

Release Notes:

June 20th, 2024: Python Cornerstone B Package updated for PyVISA Version 1.14. Below application notes were compiled with an older version of PyVISA and Windows operating system, modifications may be required for current and future PyVISA iterations.

Controlling the Cornerstone B Monochromator from an Application

This application note describes some of the steps required to communicate with the Cornerstone B monochromator using its USB port. Some example code is given for C++ and Python. For LabView, example code in the form of the “MonoTerm” application is found in the information files that are included with the instrument. This document does not address LabView programming.

The methods require the NI-VISA package. NI-VISA is National Instrument’s collection of “drivers” that connects to instruments using multiple interfaces: GPIB, RS-232 (serial), Ethernet, PXI, USB, and others. National provides NI-VISA free of charge for many uses.

Python

Communicating with instruments using Python is simple using NI-VISA and PyVisa. PyVisa is a python “wrapper” for NI-VISA. PyVisa is sometimes called the “front end” and NI-VISA is called the “back end”. Suggestions for installing them are included in a later section of this document.

Once these two packages are installed on a computer, a python program needs only three PyVisa functions to communicate with an instrument: an “open” function, a “write”, and a “read”.

While NI-VISA supports “raw” USB communication with some instruments, it requires a detailed understanding of an instrument’s firmware USB implementation. On the other hand, it treats instruments that implement the USB Test and Measurement Class (USBTMC) similarly to a GPIB instrument using basic open, write, and read functions, typically sending and receiving character-based command strings and responses. This greatly simplifies the process of connecting to and communicating with a USBTMC instrument, such as the Cornerstone ‘B’ monochromators.

The following discussion simplistically explains some of the basic functions and methods supplied by PyVisa version 1.9.1, using Python version 3.7 under Windows 7. It is strongly recommended to read some of the excellent on-line tutorials and reference materials provided by the PyVisa development group (e.g. pyvisa.readthedocs.io). It should be noted that PyVisa is continually being improved, and some of the methods listed here may be from versions that are older than the current one.

Other resources that can be extremely useful are National Instruments’ NI-VISA User Manual and Programmer’s Reference Manual. It also includes some C programming examples.

GPIO Example

Starting with version 1.5¹, PyVisa requires the initialization of a “resource manager” for all instruments that will be used:

```
>>> import visa
>>> rm = visa.ResourceManager( )
```

The resource manager is then used to create virtual instruments that your program uses to communicate with actual instruments. For a GPIO instrument, assuming its address is 4, use this statement:

```
>>>my_gpio_instr = rm.open_resource( 'GPIO0::4::INSTR' )
```

The ‘open_resource’ method instantiates a virtual instrument class containing read and write methods.

To send a command to an actual GPIO instrument, use the write method of the virtual instrument:

```
>>> my_gpio_instrument.write( command )
```

where “command” is the string to send.

To read a response from the instrument, use the instrument’s read method:

```
>>>reply_string = my_gpio_instrument.read( )
```

Serial Example

PyVisa treats a PC’s COM port as an instrument. To open communication with a device on COM1, assuming the resource manager has already been initialized, use:

```
>>>my_serial_instrument = rm.open_resource( 'COM1' )
```

Then use the write and read methods of my_serial_instrument to communicate with the device.

Another perfectly valid approach is to use PySerial instead of PyVISA. PySerial has a much smaller installation size since it does not require NI-VISA.

USBTCM Example

A serial port (RS-232) typically has only one instrument connected to a COM port, so there is no need to identify the instrument to create a connection to it. The GPIO bus can have several devices on it, but they are uniquely defined by an address number. The USB bus, however, can have several devices connected to it, including several that implement USBTCM; information identifying the device is required to make a correct connection.

When PyVisa polls the USBTCM devices, it automatically queries them for some identifying information. Some of this information includes the vendor ID number, the product ID number, and the device’s serial number. Vendors (manufacturers) must register with the USB Consortium to obtain a unique Vendor ID, and they must also maintain a list of unique product ID numbers. This is sufficient to identify a single product type if several different devices are connected. In the case where there are multiple products of

¹ For versions earlier than 1.5, we recommend reading the PyVisa tutorial “Migrating from PyVISA < 1.5”.

the same type on the bus, the serial number should provide sufficient identifying information to be sure that the connection is for a specific instrument.

To instantiate an instrument class for a USBTMC instrument, open the resource, e.g.:

```
>>>my_usb_instrument = rm.open_resource ( 'USB0::0x1FDE::0x0006::1042::INSTR' )
```

The string in quotes sent as a parameter to the `open_resource` method contains the following fields, separated by double-colons:

1. The string "USB0".
2. A string representing the vendor ID number (0x1FDE, Newport, in this example).
3. A string representing the product ID number (0x0006: a CS130B monochromator²).
4. A string representing the product serial number (1042 in this case).
5. Optionally the word 'INSTR'.

Resource Strings

To find these identification strings, use the resource manager's "list_resources" method, typically before instantiating each instrument.

```
>>>instr_list = rm.list_resources()
>>>print( instr_list )
('USB0::0x1FDE::0x0006::1042::INSTR', 'ASRL1::INSTR', 'GPIB0::22:INSTR')
```

The first instrument in the list above is a USBTMC instrument from manufacturer 0x1FDE (Newport), its product ID is 0x0006 (CS130B monochromator), and its serial number is 1042.

The 'ASRL1::INSTR' string is PyVisa's terminology for 'COM1', the serial port installed in the computer. Note that PyVisa accepts either 'ASRL1::INSTR' or 'COM1'.

'GPIB0:22:INSTR' is the GPIB instrument (e.g. voltmeter) connected to the computer. Each of the three strings is acceptable as a resource string parameter for the `open_resource()` method.

When there are multiple USBTMC instruments connected, a program must parse the resource strings reported by the resource manager:

```
>>>instr_list = rm.list_resources()
>>>print( instr_list )
('USB0::0x1FDE::0x0006::1042::INSTR', 'USB0::0x1FDE::0x000A::71200000::INSTR',
'ASRL1::INSTR', 'GPIB0::22:INSTR')
```

This shows that in addition to the CornerstoneB monochromator there is also a Newport/ILX Verasol solar simulator attached. This is the instrument with the product ID '0x000A'.

² The product ID is 0x0006 for the Newport CS130B and 0x0014 for the CS260B.

Here is one way to get the ID string for the required instrument, once all the resource strings are in `instr_list`:

```
mono_string = "nothing"
# Check each field in each string in the instrument list:
for instr_string in instr_list :
    fields = instr_string.split(':')
    if( ( len( fields ) == 5 ) and
        ( fields[0] == "USB0" ) and
        ( fields[1] == "0x1FDE" ) and # Newport/ILX
        ( fields[2] == "0x0006" ) and # CS130B
        ( fields[3] == "1042" ) ):
        mono_string = instr_string # now it is something
if( mono_string != "nothing" ):
    mono_instr = rm.open_resource( mono_string )
else:
    print( 'Could not find Newport CS130B' )
```

It should be noted that the first field "USB0" in the above resource strings can be different on another computer: the '0' may be a different number, so it might be best just to look for the first three characters 'U', 'S', and 'B' (or just ignore the first field).

C++

You can interface directly to NI-VISA from C++: a "wrapper" like PyVisa is not needed.

The following C++ code opens and closes a pair of instruments by interfacing with National Instruments' NI-VISA. See the "Python" section for more information about NI-VISA. This code uses the same NI-VISA version mentioned above. It was compiled using Visual Studio Community 2017 version 15.9.9, on a PC running Windows 7.

```
#include <iostream>
// The library corresponding to the following visa header is in:
// C:\Program Files (x86)\IVI Foundation\VISA\WinNT\lib\msc\visa32.lib.
#include "C:\Program Files (x86)\IVI Foundation\VISA\WinNT\Include\visa.h"

#define BUFF_SIZE 200

int main( void )
{
    ViStatus      viStatus; // for checking errors
    ViSession     viRsrc;   // the visa resource manager
    ViSession     mono_instr; // the monochromator virtual instrument
    ViSession     meter_instr; // the voltmeter virtual instrument
    ViUInt32      retCount; // the number of characters returned by a read
    ViChar        buffer[BUFF_SIZE]; // will contain the characters read

    // Modify these resource names to match your instruments:
    ViChar        meter_name[] = "GPIB0::22::INSTR";
    // Change the 0 below to your instrument's serial number.
    ViChar        mono_name[] = "USB0::0x1FDE::0x0006::0::INSTR";
```

```
// Open the VISA resource manager.
viStatus = viOpenDefaultRM( &viRsrc );
if (viStatus < VI_SUCCESS)
{
    std::cout << "failed to open resource manager\n";
    return -1;
}

// Open the voltmeter (on GPIB).
viStatus = viOpen(viRsrc, meter_name, VI_NULL, 5000, &meter_instr);
if (viStatus < VI_SUCCESS)
{
    std::cout << "failed to open voltmeter\n";
    return -1;
}

// Open the monochromator (on USB).
viStatus = viOpen(viRsrc, mono_name, VI_NULL, 5000, &mono_instr);
if (viStatus < VI_SUCCESS)
{
    std::cout << "failed to open monochromator\n";
    return -1;
}

// Get (and print) the voltmeter's identification string.
viStatus = viWrite(meter_instr, (ViConstBuf)"*IDN?", 6, &retCount);
memset(buffer, 0, sizeof(buffer));
viStatus = viRead(meter_instr, (ViPBuf)buffer, BUFF_SIZE, &retCount);
std::cout << buffer;

// Get (and print) the monochromator's identification string.
viStatus = viWrite(mono_instr, (ViConstBuf)"*IDN?", 6, &retCount);
memset(buffer, 0, sizeof(buffer));
viStatus = viRead(mono_instr, (ViPBuf)buffer, BUFF_SIZE, &retCount);
std::cout << buffer;

// Close the instruments.
viStatus = viClose(meter_instr);
viStatus = viClose(mono_instr);

// Finally close the resource manager.
viStatus = viClose(viRsrc);

return 0;
}
```

Installation of PyVisa and NI-VISA

NI-VISA

The NI-VISA download is large, so it's a good idea to make sure your computer will stay turned on while it's downloading. National Instruments requires that you create an account before you can download anything, and they also reasonably ask you to verify that you won't be selling any product containing the items they freely allow you to use. As of March 2019, their downloads are located at <http://www.ni.com/en-us/support/downloads/drivers/download.ni-visa.html>.

Choose the download that is right for your operating system and computer. For this document, we chose version 18.5 Runtime package. This downloads an installation executable. Run the executable. It will ask you to reboot, after which the NI-VISA drivers will be installed on your computer.

PyVisa

If you already have a distribution of Python containing pip, you can just execute 'pip install PyVISA'. Alternatively, you can download a compressed package from <https://pypi.org/project/PyVISA/#files>. This is a 'compressed tarball' which can be extracted using 7-zip or another suitable compression tool. Make sure the distribution winds up under your <python>\Lib\site-packages directory.

As with all installed software, there is no way to troubleshoot in advance problems that crop up. However, the PyVisa pages have several tutorial and FAQ pages, and so does the National Instruments site.