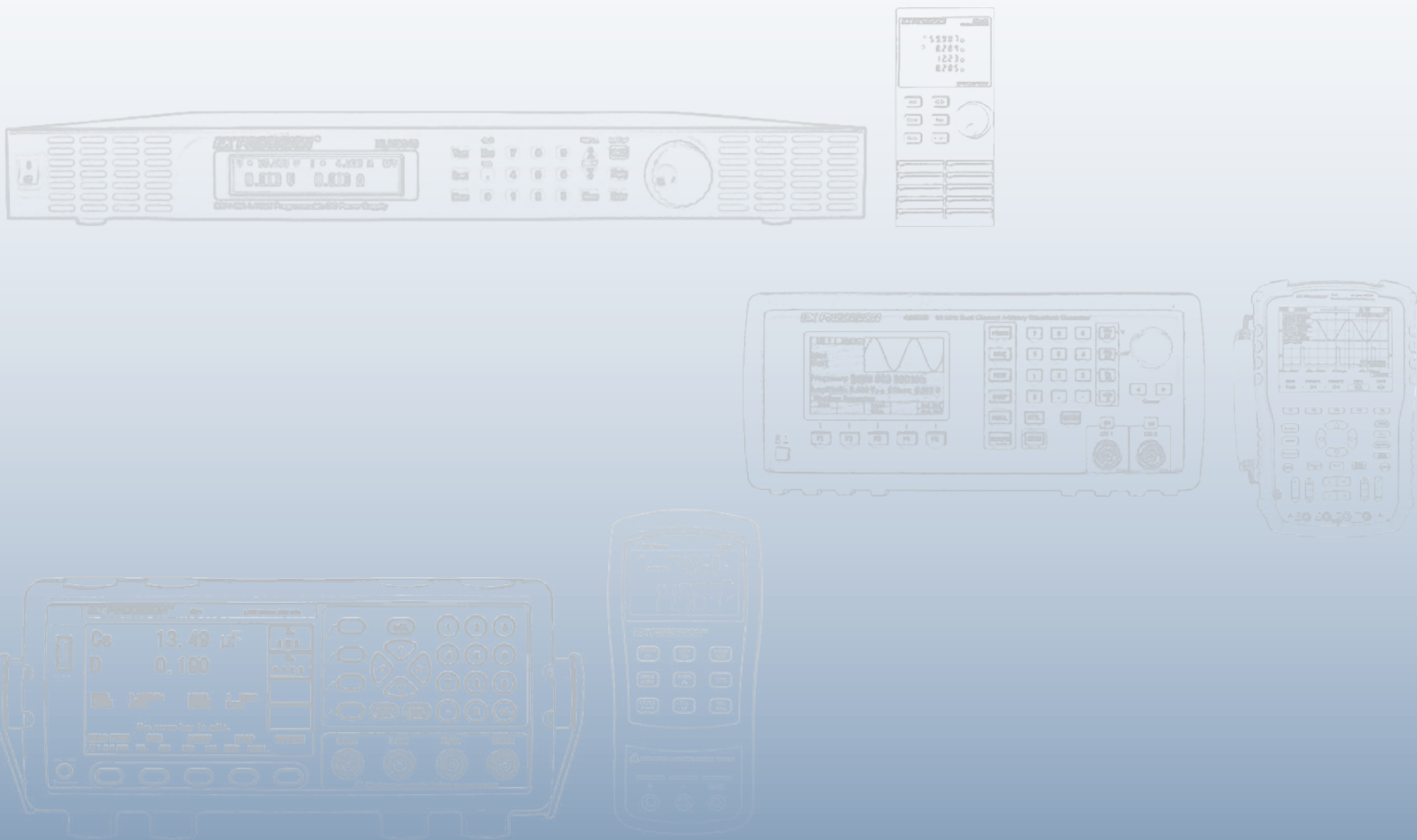


# *VISA Remote Control*

## *C++*

## *C#*

## *Python*



# Contents

1	Introduction	3
1.1	Getting Started	3
1.2	Installing NI VISA	4
1.2.1	Installing .NET Development Support if NI-VISA is already installed	6
2	C++	7
2.1	Include NI VISA in C++	7
2.1.1	Headers	10
2.2	Initializing Your Instrument	11
2.3	Event Handling	16
2.3.1	Queuing Mechanism	16
2.3.2	Callback Handler	16
2.3.3	Enabling Events	16
2.4	Data Logging	21
3	C#	22
3.1	Using NI VISA in C#	22
3.1.1	Headers	26
3.2	Initialization of an Instrument	27
3.2.1	Opening A Session	29
3.3	Establishing Communication	30
3.4	Event Handling	31
3.4.1	Queuing Mechanism	31
3.4.2	Callback Handler	31
3.4.3	Event Handling Example	31
3.5	Data Logging	33
3.5.1	Creating Exel File	33
4	Python	37
4.1	Import NI VISA to Python	37
4.1.1	Headers	39
4.2	Initialization	39
4.2.1	Opening a Session	40
4.3	Python Event Handling	41
4.3.1	Queuing Mechanism	41
4.3.2	Callback Handler	41
4.3.3	Event Handling Example	41
4.4	Data Logging	43
4.4.1	Data Acquisition	45

# Introduction

This application note describes the setup required in order to remotely control a BK Precision instrument using the NI-VISA packages on either: Python, C++, or C#.

This chapter lists what you need to get started, and provides a brief overview of NI VISA.

---

## 1.1 Getting Started

---

To facilitate communication between the instrument and the pc the VISA Application Programming Interface (API) will be used.

VISA is a high level driver that abstract lower lever drivers to establish communication with the following bus types: Serial, USB, Ethernet / LAN, or GPIB. For communication with GPIB products NI-488.2 will also be required. Both NI-VISA and NI-488.2 are free and can be downloaded from the National Instruments Website.

**[NI-VISA Download - NI](#)**

**[NI-488.2 Download - NI](#)**

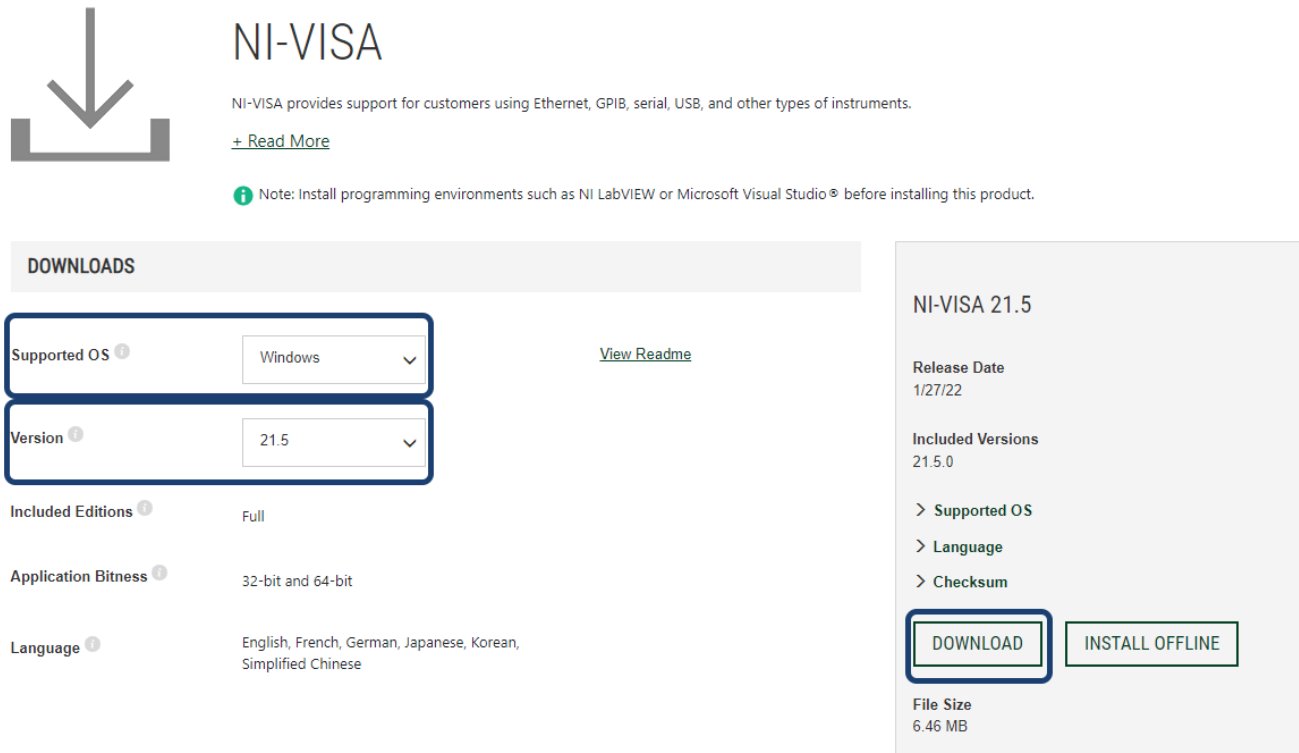
### Note:

Before installing NI-VISA verify that your operating system is compatible with the selected NI-VISA Version. National instruments provides compatibility lists for Microsoft Windows operating systems, OS, and Linux OS. You can find these tables **[NI-VISA and Operating System Compatibility - NI](#)**.

---

## 1.2 Installing NI VISA

After finding a compatible VISA version select the appropriate OS and Version. Then click the Download button to download the ni-visa\_<XX.X>\_online.exe file.



NI-VISA

NI-VISA provides support for customers using Ethernet, GPIB, serial, USB, and other types of instruments.

[+ Read More](#)

**Note:** Install programming environments such as NI LabVIEW or Microsoft Visual Studio® before installing this product.

### DOWNLOADS

Supported OS <sup>?</sup> Windows <sup>?</sup>

Version <sup>?</sup> 21.5 <sup>?</sup>

[View Readme](#)

Included Editions <sup>?</sup> Full

Application Bitness <sup>?</sup> 32-bit and 64-bit

Language <sup>?</sup> English, French, German, Japanese, Korean, Simplified Chinese

### NI-VISA 21.5

Release Date  
1/27/22

Included Versions  
21.5.0

> Supported OS

> Language

> Checksum

**DOWNLOAD** **INSTALL OFFLINE**

File Size  
6.46 MB

**Figure 1.1** NI VISA Download

Once the download is complete, run the executable to launch the NI Package Manager. The NI Package Manager is an access hub to download, upgrade, and manage all National Instruments' software.

The Package Manager will display the **Select** tab. In this tab you can select the items you wish to install. The **NI-VISA .NET Development Support** item contains the required libraries to connect NI-VISA with the chosen programming language.

Not all options are required to establish communication with the PC. However, it is recommended that all items be selected. Items such as **NI I/O Trace** and **NI Measurement & Automation Explorer** provide unique software that facilitate creating connection with instruments as well as troubleshooting.

Items such as the **NI-VISA C Examples** and **PXI Platforms Services** are not required but can be. PXI platform is not required since BK Precision currently does not provide any items that support this platform. One of the

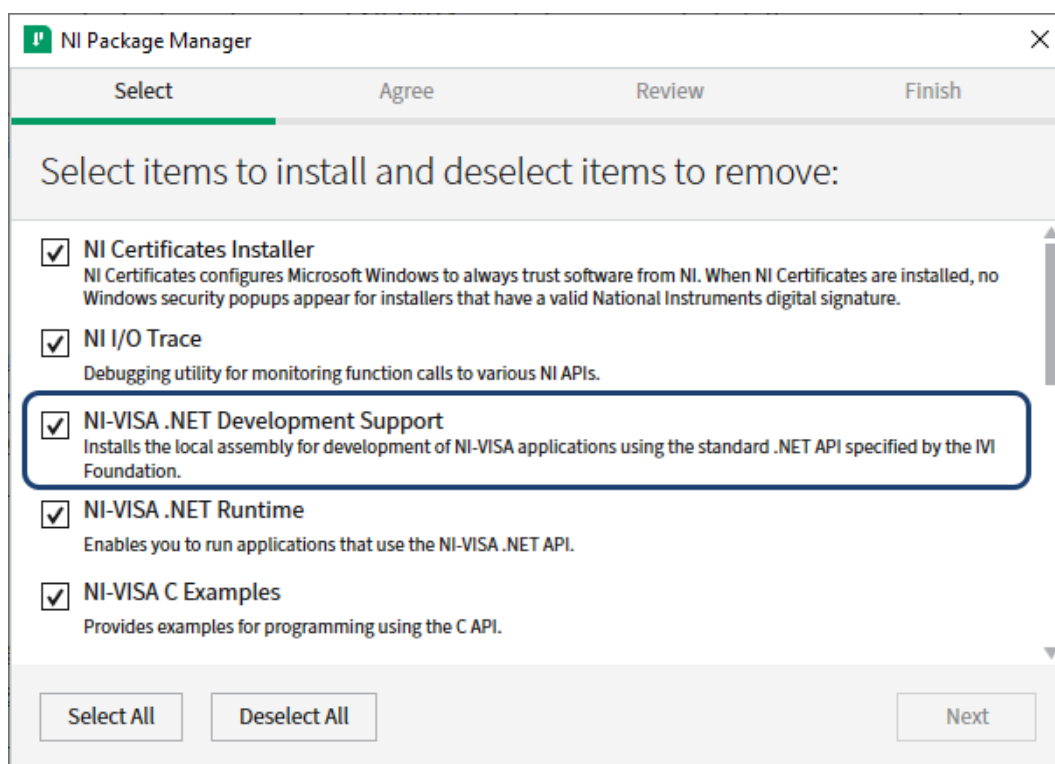


Figure 1.2 NI Package Manager

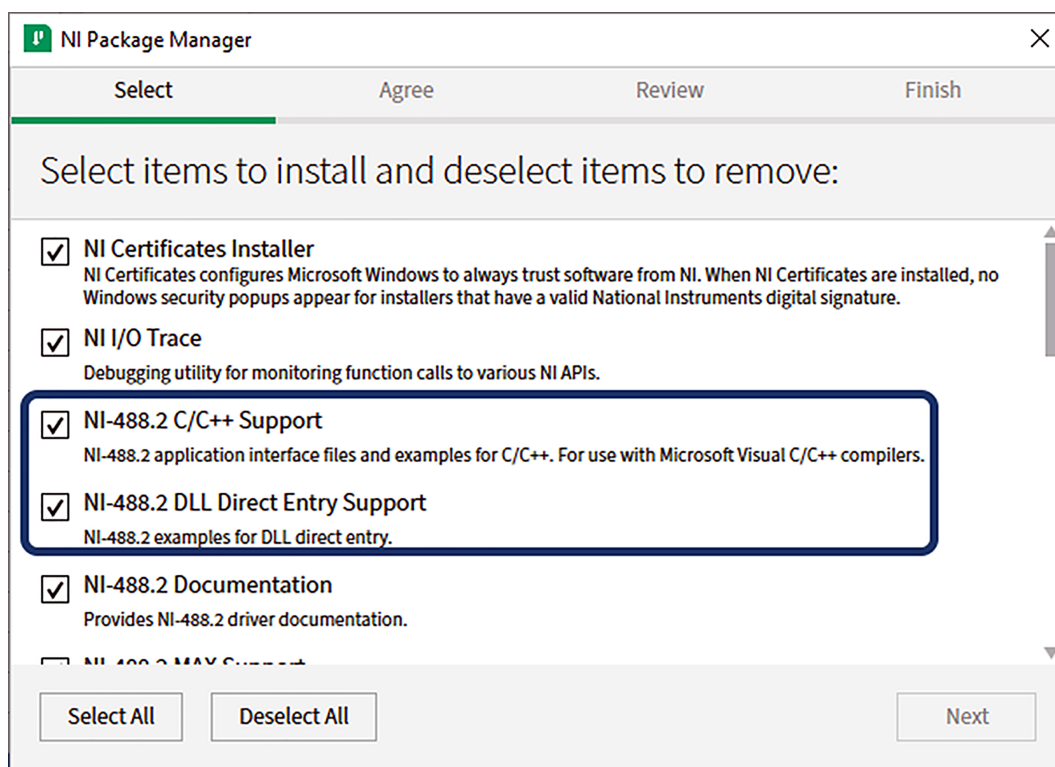


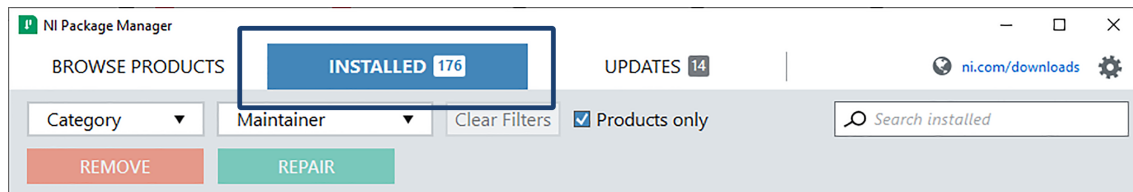
Figure 1.3 NI 488.2 Package Manager

Proceed with the installation. After installing the selected items restart the computer.

### 1.2.1 Installing .NET Development Support if NI-VISA is already installed

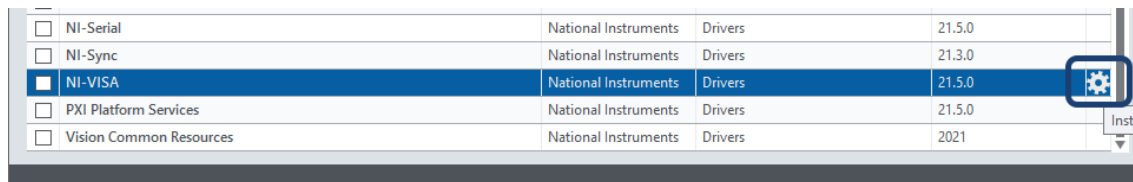
If the NI-VISA driver was previously installed, and NI-VISA.NET Development Support was not installed follow the step below to install it.

1. Open **NI Package Manager**
2. Select the **INSTALLED** tab.



**Figure 1.4** Package Manager INSTALLED

3. Search for **NI-VISA** and hover your mouse over the right most section.
  - A settings emblem should appear. (**install or remove related packages**)



**Figure 1.5** NI-VISA Settings Emblem

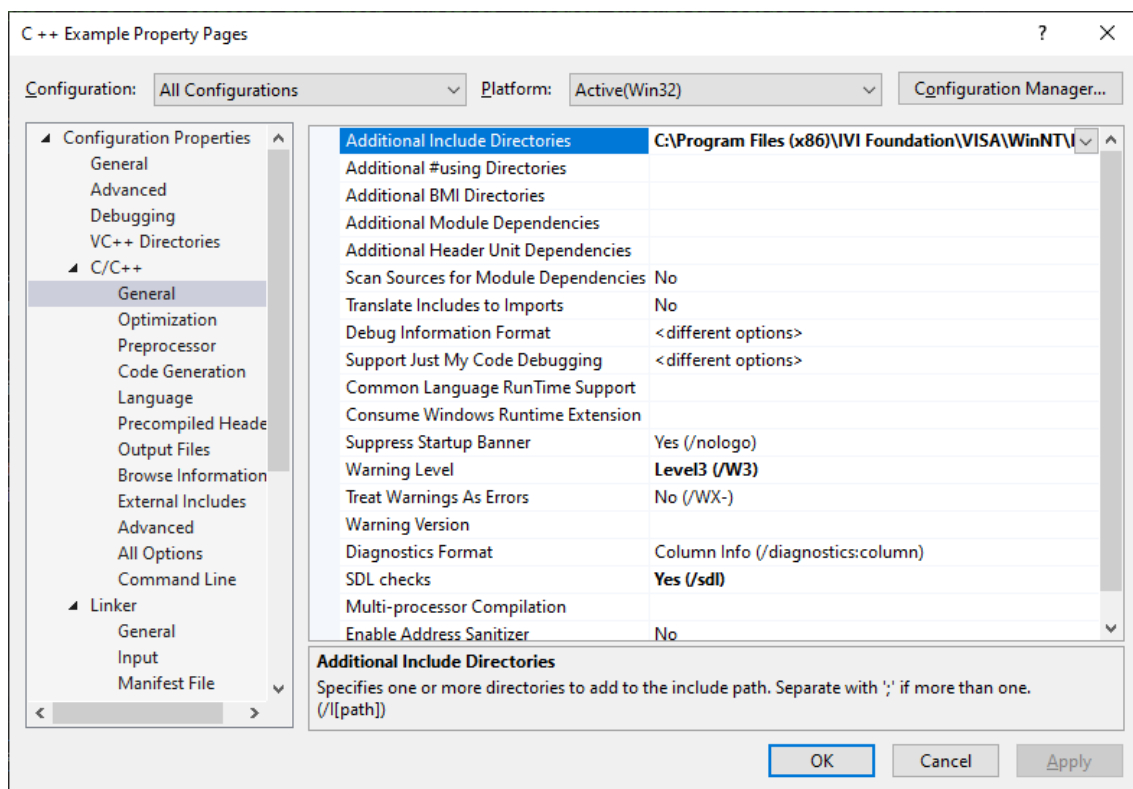
4. The NI Package Manager tab shown in image 1.2 will be displayed.
  - Select **NI-VISA .NET Development Support** and **NI-VISA .NET Runtime** to install the required packages.

# C++

## 2.1 Include NI VISA in C++

Unlike C# and python, C++ requires some extra configuration. Follow the instructions below to add the necessary references to your project.

1. Open/ create the C++ project in Visual Studio.
2. From the menu at the top of the screen, select **Project > Properties**.
3. Click the C/C++ > General > Additional Include Directories.
4. Add the following to the "**Additional Include Directories**".
  - If you are using x86 CPU architecture enter:  
**C:\Program Files (x86)\IVI Foundation\VISA\WinNT\Include**
  - If you are using x64 CPU architecture enter:  
**C:\Program Files \IVI Foundation\VISA\WinNT\Include**



**Figure 2.1** Additional Include Directories

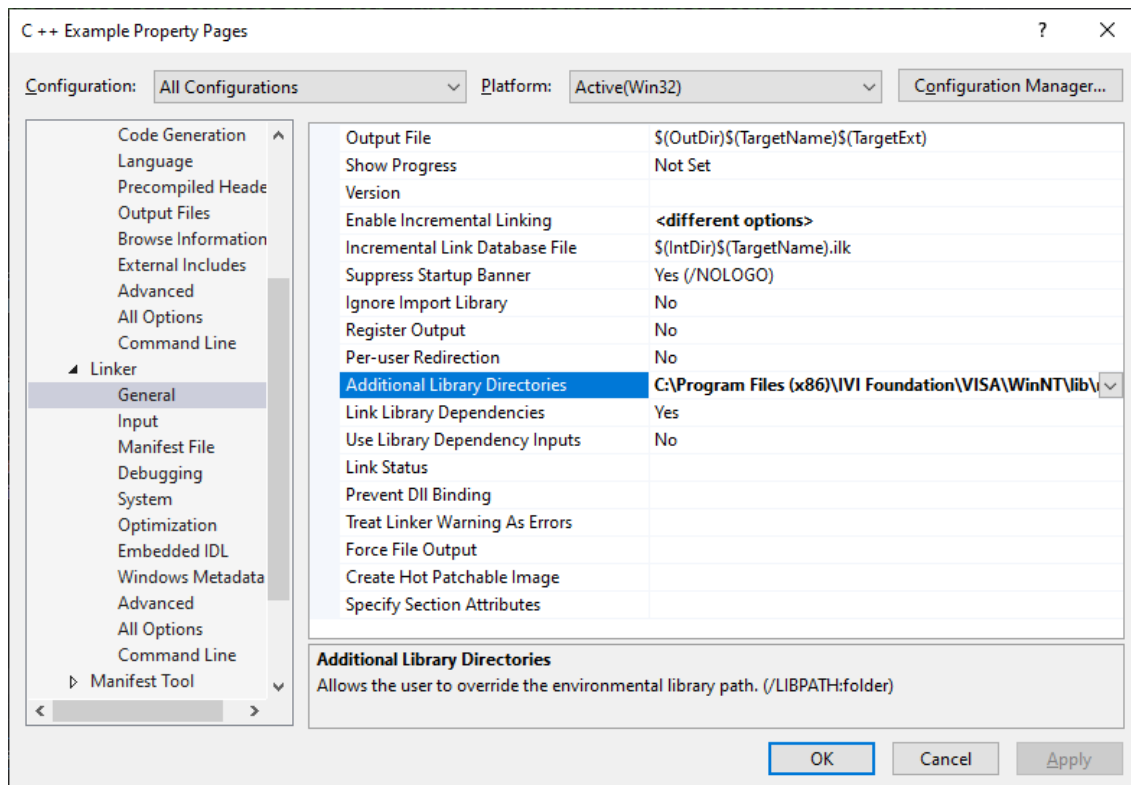
5. Click the **Linker > General > Additional Library Directories**

- If you are using x86 CPU architecture enter:

**C:\Program Files (x86)\IVI Foundation\VISA\WinNT\lib\msc**

- If you are using x64 CPU architecture enter:

**C:\Program Files\IVI Foundation\VISA\WinNT\lib\msc**



**Figure 2.2** Additional Library Directories



6. Add visa32.lib to the **Additional Options**.

- Click the **Linker > Command Line > Additional Options**.

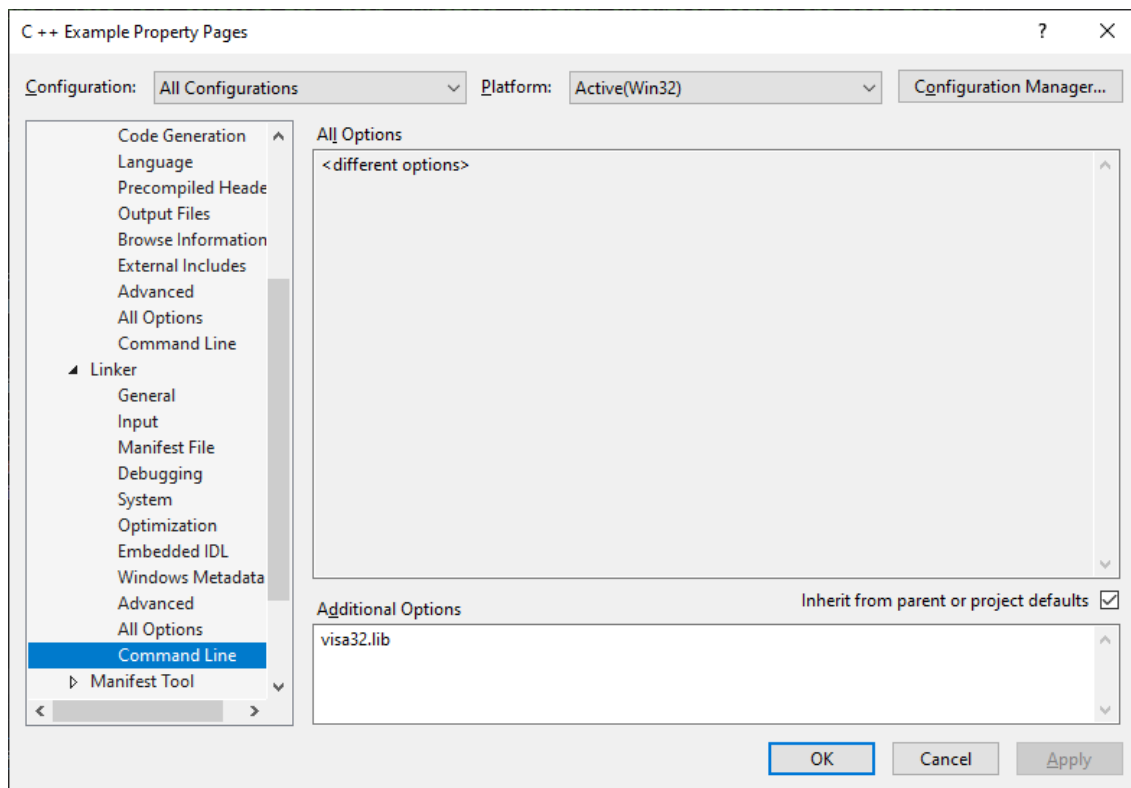


Figure 2.3 Command Line

**visa.h** should now appear in your **External Dependencies**.

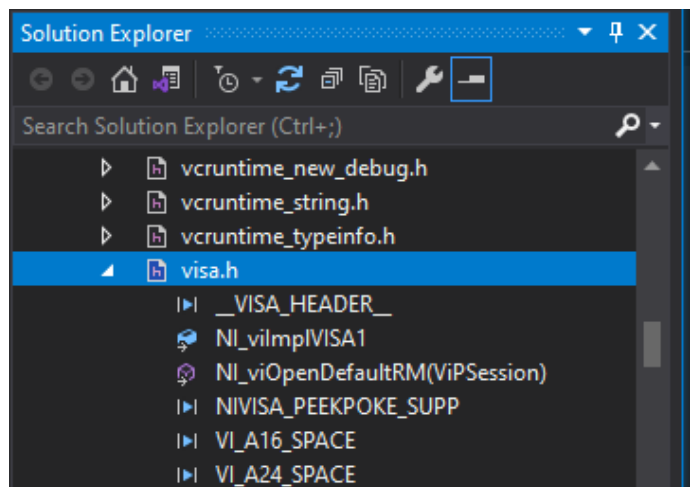


Figure 2.4 External Dependencies

### **2.1.1 Headers**

---

Once the necessary references have been added to the project the next step is to include the header files. The "include" keyword specifies the header file being included in the project.

Add the following statements to the header of the code:

```
#include <stdio.h>
```

```
#include <tchar.h>
```

```
#include <string.h>
```

```
#include <stdlib.h>
```

```
#include <Windows.h>
```

```
#include "visa.h"
```

```
#include "xlsxwriter.h"
```

## 2.2 Initializing Your Instrument

This section describes the steps required to prepare your application for communication with your device. VISA's capability of accessing devices on various interfaces allows user to reuse code regardless of the interface used for communication. This is accomplished by the VISA Resource Manager, which locates resources, creates sessions, and assigns unique resource addresses.

When trying to access any of the VISA resources, the first step is to get a reference to the default Resource Manager by calling **viOpenDefaultRM()**. The application can then use the session returned to open sessions to resources controlled by that Resource Manager.

```
status = viOpenDefaultRM(&defaultRM);  
if (status < VI_SUCCESS) {  
    /* Error initializing VISA ... exiting */  
    return -1;  
}
```

The **viFindRsrc()** function is then called to locate the resources associated with a specified interface. **Table 2.1** list the parameters and the interface they specify.

```
status = viFindRsrc(defaultRM, "?*", &fList, &numInstrs, desc);  
strcpy_s(deviceIDs[0], 256, desc);  
if (status < VI_SUCCESS) {  
    viStatusDesc(defaultRM, status, errorText);  
    printf(errorText);  
    return -1;  
}
```

The **viFindNext()** function is then called to return the next resource from the list of resources found during the previous call to **viFindRsrc()**.

```
viFindNext(fList, desc);
```

The **Finding All Resources Example** implements the functions listed above to search and list all available resources. The user will then be able to select the index of the desired resource to open a session.

Regular Expression	Sample Matches
<code>GPIB?*INSTR</code>	Matches <code>GPIB0::2::INSTR</code> , <code>GPIB1::1::1::INSTR</code> , and <code>GPIB-VXI1::8::INSTR</code> .
<code>GPIB[0-9]*::*INSTR</code>	Matches <code>GPIB0::2::INSTR</code> and <code>GPIB1::1::1::INSTR</code> but not <code>GPIB-VXI1::8::INSTR</code> .
<code>GPIB[0]::*INSTR</code>	Matches <code>GPIB1::1::1::INSTR</code> but not <code>GPIB0::2::INSTR</code> or <code>GPIB12::8::INSTR</code> .
<code>VXI?*INSTR</code>	Matches <code>VXI0::1::INSTR</code> but not <code>GPIB-VXI0::1::INSTR</code> .
<code>GPIB-VXI?*INSTR</code>	Matches <code>GPIB-VXI0::1::INSTR</code> but not <code>VXI0::1::INSTR</code> .
<code>?*VXI[0-9]*::*INSTR</code>	Matches <code>VXI0::1::INSTR</code> and <code>GPIB-VXI0::1::INSTR</code> .
<code>ASRL[0-9]*::*INSTR</code>	Matches <code>ASRL1::INSTR</code> but not <code>VXI0::5::INSTR</code> .
<code>ASRL1+::INSTR</code>	Matches <code>ASRL1::INSTR</code> and <code>ASRL11::INSTR</code> but not <code>ASRL2::INSTR</code> .
<code>(GPIB VXI)?*INSTR</code>	Matches <code>GPIB1::5::INSTR</code> and <code>VXI0::3::INSTR</code> but not <code>ASRL2::INSTR</code> .
<code>(GPIB0 VXI0)::1::INSTR</code>	Matches <code>GPIB0::1::INSTR</code> and <code>VXI0::1::INSTR</code> .
<code>?*INSTR</code>	Matches all <code>INSTR</code> (device) resources.
<code>?*VXI[0-9]*::*MEMACC</code>	Matches <code>VXI0::MEMACC</code> and <code>GPIB-VXI1::MEMACC</code> .
<code>VXI0::?*</code>	Matches <code>VXI0::1::INSTR</code> , <code>VXI0::2::INSTR</code> , and <code>VXI0::MEMACC</code> .
<code>?*</code>	Matches all resources.
<code>visa://hostname/?*</code>	Matches all resources on the specified remote system. The hostname can be represented as either an IP address (dot-notation) or network machine name. This remote system need not be a configured remote system.
<code>/?*</code>	Matches all resources on the local machine. Configured remote systems are not queried.
<code>visa:/ASRL?*INSTR</code>	Matches all <code>ASRL</code> resources on the local machine and returns them in URL format (for example, <code>visa:/ASRL1::INSTR</code> ).
<code>USB?*</code>	Matches all <code>USBTMC</code> resources on the local machine
<code>TCP?*</code>	Matches all <code>TCP</code> resources on the local machine

Table 2.1 FindRsrc

## Finding All Resources Example

```
int main()
{
    ViStatus    status;                // Checks Errors
    ViSession   defaultRM, instr;      // Communication channels
    ViUInt32    retCount = 0;          // Return count from string
    const int   bufferSz = 256;        // I/O
    ViChar      buffer[bufferSz];      // States buffer size
    ViUInt32    numInstrs = 0;         // Buffer for string I/O
    ViFindList  fList = 0;             // Number of instruments
                                        // found
    ViChar      desc[VI_FIND_BUFLen];  // List of instruments found
    ViChar      errorText[256];        // Description of inst
    int         i, j, k, l;            // Return error
    char*       deviceIDs[256];        // counter integers
    int         maxChoice = 0;         // Used to create array of
    int         instNum = 999;         // max number of instruments
    ViPUInt32   retCnt = 0;            found
    char        cmd[128];              // index selected

    for ( i = 0; i < 256; i++) {
        deviceIDs[i] = (char*)malloc(256 * (sizeof(deviceIDs)));
    }
    status = viOpenDefaultRM(&defaultRM);
    if (status < VI_SUCCESS) {
        /* Error Initializing VISA...exiting*/
        return -1;
    }
}
```

```
status = viFindRsrc(defaultRM, "?*", &fList, &numInstrs, desc);
strcpy_s(deviceIDs[0], 256, desc);
if (status < VI_SUCCESS) {
    viStatusDesc(defaultRM, status, errorText);
    printf(errorText);
    printf("viFindRsrc: %d", status);
    return -1;
}
printf("Found %d instruments\n\n", numInstrs);
/*Finds all available resources*/
for (int i= numInstrs - 1; i> 0; i--) {
    viFindNext(fList, desc);
    strcpy_s(deviceIDs[i], 256, desc);
    maxChoice++;
}
/*List all resources found*/
for (i= 0; i< numInstrs; i+ +) {
    printf("inst %d: %s\n\n", i, deviceIDs[i]);
}
printf("Select an instrument (enter 999 to quit) : ");
scanf_s("%d", &instNum);
if (0 <= instNum && instNum <= maxChoice) {
}
else if (instNum == 999) {
    printf("Quitting\n");
    return 0;
}
else {
    printf("Invalid choice! Please select one of the available resources");
    return 1;
}
```

```
}  
printf("Device String: %s\n", deviceIDs[instNum]);  
viOpen(defaultRM, deviceIDs[instNum], VI_NULL, VI_NULL, &instr);  
if (status < VI_SUCCESS) {  
    viStatusDesc(defaultRM, status, errorText);  
    printf(errorText);  
    printf("viFindRsrc: %d", status);  
    return -1;  
}  
printf("Device connection opened\n");  
status = viSetAttribute(instr, VI_ATTR_TERMCHAR, (ViChar)'\n');  
status = viSetAttribute(instr, VI_ATTR_TERMCHAR_EN, VI_TRUE);  
/* Send the commands to the instrument*/  
}
```

## 2.3 Event Handling

This section describes the VISA event model and the various events VISA supports.

An event is a means of communication between a VISA resource and its applications. Events occur because of a condition requiring the attention of applications.

There are two ways for an application to receive event notifications:

### 2.3.1 Queuing Mechanism

This method places all occurrences of the specified event in a queue. This method is recommended for noncritical events that do not need immediate servicing.

The event queuing process requires that you first enable the session to sense the particular event type. When enabled, the session can automatically queue the event occurrences as they happen. A session can later dequeue these events using the **viWaitOnEvent()** operation. The timeout to **VI\_TMO\_IMMEDIATE** if you want your application to check if any event of the specified event type exists in the queue.

The event queues in VISA do not dynamically grow as new events arrive. The default queue length is 50, but you can change the size of a queue by using the **VI\_ATTR\_MAX\_QUEUE\_LENGTH** template attribute.

### 2.3.2 Callback Handler

The Callback Handler invokes a function that the program specifies prior to enabling the event. This method is recommended for applications that require an immediate response.

### 2.3.3 Enabling Events

Use the **viEnableEvent()** operation to enable an event type using either of the methods mentioned before. For example, to enable the **VI\_EVENT\_SERVICE\_REQ** event for queuing, use the following code:

```
status = viEnableEvent(instr,VI_EVENT_SERVICE_REQ,VI_QUEUE,VI_NULL);
```

Use **viDisableEvent()** to stop a session from receiving events of a specified type. For example, to disable the **VI\_EVENT\_SERVICE\_REQ** event regardless of the mechanism for which it was enabled, use the following code:

```
status = viDisableEvent(instr,VI_EVENT_SERVICE_REQ,VI_ALL_MECH);
```

**Table 2.2** list all available event types.



Event Type	Description	Notes
VI_EVENT_IO_COMPLETION	Notification that an asynchronous I/O operation has completed.	The I/O Completion event applies to all asynchronous operations, which for INSTR includes viReadAsync(), viWriteAsync(), and viMoveAsync(). For resource classes that do not support asynchronous operations, this event type is not applicable.
VI_EVENT_EXCEPTION	Notification that an error condition (exception) has occurred during an operation invocation.	The exception event supports only the callback model. Refer to the Exception Handling section at the end of this chapter for more information about this event type.
VI_EVENT_SERVICE_REQ	Notification of a service request (SRQ) from the device.	Supported for message based INSTR classes, including GPIB, VXI, GPIB-VXI, and TCPIP.
VI_EVENT_VXI_SIGP	Notification of a VXIbus signal or VXIbus interrupt from the device.	Supported for VXI INSTR only.
VI_EVENT_VXI_VME_INTR	Notification of a VXIbus interrupt from the device.	Supported for VXI INSTR only. This applies to both VXI and VME devices.
VI_EVENT_TRIG	Notification of a VXIbus trigger.	Supported for VXI INSTR and VXI BACKPLANE only.
VI_EVENT_PXI_INTR	Notification of a PCI/PXI interrupt from the device.	Supported for PXI INSTR only. Not supported on all platforms.
VI_EVENT_ASRL_BREAK	Notification that a break signal was received.	Supported for Serial INSTR only. Not supported on all platforms.
VI_EVENT_ASRL_CTS	Notification that the Clear To Send (CTS) line changed state.	Supported for Serial INSTR only. Not supported on all platforms. If the CTS line changes state quickly several times in succession, not all line state changes will necessarily result in event notifications.
VI_EVENT_ASRL_DCD	Notification that the Data Carrier Detect (DCD) line changed state.	Supported for Serial INSTR only. Not supported on all platforms. If the DCD line changes state quickly several times in succession, not all line state changes will necessarily result in event notifications.

Table 2.2 Event Types

Event Type	Description	Notes
VI_EVENT_ASRL_DSR	Notification that the Data Set Ready (DSR) line changed state.	Supported for Serial INSTR only. Not supported on all platforms. If the DSR line changes state quickly several times in succession, not all line state changes will necessarily result in event notifications.
VI_EVENT_ASRL_RI	Notification that the Ring Indicator (RI) input signal was asserted.	Supported for Serial INSTR only. Not supported on all platforms.
VI_EVENT_ASRL_CHAR	Notification that at least one data byte has been received.	Supported for Serial INSTR only. Not supported on all platforms. Each data character will not necessarily result in an event notification.
VI_EVENT_ASRL_TERMCHAR	Notification that the termination character has been received.	Supported for Serial INSTR only. Not supported on all platforms. The actual termination character is specified by setting VI_ATTR_TERMCHAR prior to enabling this event. For this event, the setting of VI_ATTR_TERMCHAR_EN is ignored.

Table 2.3 Event Types Continued

## Event Handling Example

The example below demonstrates a function invoked when an SRT event occurs, indicating that data is ready to be read.

```
ViStatus _VI_FUNCH myCallback(ViSession vi, ViEventType etype, ViEvent eventContext, ViAddr userHandle)
```

```
{
    ViJobId jobId;
    ViStatus status;
    ViUInt16 stb;
    status = viReadSTB(vi, &stb);
    status = viReadAsync(vi, (ViBuf)userHandle, MAX_CNT, &jobId);
```

```
    return VI_SUCCESS;
}
int main(void)
{
    ViStatus status;
    ViSession defaultRM, instr;
    ViBuf bufferHandle;
    ViUInt32 retCount;
    ViEventType etype;
    ViEvent eventContext;

    /* Initialize instrument as shown in section ?? the system */
    /* Allocate memory for buffer */
    /* In addition, allocate space for the ASCII NULL character */
    bufferHandle = (ViBuf)malloc(MAX_CNT + 1);
    /* Tell the driver what function to call on an event */
    status = viInstallHandler(instr, VI_EVENT_SERVICE_REQ, myCallback,bufferHandle);
    /* Enable the driver to detect events */
    status = viEnableEvent(instr, VI_EVENT_SERVICE_REQ, VI_HNDLR,VI_NULL);
    status = viEnableEvent(instr, VI_EVENT_IO_COMPLETION, VI_QUEUE,VI_NULL);
    /* Tell the device to begin acquiring data */
    status = viWrite(instr, "FETC?", 9, &retCount);
    /* The device asserts SRQ when the data is ready */
    /* The callback begins reading the data */
    /* After the data is read, an I/O completion event occurs */
    status = viWaitOnEvent(instr, VI_EVENT_IO_COMPLETION, 20000,&etype, &eventContext);
    if (status < VI_SUCCESS) {
        /* data not received...exiting */
        free(bufferHandle);
        viClose(defaultRM);
        return -1;
    }
}
```

```
/* Your code should process the data */  
/* Close the event context */  
viClose(eventContext);  
/* Stop listening for events */  
status = viDisableEvent(instr, VI_ALL_ENABLED_EVENTS, VI_ALL_MECH);  
status = viUninstallHandler(instr, VI_EVENT_SERVICE_REQ, myCallback, bufferHandle);  
/* Close down the system */  
free(bufferHandle);  
status = viClose(instr);  
status = viClose(defaultRM);  
return 0;  
}
```

## 2.4 Data Logging

Queried data will be written to an excel sheet. To accomplish this task the "**xlsxwriter.h**" library will be used. This library will allow us to create excel files, edit the file's layout/settings, and most importantly write and read to and from the file.

The example below creates a workbook and a worksheet. Once the worksheet is created we can write the returned data by specifying the row and column.

```
/* Create a new workbook and add a worksheet. */  
  
lxw_workbook* workbook = workbook_new("Log.xlsx");  
  
lxw_worksheet* worksheet = workbook_add_worksheet(workbook, NULL);  
  
/* Add a format. */  
  
lxw_format* format = workbook_add_format(workbook);  
  
/* Set the bold property for the format */  
  
format_set_bold(format);  
  
/* Change the column width for clarity. */  
  
worksheet_set_column(worksheet, 0, 0, 20, NULL);  
  
/* Write some simple text. */  
  
worksheet_write_string(worksheet, 0, 0, "Date", NULL);  
  
/* Text with formatting. */  
  
bclear(buffer, bufferSz);  
  
sprintf_s(cmd, "*IDN?\n");  
  
status = viQueryf(instr, (ViConstString)cmd, "%64c", buffer);  
  
if (status < VI_SUCCESS) {  
    viStatusDesc(defaultRM, status, errorText);  
  
    printf("%x:%s\n", status, errorText);  
  
    return -1; }  
  
worksheet_write_string(worksheet, 1, 1, buffer, format);  
  
/* Write some numbers. */  
  
worksheet_write_number(worksheet, 2, 0, 123, NULL);  
  
worksheet_write_number(worksheet, 3, 0, 123.456, NULL);  
  
workbook_close(workbook); }
```

# C#

## 3.1 Using NI VISA in C#

To incorporate the NI VISA commands add the following references to the project:

- National Instruments VISA
- National Instruments.Common
- VISA COM <VERSION> Type Library

To add the required references:

1. Right click **References** located in the **Solution Explorer**.

- Click the **Add References** on the pop-up window.

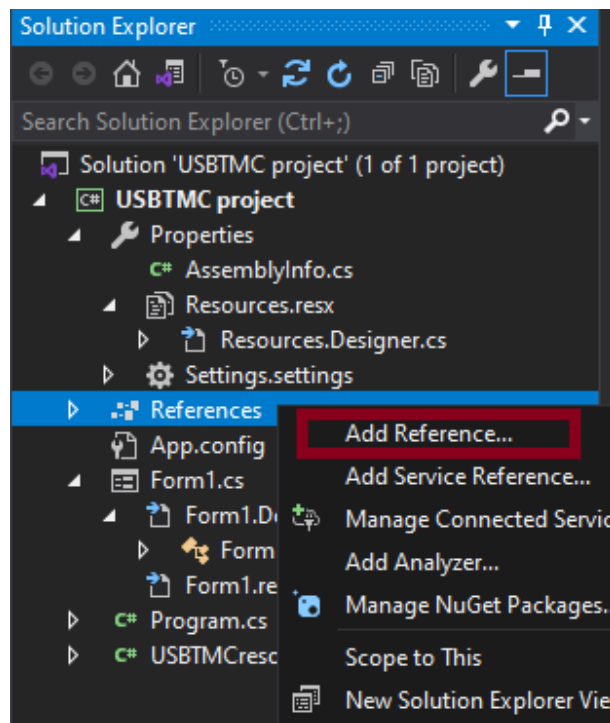
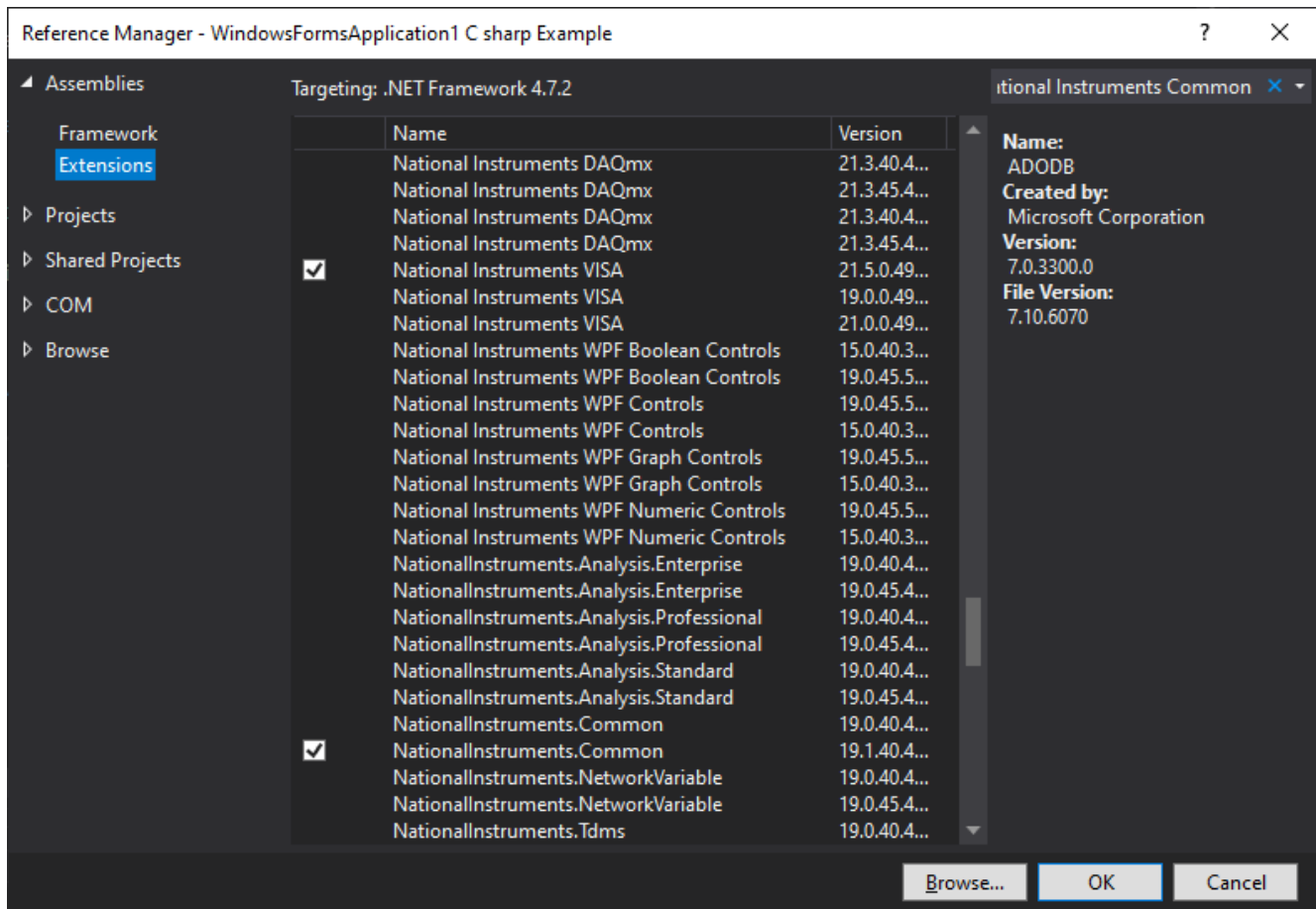


Figure 3.1 Add References

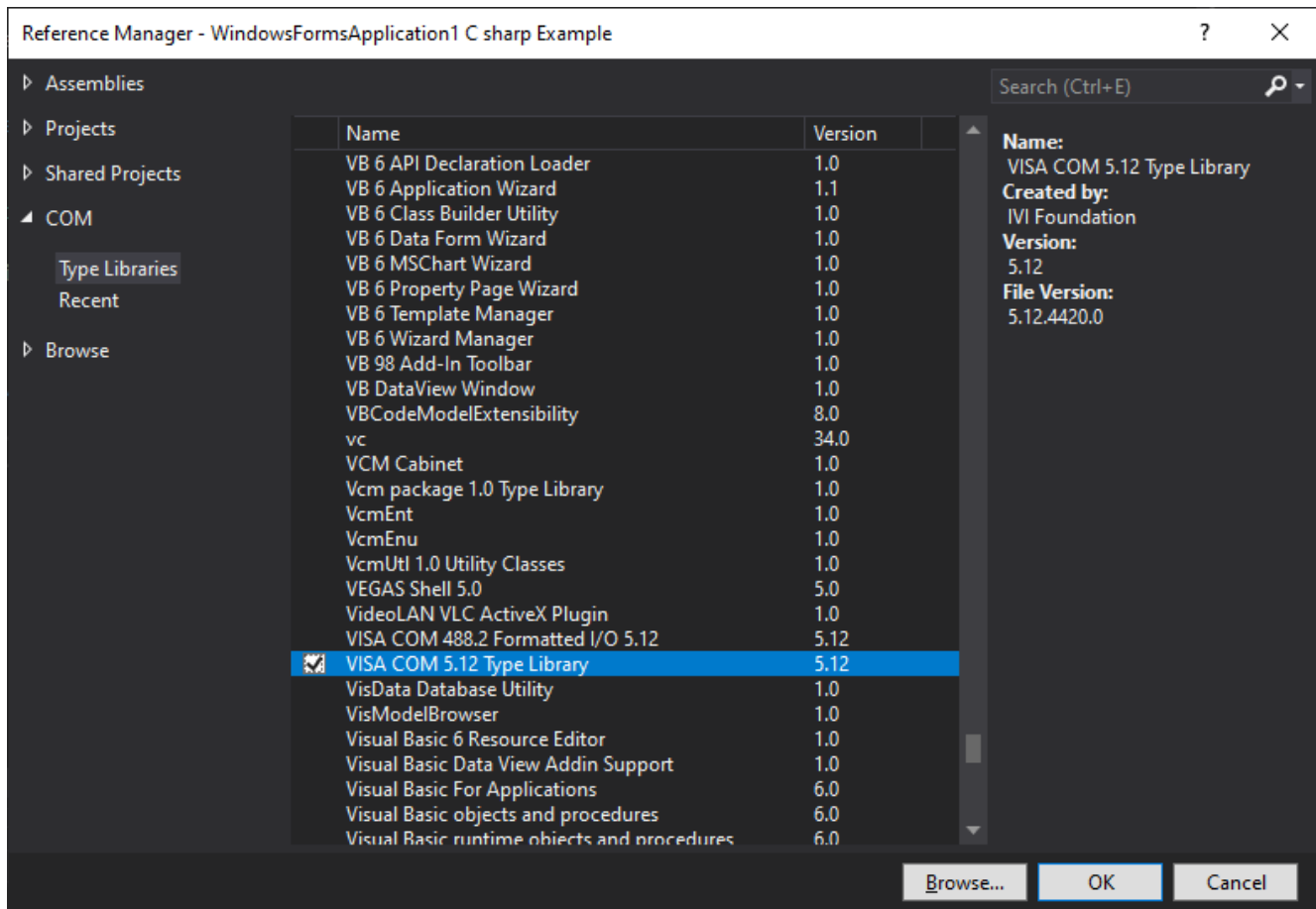
2. The Reference Manager window will open.

- Click the **Assemblies** dropdown menu.
- Click **Externsions** and check both **National Instruments VISA** and **National Instruments.Common**.



**Figure 3.2** Reference Manager Extensions

3. In the **Reference Manager** window click the **COM** dropdown and select **Type Libraries**.
  - Check the **VISA COM <version> Type Library**.
4. Click **OK** to apply the changes.



**Figure 3.3** Reference Manager Type Libraries

5. In the **Solution Explorer** click VisaComLib.
  - The **Properties** window will appear below the **Solution Explorer**.
  - Set the **Embed Interop Types** to False.



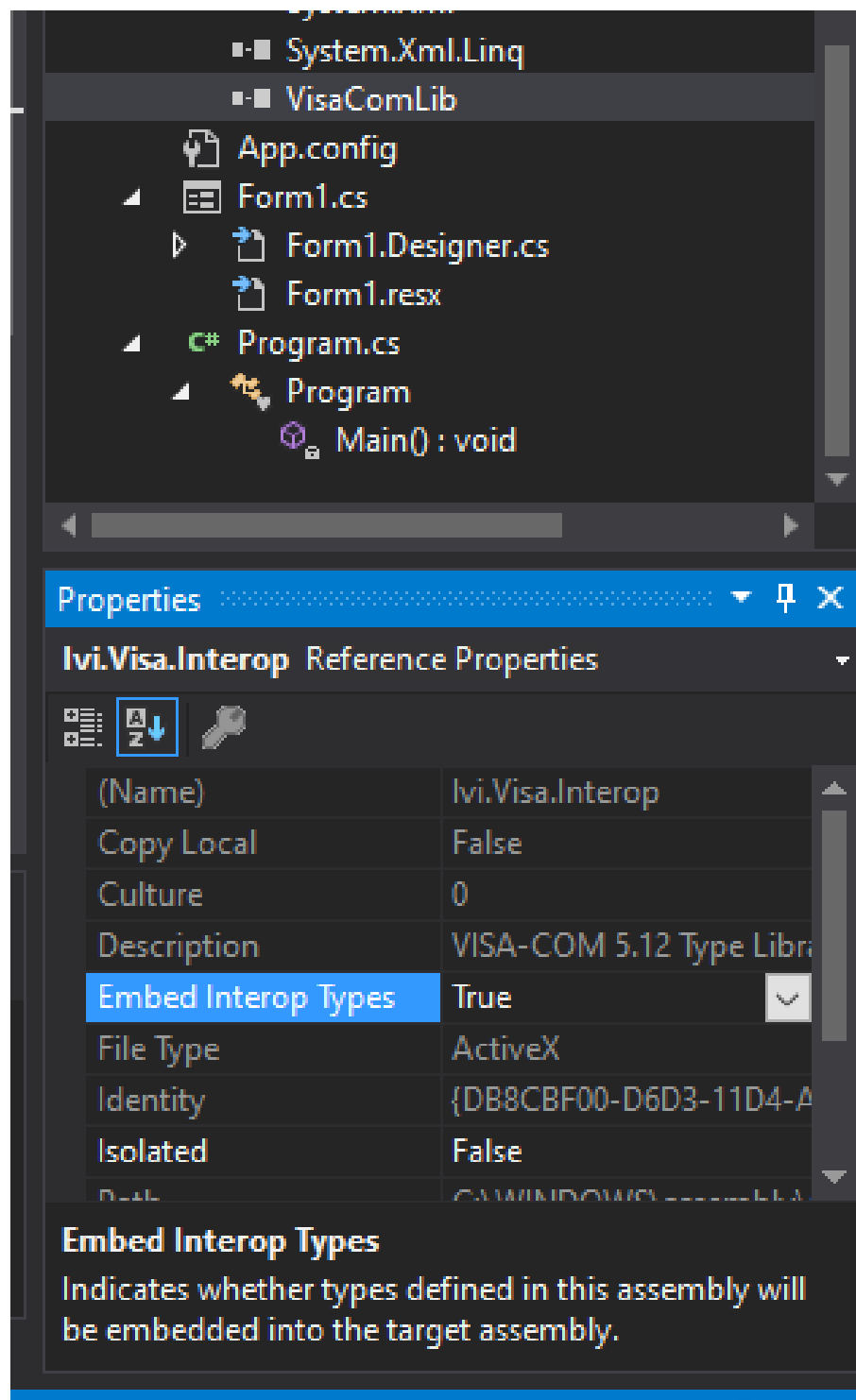


Figure 3.4 VisaComLib Properties

### **3.1.1 Headers**

---

Once the references have been added to the project the next step is to include the object namespaces in the program. The "using" keyword specifies that names are using by the program in the given namespace.

This keyword provides access to related classes and methods to use in .NET applications. The "using" keyword allows using the class without having to define the namespace.

Add the following statements to the header of the code:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using Ivi.Visa.Interop;  
using NationalInstruments.Visa;  
using System.IO;
```

## 3.2 Initialization of an Instrument

In order to initialize any product a resource manager must first be created. To create a resource manager use the following statements.

```
Ivi.Visa.Interop.ResourceMnager rMgr = new ResourceManagerClass();  
FormattedIO488 src = new Formatted IO 488Class();
```

Once the resource manager is created we can use the VISA statement FindRsrc() to find the desired resources. For this example we want to view all available resources, therefore the **FindRsc(?)** statement is used.

```
public void getAvailableResources()  
{  
    try  
    {  
        string[] resources = rMgr.FindRsrc("?*");  
        comboBox1.Items.AddRange(resources);  
    }  
    catch (Exception)  
    {  
        textBox2.Text = "No Resources Available";  
    }  
}
```

The `getAvailableResources()` function stores all the available function in the resources array. The array is then assigned to be displayed in comboBox1. If no resources are available the message No Resoruces Available is displayed in textBox2.

To filter out specific interfaces modify the parameters of the FindRsrc command. Refer to table [3.1](#) to view the available parameters.

Regular Expression	Sample Matches
<code>GPIB?*INSTR</code>	Matches GPIB0::2::INSTR, GPIB1::1::1::INSTR, and GPIB-VXI1::8::INSTR.
<code>GPIB[0-9]*::*INSTR</code>	Matches GPIB0::2::INSTR and GPIB1::1::1::INSTR but not GPIB-VXI1::8::INSTR.
<code>GPIB[0]::*INSTR</code>	Matches GPIB1::1::1::INSTR but not GPIB0::2::INSTR or GPIB12::8::INSTR.
<code>VXI?*INSTR</code>	Matches VXI0::1::INSTR but not GPIB-VXI0::1::INSTR.
<code>GPIB-VXI?*INSTR</code>	Matches GPIB-VXI0::1::INSTR but not VXI0::1::INSTR.
<code>?*VXI[0-9]*::*INSTR</code>	Matches VXI0::1::INSTR and GPIB-VXI0::1::INSTR.
<code>ASRL[0-9]*::*INSTR</code>	Matches ASRL1::INSTR but not VXI0::5::INSTR.
<code>ASRL1+::INSTR</code>	Matches ASRL1::INSTR and ASRL11::INSTR but not ASRL2::INSTR.
<code>(GPIB VXI)?*INSTR</code>	Matches GPIB1::5::INSTR and VXI0::3::INSTR but not ASRL2::INSTR.
<code>(GPIB0 VXI0)::1::INSTR</code>	Matches GPIB0::1::INSTR and VXI0::1::INSTR.
<code>?*INSTR</code>	Matches all INSTR (device) resources.
<code>?*VXI[0-9]*::*MEMACC</code>	Matches VXI0::MEMACC and GPIB-VXI1::MEMACC.
<code>VXI0::*?</code>	Matches VXI0::1::INSTR, VXI0::2::INSTR, and VXI0::MEMACC.
<code>?*</code>	Matches all resources.
<code>visa://hostname/*?</code>	Matches all resources on the specified remote system. The hostname can be represented as either an IP address (dot-notation) or network machine name. This remote system need not be a configured remote system.
<code>//*?</code>	Matches all resources on the local machine. Configured remote systems are not queried.
<code>visa:/ASRL?*INSTR</code>	Matches all ASRL resources on the local machine and returns them in URL format (for example, visa:/ASRL1::INSTR).
<code>USB?*?</code>	Matches all USBTMC resources on the local machine
<code>TCP?*?</code>	Matches all TCP resources on the local machine

Table 3.1 FindRsrc

### 3.2.1 Opening A Session

After creating the resource manager and finding all available resources, the next step is to open a VISA session with the desired resource.

In order to open the session the user must first select the index in the `resources` array of the desired resource. The function below assigns the address of the selected resource to `label1`, and then enables the **OPEN** (button3) and **CLOSE** (button4) session buttons.

```
public void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    label1.Text = comboBox1.Text;
    button3.Enabled = true;
    button4.Enabled = true;
}
```

With the resource address assigned and the **OPEN** button being enabled the user can now click the **OPEN** button to open the VISA session. Clicking the **OPEN** button will call the following function in the example provided.

```
public void button3_Click(object sender, EventArgs e)
{
    string srcAddress = label1.Text;

    src.IO = (IMessage)rmMgr.Open(srcAddress, AccessMode.NO_LOCK, 2000, "");
    src.IO.Timeout = 10000;

    comboBox1.Enabled = false;
    groupBox1.Enabled = true;
    groupBox2.Enabled = true;
    groupBox3.Enabled = true;
}
```

The `button3_Click` function opens the VISA session for the selected resource. A time out of 10 seconds is set and the remaining buttons and dialog boxes are enabled. The resources combobox is disabled once the session is opened.

### 3.3 Establishing Communication

To ensure communication has been established with the instrument we can query the command **\*IDN?**. This query returns a string that uniquely identifies the instrument. If the proper string is returned we can conclude the session was open successfully.

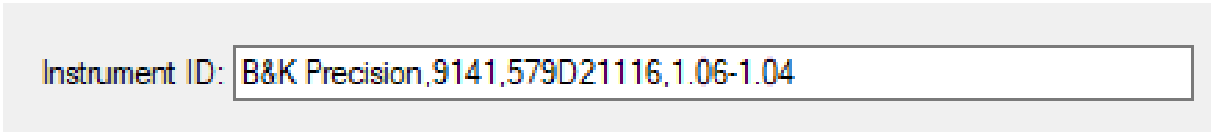
The example provided includes the **\*IDN?** query in the **button3\_Click**, immediately after opening the session. The query involves two parts:

1. Writing the **\*IDN?** command to the instruments.
2. Reading the instrument's buffer.

This is accomplished in the example below.

```
src.WriteString("*IDN?\n");  
try  
{  
    textBox8.Text += src.ReadString() + "\r\n";  
}  
catch ( TimeoutException)  
{  
    textBox2.Text = "timeout exception";  
}
```

The example provided will display the returned string in the bottom leftside of the window. See image 3.5

A screenshot of a software window with a light gray background. On the left, the text "Instrument ID:" is displayed in a dark blue font. To its right is a rectangular text box with a thin black border. Inside the text box, the string "B&K Precision,9141,579D21116,1.06-1.04" is displayed in a monospaced font, with each character in a different color (multi-colored text).

Instrument ID: B&K Precision,9141,579D21116,1.06-1.04

**Figure 3.5** \*IDN? Query

If an issue occurs the message **"timeout exception"** will be displayed. If this occurs use the NI I/O software that was installed along with NI VISA to troubleshoot.

---

## 3.4 Event Handling

---

This section describes the VISA event model and the various events VISA supports.

An event is a means of communication between a VISA resource and its applications. Events occur because of a condition requiring the attention of applications.

There are two ways for an application to receive event notifications.

### 3.4.1 Queuing Mechanism

---

This method places all occurrences of the specified event in a queue. This method is recommended for noncritical events that do not need immediate servicing.

### 3.4.2 Callback Handler

---

The Callback Handler invokes a function that the program specifies prior to enabling the event. This method is recommended for applications that require an immediate response.

### 3.4.3 Event Handling Example

---

When implementing an event handler the first step is to specify the type of event we want to be notified of as well as the mechanism we are implementing.

For this example we will be implementing a service request event. A service request can be generated when an operation is complete.

```
MessageBasedSession src = (MessageBasedSession)rm;
```

```
MessageBasedSessionEventType srq = MessageBasedSessionEventType.ServiceRequest;
```

The queue mechanism will be used for this example

```
src.EnableEvent(srq, EventMechanism.Queue);
```

For events that require immediate action creating a function that is called by the VISA library is useful. When creating a handler function, it needs to have a specific signature to be used by VISA. The expected signature is (session, event\_type, event\_context, user\_handle).

```
private static void DataReceivedHandler(object sender,
    MessageBasedSessionEventArgs e)
{
    string indata = src.ReadExisting();
    Console.WriteLine("Service Request Received!");
}
```

The example below gathers all the function listed above and queries multiple measurements from a DMM.

```
MessageBasedSession src = (MessageBasedSession)rm;
MessageBasedSessionEventType srq = MessageBasedSessionEventType.ServiceRequest;
src.EnableEvent(srq, EventMechanism.Queue);
int timeout = 10000;
// Set sample size
src.Write(":SAMP:COUN 999999");
// INIT trigger
src.Write(":SAMP:COUN 999999");
// This waits for the Service Request
src.WaitOnEvent(srq, timeout);
string data = src.Query(":FETC?");
src.Dispose();
```



## 3.5 Data Logging

Queried data will be written to an excel sheet in order to save all queried measurements. To accomplish this task the **Microsoft.Office.Interop.Excel** package will be used. This package will allow us to create excel files, edit the file's layout/settings, and most importantly write and read to and from the excel files. Before creating the excel file, a couple of variables are created in order to store values called through the code.

```
public string FilePath{ get; }
```

```
public string ID;
```

The variables defined above are used to specify the directory where the excel sheet will be save, and to store the identification string of the instrument being queried respectively.

### 3.5.1 Creating Excel File

To create an exeel file the function below will be used.

```
public void CreateExcel()
```

```
{
```

```
    this.StartButton.Text = "Stop Logging";
```

```
    DateTime date1 = new DateTime();
```

```
    DateTime dateOnly = date1.Date;
```

```
    DateTime timeOnly = date1.ToLocalTime();
```

```
    string dateOnly1 = dateOnly.ToString("d");
```

```
    string FilePath = textBox7.Text;
```

```
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
```

```
    excel.Visible = false;
```

```
    excel.DisplayAlerts = false;
```

```
    Workbook wb;
```

```
    Worksheet ws;
```

```
wb = excel.Workbooks.Add(Type.Missing);
ws = (Microsoft.Office.Interop.Excel.Worksheet)wb.ActiveSheet;
ws.Name = "Log";

if (ws == null)
{
    Console.WriteLine("Worksheet could not be created.");
    return;
}

ws.get_Range("A1", "C1").Merge();
Range Date = ws.Range["A1:D1"];
Date.Value = dateOnly;

ws.get_Range("A2", "D2").Merge();
Range IDN = ws.Range["A1:D1"];
IDN.Value = ID;

wb.SaveAs(FilePath);
wb.Close();
}
```

## WriteToExcel

After creating the excel file the next step is to write the data to be logged to the excel file. To write the data to the excel file the file must first be opened. Once the file is opened a loop can be used to specify how many points are to be logged.

Upon writing the returned string to the excel file the file will be saved and closed. This is done to assure no logged data is lost if any issues occur between readings that can cause all previous data to not be saved properly.

```
public void WriteToExcel()
```

```
{
```

```
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
```

```
    excel.Visible = false;
```

```
    excel.DisplayAlerts = false;
```

```
    Workbook wb;
```

```
    Worksheet ws;
```

```
    wb = excel.Workbooks.Open(Path.GetFileName(textBox7.Text));
```

```
    ws = (Microsoft.Office.Interop.Excel.Worksheet)wb.ActiveSheet;
```

```
    for (int i = 3; i < Convert.ToInt32(textBox5.Text); i = i + 1)
```

```
    try
```

```
    {
```

```
        resp = src.ReadString() + "\r\n";
```

```
        ws.get_Range("Ai", "Di").Merge();
```

```
        Range RES = ws.Range["Ai:Di"];
```

```
        RES.Value = resp;
```

```
        wb.SaveAs(FilePath);
```

```
        wb.Close();
```

```
    }
```

```
    catch (TimeoutException)
```

```
{  
    textBox2.Text = "timeout exception";  
}
```

```
{
```

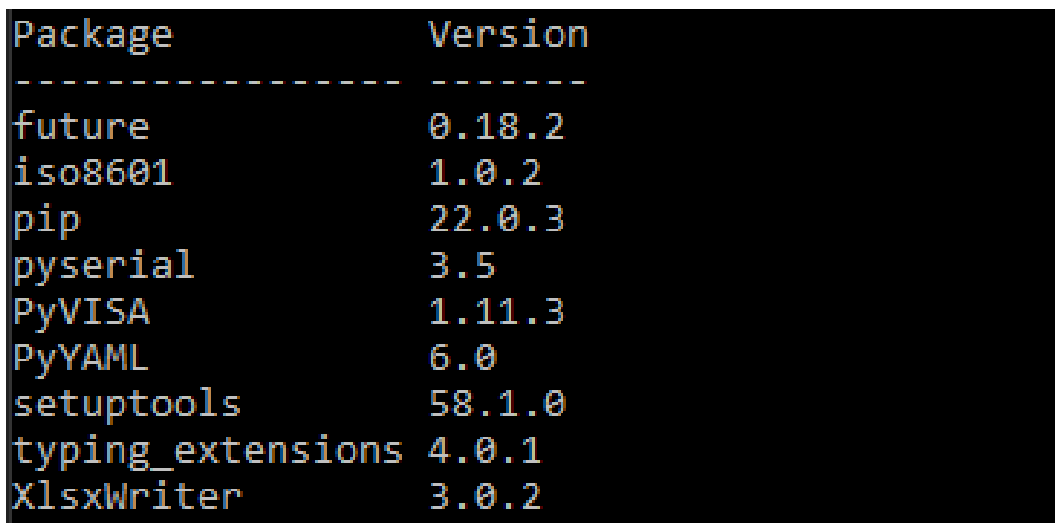
With the functions written we can now call them

# Python

## 4.1 Import NI VISA to Python

For python the PyVISA package is required along with NI VISA. PyVISA is a frontend to the VISA library which runs on Python 3.6+. PyVISA can be installed using pip. If pip is installed:

1. Open the command line.
2. Use **\$ pip install pyvisa**
  - PyVISA will install without any further action needed.
3. When installation is complete use the command **pip list** to verify PyVISA is installed properly.
  - PyVISA will appear under the package section



Package	Version
future	0.18.2
iso8601	1.0.2
pip	22.0.3
pyserial	3.5
PyVISA	1.11.3
PyYAML	6.0
setuptools	58.1.0
typing_extensions	4.0.1
XlsxWriter	3.0.2

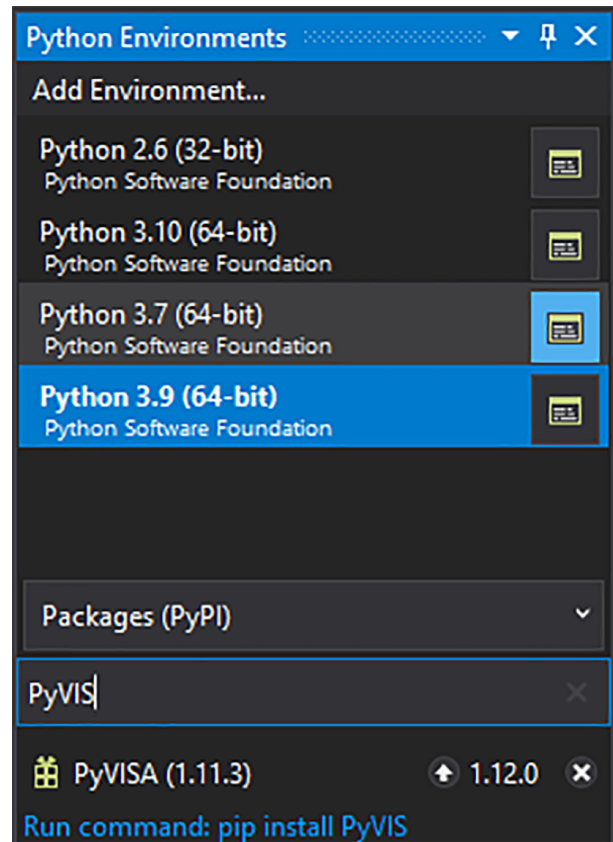
**Figure 4.1** Verify PyVISA

### Note:

PyVISA works with 32- and 64- bit Python and can deal with 32- and 64-bit VISA libraries without any extra configuration. What PyVISA cannot do is open a 32-bit VISA library while running in 64-bit Python (or the other way around).

If you are using Visual Studio, the PyVISA package can be installed from the **Python Environments** window.

1. Select the **Python Environment**.
  - The example provided uses Python 3.9
2. Select Packages (PyPI) in the drop-down menu.
3. Search for PyVISA.
  - Click **Run command: pip install pyvisa** to install the package.
  - If the package is already installed it will appear above the command.
  - Packages already installed can be updated by clicking the up arrow located to the right of the version. See [figure 4.2](#)



**Figure 4.2** PyVISA in Visual Studio

### Note:

All other packages required for the example can be included at this point. (**XlsxWrite**)

### 4.1.1 Headers

---

Upon installing the required packages, the next step is to import the packages to the project. To import the packages use the **"import"** keyword followed by the name of the package.

For the example provided the following packages are needed:

```
import pyvisa
import csv
import time
from datetime import datetime
import xlswriter
```

## 4.2 Initialization

---

To initialize any product a resource manager must first be created. To create a resource manager use the following statements.

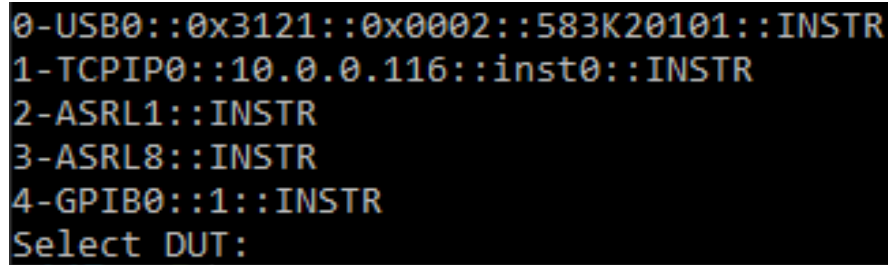
```
rm = pyvisa.ResourceManager()
```

Once the resource manager is created we must scan for all available sources and store them. To store the scanned resources and display them use the following statements:

```
li = rm.list_resources()
choice = ""

while (choice == ""):
    for index in range(len(li)):
        print(str(index) + "-" + li[index])
    choice = input ("Select DUT:")
    try:
        if(int(choice) > len(li) -1 or int (choice) < 0):
            choice = ""
            print("Invalid Input \n")
    except:
        print("Invalid Input \n")
        choice = ""
```

The code above prints a list consisting of all available resources.(see [figure 4.3](#)) The user can then input the index of the desired resource to stored that index value in variable choice.



```
0-USB0::0x3121::0x0002::583K20101::INSTR
1-TCPIP0::10.0.0.116::inst0::INSTR
2-ASRL1::INSTR
3-ASRL8::INSTR
4-GPIB0::1::INSTR
Select DUT:
```

**Figure 4.3** Available Resources

USBTMC, USBVCP, and serial devices will automatically be available without having to add the devices. GPIB and LAN devices will have to be added manually added to view them in the resource manager. Refer

### 4.2.1 Opening a Session

After creating the source manager and finding all available resources, the next step is to open a VISA session with the desired resource.

In the Initialization section the user stored the index containing the address of the desired device. To open the session we use the statements in the example below to call that index. In this section we will also set a timeout. This timeout will raise an exception when the operation takes longer than the set timeout. The timeout is specified in milliseconds. For this example a 10 second timeout is set, therefore, 10000 milliseconds is specified.

```
src = rm.open_resource(l[ int(choice)])
```

```
src.timeout = 10000
```

Once we open the session we will then verify that the session was successfully open by querying the Identification string. If the query is successful the string will be printed in the command line and parsed based on commands. The parse string will be stored into the IDN\_list array to be used later.

```
IDN = src.query("*IDN?\n")
```

```
print(IDN)
```

```
IDN_list = IDN.split(",")
```



---

## 4.3 Python Event Handling

---

This section describes the VISA event model and the various events VISA supports.

An event is a means of communication between a VISA resource and its applications. Events occur because of a condition requiring the attention of applications.

There are two ways for an application to receive event notifications. PyVISA supports using both mechanism and tries to provide a convenient interface to both.

### 4.3.1 Queuing Mechanism

---

This method places all occurrences of the specified event in a queue. This method is recommended for noncritical events that do not need immediate servicing.

### 4.3.2 Callback Handler

---

The Callback Handler invokes a function that the program specifies prior to enabling the event. This method is recommended for applications that require an immediate response.

### 4.3.3 Event Handling Example

---

When implementing an event handler the first step is to specify the type of event we want to be notified of as well as the mechanism we are implementing.

For this example we will be implementing a service request event. A service request can be generated when an operation is complete.

```
# Type of event we want to be notified about
```

```
event_type = constants.EventType.service_request
```

The queue mechanism will be used for this example

```
# Mechanism by which we want to be notified
```

```
event_mech = constants.EventMechanism.queue
```

Once the event type and mechanism is defined the event notification must be enabled:

```
src.enable_event(event_type, event_mech)
```

For events that require immediate action creating a function that is called by the VISA library is useful. When creating a handler function, it needs to have a specific signature to be used by VISA. The expected signature is (session, event\_type, event\_context, user\_handle).

```
def handle_event(resource, event, user_handle):  
    resource.called = True  
    print(f"Handled event {event.event_type} on {resource}")
```

This signature is not exactly convenient since it forces us to deal with a number of low-level details such as session (ID of a resource in VISA) and event\_context that serves the same purpose for events, therefore, the **wrap\_handler** function is used to wrap the handler.

```
wrapped = src.wrap_handler(handle_event)
```

The handler must be installed and enabled.

```
user_handle = src.install_handler(event_type, wrapped, 42)  
src.enable_event(event_type, event_mech, None)
```

Finally, the task is complete the event handler must be disabled and uninstalled.

```
src.disable_event(event_type, event_mech)  
src.uninstall_handler(event_type, wrapped, user_handle)
```

## 4.4 Data Logging

Queried data will be written to an excel sheet in order to save all queried measurements. To accomplish this task the `xlsxwriter` package will be used. This package will allow us to create excel files, edit the file's layout/settings, and most importantly write and read to and from the excel files.

Before creating the excel file, a couple of variables are created in order to store values called through the code.

```
Date_Time = (datetime.now())
```

```
Time = str(Date_Time.time())
```

```
Time1 = Time.replace(":", "_")
```

```
Date = str(Date_Time.date())
```

After defining the variables we can then create the workbook/excel file. Both a workbook and an excel sheet must be created in order to create the file. The workbook will be given a name including the model number-Date-Time. The sheet will be name Sheet1.

```
outWorkbook = xlsxwriter.Workbook(IDN_list[1] + "-" + Date + "_" + Time1 + ".xlsx")
```

```
Sheet1 = outWorkbook.add_worksheet("Sheet1")
```

When the workbook and sheet are created we specify the default format for each cell as well as the default number format.

```
cell_format = outWorkbook.add_format()
```

```
cell_format.set_num_format("0.000")
```

Custom formats objects can be created calling the `add_format()` function. For this example 3 format objects were created: `num_format`, `merge_format`, and `centerbold_format`. To read more information about the parameters of the object please refer to [The Format Class](#).

```
num_format = outWorkbook.add_format({
```

```
    'bold': 1,
```

```
    'border': 0,
```

```
    'align': 'center',
```

```
    'valign': 'vcenter',
```

```
    'num_format': '0.000'})
```

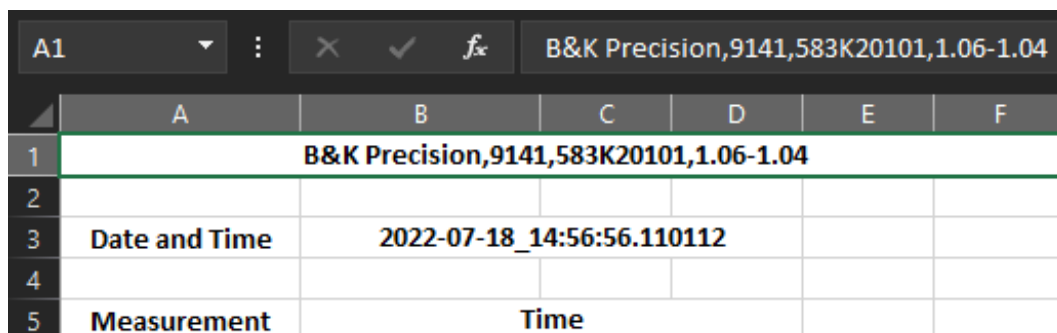
```
merge_format = outWorkbook.add_format({
    'bold': 1,
    'border': 1,
    'align': 'center',
    'valign': 'vcenter'})

centerbold_format = outWorkbook.add_format({
    'bold': 1,
    'border': 0,
    'align': 'center',
    'valign': 'vcenter'})
```

Finally, the created objects are used to set the layout of the excel document. This includes configuring the header cells of the document as well as writing basic information such as the identification string and date and time of logging.

```
Sheet1.set_column(0, 1, 16)
Sheet1.merge_range('A1:F1', IDN, merge_format)
Sheet1.write('A3', 'Date and Time', centerbold_format)
Sheet1.merge_range('B3:D3', Date + "_" + Time, centerbold_format)
Sheet1.write('A5', "Measurement", centerbold_format)
Sheet1.merge_range('B5:D5', "Time", centerbold_format)
```

The snippet above writes the basic information previously stored in variables such as the date, time and the identification string. Other strings like **Measurement** and **Time** are also written in row 5 to indicate what the data stored in this column will consist of. The snippet produces the layout shown in [figure 4.4](#)



	A	B	C	D	E	F
1	B&K Precision,9141,583K20101,1.06-1.04					
2						
3	Date and Time	2022-07-18_14:56:56.110112				
4						
5	Measurement	Time				

Figure 4.4 Excel Layout

### 4.4.1 Data Acquisition

To continuously acquire data from the instrument a while loop along with the try statement to raise an exception. The while loop will be end once a keyboard interrupt exception is raised. Pressing CTRL + C will raise the exception closing the workbook.

```
i = 7
```

```
try:
```

```
    while True:
```

```
        dt_obj2 = datetime.now()
```

```
        gettimestamp = dt_obj2.strftime("%H:%M:%S")
```

```
        meas = "%s"%(src.query("MEAS:ALL?\n"))
```

```
        meas1 = meas.replace("\n", "")
```

```
        meas_li = (meas1.split(","))
```

```
        print(float(meas_li[0]))
```

```
        Meas = "A%d"%i
```

```
        timestamp = "B%d:D%d"%(i,i)
```

```
        Sheet1.write(Meas, float(meas_li[0]), num_format)
```

```
        Sheet1.merge_range(timestamp, gettimestamp, centerbold_format)
```

```
        i += 1
```

```
        time.sleep(.050)
```

```
except KeyboardInterrupt:
```

```
    outWorkbook.close()
```