PyVISA Documentation

Release 1.11.0.dev0

PyVISA Authors

Oct 12, 2019

Contents

PyVISA is a Python package that enables you to control all kinds of measurement devices independently of the interface (e.g. GPIB, RS232, USB, Ethernet). As an example, reading self-identification from a Keithley Multimeter with GPIB number 12 is as easy as three lines of Python code:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager()
>>> rm.list_resources()
('ASRL1::INSTR', 'ASRL2::INSTR', 'GPIB0::12::INSTR')
>>> inst = rm.open_resource('GPIB0::12::INSTR')
>>> print(inst.query("*IDN?"))
```

(That's the whole program; really!) It works on Windows, Linux and Mac; with arbitrary adapters (e.g. National Instruments, Agilent, Tektronix, Stanford Research Systems).

CHAPTER 1

General overview

The programming of measurement instruments can be real pain. There are many different protocols, sent over many different interfaces and bus systems (e.g. GPIB, RS232, USB, Ethernet). For every programming language you want to use, you have to find libraries that support both your device and its bus system.

In order to ease this unfortunate situation, the Virtual Instrument Software Architecture (VISA) specification was defined in the middle of the 90ies. VISA is a standard for configuring, programming, and troubleshooting instrumentation systems comprising GPIB, VXI, PXI, Serial, Ethernet, and/or USB interfaces.

Today VISA is implemented on all significant operating systems. A couple of vendors offer VISA libraries, partly with free download. These libraries work together with arbitrary peripherical devices, although they may be limited to certain interface devices, such as the vendor's GPIB card.

The VISA specification has explicit bindings to Visual Basic, C, and G (LabVIEW's graphical language). Python can be used to call functions from a VISA shared library (*.dll*, *.so*, *.dylib*) allowing to directly leverage the standard implementations. In addition, Python can be used to directly access most bus systems used by instruments which is why one can envision to implement the VISA standard directly in Python (see the *PyVISA-Py* project for more details). PyVISA is both a Python wrapper for VISA shared libraries but can also serve as a front-end for other VISA implementation such as *PyVISA-Py*.

1.1 User guide

This section of the documentation will focus on getting you started with PyVISA. The following sections will cover how to install and configure the library, how to communicate with your instrument and how to debug standard communications issues.

1.1.1 Installation

PyVISA is a frontend to the VISA library. It runs on Python 2.7 and 3.4+.

You can install it using pip:

```
$ pip install -U pyvisa
```

Backend

In order for PyVISA to work, you need to have a suitable backend. PyVISA includes a backend that wraps the National Instruments's VISA library. However, you need to download and install the library yourself (See *NI-VISA Installation*). There are multiple VISA implementations from different vendors. PyVISA is tested only against National Instruments's VISA.

Warning: 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).

You need to make sure that the Python and VISA library have the same bitness

Alternatively, you can install PyVISA-Py which is a pure Python implementation of the VISA standard. You can install it using pip:

```
$ pip install -U pyvisa-py
```

Note: At the moment, *PyVISA-Py* implements only a limited subset of the VISA standard and does not support all protocols on all bus systems. Please refer to its documentation for more details.

Testing your installation

That's all! You can check that PyVISA is correctly installed by starting up python, and creating a ResourceManager:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager()
>>> print(rm.list_resources())
```

If you encounter any problem, take a look at the *Miscellaneous questions*. There you will find the solutions to common problem as well as useful debugging techniques. If everything fails, feel free to open an issue in our issue tracker

Using the development version

You can install the latest development version (at your own risk) directly form GitHub:

```
$ pip install -U https://github.com/pyvisa/pyvisa/zipball/master
```

Note: If you have an old system installation of Python and you don't want to mess with it, you can try Anaconda. It is a free Python distribution by Continuum Analytics that includes many scientific packages.

1.1.2 Configuring the backend

Currently there are two backends available: The one included in pyvisa, which uses the IVI library (include NI-VISA, Keysight VISA, R&S VISA, tekVISA etc.), and the backend provided by pyvisa-py, which is a pure python implementation of the VISA library. If no backend is specified, pyvisa uses the IVI backend if any IVI library has been installed (see next section for details). Failing that, it uses the pyvisa-py backend.

You can also select a desired backend by passing a parameter to the ResourceManager, shown here for pyvisa-py:

```
>>> visa.ResourceManager('@py')
```

Alternatively it can also be selected by setting the environment variable PYVISA_LIBRARY. It takes the same values as the ResourceManager constructor.

Configuring the IVI backend

Note: The IVI backend requires that you install first the IVI-VISA library. For example you can use NI-VISA or any other library in your opinion. about NI-VISA get info here: (*NI-VISA Installation*)

In most cases PyVISA will be able to find the location of the shared visa library. If this does not work or you want to use another one, you need to provide the library path to the *ResourceManager* constructor:

```
>>> rm = ResourceManager('Path to library')
```

You can make this library the default for all PyVISA applications by using a configuration file called `.pyvisarc` (mind the leading dot) in your home directory.

Operating System	Location
Windows NT	`<root>\WINNT\Profiles\<username>`
Windows 2000, XP and 2003	`<root>\Documents and Settings\<username>`
Windows Vista, 7 or 8	`<root>\Users\<username>`
Mac OS X	`/Users/<username>`
Linux	`/home/<username>` (depends on the distro)

For example in Windows XP, place it in your user folder "Documents and Settings" folder, e.g. `C:\Documents and Settings\smith\.pyvisarc` if "smith" is the name of your login account.

This file has the format of an INI file. For example, if the library is at `/usr/lib/libvisa.so.7`, the file `.pyvisarc` must contain the following:

```
[Paths]

VISA library: /usr/lib/libvisa.so.7
```

Please note that *[Paths]* is treated case-sensitively.

You can define a site-wide configuration file at `/usr/share/pyvisa/.pyvisarc` (It may also be `/usr/local/...` depending on the location of your Python). Under Windows, this file is usually placed at `c:\Python27\share\pyvisa\.pyvisarc`.

If you encounter any problem, take a look at the *Frequently asked questions*. There you will find the solutions to common problem as well as useful debugging techniques. If everything fails, feel free to open an issue in our issue tracker

1.1.3 Communicating with your instrument

Note: If you have been using PyVISA before version 1.5, you might want to read *Migrating from PyVISA < 1.5*.

An example

Let's go *in medias res* and have a look at a simple example:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager()
>>> rm.list_resources()
('ASRL1::INSTR', 'ASRL2::INSTR', 'GPIB0::14::INSTR')
>>> my_instrument = rm.open_resource('GPIB0::14::INSTR')
>>> print(my_instrument.query('*IDN?'))
```

This example already shows the two main design goals of PyVISA: preferring simplicity over generality, and doing it the object-oriented way.

After importing `pyvisa`, we create a `ResourceManager` object. If called without arguments, PyVISA will prefer the default backend (IVI) which tries to find the VISA shared library for you. If it fails it will fall back to pyvisa-py if installed. You can check what backend is used and the location of the shared library used, if relevant, simply by:

```
>>> print(rm)
<ResourceManager('/path/to/visa.so')>
```

Note: In some cases, PyVISA is not able to find the library for you resulting in an `OSError`. To fix it, find the library path yourself and pass it to the ResourceManager constructor. You can also specify it in a configuration file as discussed in *Configuring the backend*.

Once that you have a `ResourceManager`, you can list the available resources using the `list_resources` method. The output is a tuple listing the *VISA resource names*. You can use a dedicated regular expression syntax to filter the instruments discovered by this method. The syntax is described in details in *`list_resources()`*. The default value is '?*::INSTR' which means that by default only instrument whose resource name ends with '::INSTR' are listed (in particular USB RAW resources and TCPIP SOCKET resources are not listed).

In this case, there is a GPIB instrument with instrument number 14, so you ask the `ResourceManager` to open "'GPIB0::14::INSTR'" and assign the returned object to the *my_instrument*.

Notice `open_resource` has given you an instance of `GPIBInstrument` class (a subclass of the more generic `Resource`).

```
>>> print(my_instrument)
<GPIBInstrument('GPIB::14')>
```

There many `Resource` subclasses representing the different types of resources, but you do not have to worry as the `ResourceManager` will provide you with the appropriate class. You can check the methods and attributes of each class in the *Resource classes*

Then, you query the device with the following message: `'\*IDN?'`. Which is the standard GPIB message for "what are you?" or – in some cases – "what's on your display at the moment?". `query` is a short form for a `write` operation to send a message, followed by a `read`.

So:

```
>>> my_instrument.query("*IDN?")
```

is the same as:

```
>>> my_instrument.write('*IDN?')
>>> print(my_instrument.read())
```

Note: You can access all the opened resources by calling `rm.list_opened_resources()`. This will return a list of `Resource`, however note that this list is not dynamically updated.

Getting the instrument configuration right

For most instruments, you actually need to properly configure the instrument so that it understands the message sent by the computer (in particular how to identifies the end of the commands) and so that computer knows when the instrument is done talking. If you don't you are likely to see a `VisaIOError` reporting a timeout.

For message based instruments (which covers most of the use cases), this usually consists in properly setting the *`read_termination`* and *`write_termination`* attribute of the resource. Resources have more attributes described in *Resources*, but for now we will focus on those two.

The first place to look for the values you should set for your instrument is the manual. The information you are looking is usually located close to the beginning of the IO operation section of the manual. If you cannot find the value, you can try to iterate through a couple of standard values but this is not recommended approach.

Once you have that information you can try to configure your instrument and start communicating as follows:

```
>>> my_instrument.read_termination = '\n'
>>> my_instrument.write_termination = '\n'
>>> my_instrument.query('*IDN?')
```

Here we use *'n'* known as 'line feed'. This is a common value, another one is *'r'* i.e. 'carriage return', and in some cases the null byte '0' is used.

In in an ideal world, this will work and you will be able to get an answer from your instrument. If it does not, it means the settings are likely wrong (the documentation does not always shine by its clarity). In the following we will discuss common debugging tricks, if nothing works feel free to post on the PyVISA issue tracker. If you do be sure to describe in detail your setup and what you already attempted.

Note: The particular case of reading back large chunk of data either in ASCII or binary format is not discussed below but in *Reading and Writing values*.

Making sure the instrument understand the command

When using query, we are testing both writing to and reading from the instrument. The first thing to do is to try to identify if the issue occurs during the write or the read operation.

If your instrument has a front panel, you can check for errors (some instrument will display a transient message right after the read). If an error occurs, it may mean your command string contains a mistake or the instrument is using a different set of command (some instrument supports both a legacy set of commands and SCPI commands). If you see no error it means that either the instrument did not detect the end of your message or you just cannot read it. The next step is to determine in what situation we are.

To do so, you can look for a command that would produce a visible/measurable change on the instrument and send it. In the absence of errors, if the expected change did not occur it means the instrument did not understand that the command was complete. This points out to an issue with the *write_termination*. At this stage, you can go back to the manual (some instruments allow to switch between the recognized values), or try standards values (such as *'n'*, *'r'*, combination of those two, *'0'*).

Assuming you were able to confirm that the instrument understood the command you sent, it means the reading part is the issue, which is easier to troubleshoot. You can try different standard values for the *read_termination*, but if nothing works you can use the *read_bytes()* method. This method will read at most the number of bytes specified. So you can try reading one byte at a time till you encounter a time out. When that happens most likely the last character you read is the termination character. Here is a quick example:

```
my_instrument.write('*IDN?')
while True:
    print(my_instrument.read_bytes(1))
```

If *read_bytes()* times out on the first read, it actually means that the instrument did not answer. If the instrument is old it may be because your are too fast for it, so you can try waiting a bit before reading (using *time.sleep* from Python standard library). Otherwise, you either use a command that does not cause any answer or actually your write does not work (go back up a couple of paragraph).

The above focused on using only PyVISA, if you are running Windows, or MacOS you are likely to have access to third party tools that can help. Some tips to use them are given in the next section.

Note: Some instruments do not react well to a communication error, and you may have to restart it to get it to work again.

Using third-party softwares

The implementation of VISA from National Instruments and Keysight both come with tools (NIMax, Keysight Connection Expert) that can be used to figure out what is wrong with your communication setup.

In both cases, you can open an interactive communication session to your instrument and tune the settings using a GUI (which can make things easier). The basic procedure is the one described above, if you can make it work in one of those tools you should be able, in most cases, to get it to work in PyVISA. However if it does not work using those tools, it won't work in PyVISA.

Hopefully those simple tips will allow you to get through. In some cases, it may not be the case and you are always welcome to ask for help (but realize that the maintainers are unlikely to have access to the instrument you are having trouble with).

1.1.4 A more complex example

The following example shows how to use SCPI commands with a Keithley 2000 multimeter in order to measure 10 voltages. After having read them, the program calculates the average voltage and prints it on the screen.

I'll explain the program step-by-step. First, we have to initialize the instrument:

```
>>> keithley = rm.open_resource("GPIB::12")
>>> keithley.write("*rst; status:preset; *cls")
```

Here, we create the instrument variable *keithley*, which is used for all further operations on the instrument. Immediately after it, we send the initialization and reset message to the instrument.

The next step is to write all the measurement parameters, in particular the interval time (500ms) and the number of readings (10) to the instrument. I won't explain it in detail. Have a look at an SCPI and/or Keithley 2000 manual.

```
>>> interval_in_ms = 500
>>> number_of_readings = 10
>>> keithley.write("status:measurement:enable 512; *sre 1")
>>> keithley.write("sample:count %d" % number_of_readings)
>>> keithley.write("trigger:source bus")
>>> keithley.write("trigger:delay %f" % (interval_in_ms / 1000.0))
>>> keithley.write("trace:points %d" % number_of_readings)
>>> keithley.write("trace:feed sense1; trace:feed:control next")
```

Okay, now the instrument is prepared to do the measurement. The next three lines make the instrument waiting for a trigger pulse, trigger it, and wait until it sends a "service request":

```
>>> keithley.write("initiate")
>>> keithley.assert_trigger()
>>> keithley.wait_for_srq()
```

With sending the service request, the instrument tells us that the measurement has been finished and that the results are ready for transmission. We could read them with *keithley.query("trace:data?")* however, then we'd get:

```
-000.0004E+0,-000.0005E+0,-000.0004E+0,-000.0007E+0,
-000.0000E+0,-000.0007E+0,-000.0008E+0,-000.0004E+0,
-000.0002E+0,-000.0005E+0
```

which we would have to convert to a Python list of numbers. Fortunately, the *query_ascii_values()* method does this work for us:

```
>>> voltages = keithley.query_ascii_values("trace:data?")
>>> print("Average voltage: ", sum(voltages) / len(voltages))
```

Finally, we should reset the instrument's data buffer and SRQ status register, so that it's ready for a new run. Again, this is explained in detail in the instrument's manual:

```
>>> keithley.query("status:measurement?")
>>> keithley.write("trace:clear; trace:feed:control next")
```

That's it. 18 lines of lucid code. (Well, SCPI is awkward, but that's another story.)

1.1.5 Reading and Writing values

Some instruments allow to transfer to and from the computer larger datasets with a single query. A typical example is an oscilloscope, which you can query for the whole voltage trace. Or an arbitrary wave generator to which you have to transfer the function you want to generate.

Basically, data like this can be transferred in two ways: in ASCII form (slow, but human readable) and binary (fast, but more difficult to debug).

PyVISA Message Based Resources have different methods for this called *read_ascii_values()*, *query_ascii_values()* and *read_binary_values()*, *query_binary_values()*.

Reading ASCII values

If your oscilloscope (open in the variable `inst`) has been configured to transfer data in **ASCII** when the `CURV?` command is issued, you can just query the values like this:

```
>>> values = inst.query_ascii_values('CURV?')
```

`values` will be `list` containing the values from the device.

In many cases you do not want a `list` but rather a different container type such as a `numpy.array`. You can of course cast the data afterwards like this:

```
>>> values = np.array(inst.query_ascii_values('CURV?'))
```

but sometimes it is much more efficient to avoid the intermediate list, and in this case you can just specify the container type in the query:

```
>>> values = inst.query_ascii_values('CURV?', container=numpy.array)
```

In `container`, you can have any callable/type that takes an iterable.

Note: When using numpy.array or numpy.ndarray, PyVISA will use numpy routines to optimize the conversion by avoiding the use of an intermediate representation.

Some devices transfer data in ASCII but not as decimal numbers but rather hex or oct. Or you might want to receive an array of strings. In that case you can specify a `converter`. For example, if you expect to receive integers as hex:

```
>>> values = inst.query_ascii_values('CURV?', converter='x')
```

`converter` can be one of the Python string formatting codes. But you can also specify a callable that takes a single argument if needed. The default converter is `'f'`.

Finally, some devices might return the values separated in an uncommon way. For example if the returned values are separated by a `'$'` you can do the following call:

```
>>> values = inst.query_ascii_values('CURV?', separator='$')
```

You can provide a function to takes a string and returns an iterable. Default value for the separator is `','` (comma)

Reading binary values

If your oscilloscope (open in the variable `inst`) has been configured to transfer data in **BINARY** when the `CURV?` command is issued, you need to know which type datatype (e.g. uint8, int8, single, double, etc) is being used. PyVISA use the same naming convention as the struct module.

You also need to know the *endianness*. PyVISA assumes little-endian as default. If you have doubles *d* in big endian the call will be:

```
>>> values = inst.query_binary_values('CURV?', datatype='d', is_big_endian=True)
```

You can also specify the output container type, just as it was shown before.

By default, PyVISA will assume that the data block is formatted according to the IEEE convention. If your instrument uses HP data block you can pass `header_fmt='hp'` to `read_binary_values`. If your instrument does not use any header for the data simply `header_fmt='empty'`.

By default PyVISA assumes, that the instrument will add the termination character at the end of the data block and actually makes sure it reads it to avoid issues. This behavior fits well a number of devices. However some devices omit the termination character, in which cases the operation will timeout. In this situation, first makes sure you can actually read from the instrument by reading the answer using the `read_raw` function (you may need to call it multiple time), and check that the advertized length of the block match what you get from your instrument (plus the header). If it is so,

then you can safely pass `expect_termination=False`, and PyVISA will not look for a termination character at the end of the message.

If you can read without any problem from your instrument, but cannot retrieve the full message when using this method (VI_ERROR_CONN_LOST, VI_ERROR_INV_SETUP, or Python simply crashes), try passing different values for `chunk_size``(the default is 20*1024). The underlying mechanism for this issue is not clear but changing ``chunk_size` has been used to work around it. Note that using larger chunk sizes for large transfer may result in a speed up of the transfer.

In some cases, the instrument may use a protocol that does not indicate how many bytes will be transferred. The Keithley 2000 for example always return the full buffer whose size is reported by the 'trace:points?' command. Since a binary block may contain the termination character, PyVISA need to know how many bytes to expect. For those case, you can pass the expected number of points using the `data_points` keyword argument. The number of bytes will be inferred from the datatype of the block.

Writing ASCII values

To upload a function shape to arbitrary wave generator, the command might be `WLISt:WAVeform:DATA <waveform name>,<function data>` where `<waveform name>` tells the device under which name to store the data.

```
>>> values = list(range(100))
>>> inst.write_ascii_values('WLISt:WAVeform:DATA somename,', values)
```

Again, you can specify the converter code.

```
>>> inst.write_ascii_values('WLISt:WAVeform:DATA somename,', values, converter='x')
```

`converter` can be one of the Python string formatting codes. But you can also specify a callable that takes a single argument if needed. The default converter is `'f'`.

The separator can also be specified just like in `query_ascii_values`.

```
>>> inst.write_ascii_values('WLISt:WAVeform:DATA somename,', values, converter='x',␣
↪separator='$')
```

You can provide a function to takes a iterable and returns an string. Default value for the separator is `','` (comma)

Writing binary values

To upload a function shape to arbitrary wave generator, the command might be `WLISt:WAVeform:DATA <waveform name>,<function data>` where `<waveform name>` tells the device under which name to store the data.

```
>>> values = list(range(100))
>>> inst.write_binary_values('WLISt:WAVeform:DATA somename,', values)
```

Again you can specify the `datatype` and `endianness`.

```
>>> inst.write_binary_values('WLISt:WAVeform:DATA somename,', values, datatype='d',␣
↪is_big_endian=False)
```

When things are not what they should be

PyVISA provides an easy way to transfer data from and to the device. The methods described above work fine for 99% of the cases but there is always a particular device that do not follow any of the standard protocols and is so different that cannot be adapted with the arguments provided above.

In those cases, you need to get the data:

```
>>> inst.write('CURV?')
>>> data = inst.read_raw()
```

and then you need to implement the logic to parse it.

Alternatively if the *read_raw* call fails you can try to read just a few bytes using:

```
>>> inst.write('CURV?')
>>> data = inst.read_bytes(1)
```

If this call fails it may mean that your instrument did not answer, either because it needs more time or because your first instruction was not understood.

1.1.6 Resources

A resource represents an instrument, e.g. a measurement device. There are multiple classes derived from resources representing the different available types of resources (eg. GPIB, Serial). Each contains the particular set of attributes an methods that are available by the underlying device.

You do not create this objects directly but they are returned by the *open_resource()* method of a *ResourceManager*. In general terms, there are two main groups derived from *Resource*, *MessageBasedResource* and *RegisterBasedResource*.

Note: The resource Python class to use is selected automatically from the resource name. However, you can force a Resource Python class:

```
>>> from pyvisa.resources import MessageBasedResource
>>> inst = rm.open('ASRL1::INSTR', resource_pyclass=MessageBasedResource)
```

The following sections explore the most common attributes of `Resource` and `MessageBased` (Serial, GPIB, etc) which are the ones you will encounter more often. For more information, refer to the *API*.

Attributes Resource

session

Each communication channel to an instrument has a session handle which is unique. You can get this value:

```
>>> my_device.session
10442240
```

If the resource is closed, an exception will be raised:

```
>>> inst.close()
>>> inst.session
Traceback (most recent call last):
...
pyvisa.errors.InvalidSession: Invalid session handle. The resource might be closed.
```

timeout

Very most VISA I/O operations may be performed with a timeout. If a timeout is set, every operation that takes longer than the timeout is aborted and an exception is raised. Timeouts are given per instrument in **milliseconds**.

For all PyVISA objects, a timeout is set with

```
my_device.timeout = 25000
```

Here, `my_device` may be a device, an interface or whatever, and its timeout is set to 25 seconds. To set an **infinite** timeout, set it to `None` or `float('+inf')` or:

```
del my_device.timeout
```

To set it to **immediate**, set it to *0* or a negative value. (Actually, any value smaller than 1 is considered immediate)

Now every operation of the resource takes as long as it takes, even indefinitely if necessary.

Attributes of MessageBase resources

Chunk length

If you read data from a device, you must store it somewhere. Unfortunately, PyVISA must make space for the data *before* it starts reading, which means that it must know how much data the device will send. However, it doesn't know a priori.

Therefore, PyVISA reads from the device in *chunks*. Each chunk is 20 kilobytes long by default. If there's still data to be read, PyVISA repeats the procedure and eventually concatenates the results and returns it to you. Those 20 kilobytes are large enough so that mostly one read cycle is sufficient.

The whole thing happens automatically, as you can see. Normally you needn't worry about it. However, some devices don't like to send data in chunks. So if you have trouble with a certain device and expect data lengths larger than the default chunk length, you should increase its value by saying e.g.

```
my_instrument.chunk_size = 102400
```

This example sets it to 100 kilobytes.

Termination characters

Somehow the computer must detect when the device is finished with sending a message. It does so by using different methods, depending on the bus system. In most cases you don't need to worry about termination characters because the defaults are very good. However, if you have trouble, you may influence termination characters with PyVISA.

Termination characters may be one character or a sequence of characters. Whenever this character or sequence occurs in the input stream, the read operation is terminated and the read message is given to the calling application. The next read operation continues with the input stream immediately after the last termination sequence. In PyVISA, the termination characters are stripped off the message before it is given to you.

You may set termination characters for each instrument, e.g.

```
my_instrument.read_termination = '\r'
```

('r' is carriage return, usually appearing in the manuals as CR)

Alternatively you can give it when creating your instrument object:

```
my_instrument = rm.open_resource("GPIB::10", read_termination='\r')
```

The default value depends on the bus system. Generally, the sequence is empty, in particular for GPIB. For RS232 it's `\r`.

You can specify the character to add to each outgoing message using the `write_termination` attribute.

query_delay and *send_end*

There are two further options related to message termination, namely `send_end` and `query_delay`.

`send_end` is a boolean. If it's `True` (the default), the EOI line is asserted after each write operation, signalling the end of the operation. EOI is GPIB-specific but similar action is taken for other interfaces.

The argument `query_delay` is the time in seconds to wait after each write operation when performing a query. So you could write:

```
my_instrument = rm.open_resource("GPIB::10", send_end=False, delay=1.2)
```

This will set the delay to 1.2 seconds, and the EOI line is omitted. By the way, omitting EOI is *not* recommended, so if you omit it nevertheless, you should know what you're doing.

1.1.7 PyVISA Shell

The shell, moved into PyVISA from the Lantz Project is a text based user interface to interact with instruments. You can invoke it from the command-line:

```
pyvisa-shell
```

that will show something the following prompt:

```
Welcome to the VISA shell. Type help or ? to list commands.

(visa)
```

At any time, you can type `?` or `help` to get a list of valid commands:

```
(visa) help

Documented commands (type help <topic>):
========================================
EOF  attr  close  exit  help  list  open  query  read  timeout write

(visa) help list
List all connected resources.
```

Tab completion is also supported.

The most basic task is listing all connected devices:

```
(visa) list
( 0) ASRL1::INSTR
( 1) ASRL2::INSTR
( 2) USB0::0x1AB1::0x0588::DS1K00005888::INSTR
```

Each device/port is assigned a number that you can use for subsequent commands. Let's open comport 1:

```
(visa) open 0
ASRL1::INSTR has been opened.
You can talk to the device using "write", "read" or "query.
The default end of message is added to each message
(open) query *IDN?
Some Instrument, Some Company.
```

You can print timeout that is set for query/read operation:

```
(open) timeout
Timeout: 2000ms
```

Then also to change the timeout for example to 1500ms (1.5 sec):

```
(open) timeout 1500
Done
```

We can also get a list of all visa attributes:

```
(open) attr
+----------------------------+------------+----------------------------+-------------
↪-------------------------+
|          VISA name         |  Constant  |        Python name         |             ␣
↪    val                   |
+----------------------------+------------+----------------------------+-------------
↪-------------------------+
| VI_ATTR_ASRL_ALLOW_TRANSMIT | 1073676734 |       allow_transmit       |             ␣
↪     1                    |
|    VI_ATTR_ASRL_AVAIL_NUM   | 1073676460 |      bytes_in_buffer       |             ␣
↪     0                    |
|      VI_ATTR_ASRL_BAUD      | 1073676321 |         baud_rate          |             ␣
↪    9600                  |
|    VI_ATTR_ASRL_BREAK_LEN   | 1073676733 |        break_length        |             ␣
↪    250                   |
|   VI_ATTR_ASRL_BREAK_STATE  | 1073676732 |        break_state         |             ␣
↪     0                    |
|    VI_ATTR_ASRL_CONNECTED   | 1073676731 |                            |          VI_
↪ERROR_NSUP_ATTR          |
|    VI_ATTR_ASRL_CTS_STATE   | 1073676462 |                            |             ␣
↪     0                    |
|    VI_ATTR_ASRL_DATA_BITS   | 1073676322 |         data_bits          |             ␣
↪     8                    |
|    VI_ATTR_ASRL_DCD_STATE   | 1073676463 |                            |             ␣
↪     0                    |
|  VI_ATTR_ASRL_DISCARD_NULL  | 1073676464 |        discard_null        |             ␣
↪     0                    |
|    VI_ATTR_ASRL_DSR_STATE   | 1073676465 |                            |             ␣
↪     0                    |
|    VI_ATTR_ASRL_DTR_STATE   | 1073676466 |                            |             ␣
↪     1                    |
```

(continues on next page)

(continued from previous page)

```
|     VI_ATTR_ASRL_END_IN      | 1073676467 |          end_input          |                ␣
↪      2                    |
|     VI_ATTR_ASRL_END_OUT     | 1073676468 |                             |                ␣
↪      0                    |
|   VI_ATTR_ASRL_FLOW_CNTRL    | 1073676325 |                             |                ␣
↪      0                    |
|     VI_ATTR_ASRL_PARITY      | 1073676323 |           parity            |                ␣
↪      0                    |
|  VI_ATTR_ASRL_REPLACE_CHAR   | 1073676478 |        replace_char         |                ␣
↪      0                    |
|    VI_ATTR_ASRL_RI_STATE     | 1073676479 |                             |                ␣
↪      0                    |
|    VI_ATTR_ASRL_RTS_STATE    | 1073676480 |                             |                ␣
↪      1                    |
|    VI_ATTR_ASRL_STOP_BITS    | 1073676324 |         stop_bits           |                ␣
↪      10                   |
|    VI_ATTR_ASRL_WIRE_MODE    | 1073676735 |                             |                ␣
↪     128                   |
|    VI_ATTR_ASRL_XOFF_CHAR    | 1073676482 |         xoff_char           |                ␣
↪      19                   |
|    VI_ATTR_ASRL_XON_CHAR     | 1073676481 |          xon_char           |                ␣
↪      17                   |
|     VI_ATTR_DMA_ALLOW_EN     | 1073676318 |         allow_dma           |                ␣
↪      0                    |
|    VI_ATTR_FILE_APPEND_EN    | 1073676690 |                             |                ␣
↪      0                    |
|    VI_ATTR_INTF_INST_NAME    | 3221160169 |                             | ASRL1   (/
↪dev/cu.Bluetooth-PDA-Sync)  |
|       VI_ATTR_INTF_NUM       | 1073676662 |      interface_number       |                ␣
↪      1                    |
|      VI_ATTR_INTF_TYPE       | 1073676657 |                             |                ␣
↪      4                    |
|       VI_ATTR_IO_PROT        | 1073676316 |        io_protocol          |                ␣
↪      1                    |
|   VI_ATTR_MAX_QUEUE_LENGTH   | 1073676293 |                             |                ␣
↪      50                   |
|   VI_ATTR_RD_BUF_OPER_MODE   | 1073676330 |                             |                ␣
↪      3                    |
|     VI_ATTR_RD_BUF_SIZE      | 1073676331 |                             |                ␣
↪     4096                  |
|      VI_ATTR_RM_SESSION      | 1073676484 |                             |                ␣
↪  3160976                   |
|      VI_ATTR_RSRC_CLASS      | 3221159937 |       resource_class        |                ␣
↪   INSTR                    |
|  VI_ATTR_RSRC_IMPL_VERSION   | 1073676291 |   implementation_version    |                ␣
↪  5243392                   |
|   VI_ATTR_RSRC_LOCK_STATE    | 1073676292 |         lock_state          |                ␣
↪      0                    |
|     VI_ATTR_RSRC_MANF_ID     | 1073676661 |                             |                ␣
↪     4086                  |
|    VI_ATTR_RSRC_MANF_NAME    | 3221160308 | resource_manufacturer_name  |                ␣
↪National Instruments        |
|      VI_ATTR_RSRC_NAME       | 3221159938 |       resource_name         |                ␣
↪ASRL1::INSTR                |
|  VI_ATTR_RSRC_SPEC_VERSION   | 1073676656 |        spec_version         |                ␣
↪  5243136                   |
|     VI_ATTR_SEND_END_EN      | 1073676310 |          send_end           |                ␣
↪      1                    |
```

(continues on next page)

(continued from previous page)

```
|   VI_ATTR_SUPPRESS_END_EN    | 1073676342 |                              |              ␣
↪     0                    |
|       VI_ATTR_TERMCHAR       | 1073676312 |                              |              ␣
↪     10                   |
|     VI_ATTR_TERMCHAR_EN      | 1073676344 |                              |              ␣
↪     0                    |
|      VI_ATTR_TMO_VALUE       | 1073676314 |                              |              ␣
↪    2000                  |
|       VI_ATTR_TRIG_ID        | 1073676663 |                              |              ␣
↪     -1                   |
|   VI_ATTR_WR_BUF_OPER_MODE   | 1073676333 |                              |              ␣
↪     2                    |
|     VI_ATTR_WR_BUF_SIZE      | 1073676334 |                              |              ␣
↪    4096                  |
+------------------------------+------------+------------------------------+--------------
↪-------------------------+
```

To simplify the handling of VI_ATTR_TERMCHAR and VI_ATTR_TERMCHAR_EN, the command 'termchar' can be used. If only one character provided, it sets both read and write termination character to the same character. If two characters are provided, it sets read and write termination characters independently.

To setup termchar to 'r' (CR or ascii code 10):

```
(open) termchar CR
Done
```

To read what termchar is defined:

```
(open) termchar
Termchar read: CR write: CR
```

To setup read termchar to 'n' and write termchar to 'rn':

```
(open) termchar LF CRLF
Done
```

Supported termchar values are: CR ('r'), LF ('n'), CRLF ('rn') , NUL ('0'), None. None is used to disable termchar.

Finally, you can close the device:

```
(open) close
```

PyVisa Shell Backends

Based on available backend (see bellow for `info` command), it is possible to switch shell to use non-default backend via `-b BACKEND` or `--backend BACKEND`.

You can invoke:

```
pyvisa-shell -b sim
```

to use python-sim as backend instead of ni backend. This can be used for example for testing of python-sim configuration.

You can invoke:

```
pyvisa-shell -b py
```

uses python-py as backend instead of ivi backend, for situation when ivi not installed.

PyVisa Info

You can invoke it from the command-line:

```
pyvisa-info
```

that will print information to diagnose PyVISA, info about Machine, Python, backends, etc

```
Machine Details:
   Platform ID:    Windows
   Processor:      Intel64 Family 6
   ...
PyVISA Version: ...

Backends:
   ni:
      Version: 1.8 (bundled with PyVISA)
      ...
   py:
      Version: 0.2
      ...
  sim:
     Version: 0.3
     Spec version: 1.1
```

Summary

Cool, right? It will be great to have a GUI similar to NI-MAX, but we leave that to be developed outside PyVISA. Want to help? Let us know!

1.1.8 VISA resource names

If you use the method *open_resource()*, you must tell this function the *VISA resource name* of the instrument you want to connect to. Generally, it starts with the bus type, followed by a double colon `"::"`, followed by the number within the bus. For example,

```
GPIB::10
```

denotes the GPIB instrument with the number 10. If you have two GPIB boards and the instrument is connected to board number 1, you must write

```
GPIB1::10
```

As for the bus, things like `"GPIB"`, `"USB"`, `"ASRL"` (for serial/parallel interface) are possible. So for connecting to an instrument at COM2, the resource name is

```
ASRL2
```

(Since only one instrument can be connected with one serial interface, there is no double colon parameter.) However, most VISA systems allow aliases such as `"COM2"` or `"LPT1"`. You may also add your own aliases.

The resource name is case-insensitive. It doesn't matter whether you say `"ASRL2"` or `"asrl2"`. For further information, I have to refer you to a comprehensive VISA description like http://www.ni.com/pdf/manuals/370423a.pdf.

VISA Resource Syntax and Examples

(This is adapted from the VISA manual)

The following table shows the grammar for the address string. Optional string segments are shown in square brackets ([ ]).

Interface	Syntax
ENET-Serial INSTR	ASRL[0]::host address::serial port::INSTR
GPIB INSTR	GPIB[board]::primary address[::secondary address][::INSTR]
GPIB INTFC	GPIB[board]::INTFC
PXI BACKPLANE	PXI[interface]::chassis number::BACKPLANE
PXI INSTR	PXI[bus]::device[::function][::INSTR]
PXI INSTR	PXI[interface]::bus-device[.function][::INSTR]
PXI INSTR	PXI[interface]::CHASSISchassis number::SLOTslot number[::FUNCfunction][::INSTR]
PXI MEMACC	PXI[interface]::MEMACC
Remote NI-VISA	visa://host address[:server port]/remote resource
Serial INSTR	ASRLboard[::INSTR]
TCPIP INSTR	TCPIP[board]::host address[::LAN device name][::INSTR]
TCPIP SOCKET	TCPIP[board]::host address::port::SOCKET
USB INSTR	USB[board]::manufacturer ID::model code::serial number[::USB interface number][::INSTR]
USB RAW	USB[board]::manufacturer ID::model code::serial number[::USB interface number]::RAW
VXI BACKPLANE	VXI[board][::VXI logical address]::BACKPLANE
VXI INSTR	VXI[board]::VXI logical address[::INSTR]
VXI MEMACC	VXI[board]::MEMACC
VXI SERVANT	VXI[board]::SERVANT

Use the GPIB keyword to establish communication with GPIB resources. Use the VXI keyword for VXI resources via embedded, MXIbus, or 1394 controllers. Use the ASRL keyword to establish communication with an asynchronous serial (such as RS-232 or RS-485) device. Use the PXI keyword for PXI and PCI resources. Use the TCPIP keyword for Ethernet communication.

The following table shows the default value for optional string segments.

Optional String Segments	Default Value
board	0
GPIB secondary address	none
LAN device name	inst0
PXI bus	0
PXI function	0
USB interface number	lowest numbered relevant interface

The following table shows examples of address strings:

Address String	Description
ASRL::1.2.3.4::2::INSTR	A serial device attached to port 2 of the ENET Serial controller at address 1.2.3.4.
ASRL1::INSTR	A serial device attached to interface ASRL1.
GPIB::1::0::INSTR	A GPIB device at primary address 1 and secondary address 0 in GPIB interface 0.
GPIB2::INTFC	Interface or raw board resource for GPIB interface 2.
PXI::15::INSTR	PXI device number 15 on bus 0 with implied function 0.
PXI::2::BACKPLANE	Backplane resource for chassis 2 on the default PXI system, which is interface 0.
PXI::CHASSIS1::SLOT3	PXI device in slot number 3 of the PXI chassis configured as chassis 1.
PXI0::2-12.1::INSTR	PXI bus number 2, device 12 with function 1.
PXI0::MEMACC	PXI MEMACC session.
TCPIP::dev.company.com::INSTR	A TCPIP device using VXI-11 or LXI located at the specified address. This uses the default LAN Device Name of inst0.
TCPIP0::1.2.3.4::999::SOCKET	Raw TCPIP access to port 999 at the specified IP address.
USB::0x1234::125::A22-5::INSTR	A USB Test & Measurement class device with manufacturer ID 0x1234, model code 125, and serial number A22-5. This uses the device's first available USBTMC interface. This is usually number 0.
USB::0x5678::0x33::SN999::1::RAW	A USB raw class device with manufacturer ID 0x5678, model code 0x33, and serial number SN999. This uses the device's interface number 1.
visa://hostname/ASRL1::INSTR	The resource ASRL1::INSTR on the specified remote system.
VXI::1::BACKPLANE	Mainframe resource for chassis 1 on the default VXI system, which is interface 0.
VXI::MEMACC	Board-level register access to the VXI interface.
VXI0::1::INSTR	A VXI device at logical address 1 in VXI interface VXI0.
VXI0::SERVANT	Servant/device-side resource for VXI interface 0.

1.2 Advanced topics

This section of the documentation will cover the internal details of PyVISA. In particular, it will explain in details how PyVISA manage backends.

1.2.1 Architecture

PyVISA implements convenient and Pythonic programming in three layers:

1. Low-level: A wrapper around the shared visa library.

   The wrapper defines the argument types and response types of each function, as well as the conversions between Python objects and foreign types.

   You will normally not need to access these functions directly. If you do, it probably means that we need to improve layer 2.

   All level 1 functions are **static methods** of *VisaLibraryBase*.

   > **Warning:** Notice however that low-level functions might not be present in all backends. For broader compatibility, do no use this layer. All the functionality should is available via the next layer.

2. Middle-level: A wrapping Python function for each function of the shared visa library.

   These functions call the low-level functions, adding some code to deal with type conversions for functions that return values by reference. These functions also have comprehensive and Python friendly documentation.

You only need to access this layer if you want to control certain specific aspects of the VISA library which are not implemented by the corresponding resource class.

All level 2 functions are **bound methods** of *VisaLibraryBase*.

3. High-level: An object-oriented layer for *ResourceManager* and *Resource*.

   The *ResourceManager* implements methods to inspect connected resources. You also use this object to open other resources instantiating the appropriate *Resource* derived classes.

   Resource and the derived classes implement functions and attributes access to the underlying resources in a Pythonic way.

Most of the time you will only need to instantiate a *ResourceManager*. For a given resource, you will use the *open_resource()* method to obtain the appropriate object. If needed, you will be able to access the `VisaLibrary` object directly using the `visalib` attribute.

The `VisaLibrary` does the low-level calls. In the default IVI Backend, levels 1 and 2 are implemented in the same package called `pyvisa.ctwrapper` (which stands for ctypes wrapper). This package is included in PyVISA.

Other backends can be used just by passing the name of the backend to `ResourceManager` after the @ symbol. See more information in *A frontend for multiple backends*.

Calling middle- and low-level functions

After you have instantiated the *ResourceManager*:

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

you can access the corresponding `VisaLibrary` instance under the `visalib` attribute.

As an example, consider the VISA function `viMapAddress`. It appears in the low-level layer as the static method `viMapAddress` of `visalib` attributed and also appears in the middle-level layer as `map_address`.

You can recognize low and middle-level functions by their names. Low-level functions carry the same name as in the shared library, and they are prefixed by **vi**. Middle-level functions have a friendlier, more pythonic but still recognizable name. Typically, camelCase names where stripped from the leading **vi** and changed to underscore separated lower case names. The docs about these methods is located here *API*.

Low-level

You can access the low-level functions directly exposed as static methods, for example:

```
>>> rm.visalib.viMapAddress(<here goes the arguments>)
```

To call this functions you need to know the function declaration and how to interface it to python. To help you out, the `VisaLibrary` object also contains middle-level functions.

It is very likely that you will need to access the VISA constants using these methods. You can find the information about these constants here *Constants module*

Middle-level

The `VisaLibrary` object exposes the middle-level functions which are one-to-one mapped from the foreign library as bound methods.

Each middle-level function wraps one low-level function. In this case:

```
>>> rm.visalib.map_address(<here goes the arguments>)
```

The calling convention and types are handled by the wrapper.

1.2.2 A frontend for multiple backends

A small historical note might help to make this section clearer. So bear with with me for a couple of lines. Originally PyVISA was a Python wrapper to the VISA library. More specifically, it was `ctypes` wrapper around the NI-VISA. This approach worked fine but made it difficult to develop other ways to communicate with instruments in platforms where NI-VISA was not available. Users had to change their programs to use other packages with different API.

Since 1.6, PyVISA is a frontend to VISA. It provides a nice, Pythonic API and can connect to multiple backends. Each backend exposes a class derived from VisaLibraryBase that implements the low-level communication. The ctypes wrapper around IVI-VISA is the default backend (called **ivi**) and is bundled with PyVISA for simplicity. In general, IVI-VISA can be NI-VISA, Keysight VISA, R&S VISA, tekVISA etc. By default, it calls the library that is installed on your system as VISA library.

You can specify the backend to use when you instantiate the resource manager using the `@` symbol. Remembering that **ivi** is the default, this:

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

is the same as this:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager('@ivi')
```

You can still provide the path to the library if needed:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager('/path/to/lib@ivi')
```

Under the hood, the *`ResourceManager`* looks for the requested backend and instantiate the VISA library that it provides.

PyVISA locates backends by name. If you do:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager('@somename')
```

PyVISA will try to import a package/module named `pyvisa-somename` which should be installed in your system. This is a loosely coupled configuration free method. PyVISA does not need to know about any backend out there until you actually try to use it.

You can list the installed backends by running the following code in the command line:

```
pyvisa-info
```

Developing a new Backend

What does a minimum backend looks like? Quite simple:

```
from pyvisa.highlevel import VisaLibraryBase

class MyLibrary(VisaLibraryBase):
    pass

WRAPPER_CLASS = MyLibrary
```

Additionally you can provide a staticmethod named *get_debug_info* that should return a dictionary of debug information which is printed when you call `pyvisa-info`

Note: Your backend name should not end by `-script` or it will be discarded. This is because any script generated by setuptools containing the name pyvisa will be named `pyvisa-*-script` and they are obviously not backends. Examples are the `pyvisa-shell` and `pyvisa-info` scripts.

An important aspect of developing a backend is knowing which VisaLibraryBase method to implement and what API to expose.

A **complete** implementation of a VISA Library requires a lot of functions (basically almost all level 2 functions as described in *Architecture* (there is also a complete list at the bottom of this page). But a working implementation does not require all of them.

As a **very minimum** set you need:

- **open_default_resource_manager**: returns a session to the Default Resource Manager resource.
- **open**: Opens a session to the specified resource.
- **close**: Closes the specified session, event, or find list.
- **list_resources**: Returns a tuple of all connected devices matching query.

(you can get the signature below or here *Visa Library*)

But of course you cannot do anything interesting with just this. In general you will also need:

- **get_attribute**: Retrieves the state of an attribute.
- **set_atribute**: Sets the state of an attribute.

If you need to start sending bytes to MessageBased instruments you will require:

- **read**: Reads data from device or interface synchronously.
- **write**: Writes data to device or interface synchronously.

For other usages or devices, you might need to implement other functions. Is really up to you and your needs.

These functions should raise a `pyvisa.errors.VisaIOError` or emit a `pyvisa.errors.VisaIOWarning` if necessary.

Complete list of level 2 functions to implement:

```
def read_memory(self, session, space, offset, width, extended=False):
def write_memory(self, session, space, offset, data, width, extended=False):
def move_in(self, session, space, offset, length, width, extended=False):
def move_out(self, session, space, offset, length, data, width, extended=False):
def peek(self, session, address, width):
def poke(self, session, address, width, data):
def assert_interrupt_signal(self, session, mode, status_id):
def assert_trigger(self, session, protocol):
def assert_utility_signal(self, session, line):
```

(continues on next page)

(continued from previous page)

```
def buffer_read(self, session, count):
def buffer_write(self, session, data):
def clear(self, session):
def close(self, session):
def disable_event(self, session, event_type, mechanism):
def discard_events(self, session, event_type, mechanism):
def enable_event(self, session, event_type, mechanism, context=None):
def flush(self, session, mask):
def get_attribute(self, session, attribute):
def gpib_command(self, session, data):
def gpib_control_atn(self, session, mode):
def gpib_control_ren(self, session, mode):
def gpib_pass_control(self, session, primary_address, secondary_address):
def gpib_send_ifc(self, session):
def in_8(self, session, space, offset, extended=False):
def in_16(self, session, space, offset, extended=False):
def in_32(self, session, space, offset, extended=False):
def in_64(self, session, space, offset, extended=False):
def install_handler(self, session, event_type, handler, user_handle):
def list_resources(self, session, query='?*::INSTR'):
def lock(self, session, lock_type, timeout, requested_key=None):
def map_address(self, session, map_space, map_base, map_size,
def map_trigger(self, session, trigger_source, trigger_destination, mode):
def memory_allocation(self, session, size, extended=False):
def memory_free(self, session, offset, extended=False):
def move(self, session, source_space, source_offset, source_width, destination_space,
def move_asynchronously(self, session, source_space, source_offset, source_width,
def move_in_8(self, session, space, offset, length, extended=False):
def move_in_16(self, session, space, offset, length, extended=False):
def move_in_32(self, session, space, offset, length, extended=False):
def move_in_64(self, session, space, offset, length, extended=False):
def move_out_8(self, session, space, offset, length, data, extended=False):
def move_out_16(self, session, space, offset, length, data, extended=False):
def move_out_32(self, session, space, offset, length, data, extended=False):
def move_out_64(self, session, space, offset, length, data, extended=False):
def open(self, session, resource_name,
def open_default_resource_manager(self):
def out_8(self, session, space, offset, data, extended=False):
def out_16(self, session, space, offset, data, extended=False):
def out_32(self, session, space, offset, data, extended=False):
def out_64(self, session, space, offset, data, extended=False):
def parse_resource(self, session, resource_name):
def parse_resource_extended(self, session, resource_name):
def peek_8(self, session, address):
def peek_16(self, session, address):
def peek_32(self, session, address):
def peek_64(self, session, address):
def poke_8(self, session, address, data):
def poke_16(self, session, address, data):
def poke_32(self, session, address, data):
def poke_64(self, session, address, data):
def read(self, session, count):
def read_asynchronously(self, session, count):
def read_stb(self, session):
def read_to_file(self, session, filename, count):
def set_attribute(self, session, attribute, attribute_state):
def set_buffer(self, session, mask, size):
```

(continues on next page)

(continued from previous page)

```
def status_description(self, session, status):
def terminate(self, session, degree, job_id):
def uninstall_handler(self, session, event_type, handler, user_handle=None):
def unlock(self, session):
def unmap_address(self, session):
def unmap_trigger(self, session, trigger_source, trigger_destination):
def usb_control_in(self, session, request_type_bitmap_field, request_id, request_
↪value,
def usb_control_out(self, session, request_type_bitmap_field, request_id, request_
↪value,
def vxi_command_query(self, session, mode, command):
def wait_on_event(self, session, in_event_type, timeout):
def write(self, session, data):
def write_asynchronously(self, session, data):
def write_from_file(self, session, filename, count):
```

1.3 Frequently asked questions

This section covers frequently asked questions in relation with PyVISA. You will find first miscellaneous questions and next a set of questions that requires more in depth answers.

1.3.1 Miscellaneous questions

Is *PyVISA* endorsed by National Instruments?

No. *PyVISA* is developed independently of National Instrument as a wrapper for the VISA library.

Who makes *PyVISA*?

PyVISA was originally programmed by Torsten Bronger and Gregor Thalhammer. It is based on earlier experiences by Thalhammer.

It was maintained from March 2012 to August 2013 by Florian Bauer. It was maintained from August 2013 to December 2017 by Hernan E. Grecco <hernan.grecco@gmail.com>. It is currently maintained by Matthieu Dartiailh <m.dartiailh@gmail.com>

Take a look at AUTHORS for more information

Is PyVISA thread-safe?

Yes, PyVISA is thread safe starting from version 1.6.

I have an error in my program and I am having trouble to fix it

PyVISA provides useful logs of all operations. Add the following commands to your program and run it again:

```
import pyvisa
pyvisa.log_to_screen()
```

I found a bug, how can I report it?

Please report it on the Issue Tracker, including operating system, python version and library version. In addition you might add supporting information by pasting the output of this command:

```
pyvisa-info
```

Error: Image not found

This error occurs when you have provided an invalid path for the VISA library. Check that the path provided to the constructor or in the configuration file

Error: Could not found VISA library

This error occurs when you have not provided a path for the VISA library and PyVISA is not able to find it for you. You can solve it by providing the library path to the `VisaLibrary` or `ResourceManager` constructor:

```
>>> visalib = VisaLibrary('/path/to/library')
```

or:

```
>>> rm = ResourceManager('Path to library')
```

or creating a configuration file as described in *Configuring the backend*.

Error: *visa* module has no attribute *ResourceManager*

The https://github.com/visa-sdk/visa-python provides a visa package that can conflict with `visa` module provided by PyVISA, which is why the `visa` module is deprecated and it is preferred to import `pyvisa` instead of `visa`. Both modules provides the same interface and no other changes should be needed.

Error: No matching architecture

This error occurs when you the Python architecture does not match the VISA architecture.

Note: PyVISA tries to parse the error from the underlying foreign function library to provide a more useful error message. If it does not succeed, it shows the original one.

In Mac OS X the original error message looks like this:

```
OSError: dlopen(/Library/Frameworks/visa.framework/visa, 6): no suitable image found.␣
↪ Did find:
    /Library/Frameworks/visa.framework/visa: no matching architecture in universal␣
↪wrapper
    /Library/Frameworks/visa.framework/visa: no matching architecture in universal␣
↪wrapper
```

In Linux the original error message looks like this:

```
OSError: Could not open VISA library:
    Error while accessing /usr/local/vxipnp/linux/bin/libvisa.so.7:/usr/local/vxipnp/
↪linux/bin/libvisa.so.7: wrong ELF class: ELFCLASS32
```

First, determine the details of your installation with the help of the following debug command:

```
pyvisa-info
```

You will see the 'bitness' of the Python interpreter and at the end you will see the list of VISA libraries that PyVISA was able to find.

The solution is to:

1. Install and use a VISA library matching your Python 'bitness'

   Download and install it from **National Instruments's VISA**. Run the debug command again to see if the new library was found by PyVISA. If not, create a configuration file as described in *Configuring the backend*.

   If there is no VISA library with the correct bitness available, try solution 2.

or

2. Install and use a Python matching your VISA library 'bitness'

   In Windows and Linux: Download and install Python with the matching bitness. Run your script again using the new Python

   In Mac OS X, Python is usually delivered as universal binary (32 and 64 bits).

   You can run it in 32 bit by running:

   ```
   arch -i386 python myscript.py
   ```

   or in 64 bits by running:

   ```
   arch -x86_64 python myscript.py
   ```

   You can create an alias by adding the following line

   alias python32="arch -i386 python"

   into your .bashrc or .profile or ~/.bash_profile (or whatever file depending on which shell you are using.)

   You can also create a virtual environment for this.

Where can I get more information about VISA?

- The original VISA docs:
  - VISA specification (scroll down to the end)
  - VISA library specification
  - VISA specification for textual languages
- The very good VISA manuals from National Instruments's VISA:
  - NI-VISA User Manual
  - NI-VISA Programmer Reference Manual
  - NI-VISA help file in HTML

1.3.2 NI-VISA Installation

In every OS, the NI-VISA library bitness (i.e. 32- or 64-bit) has to match the Python bitness. So first you need to install a NI-VISA that works with your OS and then choose the Python version matching the installed NI-VISA bitness.

PyVISA includes a debugging command to help you troubleshoot this (and other things):

```
pyvisa-info
```

According to National Instruments, NI VISA **17.5** is available for the following platforms.

Note: If NI-VISA is not available for your system, take a look at the *Frequently asked questions*.

Mac OS X

Download NI-VISA for Mac OS X

Supports:

- Mac OS X 10.7.x x86 and x86-64
- Mac OS X 10.8.x

64-bit VISA applications are supported for a limited set of instrumentation buses. The supported buses are ENET-Serial, USB, and TCPIP. Logging VISA operations in NI I/O Trace from 64-bit VISA applications is not supported.

Windows

Download NI-VISA for Windows

Suports:

- Windows Server 2003 R2 (32-bit version only)
- Windows Server 2008 R2 (64-bit version only)
- Windows 8 x64 Edition (64-bit version)
- Windows 8 (32-bit version)
- Windows 7 x64 Edition (64-bit version)
- Windows 7 (32-bit version)
- Windows Vista x64 Edition (64-bit version)
- Windows Vista (32-bit version)
- Windows XP Service Pack 3

Support for Windows Server 2003 R2 may require disabling physical address extensions (PAE).

Linux

Download NI-VISA for Linux

Supports:

- openSUSE 12.2

- openSUSE 12.1
- Red Hat Enterprise Linux Desktop + Workstation 6
- Red Hat Enterprise Linux Desktop + Workstation 5
- Scientific Linux 6.x
- Scientific Linux 5.x

More details details can be found in the README of the installer.

Note: NI-VISA runs on other linux distros but the installation is more cumbersome. On Arch linux and related distributions, the AUR package ni-visa (early development) is known to work for the USB and TCPIP interfaces. Please note that you should restart after the installation for things to work properly.

1.3.3 Migrating from PyVISA < 1.5

Note: if you want PyVISA 1.4 compatibility use PyVISA 1.5 that provides Python 3 support, better visa library detection heuristics, Windows, Linux and OS X support, and no singleton object. PyVISA 1.6+ introduces a few compatibility breaks.

Some of these decisions were inspired by the `visalib` package as a part of Lantz

Short summary

PyVISA 1.5 has full compatibility with previous versions of PyVISA using the legacy module (changing some of the underlying implementation). But you are encouraged to do a few things differently if you want to keep up with the latest developments and be compatible with PyVISA > 1.5.

Indeed PyVISA 1.6 breaks compatibility to bring across a few good things.

If you are doing:

```
>>> import pyvisa
>>> keithley = pyvisa.instrument("GPIB::12")
>>> print(keithley.ask("*IDN?"))
```

change it to:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager()
>>> keithley = rm.open_resource("GPIB::12")
>>> print(keithley.query("*IDN?"))
```

If you are doing:

```
>>> print(pyvisa.get_instruments_list())
```

change it to:

```
>>> print(rm.list_resources())
```

If you are doing:

```
>>> import pyvisa.vpp43 as vpp43
>>> vpp43.visa_library.load_library("/path/to/my/libvisa.so.7")
```

change it to:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager("/path/to/my/libvisa.so.7")
>>> lib = rm.visalib
```

If you are doing::

```
>>> vpp43.lock(session)
```

change it to:

```
>>> lib.lock(session)
```

or better:

```
>>> resource.lock()
```

If you are doing::

```
>>> inst.term_chars = '\r'
```

change it to:

```
>>> inst.read_termination = '\r'
>>> inst.write_termination = '\r'
```

If you are doing::

```
>>> print(lib.status)
```

change it to:

```
>>> print(lib.last_status)
```

or even better, do it per resource:

```
>>> print(rm.last_status) # for the resource manager
>>> print(inst.last_status) # for a specific instrument
```

If you are doing::

```
>>> inst.timeout = 1  # Seconds
```

change it to:

```
>>> inst.timeout = 1000  # Milliseconds
```

As you see, most of the code shown above is making a few things explict. It adds 1 line of code (instantiating the ResourceManager object) which is not a big deal but it makes things cleaner.

If you were using `printf`, `queryf`, `scanf`, `sprintf` or `sscanf` of `vpp43`, rewrite as pure Python code (see below).

If you were using `Instrument.delay`, change your code or use `Instrument.query_delay` (see below).

A few alias has been created to ease the transition:

- ask -> query
- ask_delay -> query_delay
- get_instrument -> open_resource

A more detailed description

Dropped support for string related functions

The VISA library includes functions to search and manipulate strings such as `printf`, `queryf`, `scanf`, `sprintf` and `sscanf`. This makes sense as VISA involves a lot of string handling operations. The original PyVISA implementation wrapped these functions. But these operations are easily expressed in pure python and therefore were rarely used.

PyVISA 1.5 keeps these functions for backwards compatibility but they are removed in 1.6.

We suggest that you replace such functions by a pure Python version.

Isolated low-level wrapping module

In the original PyVISA implementation, the low level implementation (`vpp43`) was mixed with higher level constructs. The VISA library was wrapped using ctypes.

In 1.5, we refactored it as `ctwrapper`. This allows us to test the foreign function calls by isolating them from higher level abstractions. More importantly, it also allows us to build new low level modules that can be used as drop in replacements for `ctwrapper` in high level modules.

In 1.6, we made the `ResourceManager` the object exposed to the user. The type of the `VisaLibrary` can selected depending of the `library_path` and obtained from a plugin package.

We have two of such packages planned:

- a Mock module that allows you to test a PyVISA program even if you do not have VISA installed.
- a CFFI based wrapper. CFFI is new python package that allows easier and more robust wrapping of foreign libraries. It might be part of Python in the future.

PyVISA 1.5 keeps `vpp43` in the legacy subpackage (reimplemented on top of `ctwrapper`) to help with the migration. This module is gone in 1.6.

All functions that were present in `vpp43` are now present in `ctwrapper` but they take an additional first parameter: the foreign library wrapper.

We suggest that you replace `vpp43` by accessing the `VisaLibrary` object under the attribute visalib of the resource manager which provides all foreign functions as bound methods (see below).

No singleton objects

The original PyVISA implementation relied on a singleton, global objects for the library wrapper (named `visa_library`, an instance of the old `pyvisa.vpp43.VisaLibrary`) and the resource manager (named `resource_manager`, and instance of the old `pyvisa.visa.ResourceManager`). These were instantiated on import and the user could rebind to a different library using the `load_library` method. Calling this method however did not affect `resource_manager` and might lead to an inconsistent state.

There were additionally a few global structures such a `status` which stored the last status returned by the library and the warning context to prevent unwanted warnings.

In 1.5, there is a new `VisaLibrary` class and a new `ResourceManager` class (they are both in `pyvisa.highlevel`). The new classes are not singletons, at least not in the strict sense. Multiple instances of `VisaLibrary` and `ResourceManager` are possible, but only if they refer to different foreign libraries. In code, this means:

```
>>> lib1 = pyvisa.VisaLibrary("/path/to/my/libvisa.so.7")
>>> lib2 = pyvisa.VisaLibrary("/path/to/my/libvisa.so.7")
>>> lib3 = pyvisa.VisaLibrary("/path/to/my/libvisa.so.8")
>>> lib1 is lib2
True
>>> lib1 is lib3
False
```

Most of the time, you will not need access to a `VisaLibrary` object but to a `ResourceManager`. You can do:

```
>>> lib = pyvisa.VisaLibrary("/path/to/my/libvisa.so.7")
>>> rm = lib.resource_manager
```

or equivalently:

```
>>> rm = pyvisa.ResourceManager("/path/to/my/libvisa.so.7")
```

Note: If the path for the library is not given, the path is obtained from the user settings file (if exists) or guessed from the OS.

In 1.6, the state returned by the library is stored per resource. Additionally, warnings can be silenced by resource as well. You can access with the `last_status` property.

All together, these changes makes PyVISA thread safe.

VisaLibrary methods as way to call Visa functions

In the original PyVISA implementation, the `VisaLibrary` class was just having a reference to the ctypes library and a few functions.

In 1.5, we introduced a new `VisaLibrary` class (`pyvisa.highlevel`) which has every single low level function defined in `ctwrapper` as bound methods. In code, this means that you can do:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager("/path/to/my/libvisa.so.7")
>>> lib = rm.visalib
>>> print(lib.read_stb(session))
```

(But it is very likely that you do not have to do it as the resource should have the function you need)

It also has every single VISA foreign function in the underlying library as static method. In code, this means that you can do:

```
>>> status = ctypes.c_ushort()
>>> ret lib.viReadSTB(session, ctypes.byref(status))
>>> print(ret.value)
```

Ask vs. query

Historically, the method `ask` has been used in PyVISA to do a `write` followed by a `read`. But in many other programs this operation is called `query`. Thereby we have decided to switch the name, keeping an alias to help with the transition.

However, `ask_for_values` has not been aliased to `query_values` because the API is different. `ask_for_values` still uses the old formatting API which is limited and broken. We suggest that you migrate everything to `query_values`

Seconds to milliseconds

The timeout is now in milliseconds (not in seconds as it was before). The reason behind this change is to make it coherent with all other VISA implementations out there. The C-API, LabVIEW, .NET: all use milliseconds. Using the same units not only makes it easy to migrate to PyVISA but also allows to profit from all other VISA docs out there without extra cognitive effort.

Removal of Instrument.delay and added Instrument.query_delay

In the original PyVISA implementation, `Instrument` takes a `delay` argument that adds a pause after each write operation (This also can be changed using the `delay` attribute).

In PyVISA 1.6, `delay` is removed. Delays after write operations must be added to the application code. Instead, a new attribute and argument `query_delay` is available. This allows you to pause between `write` and ``read` operations inside `query`. Additionally, `query` takes an optional argument called `query` allowing you to change it for each method call.

Deprecated term_chars and automatic removal of CR + LF

In the original PyVISA implementation, `Instrument` takes a `term_chars` argument to change at the read and write termination characters. If this argument is `None`, `CR + LF` is appended to each outgoing message and not expected for incoming messages (although removed if present).

In PyVISA 1.6, `term_chars` is replaced by `read_termination` and ``write_termination`. In this way, you can set independently the termination for each operation. Automatic removal of `CR + LF` is also gone in 1.6.

1.3.4 Contributing to PyVISA

You can contribute in different ways:

Report issues

You can report any issues with the package, the documentation to the PyVISA issue tracker. Also feel free to submit feature requests, comments or questions. In some cases, platform specific information is required. If you think this is the case, run the following command and paste the output into the issue:

```
pyvisa-info
```

It is useful that you also provide the log output. To obtain it, add the following lines to your code:

```
import pyvisa
pyvisa.log_to_screen()
```

If your issue concern a specific instrument please be sure to indicate the manufacturer and the model.

Contribute code

To contribute fixes, code or documentation to PyVISA, send us a patch, or fork PyVISA in github and submit the changes using a pull request.

You can also get the code from PyPI or GitHub. You can either clone the public repository:

```
$ git clone git://github.com/pyvisa/pyvisa.git
```

Download the tarball:

```
$ curl -OL https://github.com/pyvisa/pyvisa/tarball/master
```

Or, download the zipball:

```
$ curl -OL https://github.com/pyvisa/pyvisa/zipball/master
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

Note: If you have an old system installation of Python and you don't want to mess with it, you can try Anaconda. It is a free Python distribution by Continuum Analytics that includes many scientific packages.

Contributing to an existing backend

Backends are the central piece of PyVISA as they provide the low level communication over the different interfaces. There a couple of backends in the wild which can use your help. Look them up in PyPI (try *pyvisa* in the search box) and see where you can help.

Contributing a new backend

If you think there is a new way that low level communication can be achieved, go for it. You can use any of the existing backends as a template or start a thread in the issue tracker and we will be happy to help you.

1.4 API

1.4.1 Visa Library

class pyvisa.highlevel.**VisaLibraryBase**

Base for VISA library classes.

A class derived from *VisaLibraryBase* library provides the low-level communication to the underlying devices providing Pythonic wrappers to VISA functions. But not all derived class must/will implement all methods.

The default VisaLibrary class is `pyvisa.ctwrapper.highlevel.IVIVisaLibrary`, which implements a ctypes wrapper around the IVI-VISA library. Certainly, IVI-VISA can be NI-VISA, Keysight VISA, R&S VISA, tekVISA etc.

In general, you should not instantiate it directly. The object exposed to the user is the *pyvisa.highlevel.ResourceManager*. If needed, you can access the VISA library from it:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager("/path/to/my/libvisa.so.7")
>>> lib = rm.visalib
```

assert_interrupt_signal (*session*, *mode*, *status_id*)
Asserts the specified interrupt or signal.

Corresponds to viAssertIntrSignal function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **mode** – How to assert the interrupt. (Constants.ASSERT*)
- **status_id** – This is the status value to be presented during an interrupt acknowledge cycle.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

assert_trigger (*session*, *protocol*)
Asserts software or hardware trigger.

Corresponds to viAssertTrigger function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **protocol** – Trigger protocol to use during assertion. (Constants.PROT*)

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

assert_utility_signal (*session*, *line*)
Asserts or deasserts the specified utility bus signal.

Corresponds to viAssertUtilSignal function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **line** – specifies the utility bus signal to assert. (Constants.VI_UTIL_ASSERT*)

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

buffer_read (*session*, *count*)
Reads data from device or interface through the use of a formatted I/O read buffer.

Corresponds to viBufRead function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.

- **count** – Number of bytes to be read.

Returns data read, return value of the library call.

Return type bytes, *pyvisa.constants.StatusCode*

buffer_write(*session*, *data*)
Writes data to a formatted I/O write buffer synchronously.

Corresponds to viBufWrite function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **data** (*bytes*) – data to be written.

Returns number of written bytes, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

clear(*session*)
Clears a device.

Corresponds to viClear function of the VISA library.

Parameters **session** – Unique logical identifier to a session.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

close(*session*)
Closes the specified session, event, or find list.

Corresponds to viClose function of the VISA library.

Parameters **session** – Unique logical identifier to a session, event, or find list.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

disable_event(*session*, *event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Corresponds to viDisableEvent function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

discard_events(*session*, *event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in a session.

Corresponds to viDiscardEvents function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be discarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

enable_event(*session*, *event_type*, *mechanism*, *context=None*)

Enable event occurrences for specified event types and mechanisms in a session.

Corresponds to viEnableEvent function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** –

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

flush(*session*, *mask*)

Manually flushes the specified buffers associated with formatted I/O operations and/or serial communication.

Corresponds to viFlush function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **mask** – Specifies the action to be taken with flushing the buffer. The following values (defined in the constants module can be combined using the | operator. However multiple operations on a single buffer cannot be combined.
  - VI_READ_BUF: Discard the read buffer contents and if data was present in the read buffer and no END-indicator was present, read from the device until encountering an END indicator (which causes the loss of data).
  - VI_READ_BUF_DISCARD: Discard the read buffer contents (does not perform any I/O to the device).
  - VI_WRITE_BUF: Flush the write buffer by writing all buffered data to the device.
  - VI_WRITE_BUF_DISCARD: Discard the write buffer contents (does not perform any I/O to the device).
  - VI_IO_IN_BUF: Discards the receive buffer contents (same as VI_IO_IN_BUF_DISCARD).
  - VI_IO_IN_BUF_DISCARD: Discard the receive buffer contents (does not perform any I/O to the device).
  - VI_IO_OUT_BUF: Flush the transmit buffer by writing all buffered data to the device.
  - VI_IO_OUT_BUF_DISCARD: Discard the transmit buffer contents (does not perform any I/O to the device).

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

get_attribute(*session*, *attribute*)
Retrieves the state of an attribute.

Corresponds to viGetAttribute function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session, event, or find list.
- **attribute** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource, return value of the library call.

Return type unicode (Py2) or str (Py3), list or other type, *pyvisa.constants.StatusCode*

static get_debug_info()
Override this method to return an iterable of lines with the backend debug details.

get_last_status_in_session(*session*)
Last status in session.

Helper function to be called by resources properties.

static get_library_paths()
Override this method to return an iterable of possible library_paths to try in case that no argument is given.

gpib_command(*session*, *data*)
Write GPIB command bytes on the bus.

Corresponds to viGpibCommand function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **data** (*bytes*) – data tor write.

Returns Number of written bytes, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

gpib_control_atn(*session*, *mode*)
Specifies the state of the ATN line and the local active controller state.

Corresponds to viGpibControlATN function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **mode** – Specifies the state of the ATN line and optionally the local active controller state. (Constants.VI_GPIB_ATN*)

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

gpib_control_ren(*session*, *mode*)
Controls the state of the GPIB Remote Enable (REN) interface line, and optionally the remote/local state of the device.

Corresponds to viGpibControlREN function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **mode** – Specifies the state of the REN line and optionally the device remote/local state. (Constants.VI_GPIB_REN*)

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

gpib_pass_control(*session*, *primary_address*, *secondary_address*)
Tell the GPIB device at the specified address to become controller in charge (CIC).

Corresponds to viGpibPassControl function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **primary_address** – Primary address of the GPIB device to which you want to pass control.
- **secondary_address** – Secondary address of the targeted GPIB device. If the targeted device does not have a secondary address, this parameter should contain the value Constants.VI_NO_SEC_ADDR.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

gpib_send_ifc(*session*)
Pulse the interface clear line (IFC) for at least 100 microseconds.

Corresponds to viGpibSendIFC function of the VISA library.

Parameters **session** – Unique logical identifier to a session.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

handlers = None
Contains all installed event handlers. Its elements are tuples with three elements: The handler itself (a Python callable), the user handle (as a ct object) and the handler again, this time as a ct object created with CFUNCTYPE.

ignore_warning(*session*, **warnings_constants*)
A session dependent context for ignoring warnings

Parameters

- **session** – Unique logical identifier to a session.
- **warnings_constants** – constants identifying the warnings to ignore.

in_16(*session*, *space*, *offset*, *extended=False*)
Reads in an 16-bit value from the specified memory space and offset.

Corresponds to viIn16* function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

in_32 (*session*, *space*, *offset*, *extended=False*)
Reads in an 32-bit value from the specified memory space and offset.

Corresponds to viIn32* function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

in_64 (*session*, *space*, *offset*, *extended=False*)
Reads in an 64-bit value from the specified memory space and offset.

Corresponds to viIn64* function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

in_8 (*session*, *space*, *offset*, *extended=False*)
Reads in an 8-bit value from the specified memory space and offset.

Corresponds to viIn8* function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

install_handler (*session*, *event_type*, *handler*, *user_handle*)
Installs handlers for event callbacks.

Corresponds to viInstallHandler function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns a handler descriptor which consists of three elements: - handler (a python callable) - user handle (a ctypes object) - ctypes handler (ctypes object wrapping handler) and return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

install_visa_handler(*session*, *event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks.

Parameters

- **session** – Unique logical identifier to a session.
- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

issue_warning_on = None
Set error codes on which to issue a warning. set

last_status
Last return value of the library.

list_resources(*session*, *query='?*::INSTR'*)
Returns a tuple of all connected devices matching query.

Parameters **query** – regular expression used to match devices.

lock(*session*, *lock_type*, *timeout*, *requested_key=None*)
Establishes an access mode to the specified resources.

Corresponds to viLock function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **lock_type** – Specifies the type of lock requested, either Constants.EXCLUSIVE_LOCK or Constants.SHARED_LOCK.
- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error.
- **requested_key** – This parameter is not used and should be set to VI_NULL when lockType is VI_EXCLUSIVE_LOCK.

Returns access_key that can then be passed to other sessions to share the lock, return value of the library call.

Return type str, *pyvisa.constants.StatusCode*

map_address(*session*, *map_space*, *map_base*, *map_size*, *access=False*, *suggested=None*)
Maps the specified memory space into the process's address space.

Corresponds to viMapAddress function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **map_space** – Specifies the address space to map. (Constants.*SPACE*)
- **map_base** – Offset (in bytes) of the memory to be mapped.
- **map_size** – Amount of memory to map (in bytes).
- **access** –
- **suggested** – If not Constants.VI_NULL (0), the operating system attempts to map the memory to the address specified in suggested. There is no guarantee, however, that the memory will be mapped to that address. This operation may map the memory into an address region different from suggested.

Returns address in your process space where the memory was mapped, return value of the library call.

Return type address, *pyvisa.constants.StatusCode*

map_trigger(*session*, *trigger_source*, *trigger_destination*, *mode*)
Map the specified trigger source line to the specified destination line.

Corresponds to viMapTrigger function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **trigger_source** – Source line from which to map. (Constants.VI_TRIG*)
- **trigger_destination** – Destination line to which to map. (Constants.VI_TRIG*)
- **mode** –

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

memory_allocation(*session*, *size*, *extended=False*)
Allocates memory from a resource's memory region.

Corresponds to viMemAlloc* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **size** – Specifies the size of the allocation.
- **extended** – Use 64 bits offset independent of the platform.

Returns offset of the allocated memory, return value of the library call.

Return type offset, *pyvisa.constants.StatusCode*

memory_free(*session*, *offset*, *extended=False*)
Frees memory previously allocated using the memory_allocation() operation.

Corresponds to viMemFree* function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **offset** – Offset of the memory to free.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

move(*session*, *source_space*, *source_offset*, *source_width*, *destination_space*, *destination_offset*, *destination_width*, *length*)
Moves a block of data.

Corresponds to viMove function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **source_space** – Specifies the address space of the source.
- **source_offset** – Offset of the starting address or register from which to read.
- **source_width** – Specifies the data width of the source.
- **destination_space** – Specifies the address space of the destination.
- **destination_offset** – Offset of the starting address or register to which to write.
- **destination_width** – Specifies the data width of the destination.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

move_asynchronously(*session*, *source_space*, *source_offset*, *source_width*, *destination_space*, *destination_offset*, *destination_width*, *length*)
Moves a block of data asynchronously.

Corresponds to viMoveAsync function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **source_space** – Specifies the address space of the source.
- **source_offset** – Offset of the starting address or register from which to read.
- **source_width** – Specifies the data width of the source.
- **destination_space** – Specifies the address space of the destination.
- **destination_offset** – Offset of the starting address or register to which to write.
- **destination_width** – Specifies the data width of the destination.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.

Returns Job identifier of this asynchronous move operation, return value of the library call.

Return type jobid, *pyvisa.constants.StatusCode*

move_in(*session*, *space*, *offset*, *length*, *width*, *extended=False*)
Moves a block of data to local memory from the specified address space and offset.

Corresponds to viMoveIn* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from the bus, return value of the library call.

Return type list, *pyvisa.constants.StatusCode*

move_in_16(*session*, *space*, *offset*, *length*, *extended=False*)
Moves an 16-bit block of data from the specified address space and offset to local memory.

Corresponds to viMoveIn16* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from the bus, return value of the library call.

Return type list, *pyvisa.constants.StatusCode*

move_in_32(*session*, *space*, *offset*, *length*, *extended=False*)
Moves an 32-bit block of data from the specified address space and offset to local memory.

Corresponds to viMoveIn32* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from the bus, return value of the library call.

Return type list, *pyvisa.constants.StatusCode*

move_in_64(*session*, *space*, *offset*, *length*, *extended=False*)
Moves an 64-bit block of data from the specified address space and offset to local memory.

Corresponds to viMoveIn64* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from the bus, return value of the library call.

Return type list, *pyvisa.constants.StatusCode*

move_in_8(*session*, *space*, *offset*, *length*, *extended=False*)
Moves an 8-bit block of data from the specified address space and offset to local memory.

Corresponds to viMoveIn8* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from the bus, return value of the library call.

Return type list, *pyvisa.constants.StatusCode*

move_out(*session*, *space*, *offset*, *length*, *data*, *width*, *extended=False*)
Moves a block of data from local memory to the specified address space and offset.

Corresponds to viMoveOut* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

move_out_16(*session*, *space*, *offset*, *length*, *data*, *extended=False*)
Moves an 16-bit block of data from local memory to the specified address space and offset.

Corresponds to viMoveOut16* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

move_out_32(*session*, *space*, *offset*, *length*, *data*, *extended=False*)
Moves an 32-bit block of data from local memory to the specified address space and offset.

Corresponds to viMoveOut32* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

move_out_64(*session*, *space*, *offset*, *length*, *data*, *extended=False*)
Moves an 64-bit block of data from local memory to the specified address space and offset.

Corresponds to viMoveOut64* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

move_out_8(*session*, *space*, *offset*, *length*, *data*, *extended=False*)
Moves an 8-bit block of data from local memory to the specified address space and offset.

Corresponds to viMoveOut8* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

Corresponds to viMoveOut8 function of the VISA library.

open(*session*, *resource_name*, *access_mode=<AccessModes.no_lock: 0>*, *open_timeout=0*)
Opens a session to the specified resource.

Corresponds to viOpen function of the VISA library.

Parameters

- **session** – Resource Manager session (should always be a session returned from open_default_resource_manager()).
- **resource_name** – Unique symbolic name of a resource.
- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the `access_mode` parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

Returns Unique logical identifier reference to a session, return value of the library call.

Return type session, *pyvisa.constants.StatusCode*

open_default_resource_manager()
This function returns a session to the Default Resource Manager resource.

Corresponds to viOpenDefaultRM function of the VISA library.

Returns Unique logical identifier to a Default Resource Manager session, return value of the library call.

Return type session, *pyvisa.constants.StatusCode*

out_16(*session*, *space*, *offset*, *data*, *extended=False*)
Write in an 16-bit value from the specified memory space and offset.

Corresponds to viOut16* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

out_32(*session*, *space*, *offset*, *data*, *extended=False*)
Write in an 32-bit value from the specified memory space and offset.

Corresponds to viOut32* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

out_64(*session*, *space*, *offset*, *data*, *extended=False*)
Write in an 64-bit value from the specified memory space and offset.

Corresponds to viOut64* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

out_8(*session*, *space*, *offset*, *data*, *extended=False*)
Write in an 8-bit value from the specified memory space and offset.

Corresponds to viOut8* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.

- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

parse_resource(*session*, *resource_name*)
Parse a resource string to get the interface information.

Corresponds to viParseRsrc function of the VISA library.

Parameters

- **session** – Resource Manager session (should always be the Default Resource Manager for VISA returned from open_default_resource_manager()).
- **resource_name** – Unique symbolic name of a resource.

Returns Resource information with interface type and board number, return value of the library call.

Return type *pyvisa.highlevel.ResourceInfo*, *pyvisa.constants.StatusCode*

parse_resource_extended(*session*, *resource_name*)
Parse a resource string to get extended interface information.

Corresponds to viParseRsrcEx function of the VISA library.

Parameters

- **session** – Resource Manager session (should always be the Default Resource Manager for VISA returned from open_default_resource_manager()).
- **resource_name** – Unique symbolic name of a resource.

Returns Resource information, return value of the library call.

Return type *pyvisa.highlevel.ResourceInfo*, *pyvisa.constants.StatusCode*

peek(*session*, *address*, *width*)
Read an 8, 16, 32, or 64-bit value from the specified address.

Corresponds to viPeek* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.
- **width** – Number of bits to read.

Returns Data read from bus, return value of the library call.

Return type bytes, *pyvisa.constants.StatusCode*

peek_16(*session*, *address*)
Read an 16-bit value from the specified address.

Corresponds to viPeek16 function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.

Returns Data read from bus, return value of the library call.

Return type bytes, *pyvisa.constants.StatusCode*

peek_32 (*session*, *address*)
Read an 32-bit value from the specified address.

Corresponds to viPeek32 function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.

Returns Data read from bus, return value of the library call.

Return type bytes, *pyvisa.constants.StatusCode*

peek_64 (*session*, *address*)
Read an 64-bit value from the specified address.

Corresponds to viPeek64 function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.

Returns Data read from bus, return value of the library call.

Return type bytes, *pyvisa.constants.StatusCode*

peek_8 (*session*, *address*)
Read an 8-bit value from the specified address.

Corresponds to viPeek8 function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.

Returns Data read from bus, return value of the library call.

Return type bytes, *pyvisa.constants.StatusCode*

poke (*session*, *address*, *width*, *data*)
Writes an 8, 16, 32, or 64-bit value from the specified address.

Corresponds to viPoke* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.
- **width** – Number of bits to read.
- **data** – Data to be written to the bus.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

poke_16 (*session*, *address*, *data*)
Write an 16-bit value from the specified address.

Corresponds to viPoke16 function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.
- **data** – value to be written to the bus.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

poke_32 (*session*, *address*, *data*)
Write an 32-bit value from the specified address.

Corresponds to viPoke32 function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.
- **data** – value to be written to the bus.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

poke_64 (*session*, *address*, *data*)
Write an 64-bit value from the specified address.

Corresponds to viPoke64 function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.
- **data** – value to be written to the bus.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

poke_8 (*session*, *address*, *data*)
Write an 8-bit value from the specified address.

Corresponds to viPoke8 function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **address** – Source address to read the value.
- **data** – value to be written to the bus.

Returns Data read from bus.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

read(*session*, *count*)
Reads data from device or interface synchronously.

Corresponds to viRead function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **count** – Number of bytes to be read.

Returns data read, return value of the library call.

Return type bytes, *pyvisa.constants.StatusCode*

read_asynchronously(*session*, *count*)
Reads data from device or interface asynchronously.

Corresponds to viReadAsync function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **count** – Number of bytes to be read.

Returns result, jobid, return value of the library call.

Return type ctypes buffer, jobid, *pyvisa.constants.StatusCode*

read_memory(*session*, *space*, *offset*, *width*, *extended=False*)
Reads in an 8-bit, 16-bit, 32-bit, or 64-bit value from the specified memory space and offset.

Corresponds to viIn* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

read_stb(*session*)
Reads a status byte of the service request.

Corresponds to viReadSTB function of the VISA library.

Parameters **session** – Unique logical identifier to a session.

Returns Service request status byte, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

read_to_file(*session*, *filename*, *count*)
Read data synchronously, and store the transferred data in a file.

Corresponds to viReadToFile function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.

- **filename** – Name of file to which data will be written.
- **count** – Number of bytes to be read.

Returns Number of bytes actually transferred, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

resource_manager = None
Default ResourceManager instance for this library.

set_attribute(*session*, *attribute*, *attribute_state*)
Sets the state of an attribute.

Corresponds to viSetAttribute function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **attribute** – Attribute for which the state is to be modified. (Attributes.*)
- **attribute_state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

set_buffer(*session*, *mask*, *size*)
Sets the size for the formatted I/O and/or low-level I/O communication buffer(s).

Corresponds to viSetBuf function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **mask** – Specifies the type of buffer. (Constants.VI_READ_BUF, .VI_WRITE_BUF, .VI_IO_IN_BUF, .VI_IO_OUT_BUF)
- **size** – The size to be set for the specified buffer(s).

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

status_description(*session*, *status*)
Returns a user-readable description of the status code passed to the operation.

Corresponds to viStatusDesc function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **status** – Status code to interpret.

Returns

- The user-readable string interpretation of the status code passed to the operation,
- return value of the library call.

Return type

- unicode (Py2) or str (Py3)
- *pyvisa.constants.StatusCode*

terminate(*session*, *degree*, *job_id*)
Requests a VISA session to terminate normal execution of an operation.

Corresponds to viTerminate function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **degree** – Constants.NULL
- **job_id** – Specifies an operation identifier.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

uninstall_all_visa_handlers(*session*)
Uninstalls all previously installed handlers for a particular session.

Parameters **session** – Unique logical identifier to a session. If None, operates on all sessions.

uninstall_handler(*session*, *event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events.

Corresponds to viUninstallHandler function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely in a session for an event.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

uninstall_visa_handler(*session*, *event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events.

Parameters

- **session** – Unique logical identifier to a session.
- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_visa_handler.

unlock(*session*)
Relinquishes a lock for the specified resource.

Corresponds to viUnlock function of the VISA library.

Parameters **session** – Unique logical identifier to a session.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

unmap_address(*session*)
Unmaps memory space previously mapped by map_address().

Corresponds to viUnmapAddress function of the VISA library.

Parameters **session** – Unique logical identifier to a session.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

unmap_trigger(*session*, *trigger_source*, *trigger_destination*)
Undo a previous map from the specified trigger source line to the specified destination line.

Corresponds to viUnmapTrigger function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **trigger_source** – Source line used in previous map. (Constants.VI_TRIG*)
- **trigger_destination** – Destination line used in previous map. (Constants.VI_TRIG*)

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

usb_control_in(*session*, *request_type_bitmap_field*, *request_id*, *request_value*, *index*, *length=0*)
Performs a USB control pipe transfer from the device.

Corresponds to viUsbControlIn function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **request_type_bitmap_field** – bmRequestType parameter of the setup stage of a USB control transfer.
- **request_id** – bRequest parameter of the setup stage of a USB control transfer.
- **request_value** – wValue parameter of the setup stage of a USB control transfer.
- **index** – wIndex parameter of the setup stage of a USB control transfer. This is usually the index of the interface or endpoint.
- **length** – wLength parameter of the setup stage of a USB control transfer. This value also specifies the size of the data buffer to receive the data from the optional data stage of the control transfer.

Returns

- The data buffer that receives the data from the optional data stage of the control transfer
- return value of the library call.

Return type

- bytes
- *pyvisa.constants.StatusCode*

usb_control_out(*session*, *request_type_bitmap_field*, *request_id*, *request_value*, *index*, *data=''*)
Performs a USB control pipe transfer to the device.

Corresponds to viUsbControlOut function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **request_type_bitmap_field** – bmRequestType parameter of the setup stage of a USB control transfer.
- **request_id** – bRequest parameter of the setup stage of a USB control transfer.
- **request_value** – wValue parameter of the setup stage of a USB control transfer.
- **index** – wIndex parameter of the setup stage of a USB control transfer. This is usually the index of the interface or endpoint.
- **data** – The data buffer that sends the data in the optional data stage of the control transfer.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

vxi_command_query(*session*, *mode*, *command*)

Sends the device a miscellaneous command or query and/or retrieves the response to a previous query.

Corresponds to viVxiCommandQuery function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **mode** – Specifies whether to issue a command and/or retrieve a response. (Constants.VI_VXI_CMD*, .VI_VXI_RESP*)
- **command** – The miscellaneous command to send.

Returns The response retrieved from the device, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

wait_on_event(*session*, *in_event_type*, *timeout*)

Waits for an occurrence of the specified event for a given session.

Corresponds to viWaitOnEvent function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds.

Returns

- Logical identifier of the event actually received
- A handle specifying the unique occurrence of an event
- return value of the library call.

Return type

- eventtype
- event
- *pyvisa.constants.StatusCode*

write(*session*, *data*)
Writes data to device or interface synchronously.

Corresponds to viWrite function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **data** (*str*) – data to be written.

Returns Number of bytes actually transferred, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

write_asynchronously(*session*, *data*)
Writes data to device or interface asynchronously.

Corresponds to viWriteAsync function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **data** – data to be written.

Returns Job ID of this asynchronous write operation, return value of the library call.

Return type jobid, *pyvisa.constants.StatusCode*

write_from_file(*session*, *filename*, *count*)
Take data from a file and write it out synchronously.

Corresponds to viWriteFromFile function of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **filename** – Name of file from which data will be read.
- **count** – Number of bytes to be written.

Returns Number of bytes actually transferred, return value of the library call.

Return type int, *pyvisa.constants.StatusCode*

write_memory(*session*, *space*, *offset*, *data*, *width*, *extended=False*)
Write in an 8-bit, 16-bit, 32-bit, 64-bit value to the specified memory space and offset.

Corresponds to viOut* functions of the VISA library.

Parameters

- **session** – Unique logical identifier to a session.
- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

1.4.2 Resource Manager

class pyvisa.highlevel.**ResourceInfo**(*interface_type*, *interface_board_number*, *resource_class*, *resource_name*, *alias*)

Resource extended information

Named tuple with information about a resource. Returned by some *ResourceManager* methods.

Interface_type Interface type of the given resource string. *pyvisa.constants.InterfaceType*

Interface_board_number Board number of the interface of the given resource string.

Resource_class Specifies the resource class (for example, "INSTR") of the given resource string.

Resource_name This is the expanded version of the given resource string. The format should be similar to the VISA-defined canonical resource name.

Alias Specifies the user-defined alias for the given resource string.

class pyvisa.highlevel.**ResourceManager**

VISA Resource Manager

Parameters **visa_library** – VisaLibrary Instance, path of the VISA library or VisaLibrary spec string. (if not given, the default for the platform will be used).

close()

Close the resource manager session.

last_status

Last status code returned for an operation with this Resource Manager

Return type *pyvisa.constants.StatusCode*

list_resources(*query='?*::INSTR'*)

Returns a tuple of all connected devices matching query.

note: The query uses the VISA Resource Regular Expression syntax - which is not the same

as the Python regular expression syntax. (see below)

The VISA Resource Regular Expression syntax is defined in the VISA Library specification: http://www.ivifoundation.org/docs/vpp43.pdf

Symbol Meaning ———- ———-

? Matches any one character.

Makes the character that follows it an ordinary character instead of special character. For example, when a question mark follows a backslash (?), it matches the ? character instead of any one character.

[list] Matches any one character from the enclosed list. You can use a hyphen to match a range of characters.

[^list] Matches any character not in the enclosed list. You can use a hyphen to match a range of characters.

- Matches 0 or more occurrences of the preceding character or expression.
- Matches 1 or more occurrences of the preceding character or expression.

Exp|exp Matches either the preceding or following expression. The or operator | matches the entire expression that precedes or follows it and not just the character that precedes or follows it. For example, VXI|GPIB means (VXI)|(GPIB), not VX(I|G)PIB.

(exp) Grouping characters or expressions.

Thus the default query, '?*::INSTR', matches any sequences of characters ending ending with '::INSTR'.

Parameters **query** – a VISA Resource Regular Expression used to match devices.

list_resources_info(*query='?*::INSTR'*)

Returns a dictionary mapping resource names to resource extended information of all connected devices matching query.

For details of the VISA Resource Regular Expression syntax used in query, refer to list_resources().

Parameters **query** – a VISA Resource Regular Expression used to match devices.

Returns Mapping of resource name to ResourceInfo

Return type dict[str, *pyvisa.highlevel.ResourceInfo*]

open_bare_resource(*resource_name*, *access_mode=<AccessModes.no_lock: 0>*, *open_timeout=0*)

Open the specified resource without wrapping into a class

Parameters

- **resource_name** – Name or alias of the resource to open.
- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

Returns Unique logical identifier reference to a session.

open_resource(*resource_name*, *access_mode=<AccessModes.no_lock: 0>*, *open_timeout=0*, *resource_pyclass=None*, ***kwargs*)

Return an instrument for the resource name.

Parameters

- **resource_name** – Name or alias of the resource to open.
- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.
- **resource_pyclass** – Resource Python class to use to instantiate the Resource. Defaults to None: select based on the resource name.
- **kwargs** – Keyword arguments to be used to change instrument attributes after construction.

Return type *pyvisa.resources.Resource*

resource_info(*resource_name*, *extended=True*)
Get the (extended) information of a particular resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

session
Resource Manager session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

1.4.3 Resource classes

Resources are high level abstractions to managing specific sessions. An instance of one of these classes is returned by the *open_resource()* depending on the resource type.

Generic classes

- *Resource*
- *MessageBasedResource*
- *RegisterBasedResource*

Specific Classes

- *SerialInstrument*
- *TCPIPInstrument*
- *TCPIPSocket*
- *USBInstrument*
- *USBRaw*
- *GPIBInstrument*
- *GPIBInterface*
- *FirewireInstrument*
- *PXIInstrument*
- *PXIInstrument*
- *VXIInstrument*
- *VXIMemory*
- *VXIBackplane*

class pyvisa.resources.**Resource**(*resource_manager*, *resource_name*)
Base class for resources.

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

Parameters

- **resource_manager** – A resource manager instance.
- **resource_name** – the VISA name for the resource (eg. “GPIB::10”)

before_close()
Called just before closing an instrument.

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)

Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info

Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

timeout
The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

class pyvisa.resources.**MessageBasedResource**(**args*, ***kwargs*)
Base class for resources that use message based communication.

CR = '\r'

LF = '\n'

assert_trigger()
Sends a software trigger to the device.

before_close()
Called just before closing an instrument.

chunk_size = 20480

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)

- **context** – Not currently used, leave as None.

encoding
Encoding used for read and write operations.

flush(*mask*)
Manually clears the specified buffers.

Depending on the value of the mask this can cause the buffer data to be written to the device.

Parameters **mask** – Specifies the action to be taken with flushing the buffer. See high-level.VisaLibraryBase.flush for a detailed description.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

query(*message*, *delay=None*)
A combination of write(message) and read()

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type str

query_ascii_values(*message*, *converter='f'*, *separator=','*, *container=<class 'list'>*, *delay=None*)
Query the device for values in ascii format returning an iterable of values.

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

query_binary_values(*message*, *datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *delay=None*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)
Query the device for values in binary format returning an iterable of values.

Parameters

- **message** – the message to send to the instrument.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type list

query_delay = 0.0

query_values(*message*, *delay=None*)
Query the device for values returning an iterable of values.

The datatype expected is obtained from *values_format*

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type list

read(*termination=None*, *encoding=None*)
Read a string from the device.

Reading stops when the device stops sending (e.g. by setting appropriate bus lines), or the termination characters sequence was detected. Attention: Only the last character of the termination characters is really used to stop reading, however, the whole sequence is compared to the ending of the read string message. If they don't match, a warning is issued.

All line-ending characters are stripped from the end of the string.

Return type str

read_ascii_values(*converter='f'*, *separator=','*, *container=<class 'list'>*)
Read values from the device in ascii format returning an iterable of values.

Parameters

- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

read_binary_values(*datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)
Read values from the device in binary format returning an iterable of values.

Parameters

- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'

- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type type(container)

read_bytes(*count*, *chunk_size=None*, *break_on_termchar=False*)
Read a certain number of bytes from the instrument.

Parameters

- **count** (*int*) – The number of bytes to read from the instrument.
- **chunk_size** (*int*) – The chunk size to use to perform the reading.
- **break_on_termchar** (*bool*) – Should the reading stop when a termination character is encountered.

Return type bytes

read_raw(*size=None*)
Read the unmodified string sent from the instrument to the computer.

In contrast to read(), no termination characters are stripped.

Parameters **size** – The chunk size to use when reading the data.

Return type bytes

read_stb()
Service request status register.

read_termination
Read termination character.

read_termination_context(*new_termination*)

read_values(*fmt=None*, *container=<class 'list'>*)
Read a list of floating point values from the device.

Parameters

- **fmt** – the format of the values. If given, it overrides the class attribute "values_format". Possible values are bitwise disjunctions of the above constants ascii, single, double, and big_endian. Default is ascii.
- **container** – the output datatype

Returns the list of read values

Return type list

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info
Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

stb
Service request status register.

timeout
The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

values_format

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write(*message*, *termination=None*, *encoding=None*)
Write a string message to the device.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **termination** (*unicode (Py2) or str (Py3)*) – alternative character termination to use.

- **encoding** (*unicode (Py2) or str (Py3)*) – encoding to convert from unicode to bytes.

Returns number of bytes written.

Return type int

write_ascii_values(*message*, *values*, *converter='f'*, *separator=','*, *termination=None*, *encoding=None*)

Write a string message to the device followed by values in ascii format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **converter** (*callable | str*) – function used to convert each value. String formatting codes are also accepted. Defaults to 'f'.
- **separator** – a callable that join the values in a single str. If a str is given, separator.join(values) is used.

Type separator: (collections.Iterable[T]) -> str | str

Returns number of bytes written.

Return type int

write_binary_values(*message*, *values*, *datatype='f'*, *is_big_endian=False*, *termination=None*, *encoding=None*, *header_fmt='ieee'*)

Write a string message to the device followed by values in binary format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'

Returns number of bytes written.

Return type int

write_raw(*message*)

Write a byte message to the device.

Parameters **message** (*bytes*) – the message to be sent.

Returns number of bytes written.

Return type int

write_termination

Writer termination character.

write_values(*message*, *values*, *termination=None*, *encoding=None*)

class pyvisa.resources.**RegisterBasedResource**(*resource_manager*, *resource_name*)
Base class for resources that use register based communication.

before_close()
Called just before closing an instrument.

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

move_in(*space*, *offset*, *length*, *width*, *extended=False*)
Moves a block of data to local memory from the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

move_out(*space*, *offset*, *length*, *data*, *width*, *extended=False*)
Moves a block of data from local memory to the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

read_memory(*space*, *offset*, *width*, *extended=False*)
Reads in an 8-bit, 16-bit, 32-bit, or 64-bit value from the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory.

Corresponds to viIn* functions of the visa library.

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info

Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

session

Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)

Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

timeout

The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)

Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()

Relinquishes a lock for the specified resource.

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)

Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write_memory(*space*, *offset*, *data*, *width*, *extended=False*)
Write in an 8-bit, 16-bit, 32-bit, value to the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Corresponds to viOut* functions of the visa library.

class pyvisa.resources.**SerialInstrument**(**args*, ***kwargs*)
Communicates with devices of type ASRL<board>[::INSTR]

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

CR = '\r'

LF = '\n'

allow_dma

This attribute specifies whether I/O accesses should use DMA (VI_TRUE) or Programmed I/O (VI_FALSE). In some implementations, this attribute may have global effects even though it is documented to be a local attribute. Since this affects performance and not functionality, that behavior is acceptable.

VISA Attribute VI_ATTR_DMA_ALLOW_EN (1073676318)

Type bool

allow_transmit

If set to VI_FALSE, it suspends transmission as if an XOFF character has been received. If set to VI_TRUE, it resumes transmission as if an XON character has been received.

VISA Attribute VI_ATTR_ASRL_ALLOW_TRANSMIT (1073676734)

Type bool

assert_trigger()
Sends a software trigger to the device.

baud_rate

VI_ATTR_ASRL_BAUD is the baud rate of the interface. It is represented as an unsigned 32-bit integer so that any baud rate can be used, but it usually requires a commonly used rate such as 300, 1200, 2400, or 9600 baud.

VISA Attribute VI_ATTR_ASRL_BAUD (1073676321)

Type int

Range 0 <= value <= 4294967295

before_close()
Called just before closing an instrument.

break_length

This controls the duration (in milliseconds) of the break signal asserted when VI_ATTR_ASRL_END_OUT is set to VI_ASRL_END_BREAK. If you want to control the assertion state and length of a break signal manually, use the VI_ATTR_ASRL_BREAK_STATE attribute instead.

VISA Attribute VI_ATTR_ASRL_BREAK_LEN (1073676733)

Type int

Range -32768 <= value <= 32767

break_state

If set to VI_STATE_ASSERTED, it suspends character transmission and places the transmission line in a break state until this attribute is reset to VI_STATE_UNASSERTED. This attribute lets you manually control the assertion state and length of a break signal. If you want VISA to send a break signal after each write operation automatically, use the VI_ATTR_ASRL_BREAK_LEN and VI_ATTR_ASRL_END_OUT attributes instead.

VISA Attribute VI_ATTR_ASRL_BREAK_STATE (1073676732)

Type :class:pyvisa.constants.LineState

bytes_in_buffer

VI_ATTR_ASRL_AVAIL_NUM shows the number of bytes available in the low- level I/O receive buffer.

VISA Attribute VI_ATTR_ASRL_AVAIL_NUM (1073676460)

Type int

Range 0 <= value <= 4294967295

chunk_size = 20480

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

data_bits

VI_ATTR_ASRL_DATA_BITS is the number of data bits contained in each frame (from 5 to 8). The data bits for each frame are located in the low-order bits of every byte stored in memory.

VISA Attribute VI_ATTR_ASRL_DATA_BITS (1073676322)

Type int

Range 5 <= value <= 8

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.

- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_null

If set to VI_TRUE, NUL characters are discarded. Otherwise, they are treated as normal data characters. For binary transfers, set this attribute to VI_FALSE.

VISA Attribute VI_ATTR_ASRL_DISCARD_NULL (1073676464)

Type bool

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

encoding
Encoding used for read and write operations.

end_input

VI_ATTR_ASRL_END_IN indicates the method used to terminate read operations.

VISA Attribute VI_ATTR_ASRL_END_IN (1073676467)

Type :class:pyvisa.constants.SerialTermination

flow_control

VI_ATTR_ASRL_FLOW_CNTRL indicates the type of flow control used by the transfer mechanism.

VISA Attribute VI_ATTR_ASRL_FLOW_CNTRL (1073676325)

Type int

Range 0 <= value <= 65535

flush(*mask*)
Manually clears the specified buffers.

Depending on the value of the mask this can cause the buffer data to be written to the device.

Parameters **mask** – Specifies the action to be taken with flushing the buffer. See high-level.VisaLibraryBase.flush for a detailed description.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

io_protocol

VI_ATTR_IO_PROT specifies which protocol to use. In VXI, you can choose normal word serial or fast data channel (FDC). In GPIB, you can choose normal or high-speed (HS-488) transfers. In serial, TCPIP, or USB RAW, you can choose normal transfers or 488.2-defined strings. In USB INSTR, you can choose normal or vendor-specific transfers.

VISA Attribute VI_ATTR_IO_PROT (1073676316)

Type int

Range 0 <= value <= 65535

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the `access_mode` parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

parity

VI_ATTR_ASRL_PARITY is the parity used with every frame transmitted and received.

VISA Attribute VI_ATTR_ASRL_PARITY (1073676323)

Type :class:pyvisa.constants.Parity

query(*message*, *delay=None*)
A combination of write(message) and read()

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type str

query_ascii_values(*message*, *converter='f'*, *separator=','*, *container=<class 'list'>*, *delay=None*)
Query the device for values in ascii format returning an iterable of values.

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

query_binary_values(*message*, *datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *delay=None*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)
Query the device for values in binary format returning an iterable of values.

Parameters

- **message** – the message to send to the instrument.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)

- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type list

query_delay = 0.0

query_values(*message*, *delay=None*)

Query the device for values returning an iterable of values.

The datatype expected is obtained from *values_format*

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type list

read(*termination=None*, *encoding=None*)

Read a string from the device.

Reading stops when the device stops sending (e.g. by setting appropriate bus lines), or the termination characters sequence was detected. Attention: Only the last character of the termination characters is really used to stop reading, however, the whole sequence is compared to the ending of the read string message. If they don't match, a warning is issued.

All line-ending characters are stripped from the end of the string.

Return type str

read_ascii_values(*converter='f'*, *separator=','*, *container=<class 'list'>*)

Read values from the device in ascii format returning an iterable of values.

Parameters

- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

read_binary_values(*datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)

Read values from the device in binary format returning an iterable of values.

Parameters

- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type type(container)

read_bytes(*count*, *chunk_size=None*, *break_on_termchar=False*)
Read a certain number of bytes from the instrument.

Parameters

- **count** (*int*) – The number of bytes to read from the instrument.
- **chunk_size** (*int*) – The chunk size to use to perform the reading.
- **break_on_termchar** (*bool*) – Should the reading stop when a termination character is encountered.

Return type bytes

read_raw(*size=None*)
Read the unmodified string sent from the instrument to the computer.

In contrast to read(), no termination characters are stripped.

Parameters **size** – The chunk size to use when reading the data.

Return type bytes

read_stb()
Service request status register.

read_termination
Read termination character.

read_termination_context(*new_termination*)

read_values(*fmt=None*, *container=<class 'list'>*)
Read a list of floating point values from the device.

Parameters

- **fmt** – the format of the values. If given, it overrides the class attribute "values_format". Possible values are bitwise disjunctions of the above constants ascii, single, double, and big_endian. Default is ascii.
- **container** – the output datatype

Returns the list of read values

Return type list

classmethod register(*interface_type*, *resource_class*)

replace_char

VI_ATTR_ASRL_REPLACE_CHAR specifies the character to be used to replace incoming characters that arrive with errors (such as parity error).

VISA Attribute VI_ATTR_ASRL_REPLACE_CHAR (1073676478)

Type int

Range 0 <= value <= 255

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info

Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

send_end

VI_ATTR_SEND_END_EN specifies whether to assert END during the transfer of the last byte of the buffer.

VISA Attribute VI_ATTR_SEND_END_EN (1073676310)

Type bool

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

stb
Service request status register.

stop_bits

VI_ATTR_ASRL_STOP_BITS is the number of stop bits used to indicate the end of a frame. The value VI_ASRL_STOP_ONE5 indicates one-and-one- half (1.5) stop bits.

VISA Attribute VI_ATTR_ASRL_STOP_BITS (1073676324)

Type :class:pyvisa.constants.StopBits

timeout
The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

values_format

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write(*message*, *termination=None*, *encoding=None*)
Write a string message to the device.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **termination** (*unicode (Py2) or str (Py3)*) – alternative character termination to use.
- **encoding** (*unicode (Py2) or str (Py3)*) – encoding to convert from unicode to bytes.

Returns number of bytes written.

Return type int

write_ascii_values(*message*, *values*, *converter='f'*, *separator=','*, *termination=None*, *encoding=None*)
Write a string message to the device followed by values in ascii format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **converter** (*callable | str*) – function used to convert each value. String formatting codes are also accepted. Defaults to 'f'.
- **separator** – a callable that join the values in a single str. If a str is given, separator.join(values) is used.

Type separator: (collections.Iterable[T]) -> str | str

Returns number of bytes written.

Return type int

write_binary_values(*message*, *values*, *datatype='f'*, *is_big_endian=False*, *termination=None*, *encoding=None*, *header_fmt='ieee'*)

Write a string message to the device followed by values in binary format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'

Returns number of bytes written.

Return type int

write_raw(*message*)

Write a byte message to the device.

Parameters **message** (*bytes*) – the message to be sent.

Returns number of bytes written.

Return type int

write_termination

Writer termination character.

write_values(*message*, *values*, *termination=None*, *encoding=None*)

xoff_char

VI_ATTR_ASRL_XOFF_CHAR specifies the value of the XOFF character used for XON/XOFF flow control (both directions). If XON/XOFF flow control (software handshaking) is not being used, the value of this attribute is ignored.

VISA Attribute VI_ATTR_ASRL_XOFF_CHAR (1073676482)

Type int

Range 0 <= value <= 255

xon_char

VI_ATTR_ASRL_XON_CHAR specifies the value of the XON character used for XON/XOFF flow control (both directions). If XON/XOFF flow control (software handshaking) is not being used, the value of this attribute is ignored.

VISA Attribute VI_ATTR_ASRL_XON_CHAR (1073676481)

Type int

Range 0 <= value <= 255

class pyvisa.resources.**TCPIPInstrument**(**args*, ***kwargs*)
Communicates with to devices of type TCPIP::host address[::INSTR]

More complex resource names can be specified with the following grammar: TCPIP[board]::host address[::LAN device name][::INSTR]

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

CR = '\r'

LF = '\n'

allow_dma

This attribute specifies whether I/O accesses should use DMA (VI_TRUE) or Programmed I/O (VI_FALSE). In some implementations, this attribute may have global effects even though it is documented to be a local attribute. Since this affects performance and not functionality, that behavior is acceptable.

VISA Attribute VI_ATTR_DMA_ALLOW_EN (1073676318)

Type bool

assert_trigger()
Sends a software trigger to the device.

before_close()
Called just before closing an instrument.

chunk_size = 20480

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

control_ren(*mode*)
Controls the state of the GPIB Remote Enable (REN) interface line, and optionally the remote/local state of the device.

Corresponds to viGpibControlREN function of the VISA library.

Parameters **mode** – Specifies the state of the REN line and optionally the device remote/local state. (Constants.GPIB_REN*)

Returns return value of the library call.

Return type VISAStatus

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

encoding
Encoding used for read and write operations.

flush(*mask*)
Manually clears the specified buffers.

Depending on the value of the mask this can cause the buffer data to be written to the device.

Parameters **mask** – Specifies the action to be taken with flushing the buffer. See high-level.VisaLibraryBase.flush for a detailed description.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.

- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

query(*message*, *delay=None*)
A combination of write(message) and read()

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type str

query_ascii_values(*message*, *converter='f'*, *separator=','*, *container=<class 'list'>*, *delay=None*)
Query the device for values in ascii format returning an iterable of values.

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

query_binary_values(*message*, *datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *delay=None*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)
Query the device for values in binary format returning an iterable of values.

Parameters

- **message** – the message to send to the instrument.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type list

query_delay = 0.0

query_values(*message*, *delay=None*)

Query the device for values returning an iterable of values.

The datatype expected is obtained from *values_format*

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type list

read(*termination=None*, *encoding=None*)

Read a string from the device.

Reading stops when the device stops sending (e.g. by setting appropriate bus lines), or the termination characters sequence was detected. Attention: Only the last character of the termination characters is really used to stop reading, however, the whole sequence is compared to the ending of the read string message. If they don't match, a warning is issued.

All line-ending characters are stripped from the end of the string.

Return type str

read_ascii_values(*converter='f'*, *separator=','*, *container=<class 'list'>*)

Read values from the device in ascii format returning an iterable of values.

Parameters

- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

read_binary_values(*datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)

Read values from the device in binary format returning an iterable of values.

Parameters

- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type type(container)

read_bytes(*count*, *chunk_size=None*, *break_on_termchar=False*)

Read a certain number of bytes from the instrument.

Parameters

- **count** (*int*) – The number of bytes to read from the instrument.
- **chunk_size** (*int*) – The chunk size to use to perform the reading.
- **break_on_termchar** (*bool*) – Should the reading stop when a termination character is encountered.

Return type bytes

read_raw(*size=None*)

Read the unmodified string sent from the instrument to the computer.

In contrast to read(), no termination characters are stripped.

Parameters **size** – The chunk size to use when reading the data.

Return type bytes

read_stb()

Service request status register.

read_termination

Read termination character.

read_termination_context(*new_termination*)

read_values(*fmt=None*, *container=<class 'list'>*)

Read a list of floating point values from the device.

Parameters

- **fmt** – the format of the values. If given, it overrides the class attribute "values_format". Possible values are bitwise disjunctions of the above constants ascii, single, double, and big_endian. Default is ascii.
- **container** – the output datatype

Returns the list of read values

Return type list

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info

Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

send_end

VI_ATTR_SEND_END_EN specifies whether to assert END during the transfer of the last byte of the buffer.

VISA Attribute VI_ATTR_SEND_END_EN (1073676310)

Type bool

session

Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

stb
Service request status register.

timeout
The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

values_format

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write(*message*, *termination=None*, *encoding=None*)
Write a string message to the device.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **termination** (*unicode (Py2) or str (Py3)*) – alternative character termination to use.
- **encoding** (*unicode (Py2) or str (Py3)*) – encoding to convert from unicode to bytes.

Returns number of bytes written.

Return type int

write_ascii_values(*message*, *values*, *converter='f'*, *separator=','*, *termination=None*, *encoding=None*)
Write a string message to the device followed by values in ascii format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **converter** (*callable | str*) – function used to convert each value. String formatting codes are also accepted. Defaults to 'f'.
- **separator** – a callable that join the values in a single str. If a str is given, separator.join(values) is used.

Type separator: (collections.Iterable[T]) -> str | str

Returns number of bytes written.

Return type int

write_binary_values(*message*, *values*, *datatype='f'*, *is_big_endian=False*, *termination=None*, *encoding=None*, *header_fmt='ieee'*)
Write a string message to the device followed by values in binary format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.

- **values** – data to be writen to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'

Returns number of bytes written.

Return type int

write_raw(*message*)
Write a byte message to the device.

Parameters **message** (*bytes*) – the message to be sent.

Returns number of bytes written.

Return type int

write_termination
Writer termination character.

write_values(*message*, *values*, *termination=None*, *encoding=None*)

class pyvisa.resources.**TCPIPSocket**(**args*, ***kwargs*)
Communicates with to devices of type TCPIP::host address::port::SOCKET

More complex resource names can be specified with the following grammar: TCPIP[board]::host address::port::SOCKET

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

CR = '\r'

LF = '\n'

assert_trigger()
Sends a software trigger to the device.

before_close()
Called just before closing an instrument.

chunk_size = 20480

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

encoding
Encoding used for read and write operations.

flush(*mask*)
Manually clears the specified buffers.

Depending on the value of the mask this can cause the buffer data to be written to the device.

Parameters **mask** – Specifies the action to be taken with flushing the buffer. See high-level.VisaLibraryBase.flush for a detailed description.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.

- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

io_protocol

VI_ATTR_IO_PROT specifies which protocol to use. In VXI, you can choose normal word serial or fast data channel (FDC). In GPIB, you can choose normal or high-speed (HS-488) transfers. In serial, TCPIP, or USB RAW, you can choose normal transfers or 488.2-defined strings. In USB INSTR, you can choose normal or vendor-specific transfers.

VISA Attribute VI_ATTR_IO_PROT (1073676316)

Type int

Range 0 <= value <= 65535

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

query(*message*, *delay=None*)
A combination of write(message) and read()

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type str

query_ascii_values(*message*, *converter='f'*, *separator=','*, *container=<class 'list'>*, *delay=None*)
Query the device for values in ascii format returning an iterable of values.

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

query_binary_values(*message*, *datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *delay=None*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)
Query the device for values in binary format returning an iterable of values.

Parameters

- **message** – the message to send to the instrument.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type list

query_delay = 0.0

query_values(*message*, *delay=None*)
Query the device for values returning an iterable of values.

The datatype expected is obtained from *values_format*

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type list

read(*termination=None*, *encoding=None*)
Read a string from the device.

Reading stops when the device stops sending (e.g. by setting appropriate bus lines), or the termination characters sequence was detected. Attention: Only the last character of the termination characters is really used to stop reading, however, the whole sequence is compared to the ending of the read string message. If they don't match, a warning is issued.

All line-ending characters are stripped from the end of the string.

Return type str

read_ascii_values(*converter='f'*, *separator=','*, *container=<class 'list'>*)
Read values from the device in ascii format returning an iterable of values.

Parameters

- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float

- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

read_binary_values(*datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)

Read values from the device in binary format returning an iterable of values.

Parameters

- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type type(container)

read_bytes(*count*, *chunk_size=None*, *break_on_termchar=False*)

Read a certain number of bytes from the instrument.

Parameters

- **count** (*int*) – The number of bytes to read from the instrument.
- **chunk_size** (*int*) – The chunk size to use to perform the reading.
- **break_on_termchar** (*bool*) – Should the reading stop when a termination character is encountered.

Return type bytes

read_raw(*size=None*)

Read the unmodified string sent from the instrument to the computer.

In contrast to read(), no termination characters are stripped.

Parameters **size** – The chunk size to use when reading the data.

Return type bytes

read_stb()

Service request status register.

read_termination
Read termination character.

read_termination_context(*new_termination*)

read_values(*fmt=None*, *container=<class 'list'>*)
Read a list of floating point values from the device.

Parameters

- **fmt** – the format of the values. If given, it overrides the class attribute "values_format". Possible values are bitwise disjunctions of the above constants ascii, single, double, and big_endian. Default is ascii.
- **container** – the output datatype

Returns the list of read values

Return type list

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info
Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

stb
Service request status register.

timeout
The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

values_format

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write(*message*, *termination=None*, *encoding=None*)
Write a string message to the device.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **termination** (*unicode (Py2) or str (Py3)*) – alternative character termination to use.
- **encoding** (*unicode (Py2) or str (Py3)*) – encoding to convert from unicode to bytes.

Returns number of bytes written.

Return type int

write_ascii_values(*message*, *values*, *converter='f'*, *separator=','*, *termination=None*, *encoding=None*)
Write a string message to the device followed by values in ascii format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **converter** (*callable | str*) – function used to convert each value. String formatting codes are also accepted. Defaults to 'f'.
- **separator** – a callable that join the values in a single str. If a str is given, separator.join(values) is used.

Type separator: (collections.Iterable[T]) -> str | str

Returns number of bytes written.

Return type int

write_binary_values(*message*, *values*, *datatype='f'*, *is_big_endian=False*, *termination=None*, *encoding=None*, *header_fmt='ieee'*)
Write a string message to the device followed by values in binary format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.

- **values** – data to be writen to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'

Returns number of bytes written.

Return type int

write_raw(*message*)
Write a byte message to the device.

Parameters **message** (*bytes*) – the message to be sent.

Returns number of bytes written.

Return type int

write_termination
Writer termination character.

write_values(*message*, *values*, *termination=None*, *encoding=None*)

class pyvisa.resources.**USBInstrument**(**args*, ***kwargs*)
Communicates with devices of type USB::manufacturer ID::model code::serial number

More complex resource names can be specified with the following grammar: USB[board]::manufacturer ID::model code::serial number[::USB interface number][::INSTR]

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

CR = '\r'

LF = '\n'

assert_trigger()
Sends a software trigger to the device.

before_close()
Called just before closing an instrument.

chunk_size = 20480

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

control_in(*request_type_bitmap_field*, *request_id*, *request_value*, *index*, *length=0*)
Performs a USB control pipe transfer from the device.

Parameters

- **request_type_bitmap_field** – bmRequestType parameter of the setup stage of a USB control transfer.
- **request_id** – bRequest parameter of the setup stage of a USB control transfer.
- **request_value** – wValue parameter of the setup stage of a USB control transfer.
- **index** – wIndex parameter of the setup stage of a USB control transfer. This is usually the index of the interface or endpoint.

- **length** – wLength parameter of the setup stage of a USB control transfer. This value also specifies the size of the data buffer to receive the data from the optional data stage of the control transfer.

Returns The data buffer that receives the data from the optional data stage of the control transfer.

Return type bytes

control_out(*request_type_bitmap_field*, *request_id*, *request_value*, *index*, *data=''*)
Performs a USB control pipe transfer to the device.

Parameters

- **request_type_bitmap_field** – bmRequestType parameter of the setup stage of a USB control transfer.
- **request_id** – bRequest parameter of the setup stage of a USB control transfer.
- **request_value** – wValue parameter of the setup stage of a USB control transfer.
- **index** – wIndex parameter of the setup stage of a USB control transfer. This is usually the index of the interface or endpoint.
- **data** – The data buffer that sends the data in the optional data stage of the control transfer.

control_ren(*mode*)
Controls the state of the GPIB Remote Enable (REN) interface line, and optionally the remote/local state of the device.

Corresponds to viGpibControlREN function of the VISA library.

Parameters **mode** – Specifies the state of the REN line and optionally the device remote/local state. (Constants.GPIB_REN*)

Returns return value of the library call.

Return type VISAStatus

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)

- **context** – Not currently used, leave as None.

encoding
Encoding used for read and write operations.

flush(*mask*)
Manually clears the specified buffers.

Depending on the value of the mask this can cause the buffer data to be written to the device.

Parameters **mask** – Specifies the action to be taken with flushing the buffer. See high-level.VisaLibraryBase.flush for a detailed description.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

io_protocol

VI_ATTR_IO_PROT specifies which protocol to use. In VXI, you can choose normal word serial or fast data channel (FDC). In GPIB, you can choose normal or high-speed (HS-488) transfers. In serial, TCPIP, or USB RAW, you can choose normal transfers or 488.2-defined strings. In USB INSTR, you can choose normal or vendor-specific transfers.

VISA Attribute VI_ATTR_IO_PROT (1073676316)

Type int

Range 0 <= value <= 65535

is_4882_compliant

VI_ATTR_4882_COMPLIANT specifies whether the device is 488.2 compliant.

VISA Attribute VI_ATTR_4882_COMPLIANT (1073676703)

Type bool

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

manufacturer_id

VI_ATTR_MANF_ID is the manufacturer identification number of the device.

VISA Attribute VI_ATTR_MANF_ID (1073676505)

Type int

Range 0 <= value <= 65535

manufacturer_name

This string attribute is the manufacturer name.

VISA Attribute VI_ATTR_MANF_NAME (3221160050)

maximum_interrupt_size

VI_ATTR_USB_MAX_INTR_SIZE specifies the maximum size of data that will be stored by any given USB interrupt. If a USB interrupt contains more data than this size, the data in excess of this size will be lost.

VISA Attribute VI_ATTR_USB_MAX_INTR_SIZE (1073676719)

Type int

Range 0 <= value <= 65535

model_code

VI_ATTR_MODEL_CODE specifies the model code for the device.

VISA Attribute VI_ATTR_MODEL_CODE (1073676511)

Type int

Range 0 <= value <= 65535

model_name

This string attribute is the model name of the device.

VISA Attribute VI_ATTR_MODEL_NAME (3221160055)

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)

Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the `access_mode` parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

query(*message*, *delay=None*)

A combination of write(message) and read()

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type str

query_ascii_values(*message*, *converter='f'*, *separator=','*, *container=<class 'list'>*, *delay=None*)

Query the device for values in ascii format returning an iterable of values.

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

query_binary_values(*message*, *datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *delay=None*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)

Query the device for values in binary format returning an iterable of values.

Parameters

- **message** – the message to send to the instrument.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type list

query_delay = 0.0

query_values(*message*, *delay=None*)
Query the device for values returning an iterable of values.

The datatype expected is obtained from *values_format*

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type list

read(*termination=None*, *encoding=None*)
Read a string from the device.

Reading stops when the device stops sending (e.g. by setting appropriate bus lines), or the termination characters sequence was detected. Attention: Only the last character of the termination characters is really used to stop reading, however, the whole sequence is compared to the ending of the read string message. If they don't match, a warning is issued.

All line-ending characters are stripped from the end of the string.

Return type str

read_ascii_values(*converter='f'*, *separator=','*, *container=<class 'list'>*)
Read values from the device in ascii format returning an iterable of values.

Parameters

- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

read_binary_values(*datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)
Read values from the device in binary format returning an iterable of values.

Parameters

- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)

- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type type(container)

read_bytes(*count*, *chunk_size=None*, *break_on_termchar=False*)
Read a certain number of bytes from the instrument.

Parameters

- **count** (*int*) – The number of bytes to read from the instrument.
- **chunk_size** (*int*) – The chunk size to use to perform the reading.
- **break_on_termchar** (*bool*) – Should the reading stop when a termination character is encountered.

Return type bytes

read_raw(*size=None*)
Read the unmodified string sent from the instrument to the computer.

In contrast to read(), no termination characters are stripped.

Parameters **size** – The chunk size to use when reading the data.

Return type bytes

read_stb()
Service request status register.

read_termination
Read termination character.

read_termination_context(*new_termination*)

read_values(*fmt=None*, *container=<class 'list'>*)
Read a list of floating point values from the device.

Parameters

- **fmt** – the format of the values. If given, it overrides the class attribute "values_format". Possible values are bitwise disjunctions of the above constants ascii, single, double, and big_endian. Default is ascii.
- **container** – the output datatype

Returns the list of read values

Return type list

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info
Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

send_end

VI_ATTR_SEND_END_EN specifies whether to assert END during the transfer of the last byte of the buffer.

VISA Attribute VI_ATTR_SEND_END_EN (1073676310)

Type bool

serial_number
VI_ATTR_USB_SERIAL_NUM specifies the USB serial number of this device.

VISA Attribute VI_ATTR_USB_SERIAL_NUM (3221160352)

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

stb
Service request status register.

timeout
The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

usb_control_out(*request_type_bitmap_field*, *request_id*, *request_value*, *index*, *data=''*)
Performs a USB control pipe transfer to the device. (Deprecated)

Parameters

- **request_type_bitmap_field** – bmRequestType parameter of the setup stage of a USB control transfer.
- **request_id** – bRequest parameter of the setup stage of a USB control transfer.
- **request_value** – wValue parameter of the setup stage of a USB control transfer.
- **index** – wIndex parameter of the setup stage of a USB control transfer. This is usually the index of the interface or endpoint.
- **data** – The data buffer that sends the data in the optional data stage of the control transfer.

usb_protocol

VI_ATTR_USB_PROTOCOL specifies the USB protocol used by this USB interface.

VISA Attribute VI_ATTR_USB_PROTOCOL (1073676711)

Type int

Range 0 <= value <= 255

values_format

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)

Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write(*message*, *termination=None*, *encoding=None*)

Write a string message to the device.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **termination** (*unicode (Py2) or str (Py3)*) – alternative character termination to use.
- **encoding** (*unicode (Py2) or str (Py3)*) – encoding to convert from unicode to bytes.

Returns number of bytes written.

Return type int

write_ascii_values(*message*, *values*, *converter='f'*, *separator=','*, *termination=None*, *encoding=None*)

Write a string message to the device followed by values in ascii format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **converter** (*callable | str*) – function used to convert each value. String formatting codes are also accepted. Defaults to 'f'.
- **separator** – a callable that join the values in a single str. If a str is given, separator.join(values) is used.

Type separator: (collections.Iterable[T]) -> str | str

Returns number of bytes written.

Return type int

write_binary_values(*message*, *values*, *datatype='f'*, *is_big_endian=False*, *termination=None*, *encoding=None*, *header_fmt='ieee'*)
Write a string message to the device followed by values in binary format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'

Returns number of bytes written.

Return type int

write_raw(*message*)
Write a byte message to the device.

Parameters **message** (*bytes*) – the message to be sent.

Returns number of bytes written.

Return type int

write_termination
Writer termination character.

write_values(*message*, *values*, *termination=None*, *encoding=None*)

class pyvisa.resources.**USBRaw**(**args*, ***kwargs*)
Communicates with to devices of type USB::manufacturer ID::model code::serial number::RAW

More complex resource names can be specified with the following grammar: USB[board]::manufacturer ID::model code::serial number[::USB interface number]::RAW

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

CR = '\r'

LF = '\n'

assert_trigger()
Sends a software trigger to the device.

before_close()
Called just before closing an instrument.

chunk_size = 20480

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

encoding
Encoding used for read and write operations.

flush(*mask*)
Manually clears the specified buffers.

Depending on the value of the mask this can cause the buffer data to be written to the device.

Parameters **mask** – Specifies the action to be taken with flushing the buffer. See highlevel.VisaLibraryBase.flush for a detailed description.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

io_protocol

VI_ATTR_IO_PROT specifies which protocol to use. In VXI, you can choose normal word serial or fast data channel (FDC). In GPIB, you can choose normal or high-speed (HS-488) transfers. In serial, TCPIP, or USB RAW, you can choose normal transfers or 488.2-defined strings. In USB INSTR, you can choose normal or vendor-specific transfers.

VISA Attribute VI_ATTR_IO_PROT (1073676316)

Type int

Range 0 <= value <= 65535

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

manufacturer_id

VI_ATTR_MANF_ID is the manufacturer identification number of the device.

VISA Attribute VI_ATTR_MANF_ID (1073676505)

Type int

Range 0 <= value <= 65535

manufacturer_name
This string attribute is the manufacturer name.

VISA Attribute VI_ATTR_MANF_NAME (3221160050)

maximum_interrupt_size

VI_ATTR_USB_MAX_INTR_SIZE specifies the maximum size of data that will be stored by any given USB interrupt. If a USB interrupt contains more data than this size, the data in excess of this size will be lost.

VISA Attribute VI_ATTR_USB_MAX_INTR_SIZE (1073676719)

Type int

Range 0 <= value <= 65535

model_code
VI_ATTR_MODEL_CODE specifies the model code for the device.

VISA Attribute VI_ATTR_MODEL_CODE (1073676511)

Type int

Range 0 <= value <= 65535

model_name
This string attribute is the model name of the device.

VISA Attribute VI_ATTR_MODEL_NAME (3221160055)

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

query(*message*, *delay=None*)
A combination of write(message) and read()

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type str

query_ascii_values(*message*, *converter='f'*, *separator=','*, *container=<class 'list'>*, *delay=None*)
Query the device for values in ascii format returning an iterable of values.

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

query_binary_values(*message*, *datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *delay=None*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)
Query the device for values in binary format returning an iterable of values.

Parameters

- **message** – the message to send to the instrument.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type list

query_delay = 0.0

query_values(*message*, *delay=None*)

Query the device for values returning an iterable of values.

The datatype expected is obtained from *values_format*

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type list

read(*termination=None*, *encoding=None*)

Read a string from the device.

Reading stops when the device stops sending (e.g. by setting appropriate bus lines), or the termination characters sequence was detected. Attention: Only the last character of the termination characters is really used to stop reading, however, the whole sequence is compared to the ending of the read string message. If they don't match, a warning is issued.

All line-ending characters are stripped from the end of the string.

Return type str

read_ascii_values(*converter='f'*, *separator=','*, *container=<class 'list'>*)

Read values from the device in ascii format returning an iterable of values.

Parameters

- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

read_binary_values(*datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)

Read values from the device in binary format returning an iterable of values.

Parameters

- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type type(container)

read_bytes(*count*, *chunk_size=None*, *break_on_termchar=False*)

Read a certain number of bytes from the instrument.

Parameters

- **count** (*int*) – The number of bytes to read from the instrument.
- **chunk_size** (*int*) – The chunk size to use to perform the reading.
- **break_on_termchar** (*bool*) – Should the reading stop when a termination character is encountered.

Return type bytes

read_raw(*size=None*)

Read the unmodified string sent from the instrument to the computer.

In contrast to read(), no termination characters are stripped.

Parameters **size** – The chunk size to use when reading the data.

Return type bytes

read_stb()

Service request status register.

read_termination

Read termination character.

read_termination_context(*new_termination*)

read_values(*fmt=None*, *container=<class 'list'>*)

Read a list of floating point values from the device.

Parameters

- **fmt** – the format of the values. If given, it overrides the class attribute "values_format". Possible values are bitwise disjunctions of the above constants ascii, single, double, and big_endian. Default is ascii.
- **container** – the output datatype

Returns the list of read values

Return type list

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info

Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

serial_number

VI_ATTR_USB_SERIAL_NUM specifies the USB serial number of this device.

VISA Attribute VI_ATTR_USB_SERIAL_NUM (3221160352)

session

Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)

Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

stb

Service request status register.

timeout

The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)

Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()

Relinquishes a lock for the specified resource.

usb_protocol

VI_ATTR_USB_PROTOCOL specifies the USB protocol used by this USB interface.

VISA Attribute VI_ATTR_USB_PROTOCOL (1073676711)

Type int

Range 0 <= value <= 255

values_format

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write(*message*, *termination=None*, *encoding=None*)
Write a string message to the device.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **termination** (*unicode (Py2) or str (Py3)*) – alternative character termination to use.
- **encoding** (*unicode (Py2) or str (Py3)*) – encoding to convert from unicode to bytes.

Returns number of bytes written.

Return type int

write_ascii_values(*message*, *values*, *converter='f'*, *separator=','*, *termination=None*, *encoding=None*)
Write a string message to the device followed by values in ascii format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **converter** (*callable | str*) – function used to convert each value. String formatting codes are also accepted. Defaults to 'f'.
- **separator** – a callable that join the values in a single str. If a str is given, separator.join(values) is used.

Type separator: (collections.Iterable[T]) -> str | str

Returns number of bytes written.

Return type int

write_binary_values(*message*, *values*, *datatype='f'*, *is_big_endian=False*, *termination=None*, *encoding=None*, *header_fmt='ieee'*)
Write a string message to the device followed by values in binary format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'

Returns number of bytes written.

Return type int

write_raw(*message*)
Write a byte message to the device.

Parameters **message** (*bytes*) – the message to be sent.

Returns number of bytes written.

Return type int

write_termination
Writer termination character.

write_values(*message*, *values*, *termination=None*, *encoding=None*)

class pyvisa.resources.**GPIBInstrument**(**args*, ***kwargs*)
Communicates with to devices of type GPIB::<primary address>[::INSTR]

More complex resource names can be specified with the following grammar: GPIB[board]::primary address[::secondary address][::INSTR]

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

CR = '\r'

LF = '\n'

allow_dma

This attribute specifies whether I/O accesses should use DMA (VI_TRUE) or Programmed I/O (VI_FALSE). In some implementations, this attribute may have global effects even though it is documented to be a local attribute. Since this affects performance and not functionality, that behavior is acceptable.

VISA Attribute VI_ATTR_DMA_ALLOW_EN (1073676318)

Type bool

assert_trigger()
Sends a software trigger to the device.

before_close()
Called just before closing an instrument.

chunk_size = 20480

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

control_atn(*mode*)
Specifies the state of the ATN line and the local active controller state.

Corresponds to viGpibControlATN function of the VISA library.

Parameters **mode** – Specifies the state of the ATN line and optionally the local active controller state. (Constants.GPIB_ATN*)

Returns return value of the library call.

Return type VISAStatus

control_ren(*mode*)
Controls the state of the GPIB Remote Enable (REN) interface line, and optionally the remote/local state of the device.

Corresponds to viGpibControlREN function of the VISA library.

Parameters **mode** – Specifies the state of the REN line and optionally the device remote/local state. (Constants.GPIB_REN*)

Returns return value of the library call.

Return type VISAStatus

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

enable_repeat_addressing

VI_ATTR_GPIB_READDR_EN specifies whether to use repeat addressing before each read or write operation.

VISA Attribute VI_ATTR_GPIB_READDR_EN (1073676315)

Type bool

enable_unaddressing

VI_ATTR_GPIB_UNADDR_EN specifies whether to unaddress the device (UNT and UNL) after each read or write operation.

VISA Attribute VI_ATTR_GPIB_UNADDR_EN (1073676676)

Type bool

encoding
Encoding used for read and write operations.

flush(*mask*)
Manually clears the specified buffers.

Depending on the value of the mask this can cause the buffer data to be written to the device.

Parameters **mask** – Specifies the action to be taken with flushing the buffer. See high-level.VisaLibraryBase.flush for a detailed description.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

io_protocol

VI_ATTR_IO_PROT specifies which protocol to use. In VXI, you can choose normal word serial or fast data channel (FDC). In GPIB, you can choose normal or high-speed (HS-488) transfers. In serial, TCPIP, or USB RAW, you can choose normal transfers or 488.2-defined strings. In USB INSTR, you can choose normal or vendor-specific transfers.

VISA Attribute VI_ATTR_IO_PROT (1073676316)

Type int

Range 0 <= value <= 65535

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

pass_control(*primary_address*, *secondary_address*)
Tell the GPIB device at the specified address to become controller in charge (CIC).

Corresponds to viGpibPassControl function of the VISA library.

Parameters

- **primary_address** – Primary address of the GPIB device to which you want to pass control.
- **secondary_address** – Secondary address of the targeted GPIB device. If the targeted device does not have a secondary address, this parameter should contain the value Constants.NO_SEC_ADDR.

Returns return value of the library call.

Return type VISAStatus

primary_address

VI_ATTR_GPIB_PRIMARY_ADDR specifies the primary address of the GPIB device used by the given session. For the GPIB INTFC Resource, this attribute is Read-Write.

VISA Attribute VI_ATTR_GPIB_PRIMARY_ADDR (1073676658)

Type int

Range 0 <= value <= 30

query(*message*, *delay=None*)
A combination of write(message) and read()

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type str

query_ascii_values(*message*, *converter='f'*, *separator=','*, *container=<class 'list'>*, *delay=None*)
Query the device for values in ascii format returning an iterable of values.

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

query_binary_values(*message*, *datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *delay=None*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)

Query the device for values in binary format returning an iterable of values.

Parameters

- **message** – the message to send to the instrument.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type list

query_delay = 0.0

query_values(*message*, *delay=None*)

Query the device for values returning an iterable of values.

The datatype expected is obtained from *values_format*

Parameters

- **message** (*str*) – the message to send.
- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay

Returns the answer from the device.

Return type list

read(*termination=None*, *encoding=None*)
Read a string from the device.

Reading stops when the device stops sending (e.g. by setting appropriate bus lines), or the termination characters sequence was detected. Attention: Only the last character of the termination characters is really used to stop reading, however, the whole sequence is compared to the ending of the read string message. If they don't match, a warning is issued.

All line-ending characters are stripped from the end of the string.

Return type str

read_ascii_values(*converter='f'*, *separator=','*, *container=<class 'list'>*)
Read values from the device in ascii format returning an iterable of values.

Parameters

- **delay** – delay in seconds between write and read operations. if None, defaults to self.query_delay
- **converter** (*callable*) – function used to convert each element. Defaults to float
- **separator** – a callable that split the str into individual elements. If a str is given, data.split(separator) is used.
- **container** – container type to use for the output data.

Type separator: (str) -> collections.Iterable[int] | str

Returns the answer from the device.

Return type list

read_binary_values(*datatype='f'*, *is_big_endian=False*, *container=<class 'list'>*, *header_fmt='ieee'*, *expect_termination=True*, *data_points=0*, *chunk_size=None*)
Read values from the device in binary format returning an iterable of values.

Parameters

- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess. Defaults to False.
- **container** – container type to use for the output data.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'
- **expect_termination** – when set to False, the expected length of the binary values block does not account for the final termination character (the read termination)
- **data_points** – Number of points expected in the block. This is used only if the instrument does not report it itself. This will be converted in a number of bytes based on the datatype.
- **chunk_size** – Size of the chunks to read from the device. Using larger chunks may be faster for large amount of data.

Returns the answer from the device.

Return type type(container)

read_bytes(*count*, *chunk_size=None*, *break_on_termchar=False*)
Read a certain number of bytes from the instrument.

Parameters

- **count** (*int*) – The number of bytes to read from the instrument.
- **chunk_size** (*int*) – The chunk size to use to perform the reading.
- **break_on_termchar** (*bool*) – Should the reading stop when a termination character is encountered.

Return type bytes

read_raw(*size=None*)
Read the unmodified string sent from the instrument to the computer.

In contrast to read(), no termination characters are stripped.

Parameters **size** – The chunk size to use when reading the data.

Return type bytes

read_stb()
Service request status register.

read_termination
Read termination character.

read_termination_context(*new_termination*)

read_values(*fmt=None*, *container=<class 'list'>*)
Read a list of floating point values from the device.

Parameters

- **fmt** – the format of the values. If given, it overrides the class attribute "values_format". Possible values are bitwise disjunctions of the above constants ascii, single, double, and big_endian. Default is ascii.
- **container** – the output datatype

Returns the list of read values

Return type list

classmethod register(*interface_type*, *resource_class*)

remote_enabled

VI_ATTR_GPIB_REN_STATE returns the current state of the GPIB REN (Remote ENable) interface line.

VISA Attribute VI_ATTR_GPIB_REN_STATE (1073676673)

Type :class:pyvisa.constants.LineState

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info
Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

secondary_address

VI_ATTR_GPIB_SECONDARY_ADDR specifies the secondary address of the GPIB device used by the given session. For the GPIB INTFC Resource, this attribute is Read-Write.

VISA Attribute VI_ATTR_GPIB_SECONDARY_ADDR (1073676659)

Type int

Range 0 <= value <= 30 or in [65535]

send_command(*data*)

Write GPIB command bytes on the bus.

Corresponds to viGpibCommand function of the VISA library.

Parameters **data** (*bytes*) – data tor write.

Returns Number of written bytes, return value of the library call.

Return type int, VISAStatus

send_end

VI_ATTR_SEND_END_EN specifies whether to assert END during the transfer of the last byte of the buffer.

VISA Attribute VI_ATTR_SEND_END_EN (1073676310)

Type bool

send_ifc()

Pulse the interface clear line (IFC) for at least 100 microseconds.

Corresponds to viGpibSendIFC function of the VISA library.

Returns return value of the library call.

Return type VISAStatus

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

stb
Service request status register.

timeout
The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

values_format

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_for_srq(*timeout=25000*)
Wait for a serial request (SRQ) coming from the instrument.

Note that this method is not ended when *another* instrument signals an SRQ, only *this* instrument.

Parameters **timeout** – the maximum waiting time in milliseconds. Defaul: 25000 (milliseconds). None means waiting forever if necessary.

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write(*message*, *termination=None*, *encoding=None*)
Write a string message to the device.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **termination** (*unicode (Py2) or str (Py3)*) – alternative character termination to use.
- **encoding** (*unicode (Py2) or str (Py3)*) – encoding to convert from unicode to bytes.

Returns number of bytes written.

Return type int

write_ascii_values(*message*, *values*, *converter='f'*, *separator=','*, *termination=None*, *encoding=None*)
Write a string message to the device followed by values in ascii format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **converter** (*callable | str*) – function used to convert each value. String formatting codes are also accepted. Defaults to 'f'.
- **separator** – a callable that join the values in a single str. If a str is given, separator.join(values) is used.

Type separator: (collections.Iterable[T]) -> str | str

Returns number of bytes written.

Return type int

write_binary_values(*message*, *values*, *datatype='f'*, *is_big_endian=False*, *termination=None*, *encoding=None*, *header_fmt='ieee'*)
Write a string message to the device followed by values in binary format.

The write_termination is always appended to it.

Parameters

- **message** (*unicode (Py2) or str (Py3)*) – the message to be sent.
- **values** – data to be writen to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.
- **header_fmt** – format of the header prefixing the data. Possible values are: 'ieee', 'hp', 'empty'

Returns number of bytes written.

Return type int

write_raw(*message*)
Write a byte message to the device.

Parameters **message** (*bytes*) – the message to be sent.

Returns number of bytes written.

Return type int

write_termination
Writer termination character.

write_values(*message*, *values*, *termination=None*, *encoding=None*)

class pyvisa.resources.**GPIBInterface**(*resource_manager*, *resource_name*)
Communicates with to devices of type GPIB::INTFC

More complex resource names can be specified with the following grammar: GPIB[board]::INTFC

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

address_state

This attribute shows whether the specified GPIB interface is currently addressed to talk or listen, or is not addressed.

VISA Attribute VI_ATTR_GPIB_ADDR_STATE (1073676380)

Type :class:pyvisa.constants.AddressState

allow_dma

This attribute specifies whether I/O accesses should use DMA (VI_TRUE) or Programmed I/O (VI_FALSE). In some implementations, this attribute may have global effects even though it is documented to be a local attribute. Since this affects performance and not functionality, that behavior is acceptable.

VISA Attribute VI_ATTR_DMA_ALLOW_EN (1073676318)

Type bool

atn_state

This attribute shows the current state of the GPIB ATN (ATtentioN) interface line.

VISA Attribute VI_ATTR_GPIB_ATN_STATE (1073676375)

Type :class:pyvisa.constants.LineState

before_close()
Called just before closing an instrument.

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

control_atn(*mode*)
Specifies the state of the ATN line and the local active controller state.

Corresponds to viGpibControlATN function of the VISA library.

Parameters **mode** – Specifies the state of the ATN line and optionally the local active controller state. (Constants.GPIB_ATN*)

Returns return value of the library call.

Return type VISAStatus

control_ren(*mode*)
Controls the state of the GPIB Remote Enable (REN) interface line, and optionally the remote/local state of the device.

Corresponds to viGpibControlREN function of the VISA library.

Parameters **mode** – Specifies the state of the REN line and optionally the device remote/local state. (Constants.GPIB_REN*)

Returns return value of the library call.

Return type VISAStatus

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

flush(*mask*)
Manually clears the specified buffers.

Depending on the mask this can cause the buffer data to be written to the device.

Parameters **mask** – Specifies the action to be taken with flushing the buffer. See highlevel.VisaLibraryBase.flush for a detailed description.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

group_execute_trigger(**resources*)

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

io_protocol

VI_ATTR_IO_PROT specifies which protocol to use. In VXI, you can choose normal word serial or fast data channel (FDC). In GPIB, you can choose normal or high-speed (HS-488) transfers. In serial, TCPIP, or USB RAW, you can choose normal transfers or 488.2-defined strings. In USB INSTR, you can choose normal or vendor-specific transfers.

VISA Attribute VI_ATTR_IO_PROT (1073676316)

Type int

Range 0 <= value <= 65535

is_controller_in_charge

This attribute shows whether the specified GPIB interface is currently CIC (Controller In Charge).

VISA Attribute VI_ATTR_GPIB_CIC_STATE (1073676382)

Type bool

is_system_controller

This attribute shows whether the specified GPIB interface is currently the system controller. In some implementations, this attribute may be modified only through a configuration utility. On these systems this attribute is read-only (RO).

VISA Attribute VI_ATTR_GPIB_SYS_CNTRL_STATE (1073676392)

Type bool

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

ndac_state

This attribute shows the current state of the GPIB NDAC (Not Data ACcepted) interface line.

VISA Attribute VI_ATTR_GPIB_NDAC_STATE (1073676386)

Type :class:pyvisa.constants.LineState

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

pass_control(*primary_address*, *secondary_address*)
Tell the GPIB device at the specified address to become controller in charge (CIC).

Corresponds to viGpibPassControl function of the VISA library.

Parameters

- **primary_address** – Primary address of the GPIB device to which you want to pass control.
- **secondary_address** – Secondary address of the targeted GPIB device. If the targeted device does not have a secondary address, this parameter should contain the value Constants.NO_SEC_ADDR.

Returns return value of the library call.

Return type VISAStatus

primary_address

VI_ATTR_GPIB_PRIMARY_ADDR specifies the primary address of the GPIB device used by the given session. For the GPIB INTFC Resource, this attribute is Read-Write.

VISA Attribute VI_ATTR_GPIB_PRIMARY_ADDR (1073676658)

Type int

Range 0 <= value <= 30

classmethod register(*interface_type*, *resource_class*)

remote_enabled

VI_ATTR_GPIB_REN_STATE returns the current state of the GPIB REN (Remote ENable) interface line.

VISA Attribute VI_ATTR_GPIB_REN_STATE (1073676673)

Type :class:pyvisa.constants.LineState

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info

Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

secondary_address

VI_ATTR_GPIB_SECONDARY_ADDR specifies the secondary address of the GPIB device used by the given session. For the GPIB INTFC Resource, this attribute is Read-Write.

VISA Attribute VI_ATTR_GPIB_SECONDARY_ADDR (1073676659)

Type int

Range 0 <= value <= 30 or in [65535]

send_command(*data*)
Write GPIB command bytes on the bus.

Corresponds to viGpibCommand function of the VISA library.

Parameters **data** (*bytes*) – data tor write.

Returns Number of written bytes, return value of the library call.

Return type int, VISAStatus

send_end

VI_ATTR_SEND_END_EN specifies whether to assert END during the transfer of the last byte of the buffer.

VISA Attribute VI_ATTR_SEND_END_EN (1073676310)

Type bool

send_ifc()
Pulse the interface clear line (IFC) for at least 100 microseconds.

Corresponds to viGpibSendIFC function of the VISA library.

Returns return value of the library call.

Return type VISAStatus

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

timeout
The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (VI_TMO_IMMEDIATE): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (VI_TMO_INFINITE): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

class pyvisa.resources.**FirewireInstrument**(*resource_manager*, *resource_name*)
Communicates with to devices of type VXI::VXI logical address[::INSTR]

More complex resource names can be specified with the following grammar: VXI[board]::VXI logical address[::INSTR]

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

before_close()
Called just before closing an instrument.

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.

- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

move_in(*space*, *offset*, *length*, *width*, *extended=False*)
Moves a block of data to local memory from the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

move_out(*space*, *offset*, *length*, *data*, *width*, *extended=False*)
Moves a block of data from local memory to the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

read_memory(*space*, *offset*, *width*, *extended=False*)
Reads in an 8-bit, 16-bit, 32-bit, or 64-bit value from the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory.

Corresponds to viIn* functions of the visa library.

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info
Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

timeout
The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (VI_TMO_IMMEDIATE): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (VI_TMO_INFINITE): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write_memory(*space*, *offset*, *data*, *width*, *extended=False*)
Write in an 8-bit, 16-bit, 32-bit, value to the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Corresponds to viOut* functions of the visa library.

class pyvisa.resources.**PXIInstrument**(*resource_manager*, *resource_name*)

Communicates with to devices of type PXI::<device>[::INSTR]

More complex resource names can be specified with the following grammar:
PXI[bus]::device[::function][::INSTR]

or: PXI[interface]::bus-device[.function][::INSTR]

or: PXI[interface]::CHASSISchassis number::SLOTslot number[::FUNCfunction][::INSTR]

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

allow_dma

This attribute specifies whether I/O accesses should use DMA (VI_TRUE) or Programmed I/O (VI_FALSE). In some implementations, this attribute may have global effects even though it is documented to be a local attribute. Since this affects performance and not functionality, that behavior is acceptable.

VISA Attribute VI_ATTR_DMA_ALLOW_EN (1073676318)

Type bool

before_close()

Called just before closing an instrument.

clear()

Clears this resource

close()

Closes the VISA session and marks the handle as invalid.

destination_increment

VI_ATTR_DEST_INCREMENT is used in the viMoveOutXX() operations to specify by how many elements the destination offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the destination address will be incremented by 1 after each transfer), and the viMoveOutXX() operations move into consecutive elements. If this attribute is set to 0, the viMoveOutXX() operations will always write to the same element, essentially treating the destination as a FIFO register.

VISA Attribute VI_ATTR_DEST_INCREMENT (1073676353)

Type int

Range 0 <= value <= 1

disable_event(*event_type*, *mechanism*)

Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)

Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

manufacturer_id

VI_ATTR_MANF_ID is the manufacturer identification number of the device.

VISA Attribute VI_ATTR_MANF_ID (1073676505)

Type int

Range 0 <= value <= 65535

manufacturer_name
This string attribute is the manufacturer name.

VISA Attribute VI_ATTR_MANF_NAME (3221160050)

model_code
VI_ATTR_MODEL_CODE specifies the model code for the device.

VISA Attribute VI_ATTR_MODEL_CODE (1073676511)

Type int

Range 0 <= value <= 65535

model_name
This string attribute is the model name of the device.

VISA Attribute VI_ATTR_MODEL_NAME (3221160055)

move_in(*space*, *offset*, *length*, *width*, *extended=False*)
Moves a block of data to local memory from the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

move_out(*space*, *offset*, *length*, *data*, *width*, *extended=False*)
Moves a block of data from local memory to the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the `access_mode` parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

read_memory(*space*, *offset*, *width*, *extended=False*)
Reads in an 8-bit, 16-bit, 32-bit, or 64-bit value from the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory.

Corresponds to viIn* functions of the visa library.

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info

Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

session

Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)

Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)

- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

source_increment

VI_ATTR_SRC_INCREMENT is used in the viMoveInXX() operations to specify by how many elements the source offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the source address will be incremented by 1 after each transfer), and the viMoveInXX() operations move from consecutive elements. If this attribute is set to 0, the viMoveInXX() operations will always read from the same element, essentially treating the source as a FIFO register.

VISA Attribute VI_ATTR_SRC_INCREMENT (1073676352)

Type int

Range 0 <= value <= 1

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

timeout

The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)

Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()

Relinquishes a lock for the specified resource.

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write_memory(*space*, *offset*, *data*, *width*, *extended=False*)
Write in an 8-bit, 16-bit, 32-bit, value to the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Corresponds to viOut* functions of the visa library.

class pyvisa.resources.**PXIMemory**(*resource_manager*, *resource_name*)
Communicates with to devices of type PXI[interface]::MEMACC

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

before_close()
Called just before closing an instrument.

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

destination_increment

VI_ATTR_DEST_INCREMENT is used in the viMoveOutXX() operations to specify by how many elements the destination offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the destination address will be incremented by 1 after each transfer), and the viMoveOutXX() operations move into consecutive elements. If this attribute is set to 0, the viMoveOutXX() operations will always write to the same element, essentially treating the destination as a FIFO register.

VISA Attribute VI_ATTR_DEST_INCREMENT (1073676353)

Type int

Range 0 <= value <= 1

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.

- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

move_in(*space*, *offset*, *length*, *width*, *extended=False*)
Moves a block of data to local memory from the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

move_out(*space*, *offset*, *length*, *data*, *width*, *extended=False*)
Moves a block of data from local memory to the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

read_memory(*space*, *offset*, *width*, *extended=False*)
Reads in an 8-bit, 16-bit, 32-bit, or 64-bit value from the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory.

Corresponds to viIn* functions of the visa library.

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info
Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

source_increment

VI_ATTR_SRC_INCREMENT is used in the viMoveInXX() operations to specify by how many elements the source offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the source address will be incremented by 1 after each transfer), and the viMoveInXX() operations move from consecutive elements. If this attribute is set to 0, the viMoveInXX() operations will always read from the same element, essentially treating the source as a FIFO register.

VISA Attribute VI_ATTR_SRC_INCREMENT (1073676352)

Type int

Range 0 <= value <= 1

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

timeout

The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)

Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()

Relinquishes a lock for the specified resource.

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)

Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write_memory(*space*, *offset*, *data*, *width*, *extended=False*)
Write in an 8-bit, 16-bit, 32-bit, value to the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Corresponds to viOut* functions of the visa library.

class pyvisa.resources.**VXIInstrument**(*resource_manager*, *resource_name*)
Communicates with to devices of type VXI::VXI logical address[::INSTR]

More complex resource names can be specified with the following grammar: VXI[board]::VXI logical address[::INSTR]

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

allow_dma

This attribute specifies whether I/O accesses should use DMA (VI_TRUE) or Programmed I/O (VI_FALSE). In some implementations, this attribute may have global effects even though it is documented to be a local attribute. Since this affects performance and not functionality, that behavior is acceptable.

VISA Attribute VI_ATTR_DMA_ALLOW_EN (1073676318)

Type bool

before_close()
Called just before closing an instrument.

clear()
Clears this resource

close()
Closes the VISA session and marks the handle as invalid.

destination_increment

VI_ATTR_DEST_INCREMENT is used in the viMoveOutXX() operations to specify by how many elements the destination offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the destination address will be incremented by 1 after each transfer), and the viMoveOutXX() operations move into consecutive elements. If this attribute is set to 0, the viMoveOutXX() operations will always write to the same element, essentially treating the destination as a FIFO register.

VISA Attribute VI_ATTR_DEST_INCREMENT (1073676353)

Type int

Range 0 <= value <= 1

disable_event(*event_type*, *mechanism*)
Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.

- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

io_protocol

VI_ATTR_IO_PROT specifies which protocol to use. In VXI, you can choose normal word serial or fast data channel (FDC). In GPIB, you can choose normal or high-speed (HS-488) transfers. In serial, TCPIP, or USB RAW, you can choose normal transfers or 488.2-defined strings. In USB INSTR, you can choose normal or vendor-specific transfers.

VISA Attribute VI_ATTR_IO_PROT (1073676316)

Type int

Range 0 <= value <= 65535

is_4882_compliant

VI_ATTR_4882_COMPLIANT specifies whether the device is 488.2 compliant.

VISA Attribute VI_ATTR_4882_COMPLIANT (1073676703)

Type bool

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

manufacturer_id

VI_ATTR_MANF_ID is the manufacturer identification number of the device.

VISA Attribute VI_ATTR_MANF_ID (1073676505)

Type int

Range 0 <= value <= 65535

manufacturer_name
This string attribute is the manufacturer name.

VISA Attribute VI_ATTR_MANF_NAME (3221160050)

model_code
VI_ATTR_MODEL_CODE specifies the model code for the device.

VISA Attribute VI_ATTR_MODEL_CODE (1073676511)

Type int

Range 0 <= value <= 65535

model_name
This string attribute is the model name of the device.

VISA Attribute VI_ATTR_MODEL_NAME (3221160055)

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the access_mode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info
Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

send_end

VI_ATTR_SEND_END_EN specifies whether to assert END during the transfer of the last byte of the buffer.

VISA Attribute VI_ATTR_SEND_END_EN (1073676310)

Type bool

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

source_increment

VI_ATTR_SRC_INCREMENT is used in the viMoveInXX() operations to specify by how many elements the source offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the source address will be incremented by 1 after each transfer), and the viMoveInXX() operations move from consecutive elements. If this attribute is set to 0, the viMoveInXX() operations will always read from the same element, essentially treating the source as a FIFO register.

VISA Attribute VI_ATTR_SRC_INCREMENT (1073676352)

Type int

Range 0 <= value <= 1

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

timeout

The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)

Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()

Relinquishes a lock for the specified resource.

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)

Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

class pyvisa.resources.**VXIMemory**(*resource_manager*, *resource_name*)

Communicates with to devices of type VXI[board]::MEMACC

More complex resource names can be specified with the following grammar: VXI[board]::MEMACC

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

allow_dma

This attribute specifies whether I/O accesses should use DMA (VI_TRUE) or Programmed I/O (VI_FALSE). In some implementations, this attribute may have global effects even though it is documented to be a local attribute. Since this affects performance and not functionality, that behavior is acceptable.

VISA Attribute VI_ATTR_DMA_ALLOW_EN (1073676318)

Type bool

before_close()

Called just before closing an instrument.

clear()

Clears this resource

close()

Closes the VISA session and marks the handle as invalid.

destination_increment

VI_ATTR_DEST_INCREMENT is used in the viMoveOutXX() operations to specify by how many elements the destination offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the destination address will be incremented by 1 after each transfer), and the viMoveOutXX() operations move into consecutive elements. If this attribute is set to 0, the viMoveOutXX() operations will always write to the same element, essentially treating the destination as a FIFO register.

VISA Attribute VI_ATTR_DEST_INCREMENT (1073676353)

Type int

Range 0 <= value <= 1

disable_event(*event_type*, *mechanism*)

Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)
Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)
Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

get_visa_attribute(*name*)
Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

move_in(*space*, *offset*, *length*, *width*, *extended=False*)
Moves a block of data to local memory from the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

move_out(*space*, *offset*, *length*, *data*, *width*, *extended=False*)

Moves a block of data from local memory to the specified address space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **length** – Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
- **data** – Data to write to bus.
- **width** – Number of bits to read per element.
- **extended** – Use 64 bits offset independent of the platform.

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)

Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the `access_mode` parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

read_memory(*space*, *offset*, *width*, *extended=False*)

Reads in an 8-bit, 16-bit, 32-bit, or 64-bit value from the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Returns Data read from memory.

Corresponds to viIn* functions of the visa library.

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info
Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

session
Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)
Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

source_increment

VI_ATTR_SRC_INCREMENT is used in the viMoveInXX() operations to specify by how many elements the source offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the source address will be incremented by 1 after each transfer), and the viMoveInXX() operations move from consecutive elements. If this attribute is set to 0, the viMoveInXX() operations will always read from the same element, essentially treating the source as a FIFO register.

VISA Attribute VI_ATTR_SRC_INCREMENT (1073676352)

Type int

Range 0 <= value <= 1

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

timeout

The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)

Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()

Relinquishes a lock for the specified resource.

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)

Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

write_memory(*space*, *offset*, *data*, *width*, *extended=False*)

Write in an 8-bit, 16-bit, 32-bit, value to the specified memory space and offset.

Parameters

- **space** – Specifies the address space. (Constants.*SPACE*)
- **offset** – Offset (in bytes) of the address or register from which to read.
- **data** – Data to write to bus.
- **width** – Number of bits to read.
- **extended** – Use 64 bits offset independent of the platform.

Corresponds to viOut* functions of the visa library.

class pyvisa.resources.**VXIBackplane**(*resource_manager*, *resource_name*)

Communicates with to devices of type VXI::BACKPLANE

More complex resource names can be specified with the following grammar: VXI[board][::VXI logical address]::BACKPLANE

Do not instantiate directly, use *pyvisa.highlevel.ResourceManager.open_resource()*.

before_close()

Called just before closing an instrument.

clear()

Clears this resource

close()

Closes the VISA session and marks the handle as invalid.

disable_event(*event_type*, *mechanism*)

Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be disabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

discard_events(*event_type*, *mechanism*)

Discards event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be dicarded. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR, .VI_ALL_MECH)

enable_event(*event_type*, *mechanism*, *context=None*)

Enable event occurrences for specified event types and mechanisms in this resource.

Parameters

- **event_type** – Logical event identifier.
- **mechanism** – Specifies event handling mechanisms to be enabled. (Constants.VI_QUEUE, .VI_HNDLR, .VI_SUSPEND_HNDLR)
- **context** – Not currently used, leave as None.

get_visa_attribute(*name*)

Retrieves the state of an attribute in this resource.

Parameters **name** – Resource attribute for which the state query is made (see Attributes.*)

Returns The state of the queried attribute for a specified resource.

Return type unicode (Py2) or str (Py3), list or other type

ignore_warning(**warnings_constants*)
Ignoring warnings context manager for the current resource.

Parameters **warnings_constants** – constants identifying the warnings to ignore.

implementation_version

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

VISA Attribute VI_ATTR_RSRC_IMPL_VERSION (1073676291)

Type int

Range 0 <= value <= 4294967295

install_handler(*event_type*, *handler*, *user_handle=None*)
Installs handlers for event callbacks in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be installed by a client application.
- **user_handle** – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns user handle (a ctypes object)

interface_number
VI_ATTR_INTF_NUM specifies the board number for the given interface.

VISA Attribute VI_ATTR_INTF_NUM (1073676662)

Type int

Range 0 <= value <= 65535

interface_type
The interface type of the resource as a number.

last_status
Last status code for this session.

Return type *pyvisa.constants.StatusCode*

lock(*timeout='default'*, *requested_key=None*)
Establish a shared lock to the resource.

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – Access key used by another session with which you want your session to share a lock or None to generate a new shared access key.

Returns A new shared access key if requested_key is None, otherwise, same value as the requested_key

lock_context(*timeout='default'*, *requested_key='exclusive'*)
A context that locks

Parameters

- **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)
- **requested_key** – When using default of 'exclusive' the lock is an exclusive lock. Otherwise it is the access key for the shared lock or None to generate a new shared access key.

The returned context is the access_key if applicable.

lock_excl(*timeout='default'*)
Establish an exclusive lock to the resource.

Parameters **timeout** – Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error. (Defaults to self.timeout)

lock_state

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

VISA Attribute VI_ATTR_RSRC_LOCK_STATE (1073676292)

Type :class:pyvisa.constants.AccessModes

open(*access_mode=<AccessModes.no_lock: 0>*, *open_timeout=5000*)
Opens a session to the specified resource.

Parameters

- **access_mode** (*pyvisa.constants.AccessModes*) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (*int*) – If the `access_mode` parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.

classmethod register(*interface_type*, *resource_class*)

resource_class

VI_ATTR_RSRC_CLASS specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

VISA Attribute VI_ATTR_RSRC_CLASS (3221159937)

resource_info
Get the extended information of this resource.

Parameters **resource_name** – Unique symbolic name of a resource.

Return type *pyvisa.highlevel.ResourceInfo*

resource_manufacturer_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_MANF_NAME (3221160308)

resource_name

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Note The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

VISA Attribute VI_ATTR_RSRC_NAME (3221159938)

session

Resource session handle.

Raises `pyvisa.errors.InvalidSession` if session is closed.

set_visa_attribute(*name*, *state*)

Sets the state of an attribute.

Parameters

- **name** – Attribute for which the state is to be modified. (Attributes.*)
- **state** – The state of the attribute to be set for the specified object.

Returns return value of the library call.

Return type *pyvisa.constants.StatusCode*

spec_version

VI_ATTR_RSRC_SPEC_VERSION is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

VISA Attribute VI_ATTR_RSRC_SPEC_VERSION (1073676656)

Type int

Range 0 <= value <= 4294967295

timeout

The timeout in milliseconds for all resource I/O operations.

Special values:

- **immediate** (`VI_TMO_IMMEDIATE`): 0 (for convenience, any value smaller than 1 is considered as 0)
- **infinite** (`VI_TMO_INFINITE`): `float('+inf')` (for convenience, None is considered as `float('+inf')`)

To set an **infinite** timeout, you can also use:

```
>>> del instrument.timeout
```

uninstall_handler(*event_type*, *handler*, *user_handle=None*)
Uninstalls handlers for events in this resource.

Parameters

- **event_type** – Logical event identifier.
- **handler** – Interpreted as a valid reference to a handler to be uninstalled by a client application.
- **user_handle** – The user handle (ctypes object or None) returned by install_handler.

unlock()
Relinquishes a lock for the specified resource.

visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>,

wait_on_event(*in_event_type*, *timeout*, *capture_timeout=False*)
Waits for an occurrence of the specified event in this resource.

Parameters

- **in_event_type** – Logical identifier of the event(s) to wait for.
- **timeout** – Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds. None means waiting forever if necessary.
- **capture_timeout** – When True will not produce a VisaIOError(VI_ERROR_TMO) but instead return a WaitResponse with timed_out=True

Returns A WaitResponse object that contains event_type, context and ret value.

1.4.4 Constants module

Provides user-friendly naming to values used in different functions.

class pyvisa.constants.**AccessModes**
An enumeration.

exclusive_lock = 1
Obtains a exclusive lock on the VISA resource.

no_lock = 0
Does not obtain any lock on the VISA resource.

shared_lock = 2
Obtains a lock on the VISA resouce which may be shared between multiple VISA sessions.

class pyvisa.constants.**StopBits**
The number of stop bits that indicate the end of a frame.

one = 10

one_and_a_half = 15

two = 20

class pyvisa.constants.**Parity**
The parity types to use with every frame transmitted and received on a serial session.

even = 2

mark = 3

none = 0

odd = 1

space = 4

class pyvisa.constants.**SerialTermination**
The available methods for terminating a serial transfer.

last_bit = 1
The transfer occurs with the last bit not set until the last character is sent.

none = 0
The transfer terminates when all requested data is transferred or when an error occurs.

termination_break = 3
The write transmits a break after all the characters for the write are sent.

termination_char = 2
The transfer terminate by searching for "/" appending the termination character.

class pyvisa.constants.**InterfaceType**
The hardware interface

asrl = 4
Serial devices connected to either an RS-232 or RS-485 controller.

firewire = 9
Firewire device.

gpib = 1
GPIB Interface.

gpib_vxi = 3
GPIB VXI (VME eXtensions for Instrumentation).

pxi = 5
PXI device.

rio = 8
Rio device.

rsnrp = 33024
Rohde and Schwarz Device via Passport

tcpip = 6
TCPIP device.

unknown = -1

usb = 7
Universal Serial Bus (USB) hardware bus.

vxi = 2
VXI (VME eXtensions for Instrumentation), VME, MXI (Multisystem eXtension Interface).

class pyvisa.constants.**AddressState**
An enumeration.

listenr = 2

talker = 1

unaddressed = 0

class pyvisa.constants.**IOProtocol**
An enumeration.

fdc = 2
Fast data channel (FDC) protocol for VXI

hs488 = 3
High speed 488 transfer for GPIB

normal = 1

protocol4882_strs = 4
488 style transfer for serial

usbtmc_vendor = 5
Test measurement class vendor specific for USB

class pyvisa.constants.**LineState**
An enumeration.

asserted = 1

unasserted = 0

unknown = -1

class pyvisa.constants.**StatusCode**
Specifies the status codes that NI-VISA driver-level operations can return.

error_abort = -1073807312
The operation was aborted.

error_allocation = -1073807300
Insufficient system resources to perform necessary memory allocation.

error_attribute_read_only = -1073807329
The specified attribute is read-only.

error_bus_error = -1073807304
Bus error occurred during transfer.

error_closing_failed = -1073807338
Unable to deallocate the previously allocated data structures corresponding to this session or object reference.

error_connection_lost = -1073807194
The connection for the specified session has been lost.

error_file_access = -1073807199
An error occurred while trying to open the specified file. Possible causes include an invalid path or lack of access rights.

error_file_i_o = -1073807198
An error occurred while performing I/O on the specified file.

error_handler_not_installed = -1073807320
A handler is not currently installed for the specified event.

error_in_progress = -1073807303
Unable to queue the asynchronous operation because there is already an operation in progress.

error_input_protocol_violation = -1073807305
Device reported an input protocol error during transfer.

error_interface_number_not_configured = -1073807195
The interface type is valid but the specified interface number is not configured.

error_interrupt_pending = -1073807256
An interrupt is still pending from a previous call.

error_invalid_access_key = -1073807327
The access key to the resource associated with this session is invalid.

error_invalid_access_mode = -1073807341
Invalid access mode.

error_invalid_address_space = -1073807282
Invalid address space specified.

error_invalid_context = -1073807318
Specified event context is invalid.

error_invalid_degree = -1073807333
Specified degree is invalid.

error_invalid_event = -1073807322
Specified event type is not supported by the resource.

error_invalid_expression = -1073807344
Invalid expression specified for search.

error_invalid_format = -1073807297
A format specifier in the format string is invalid.

error_invalid_handler_reference = -1073807319
The specified handler reference is invalid.

error_invalid_job_i_d = -1073807332
Specified job identifier is invalid.

error_invalid_length = -1073807229
Invalid length specified.

error_invalid_line = -1073807200
The value specified by the line parameter is invalid.

error_invalid_lock_type = -1073807328
The specified type of lock is not supported by this resource.

error_invalid_mask = -1073807299
Invalid buffer mask specified.

error_invalid_mechanism = -1073807321
Invalid mechanism specified.

error_invalid_mode = -1073807215
The specified mode is invalid.

error_invalid_object = -1073807346
The specified session or object reference is invalid.

error_invalid_offset = -1073807279
Invalid offset specified.

error_invalid_parameter = -1073807240
The value of an unknown parameter is invalid.

error_invalid_protocol = -1073807239
The protocol specified is invalid.

error_invalid_resource_name = -1073807342
Invalid resource reference specified. Parsing error.

error_invalid_setup = -1073807302
Unable to start operation because setup is invalid due to inconsistent state of properties.

error_invalid_size = -1073807237
Invalid size of window specified.

error_invalid_width = -1073807278
Invalid source or destination width specified.

error_io = -1073807298
Could not perform operation because of I/O error.

error_library_not_found = -1073807202
A code library required by VISA could not be located or loaded.

error_line_in_use = -1073807294
The specified trigger line is currently in use.

error_machine_not_available = -1073807193
The remote machine does not exist or is not accepting any connections.

error_memory_not_shared = -1073807203
The device does not export any memory.

error_no_listeners = -1073807265
No listeners condition is detected (both NRFD and NDAC are deasserted).

error_no_permission = -1073807192
Access to the remote machine is denied.

error_nonimplemented_operation = -1073807231
The specified operation is unimplemented.

error_nonsupported_attribute = -1073807331
The specified attribute is not defined or supported by the referenced session, event, or find list.

error_nonsupported_attribute_state = -1073807330
The specified state of the attribute is not valid or is not supported as defined by the session, event, or find list.

error_nonsupported_format = -1073807295
A format specifier in the format string is not supported.

error_nonsupported_interrupt = -1073807201
The interface cannot generate an interrupt on the requested level or with the requested statusID value.

error_nonsupported_line = -1073807197
The specified trigger source line (trigSrc) or destination line (trigDest) is not supported by this VISA implementation, or the combination of lines is not a valid mapping.

error_nonsupported_mechanism = -1073807196
The specified mechanism is not supported for the specified event type.

error_nonsupported_mode = -1073807290
The specified mode is not supported by this VISA implementation.

error_nonsupported_offset = -1073807276
Specified offset is not accessible from this hardware.

error_nonsupported_offset_alignment = -1073807248
The specified offset is not properly aligned for the access width of the operation.

error_nonsupported_operation = -1073807257
The session or object reference does not support this operation.

error_nonsupported_varying_widths = -1073807275
Cannot support source and destination widths that are different.

error_nonsupported_width = -1073807242
Specified width is not supported by this hardware.

error_not_cic = -1073807264
The interface associated with this session is not currently the Controller-in-Charge.

error_not_enabled = -1073807313
The session must be enabled for events of the specified type in order to receive them.

error_not_system_controller = -1073807263
The interface associated with this session is not the system controller.

error_output_protocol_violation = -1073807306
Device reported an output protocol error during transfer.

error_queue_error = -1073807301
Unable to queue asynchronous operation.

error_queue_overflow = -1073807315
The event queue for the specified type has overflowed, usually due to not closing previous events.

error_raw_read_protocol_violation = -1073807307
Violation of raw read protocol occurred during transfer.

error_raw_write_protocol_violation = -1073807308
Violation of raw write protocol occurred during transfer.

error_resource_busy = -1073807246
The resource is valid, but VISA cannot currently access it.

error_resource_locked = -1073807345
Specified type of lock cannot be obtained or specified operation cannot be performed because the resource is locked.

error_resource_not_found = -1073807343
Insufficient location information, or the device or resource is not present in the system.

error_response_pending = -1073807271
A previous response is still pending, causing a multiple query error.

error_serial_framing = -1073807253
A framing error occurred during transfer.

error_serial_overrun = -1073807252
An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.

error_serial_parity = -1073807254
A parity error occurred during transfer.

error_session_not_locked = -1073807204
The current session did not have any lock on the resource.

error_srq_not_occurred = -1073807286
Service request has not been received for the session.

error_system_error = -1073807360
Unknown system error.

error_timeout = -1073807339
Timeout expired before operation completed.

error_trigger_not_mapped = -1073807250
The path from the trigger source line (trigSrc) to the destination line (trigDest) is not currently mapped.

error_user_buffer = -1073807247
A specified user buffer is not valid or cannot be accessed for the required size.

error_window_already_mapped = -1073807232
The specified session currently contains a mapped window.

error_window_not_mapped = -1073807273
The specified session is currently unmapped.

success = 0
Operation completed successfully.

success_device_not_present = 1073676413
Session opened successfully, but the device at the specified address is not responding.

success_event_already_disabled = 1073676291
Specified event is already disabled for at least one of the specified mechanisms.

success_event_already_enabled = 1073676290
Specified event is already enabled for at least one of the specified mechanisms.

success_max_count_read = 1073676294
The number of bytes read is equal to the input count.

success_nested_exclusive = 1073676442
Operation completed successfully, and this session has nested exclusive locks.

success_nested_shared = 1073676441
Operation completed successfully, and this session has nested shared locks.

success_no_more_handler_calls_in_chain = 1073676440
Event handled successfully. Do not invoke any other handlers on this session for this event.

success_queue_already_empty = 1073676292
Operation completed successfully, but the queue was already empty.

success_queue_not_empty = 1073676416
Wait terminated successfully on receipt of an event notification. There is still at least one more event occurrence of the requested type(s) available for this session.

success_syncronous = 1073676443
Asynchronous operation request was performed synchronously.

success_termination_character_read = 1073676293
The specified termination character was read.

success_trigger_already_mapped = 1073676414
The path from the trigger source line (trigSrc) to the destination line (trigDest) is already mapped.

warning_configuration_not_loaded = 1073676407
The specified configuration either does not exist or could not be loaded. The VISA-specified defaults are used.

warning_ext_function_not_implemented = 1073676457
The operation succeeded, but a lower level driver did not implement the extended functionality.

warning_nonsupported_attribute_state = 1073676420
Although the specified state of the attribute is valid, it is not supported by this resource implementation.

warning_nonsupported_buffer = 1073676424
The specified buffer is not supported.

warning_null_object = 1073676418
The specified object reference is uninitialized.

warning_queue_overflow = 1073676300
VISA received more event information of the specified type than the configured queue size could hold.

warning_unknown_status = 1073676421
The status code passed to the operation could not be interpreted.

Python Module Index

p

Index

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X