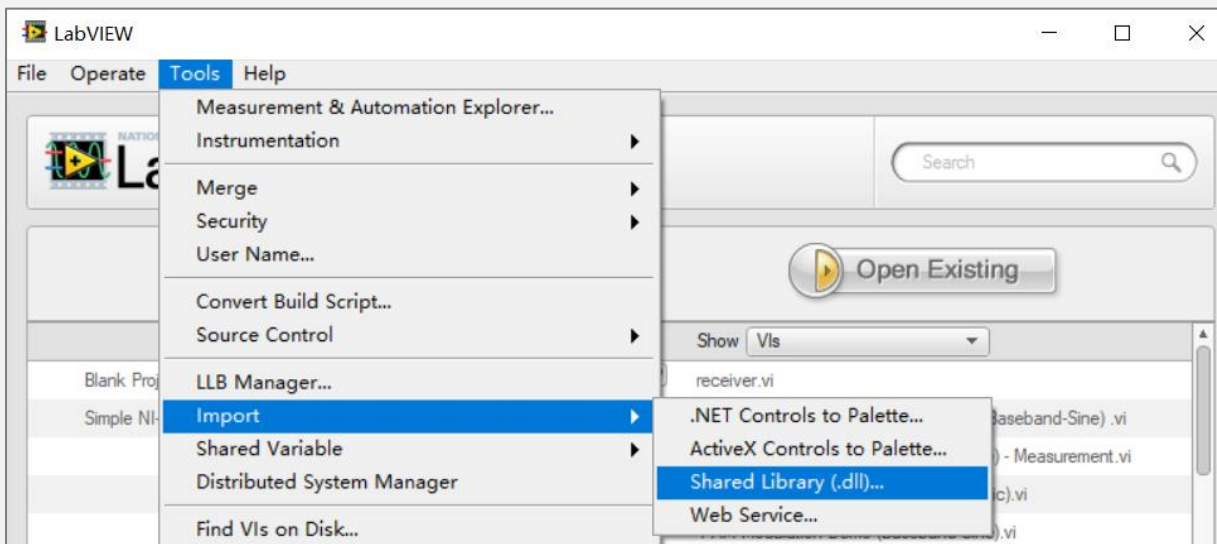




共享库向导封装接口函数

除了采用手动的方式创建LabVIEW接口函数之外，还可以采用导入共享库向导来创建接口函数的LabVIEW库。这种方法可以直接打包接口函数，能够提高工作效率，节省时间。

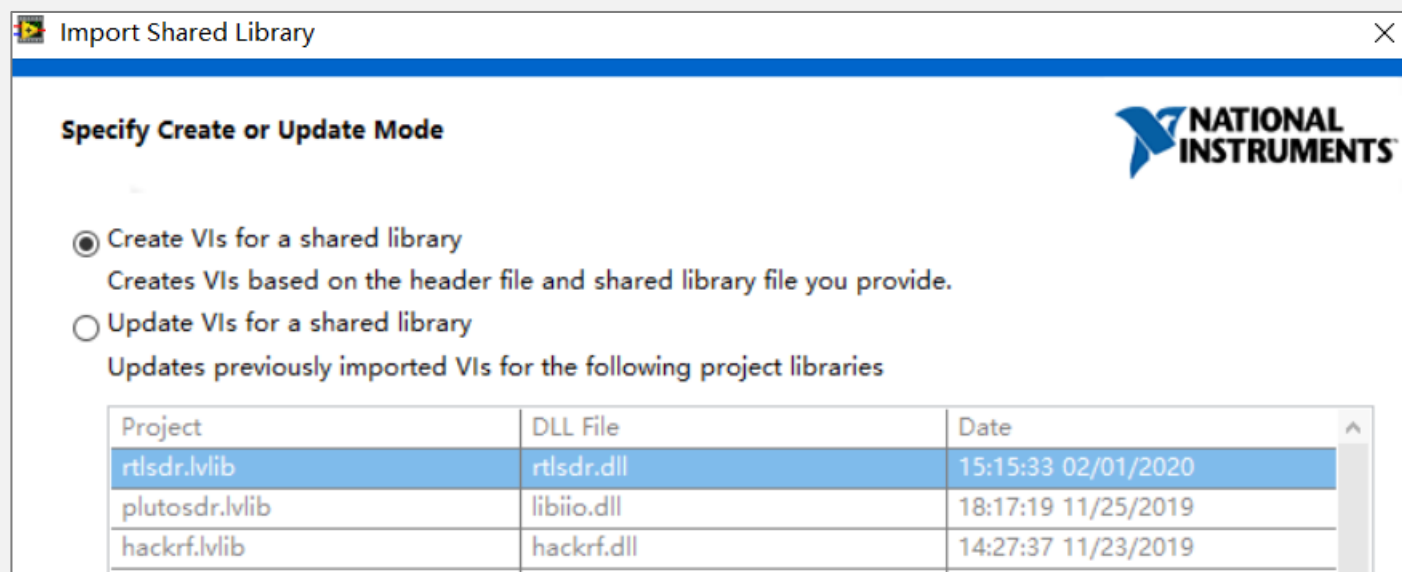
(1) 首先在“Tool”（工具）
“import”（导入）路径下找到“Shared Library”（共享库），
如图所示：





共享库向导封装接口函数

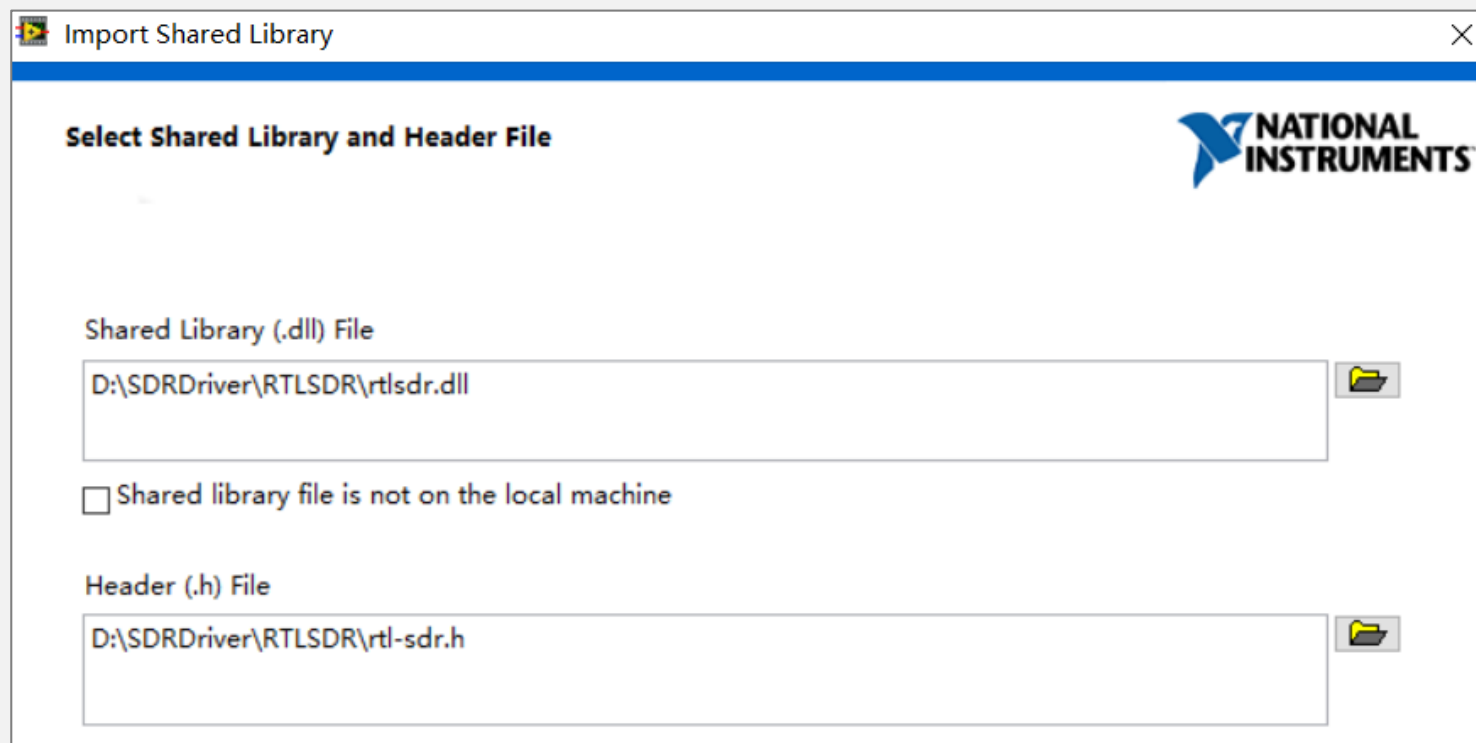
(2) 在弹出的菜单中选择“Create VIs for a shared library”（为共享库创建一个VI集），如图所示。如果需要对已经生成的VI集进行更新，则选择第二项“Update VIs for a shared library”。





共享库向导封装接口函数

(3) 点击下一步进入路径配置页面，在这个页面中，需要设置rtlsdr.dll的路径和头文件的路径，注意这里设置的头文件是rtl-sdr.h，如图。

A screenshot of the 'Import Shared Library' dialog box from National Instruments. The dialog has a title bar with a close button. The main area is titled 'Select Shared Library and Header File' and features the National Instruments logo. It contains two text input fields: 'Shared Library (.dll) File' with the path 'D:\SDRDriver\RTLSDR\rtlsdr.dll' and 'Header (.h) File' with the path 'D:\SDRDriver\RTLSDR\rtl-sdr.h'. Each field has a folder icon to its right. Below the first field is an unchecked checkbox labeled 'Shared library file is not on the local machine'.



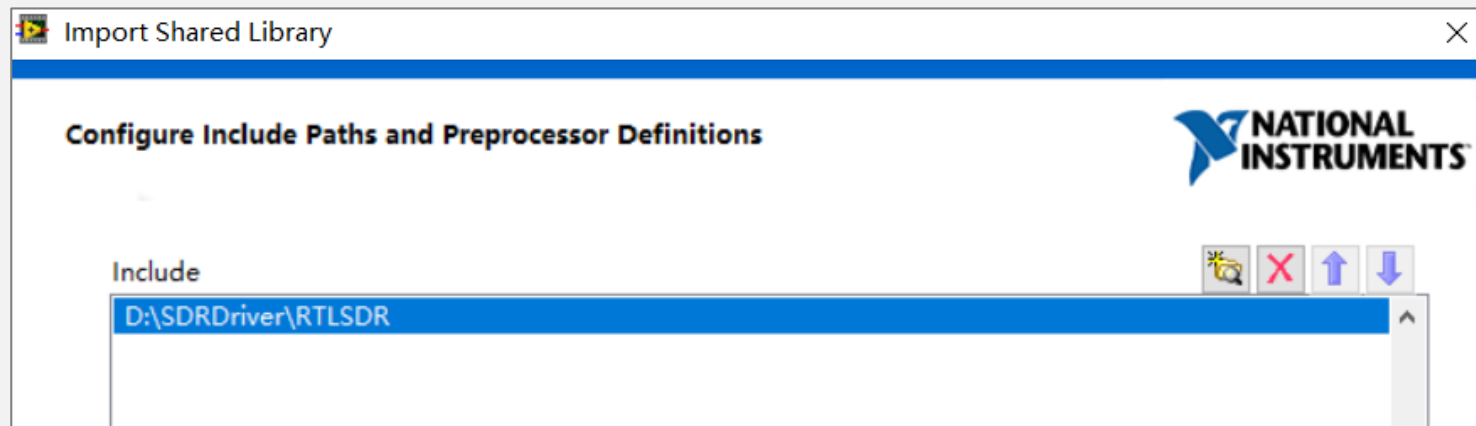
共享库向导封装接口函数

(4) 注意rtl-sdr.h还调用了两个头文件:

```
#include <stdint.h>
```

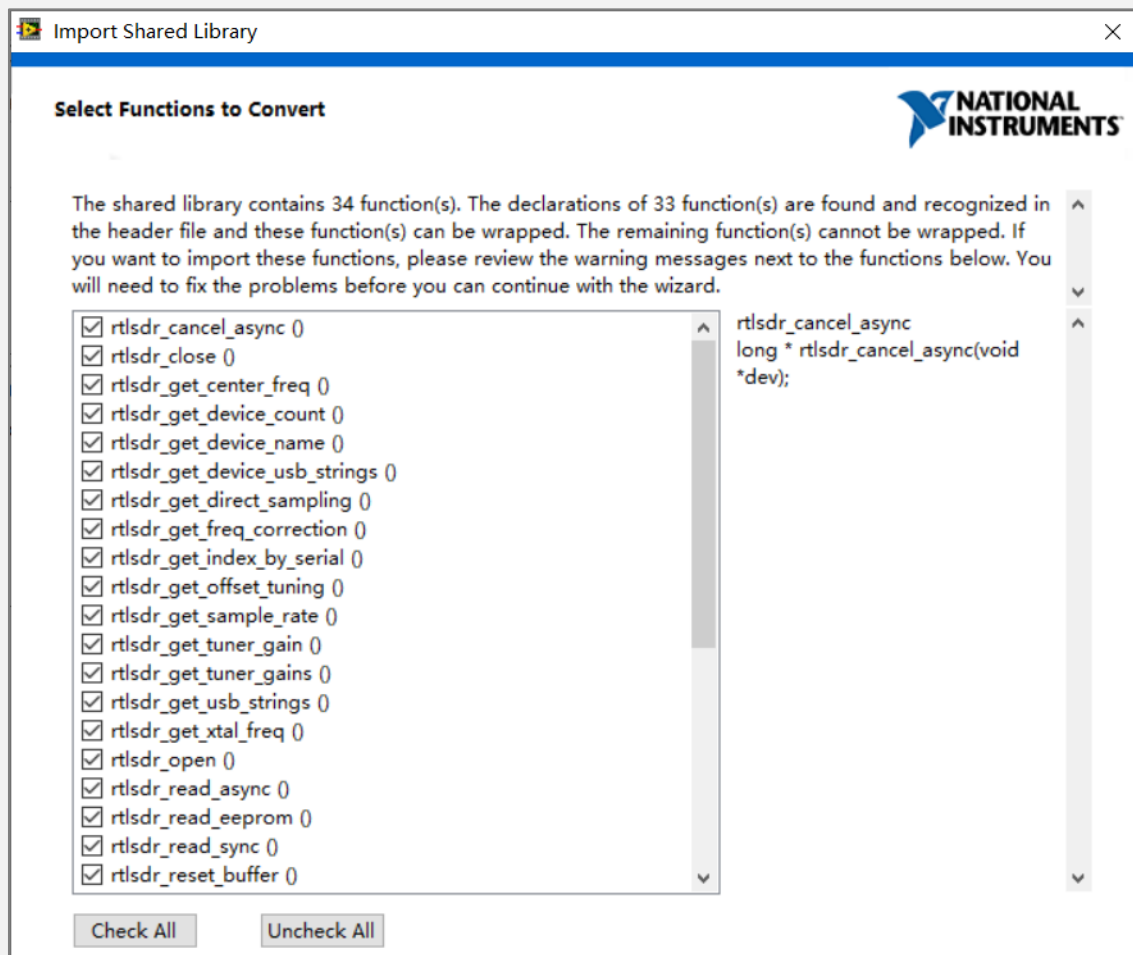
```
#include <rtl-sdr_export.h>
```

需要找到关联头文件路径, 一起导入Include文件夹中。





共享库向导封装接口函数

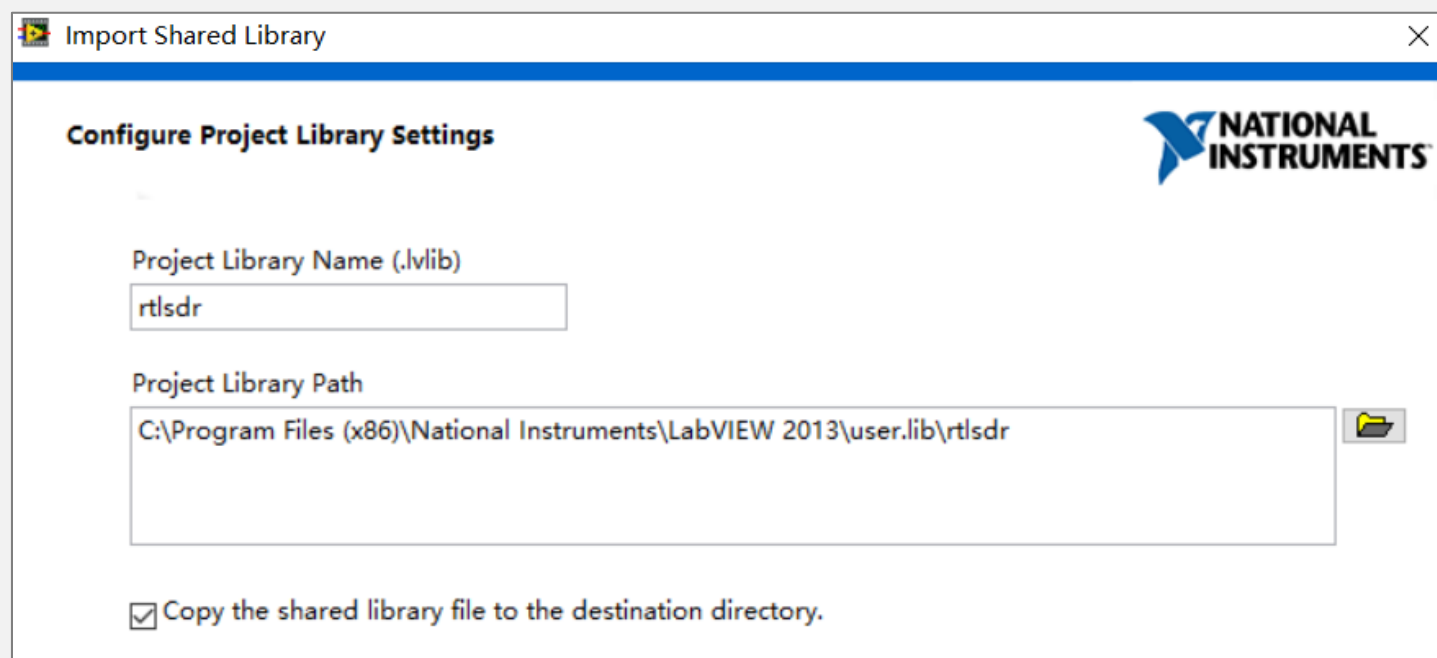


(5) 如果前面步骤配置正确，共享库向导就会根据库文件和头文件生成一个函数列表，此时正常情况下，头文件中的33个接口函数能够被识别和封装。



共享库向导封装接口函数

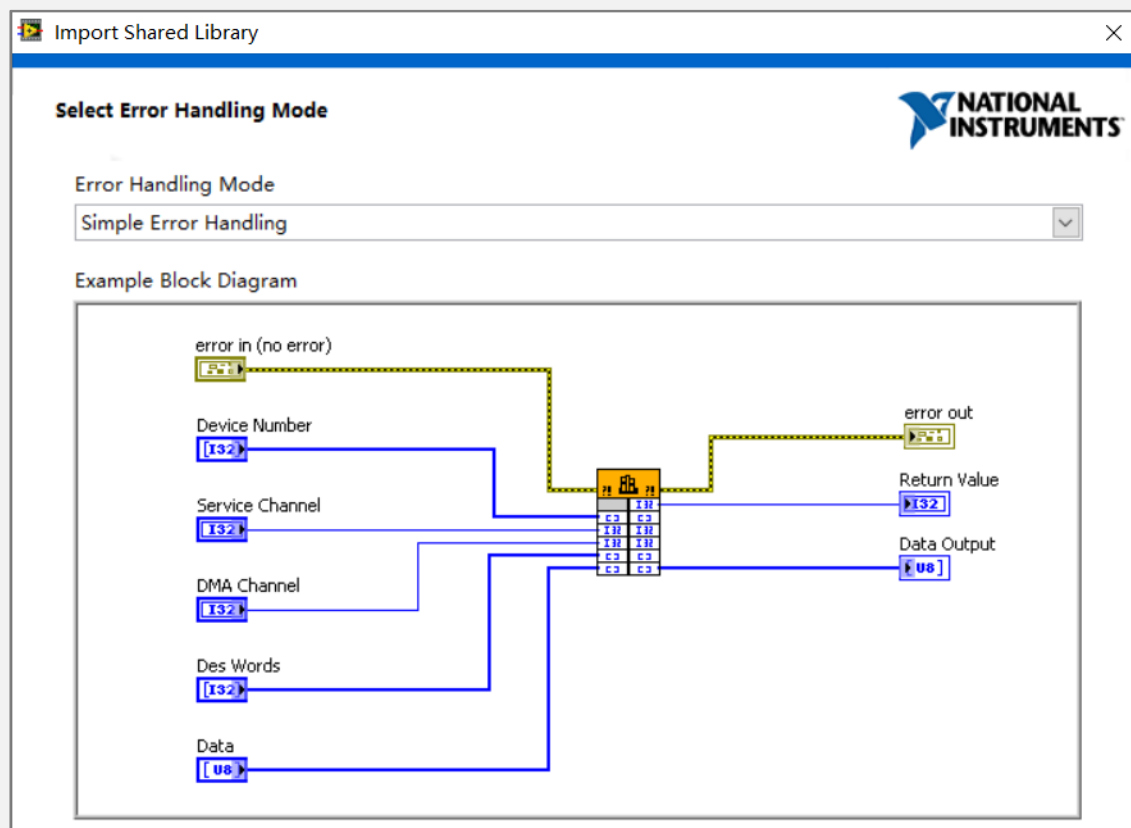
(6) 选定需要生成的函数之后，需要设置LabVIEW库函数的文件名和存放路径。





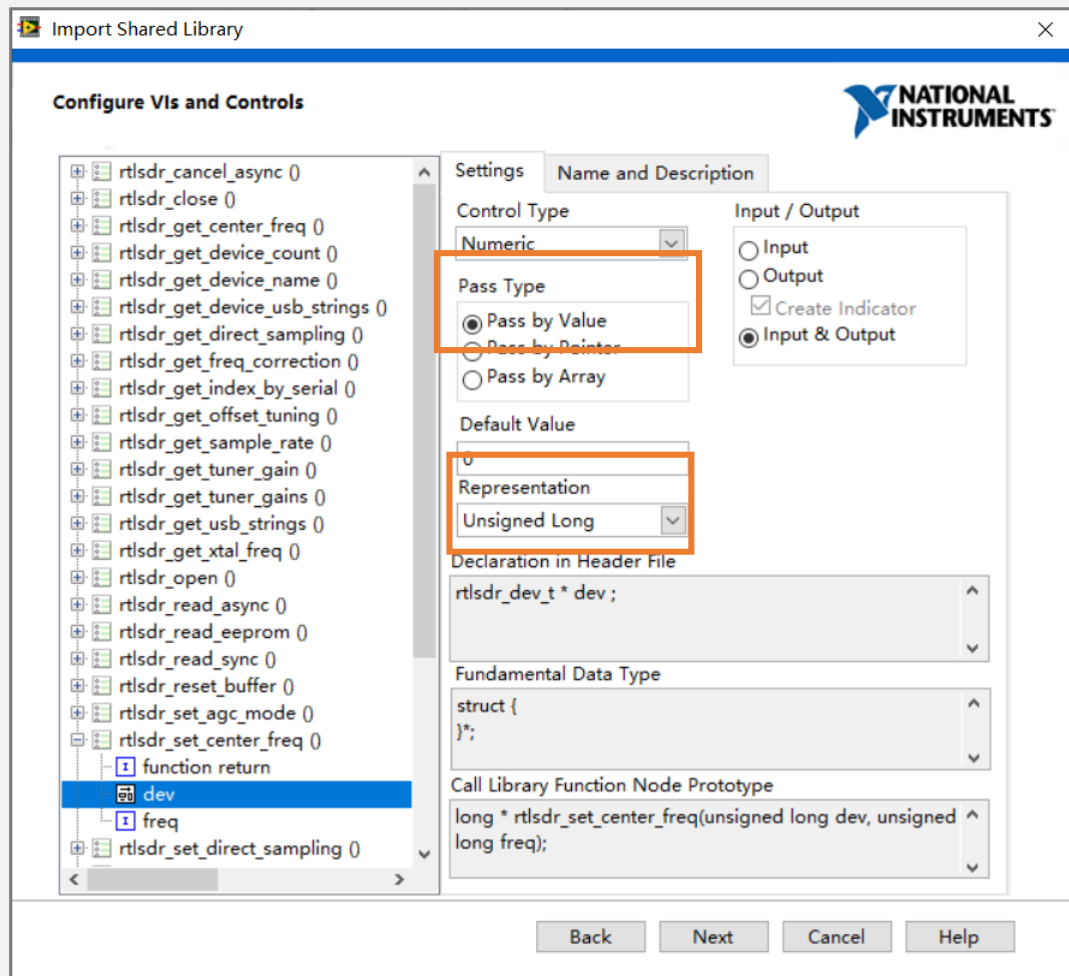
共享库向导封装接口函数

(7) 在接下来的页面中，可以配置错误输出模式，这里我们选择Simple Error Handling。





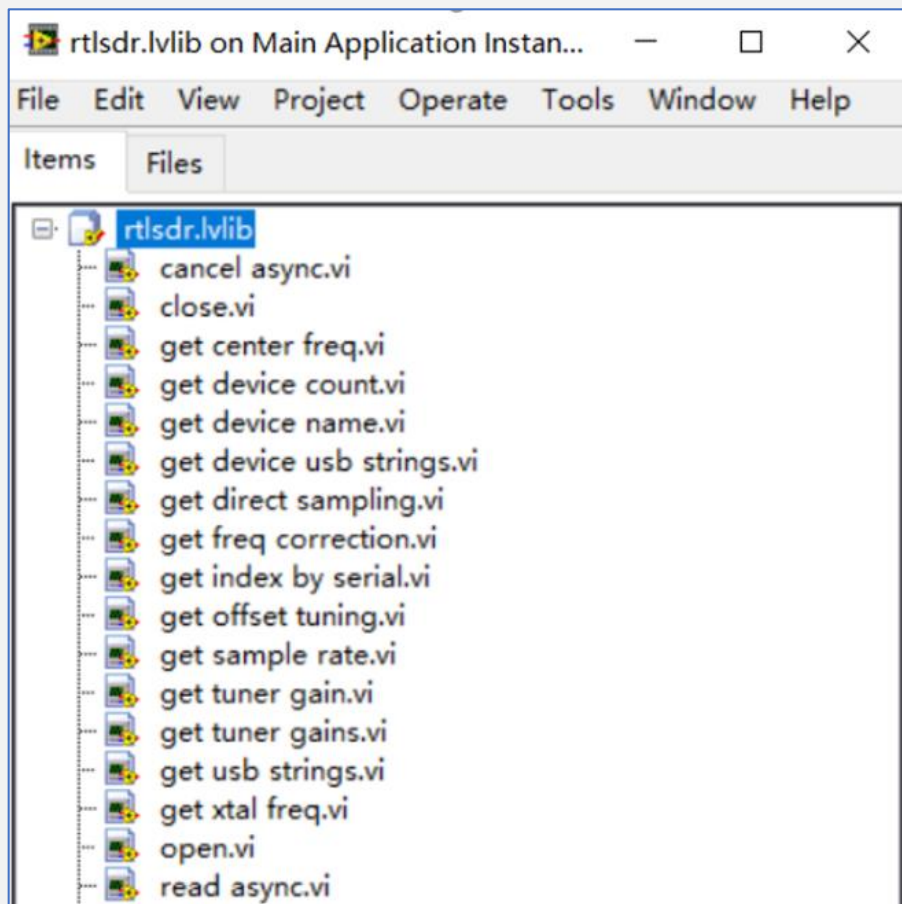
共享库向导封装接口函数



(8) 在接下来的页面中，可以对每个函数的输入参数和输出参数的类型一一进行重配置。



导入共享库向导

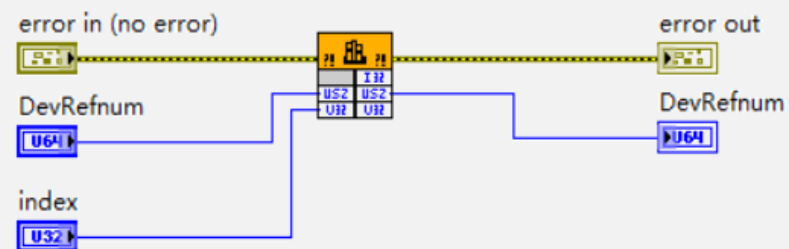


(9) 在配置完成每个函数的输入和输出参数之后，点击“Next”按钮就可以进行封装，等待几分钟之后，将会返回封装后的VI集。单独打开子VI的CLF能查看配置是否正确。



目录

- 动态链接库介绍
- RTL-SDR接口函数封装
- 导入共享库向导
- **动态链接库编译**
- LabVIEW调用动态链接库





动态链接库编译

本小节将讨论如何从源程序中编译动态链接库文件。首先在GitHub网站里搜索到rtlsdr的源文件，选择osmocom/rtl-sdr的版本，网址

<https://github.com/osmocom/rtl-sdr>

在rtl-sdr-master\src文件下，就可以找到rtlsdr库文件对应的源程序librtlsdr.c。

输出

显示输出来源(S): 生成

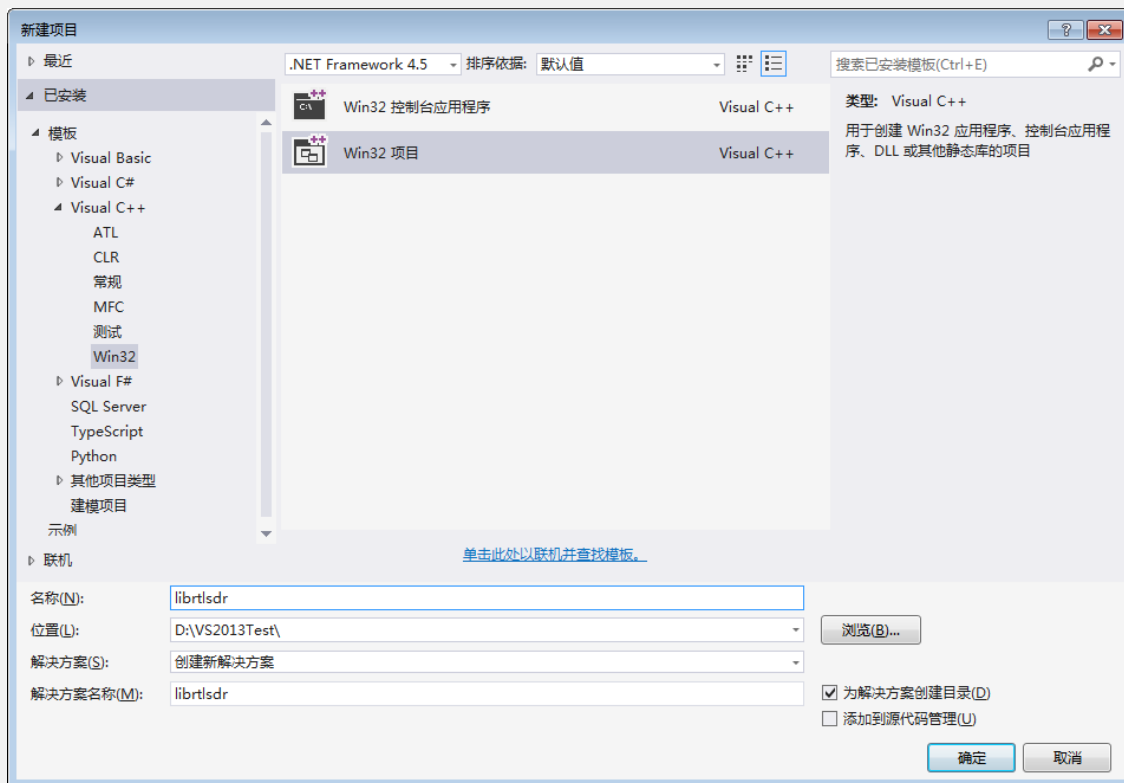
```
1>----- 已启动生成: 项目: rtlsdrdll, 配置: Debug Win32 -----
1> LINK : 没有找到 D:\VS2013Test\rtlsdrdll\Debug\rtlsdrdll.dll 或上一个增量链接没有生成它; 正在执行完全链接
1> 正在创建库 D:\VS2013Test\rtlsdrdll\Debug\rtlsdrdll.lib 和对象 D:\VS2013Test\rtlsdrdll\Debug\rtlsdrdll.exp
1> rtlsdrdll.vcxproj -> D:\VS2013Test\rtlsdrdll\Debug\rtlsdrdll.dll
===== 生成: 成功 1 个, 失败 0 个, 最新 0 个, 跳过 0 个 =====
```



动态链接库编译

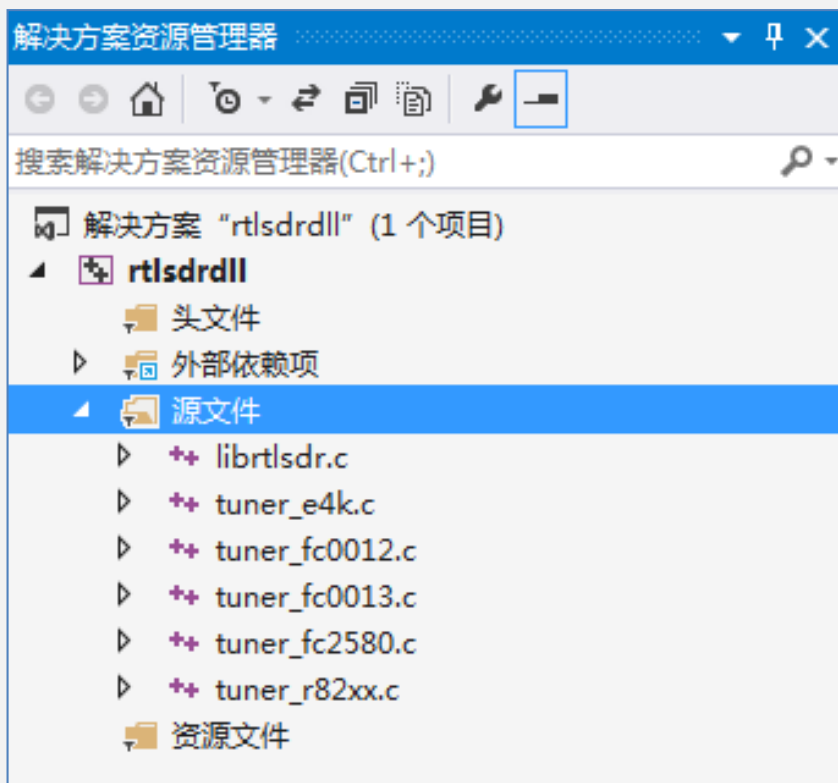
(1) 要将这个librtlsdr.c文件编译成动态链接库文件，需要安装c程序的编译软件。首先启动VS 2013，新建一个Win32项目。

(2) 在弹出的应用程序向导中选择应用程序类型，这里选择“DLL”，在附加选项中选择“空项目”，这样就完成了空项目的创建。





动态链接库编译

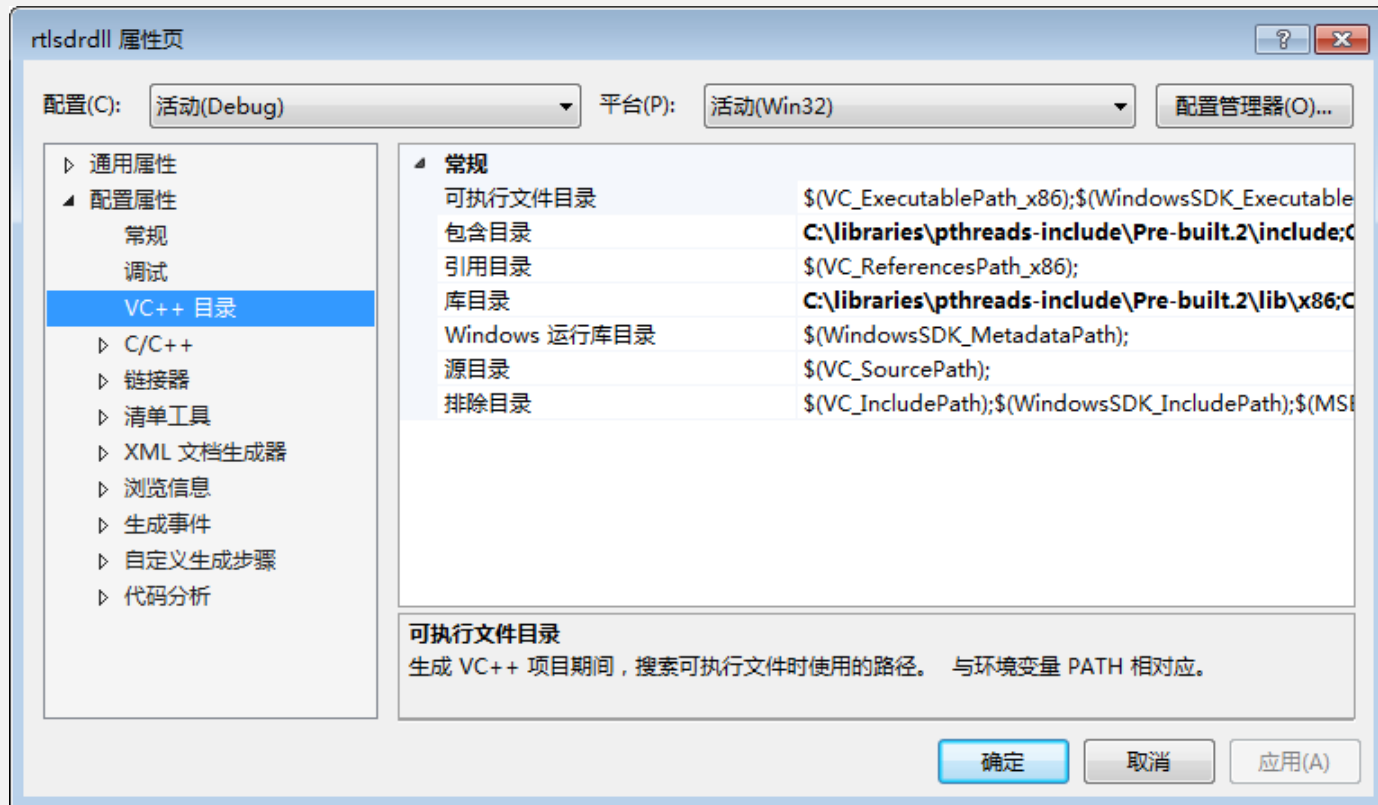


(3) 在“源文件”文件夹上点击右键，依次选择“添加”，“ 现有项”， 在弹出的对话框中将rtl-sdr-master\src文件下的六个c文件导入：

librtlsdr.c,
tuner_e4k.c, tuner_fc0012.c,
tuner_fc0013.c, tuner_fc2580.c,
tuner_r82xx.c



动态链接库编译



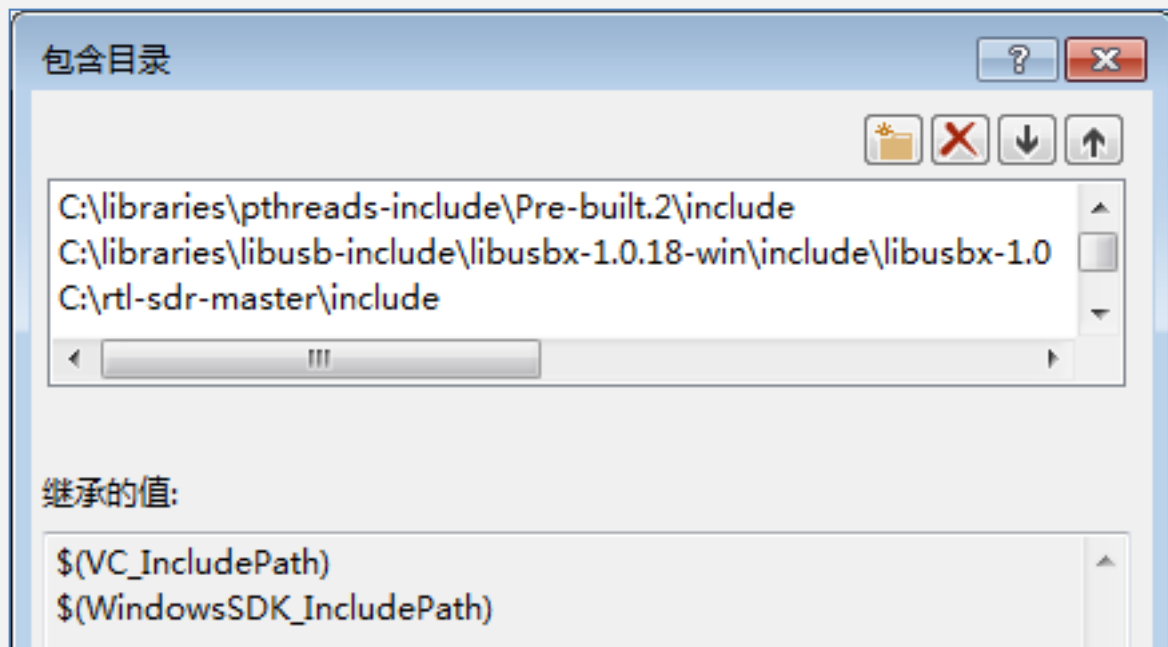
(4) 由于rtlsdr.dll的编译还依赖于libusb-1.0.lib和pthreadVC2.lib这两个静态库文件，因此还需要将这两个文件添加到库目录和包含目录中。

在项目名称“rtlsdr.dll”上单击右键，弹出的窗口中找到“配置属性”，在“VC++目录”页面下就可以配置“包含目录”和“库目录”，如图。



动态链接库编译

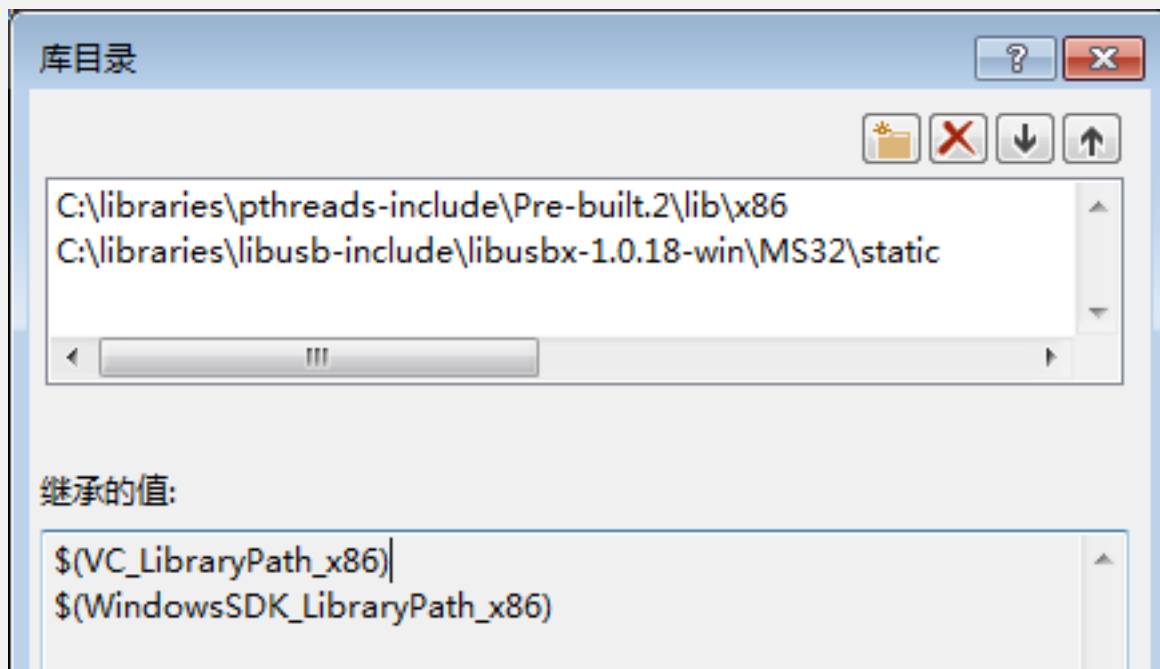
编辑包含目录，将Pre-built.2\include文件夹，libusb-1.0.18-win文件夹下的include\libusb-1.0，以及rtl-sdr-master\include文件夹添加到包含目录中。





动态链接库编译

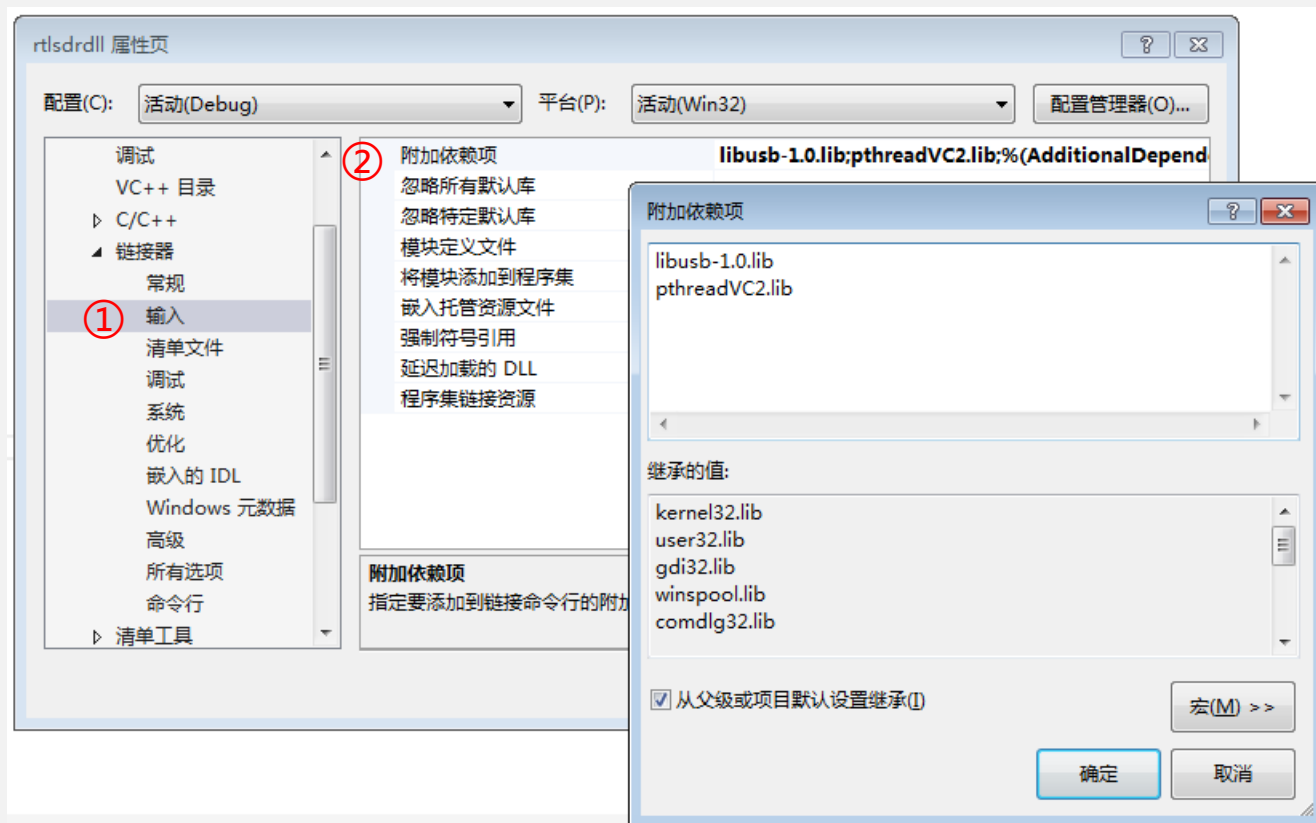
编辑库目录，将Pre-built.2\lib\x86文件夹，libusb-1.0.18-win\MS32\static文件夹添加到库目录中。





动态链接库编译

接下来，还需要在“链接器”下的“附加依赖项”中添加libusb-1.0.lib和pthreadVC2.lib这两个静态库文件，如图所示。





动态链接库编译

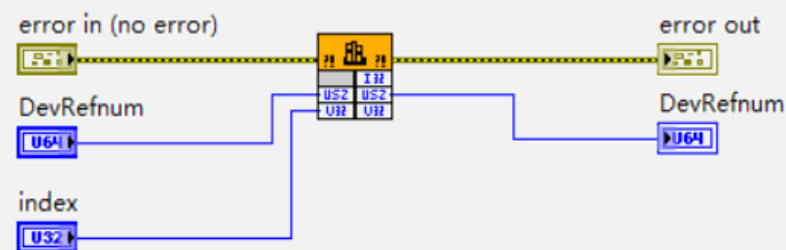
(5) 运行程序，就完成了dll的生成。需要特别注意的是所用版本的VS默认生成的动态链接库是32位的。如果需要64位文件，需要重新配置。

```
输出
显示输出来源(S): 生成
1>----- 已启动生成: 项目: rtlcdrdll, 配置: Debug Win32 -----
1> LINK : 没有找到 D:\VS2013Test\rtlsdrdll\Debug\rtlsdrdll.dll 或上一个增量链接没有生成它; 正在执行完全链接
1> 正在创建库 D:\VS2013Test\rtlsdrdll\Debug\rtlsdrdll.lib 和对象 D:\VS2013Test\rtlsdrdll\Debug\rtlsdrdll.exp
1> rtlsdrdll.vcxproj -> D:\VS2013Test\rtlsdrdll\Debug\rtlsdrdll.dll
===== 生成: 成功 1 个, 失败 0 个, 最新 0 个, 跳过 0 个 =====
```



目录

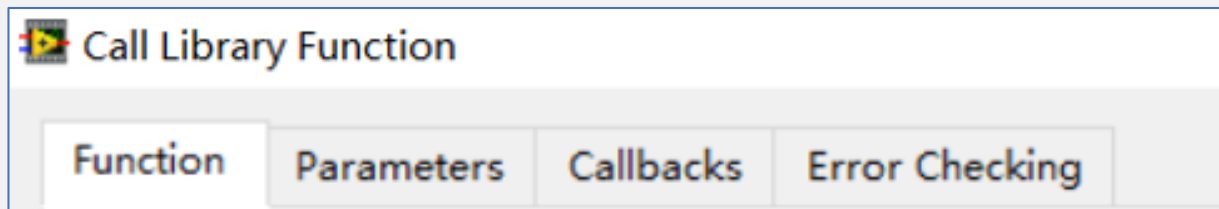
- 动态链接库介绍
- RTL-SDR接口函数封装
- 导入共享库向导
- 动态链接库编译
- LabVIEW调用动态链接库





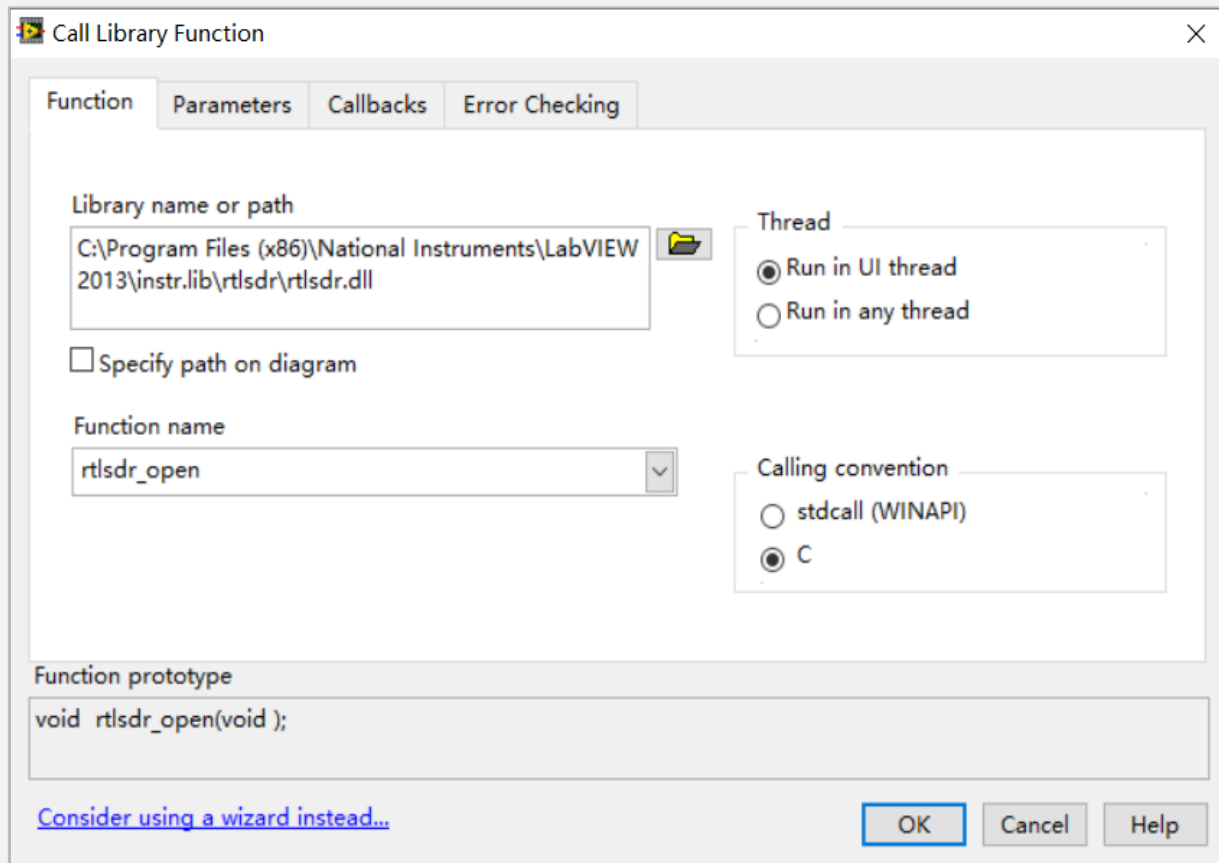
LabVIEW调用动态链接库

CLF的参数配置分为“Function”（函数）、“Parameters”（参数）、“Callbacks”（回调）、“Error Checking”（错误检查）四栏。





Function 配置

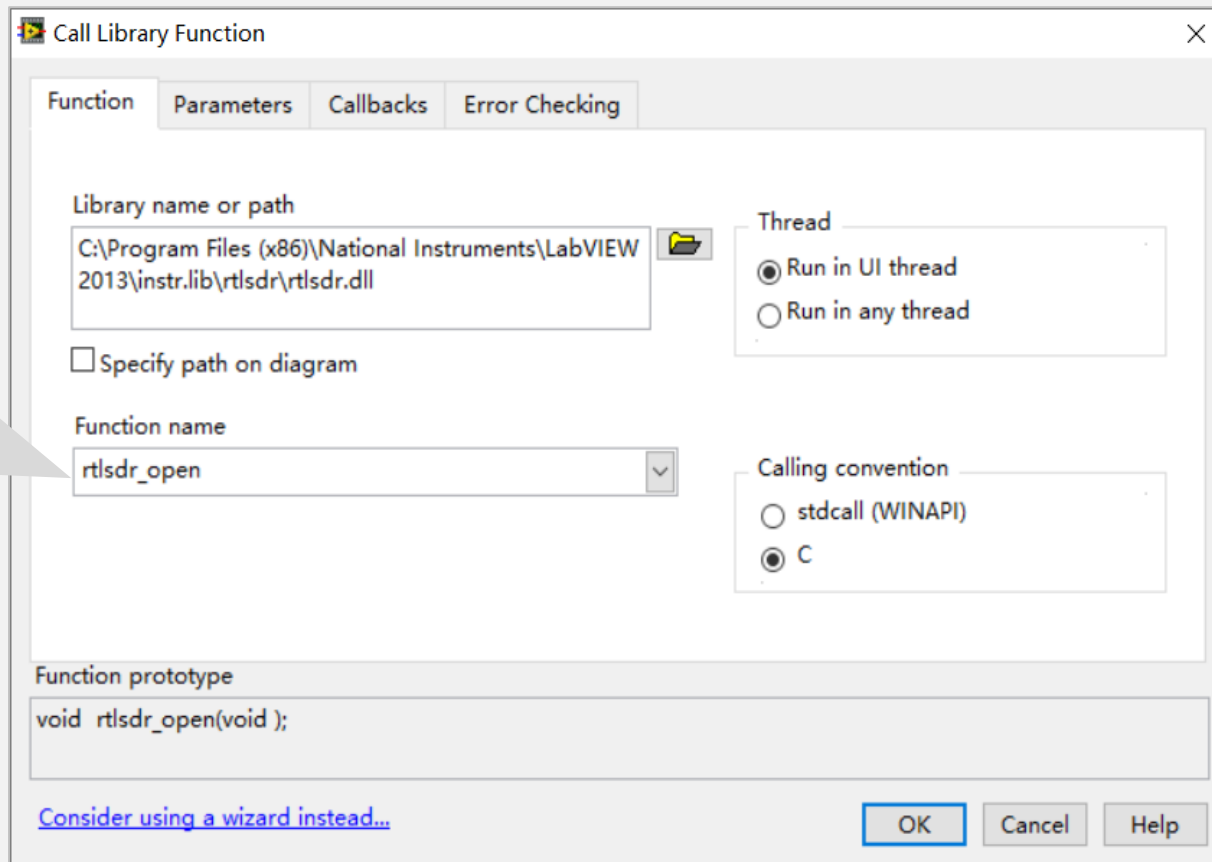


- “Library name or path”（库名/路径）文本框中，点击文件夹图标，**选择DLL所在的路径**，就可以完成设置。
- 如果库文件是LabVIEW可搜索路径下的DLL文件，**直接输入文件名也可直接调用**。
- 勾选specify path on diagram选项，LabVIEW将会“动态加载”DLL，**在程序中获取DLL地址**。



Function 配置

“Function name” 选项中，
点击下拉菜单，会列出
DLL中所有的可用函数，
在下拉框中选取所需要的
函数。



The image shows the "Call Library Function" dialog box in a software development environment. The dialog has four tabs: "Function", "Parameters", "Callbacks", and "Error Checking". The "Function" tab is active. It contains the following fields and options:

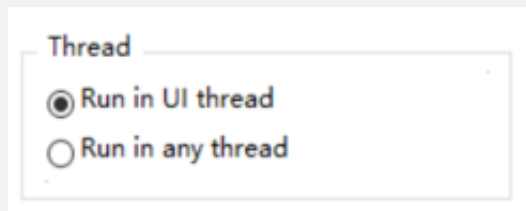
- Library name or path:** A text box containing "C:\Program Files (x86)\National Instruments\LabVIEW 2013\instr.lib\rtlsdr\rtlsdr.dll" with a folder icon to its right.
- Specify path on diagram:** An unchecked checkbox.
- Function name:** A dropdown menu showing "rtlsdr_open".
- Thread:** Two radio buttons: "Run in UI thread" (selected) and "Run in any thread".
- Calling convention:** Two radio buttons: "stdcall (WINAPI)" and "C" (selected).
- Function prototype:** A text box containing "void rtlsdr_open(void);".

At the bottom of the dialog, there is a link "Consider using a wizard instead..." and three buttons: "OK", "Cancel", and "Help".



Function 配置

在“Thread”中，有“Run in UI thread”和“Run in any thread”两个选项。

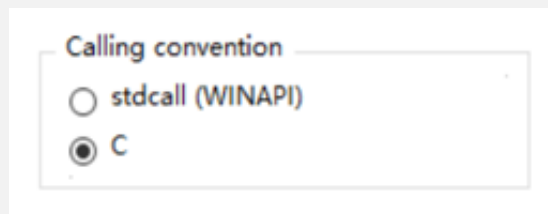


根据实际情况，如果动态链接库是多线程安全的，则选择“Run in any thread”方式。相反，如果动态链接库的源代码中存在可能会冲突全局变量、静态变量或者外部资源，那么这个动态链接库只能选择“Run in UI thread”方式。系统默认选择为“Run in UI thread”方式。



Function 配置

“Calling convention”选项，支持两种约定：“stdcall”和“C call”，

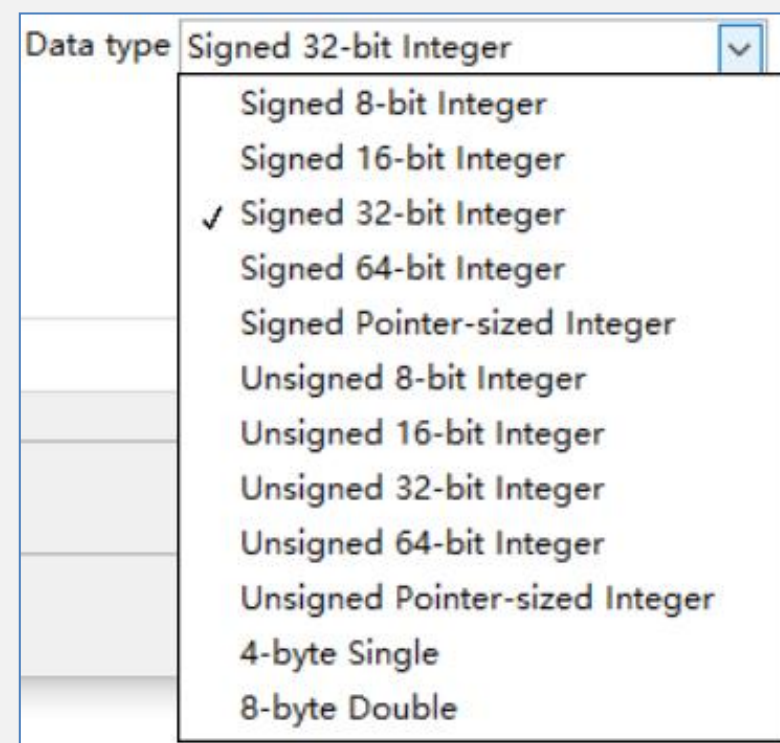


它们的区别是“stdcall”由被调用者负责清理堆栈，“C call”由调用者清理堆栈。
Windows API一般使用的都是“stdcall”；标准C的库函数则大多使用“C call”。



Parameters 配置

可以看到，Data type提供了包括数值、布尔型、数组等数据类型，我们先来理解指针类的数据类型。例如“Signed Pointer-sized Integer”。所谓的指针，指的就是变量的地址，把地址作为参数传递到DLL函数中，DLL函数就可以操作这个地址指向的变量。



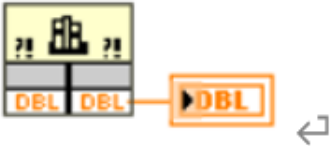
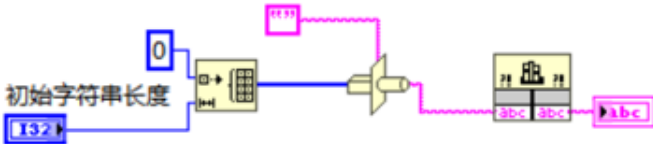
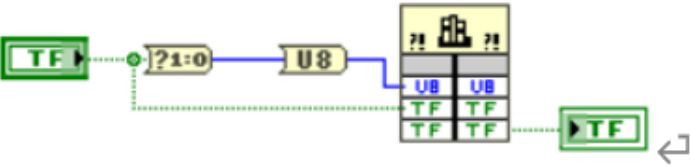


CLF中常用指针类型配置

- 使用布尔类型时，由于布尔类型在DLL函数和LabVIEW VI之间传递没有专有的数据类型，需要利用数值类型来传递的，因此输入时先要把布尔值转变为数值，再传递给DLL函数，输出时再把数值转为布尔值。**需要注意的是，如果在C语言函数参数声明中有const关键字，需要选中Constant选项。**
- 标量数据类型，在传递给DLL函数时，可能是值，也可能是指针。
- 数组这种类型，只能是指针。在传递数组类型时，“Array Format”（数组格式）要选择“Array Data Pointer”（数组数据指针）。



CLF中常用指针类型配置

C 语言 [↵]	CLF 配置 [↵]	CLF 使用 [↵]
<code>double *a</code> [↵]	<div>Type Numeric[↵]</div> <div>Constant <input type="checkbox"/></div> <div>Data type 8-byte Double[↵]</div> <div>Pass Pointer to Value[↵]</div>	
<code>Char *a</code> [↵]	<div>Type String[↵]</div> <div>Constant <input type="checkbox"/></div> <div>String format C String Pointer[↵]</div> <div>Minimum size <None>[↵]</div>	
<code>bool *a</code> [↵]	<div>Type Adapt to Type[↵]</div> <div>Constant <input type="checkbox"/></div> <div>Data format Handles by Value[↵]</div>	

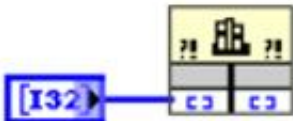
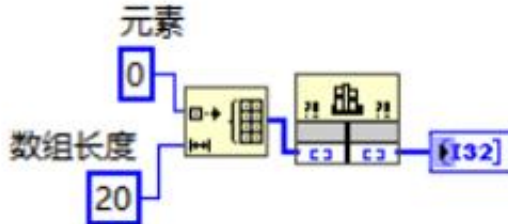


CLF中数组配置和使用

- 当DLL中没有使用到数组参数作为输出值时，需要为输出的定义数组大小。这里有两种方法，第一种方法是创建一个长度满足要求的数组，作为初始值传递给参数，输出数的数据就会被放置在输入数组的所在的内存空间内。
- 第二种方法是直接在参数配置面板上进行设置，在“Minimum size”中写入一个固定的数值，LabVIEW就会按此大小为输出的数组开辟空间。



CLF中数组配置和使用

C 语言声明↵	int a[]↵	int *a[]↵
CLF 中的配置↵	<p>Type Array ▼</p> <p>Constant <input type="checkbox"/></p> <p>Data type Signed 32-bit Integer ▼</p> <p>Dimensions 1</p> <p>Array format Array Data Pointer ▼</p> <p>Minimum size <None> ▼ ↵</p>	<p>Type Array ▼</p> <p>Constant <input type="checkbox"/></p> <p>Data type Signed 32-bit Integer ▼</p> <p>Dimensions 1</p> <p>Array format Array Data Pointer ▼</p> <p>Minimum size <None> ▼ ↵</p>
CLF 中的使用↵	↵ 	↵ 



CLF中簇的配置和使用

- 在C语言中，有一类与“cluster”（簇）类似的数据类型，也就是“struct”（结构）这种数据类型。由单个或者多个不同类型的成员变量组成。
- 但是在CLF节点的配置面板中，没有专门命名为“struct”或者“cluster”参数类型，我们在parameters中选择“Adapt to Type”就可以。
- 需要注意的是，“cluster”由不同数据类型构成时，需要考虑字节对齐问题。



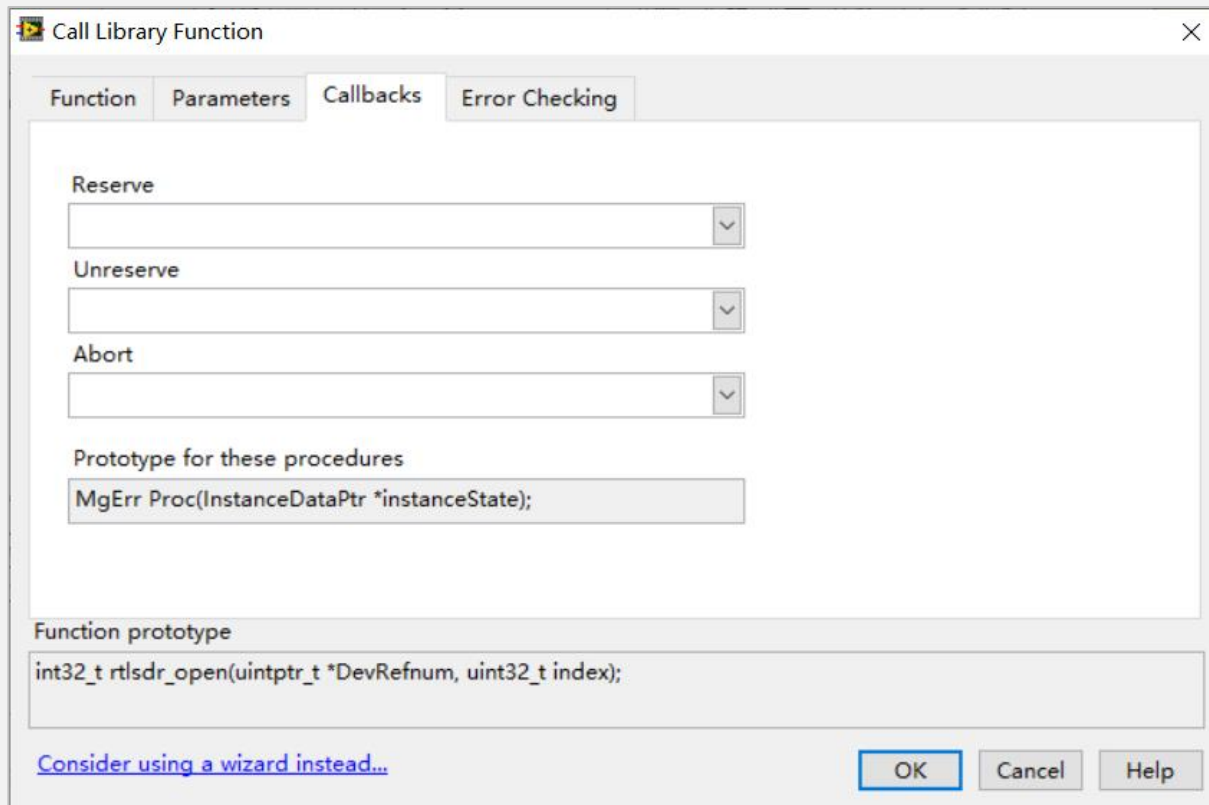
字节对齐问题

- 在C语言中，根据数据类型的不同，在结构体分配空间时也会有不同的对齐值：“char”对应LabVIEW中U8类型，其自身的对齐值为1；“int”对应LabVIEW中I32类型，其自身对齐值为4；“double”对应了LabVIEW中DBL类型，其自身对齐值为8。
- 在LabVIEW的“cluster”中，所有元素均是1字节对齐，也就是簇实际分配空间与各个元素所需空间之和相等。因此，当C语言中“struct”与LabVIEW中的“cluster”对应时，需要做适当调整。



Callbacks配置

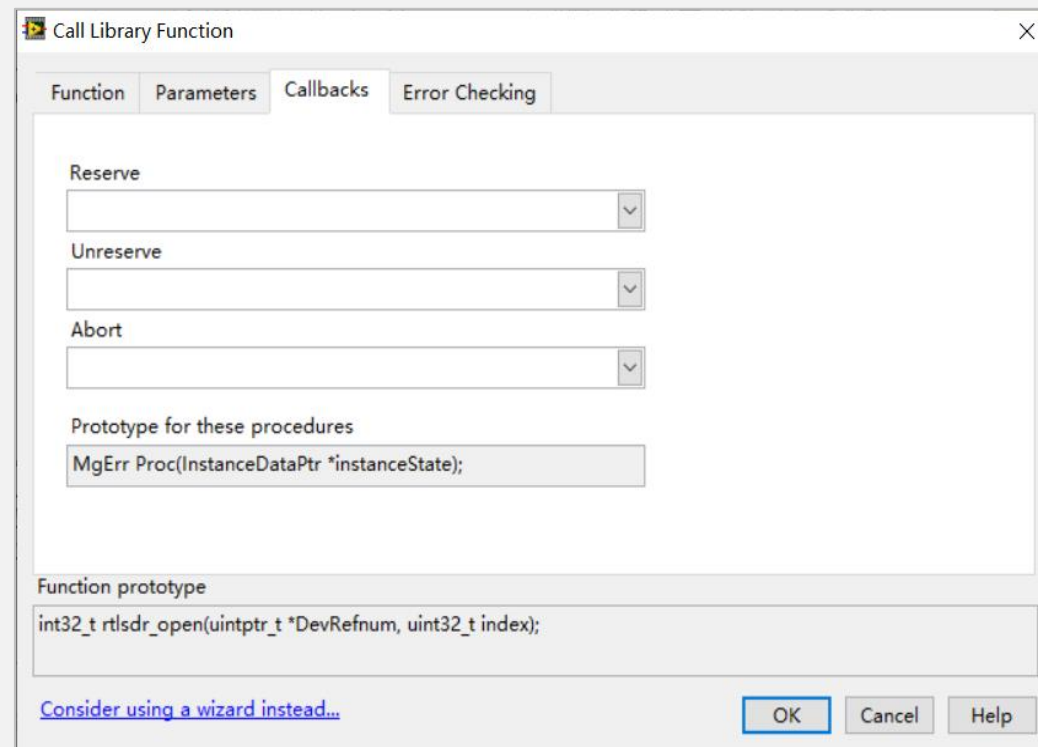
- 有时候我们需要将应用程序的某些功能提供给DLL使用，这时就可以使用回调函数。回调函数是一个通过函数指针调用的函数，这些回调函数可以用于在特定的情形下完成初始化、清理资源等工作。





Callbacks配置

- 如果为“Reserve”选择了一个回调函数，那么当一个新的线程开始调用这个DLL时，这个回调函数首先被调用。
可以利用这个函数为新线程使用到的数据做初始化工作。
- 线程在使用完这个DLL之后，它会去调用Unreserve中指定的回调函数。
- Abort中指定的函数用于VI非正常结束时被调用。





Error Checking配置

- 系统提供了三种错误检查的方案，从最高级到不检测会降低程序的排查成本，同时对节点运行速度的影响逐渐减小。

The image shows a screenshot of the 'Error Checking' configuration dialog box in LabVIEW. The dialog has four tabs: 'Function', 'Parameters', 'Callbacks', and 'Error Checking'. The 'Error Checking' tab is selected. Inside the dialog, there is a section titled 'Error Checking Level' with three radio button options:

Error Checking Level	Description
<input type="radio"/> Maximum	Enables the maximum level of error checking for the Call Library Function Node. LabVIEW provides feedback on any errors that are detected. This error checking level slows down the execution speed of the Call Library Function Node.
<input checked="" type="radio"/> Default	Enables a minimum level of error checking for the Call Library Function Node. This error checking level has minimal effects on the execution speed of the Call Library Function Node.
<input type="radio"/> Disabled	Disables all error checking for the Call Library Function Node to optimize the execution speed of the Call Library Function Node.



- Question ?





【通信新说】



腾讯课堂