# Contents

1 General overview 3
2 User guide 5
3 More information 25
Python Module Index 187
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:

```python
>>> import visa
>>> rm = visa.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 peripheral 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.
2.1 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
```

2.1.1 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.
2.1.2 Testing your installation

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

```python
>>> import visa
>>> rm = visa.ResourceManager()
>>> print(rm.list_resources())
```

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.

2.1.3 Using the development version

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

```bash
$ 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 CE. It is a free Python distribution by Continuum Analytics that includes many scientific packages.

2.2 Configuring the backend

Currently there are two backends available: The one included in pyvisa, which uses the NI library, 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 NI backend if the NI 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:

```python
>>> 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.

2.3 Configuring the NI backend

**Note:** The NI backend requires that you install first the NI-VISA library. You can 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 `pyvisa.highlevel.ResourceManager` constructor:

```python
>>> 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.
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.

## 2.4 Tutorial

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

### 2.4.1 An example

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

```python
>>> import visa
>>> rm = visa.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 visa, we create a ResourceManager object. If called without arguments, PyVISA will use the default backend (NI) which tries to find the VISA shared library for you. You can check, the location of the shared library used simply by:

```python
>>> 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’ (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).

```python
>>> 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:

```python
>>> my_instrument.query('*IDN?')
```

is the same as:

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

2.4.2 Example for serial (RS232) device

Consider an Oxford ITC4 temperature controller, which is connected through COM2 with my computer. The following code prints its self-identification on the screen:

```python
itc4 = rm.open_resource("COM2")
itc4.write("V")
print(itc4.read())
```

Instead of separate write and read operations, you can do both with one query() call. Thus, the above source code is equivalent to:

```python
print(itc4.query("V"))
```

It couldn’t be simpler.
2.5 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 two different methods for this called `pyvisa.resources.MessageBasedResource.query_ascii_values()` and `pyvisa.resources.MessageBasedResource.query_binary_values()`. It also has the convenient `pyvisa.resources.MessageBasedResource.query_values()` which will use follow a previously established configuration.

2.5.1 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:

```python
>>> 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:

```python
>>> 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:

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

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

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:

```python
>>> 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:

```python
>>> 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)

2.5.2 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`.

2.5. Reading and Writing values
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:

```python
>>> 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 advertised 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.

### 2.5.3 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.

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

Again, you can specify the converter code.

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

decl · 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`.

```python
>>> 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)

### 2.5.4 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.

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

Again you can specify the datatype and *endianness*.

```python
>>> inst.write_binary_values('WLIST:WAVeform:DATA somename,', values, datatype='d',
˓is_big_endian=False)
```
2.5.5 Preconfiguring the transfer format

Most of the cases, each device will transfer data in the same format every time. And making the call so detailed
everytime can be annoying. For this purpose, PyVISA provides a way to preconfigure the default. Each Message
Based Resources exposes an attribute named values_format which is an object with the following properties:
is_binary, datatype, is_big_endian, container. For example to set e.g. little-endian doubles and a
numpy array:

```python
>>> inst.values_format.is_binary = True
>>> inst.values_format.datatype = 'd'
>>> inst.values_format.is_big_endian = False
>>> inst.values_format.container = numpy.array
```

or shorter:

```python
>>> inst.values_format.use_binary('d', False, numpy.array)
```

After doing this, you can simply call:

```python
>>> inst.query_values('CURV?')
```

which will dispatch to the appropriate function and arguments.

If you want to default to ASCII transfer, preconfiguring is a little bit more cumbersome as you need to specify the
converters for both ways.

For example with hex, with '$' as separator:

```python
>>> inst.values_format.is_binary = False
>>> inst.values_format.converter = 'x'
>>> inst.values_format.separator = '$'
>>> inst.values_format.container = numpy.array
```

or shorter:

```python
>>> inst.values_format.use_ascii('x', '$', numpy.array)
```

This works for both query and write operations.

2.5.6 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:

```python
>>> 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:

```python
>>> 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.

### 2.6 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 initialise the instrument:

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

Here, we create the instrument variable `keithley`, which is used for all further operations on the instrument. Immediately after it, we send the initialisation 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.

```python
>>> 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 sensel; 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”:

```python
>>> 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:

```python
>>> 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:

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

That’s it. 18 lines of lucid code. (Well, SCPI is awkward, but that’s another story.)
2.7 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 `pyvisa.highlevel.ResourceManager.open_resource()` method of a `pyvisa.highlevel.ResourceManager`. In general terms, there are two main groups derived from `pyvisa.resources.Resource`, `pyvisa.resources.RegisterBasedResource` and `pyvisa.resources.MessageBasedResource`.

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

```python
>>> 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 encounte more often. For more information, refer to the API.

2.7.1 Attributes Resource

**session**

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

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

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

```python
>>> 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

```python
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:

```python
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.

### 2.7.2 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.

```python
my_instrument.chunk_size = 102400
```

This example sets it to 100 kilobytes.

### 2.7.3 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.

```python
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:

```python
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. So you could write:

```python
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.

### 2.8 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 NI-VISA is the default backend (called *ni*) and is bundled with PyVISA for simplicity.

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

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

is the same as this:

```python
>>> import visa
>>> rm = visa.ResourceManager('@ni')
```

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

```python
>>> import visa
>>> rm = visa.ResourceManager('/path/to/lib@ni')
```

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

PyVISA locates backends by name. If you do:

```python
>>> import visa
>>> rm = visa.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:

```
python -m visa info
```

### 2.8.1 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 python -m visa info or 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_attribute: 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)
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, 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):
de open(self, session, resource_name,
de open_default_resource_manager(self):
de out_8(self, session, space, offset, data, extended=False):
de out_16(self, session, space, offset, data, extended=False):
de out_32(self, session, space, offset, data, extended=False):
de out_64(self, session, space, offset, data, extended=False):
de parse_resource(self, session, resource_name):
de parse_resource_extended(self, session, resource_name):
de peek_8(self, session, address):
de peek_16(self, session, address):
de peek_32(self, session, address):
de peek_64(self, session, address):
de poke_8(self, session, address, data):
de poke_16(self, session, address, data):
de poke_32(self, session, address, data):
de poke_64(self, session, address, data):
de read(self, session, count):
de read_asynchronously(self, session, count):
de read_stb(self, session):
de read_to_file(self, session, filename, count):
de set_attribute(self, session, attribute, attribute_state):
de set_buffer(self, session, mask, size):

2.8. A frontend for multiple backends

(continues on next 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):

2.9 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:

```python
python -m visa shell
```

or:

```python
pyvisa-shell
```

that will show something the following prompt:

```plaintext
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:

```plaintext
(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:

```plaintext
(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:
You can talk to the device using "write", "read" or "query.
The default end of message is added to each message

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

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

We can also get a list of all visa attributes:

(continues on next page)
<table>
<thead>
<tr>
<th>VI_ATTR_VALUE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI_ATTR_ASRL_REPLACE_CHAR</td>
<td>replace_char</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_ASRL_RI_STATE</td>
<td>1073676479</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_ASRL_RTS_STATE</td>
<td>1073676480</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_ASRL_STOP_BITS</td>
<td>1073676324</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_ASRL_WIRE_MODE</td>
<td>1073676735</td>
</tr>
<tr>
<td>128</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_ASRL_XOFF_CHAR</td>
<td>1073676482</td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_ASRL_XON_CHAR</td>
<td>1073676481</td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_DMA_ALLOW_EN</td>
<td>1073676318</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_FILE_APPEND_EN</td>
<td>1073676690</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_INTF_INST_NAME</td>
<td>3221160169</td>
</tr>
<tr>
<td>ASRL1 (/dev/cu.Bluetooth-PDA-Sync)</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_INTF_NUM</td>
<td>1073676662</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_INTF_TYPE</td>
<td>1073676657</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_IO_PROT</td>
<td>1073676316</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_MAX_QUEUE_LENGTH</td>
<td>1073676293</td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RD_BUF_OPER_MODE</td>
<td>1073676330</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RD_BUF_SIZE</td>
<td>1073676331</td>
</tr>
<tr>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RM_SESSION</td>
<td>1073676484</td>
</tr>
<tr>
<td>3160976</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RSRC_CLASS</td>
<td>3221159937</td>
</tr>
<tr>
<td>INSTR</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RSRC_IMPL_VERSION</td>
<td>1073676291</td>
</tr>
<tr>
<td>5243392</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RSRC_LOCK_STATE</td>
<td>1073676292</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RSRC_MANF_ID</td>
<td>1073676661</td>
</tr>
<tr>
<td>4086</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RSRC_MANF_NAME</td>
<td>3221160308</td>
</tr>
<tr>
<td>National Instruments</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RSRC_NAME</td>
<td>3221159938</td>
</tr>
<tr>
<td>ASRL1::INSTR</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_RSRC_SPEC_VERSION</td>
<td>1073676656</td>
</tr>
<tr>
<td>5243136</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_SEND_END_EN</td>
<td>1073676310</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR.Suppress_END_EN</td>
<td>1073676342</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_TERMCHAR</td>
<td>1073676312</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_TERMCHAR_LEN</td>
<td>1073676344</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>VI_ATTR_TMO_VALUE</td>
<td>1073676314</td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
</tbody>
</table>
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
```

## 2.10 PyVisa Shell Backends

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

You can invoke:

```
python -m visa -b sim shell
```

or:

```
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:

```
python -m visa -b py shell
```

or:
uses python-py as backend instead of ni backend, for situation when ni not installed.

## 2.11 PyVisa Info

You can invoke it from the command-line:

```bash
python -m visa info
```

or:

```bash
pyvisa-info
```

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

<table>
<thead>
<tr>
<th>Machine Details:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform ID:</td>
</tr>
<tr>
<td>Processor:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>PyVISA Version:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Backends:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ni:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>py:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>sim:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

## 2.12 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!

## 2.13 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 `pyvisa.highlevel.VisaLibrary`. 
Warning: Notice however that low-level functions might not be present in all backends. For broader compatibility, do not use this layer. All the functionality should be 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 `pyvisa.highlevel.VisaLibrary`.


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 `pyvisa.highlevel.ResourceManager.open_resource()` method to obtain the appropriate object. If needed, you will be able to access the `VisaLibrary` object directly using the `pyvisa.highlevel.ResourceManager.visalib` attribute.

The `VisaLibrary` does the low-level calls. In the default NI 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.

### 2.13.1 Calling middle- and low-level functions

After you have instantiated the `ResourceManager`:

```python
>>> import visa

>>> rm = visa.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:

```python
>>> 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:

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

The calling convention and types are handled by the wrapper.
3.1 VISA resource names

If you use the function 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,

\[
\text{GPIB::10}
\]

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

\[
\text{GPIB1::10}
\]

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

\[
\text{ASRL2}
\]

(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.

3.1.1 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 ([ ]).

<table>
<thead>
<tr>
<th>Address String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIB::10</td>
<td>GPIB instrument with number 10</td>
</tr>
<tr>
<td>GPIB1::10</td>
<td>GPIB instrument on board 1</td>
</tr>
<tr>
<td>ASRL2</td>
<td>ASRL instrument at COM2</td>
</tr>
<tr>
<td>COM2</td>
<td>COM2 interface</td>
</tr>
<tr>
<td>LPT1</td>
<td>LPT1 interface</td>
</tr>
</tbody>
</table>

25
<table>
<thead>
<tr>
<th>Interface</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENET-Serial IN- STR</td>
<td>ASRL[0]:host address::serial port::INSTR</td>
</tr>
<tr>
<td>GPIB INSTR</td>
<td>GPIB[board][::primary address[::secondary address][::INSTR]]</td>
</tr>
<tr>
<td>GPIB INTFC</td>
<td>GPIB[board][::INTFC]</td>
</tr>
<tr>
<td>PXI BACKPLANE</td>
<td>PXI[interface][::chassis number::BACKPLANE]</td>
</tr>
<tr>
<td>PXI INSTR</td>
<td>PXI[board][::bus-device::function][::INSTR]</td>
</tr>
<tr>
<td>PXI INSTR</td>
<td>PXI[interface][::bus-device::function][::INSTR]</td>
</tr>
<tr>
<td>PXI INSTR</td>
<td>PXI[interface][::CHASSIS::chassis number::SLOT::slot number::FUNCTION][::INSTR]</td>
</tr>
<tr>
<td>PXI MEMACC</td>
<td>PXI[interface][::MEMACC]</td>
</tr>
<tr>
<td>Remote NI-VISA</td>
<td>visa://host address[:server port][remote resource]</td>
</tr>
<tr>
<td>Serial INSTR</td>
<td>ASRL[board][::INSTR]</td>
</tr>
<tr>
<td>TCPIP INSTR</td>
<td>TCPIP[board]:host address::LAN device name[::INSTR]</td>
</tr>
<tr>
<td>TCPIP SOCKET</td>
<td>TCPIP[board]:host address::port::SOCKET</td>
</tr>
<tr>
<td>USB INSTR</td>
<td>USB[board][::manufacturer ID::model code::serial number[::USB interface number]][::INSTR]</td>
</tr>
<tr>
<td>USB RAW</td>
<td>USB[board][::manufacturer ID::model code::serial number[::USB interface number]:RAW]</td>
</tr>
<tr>
<td>VXI BACKPLANE</td>
<td>VXI[board][::VXI logical address]:BACKPLANE</td>
</tr>
<tr>
<td>VXI INSTR</td>
<td>VXI[board][::VXI logical address][::INSTR]</td>
</tr>
<tr>
<td>VXI MEMACC</td>
<td>VXI[board][::MEMACC]</td>
</tr>
<tr>
<td>VXI SERVANT</td>
<td>VXI[board][::SERVANT]</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Optional String Segments</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>board</td>
<td>0</td>
</tr>
<tr>
<td>GPIB secondary address</td>
<td>none</td>
</tr>
<tr>
<td>LAN device name</td>
<td>inst0</td>
</tr>
<tr>
<td>PXI bus</td>
<td>0</td>
</tr>
<tr>
<td>PXI function</td>
<td>0</td>
</tr>
<tr>
<td>USB interface number</td>
<td>lowest numbered relevant interface</td>
</tr>
</tbody>
</table>

The following table shows examples of address strings:
### 3.2 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](https://github.com/Lantz/visalib) package as a part of Lantz

#### 3.2.1 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:

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

change it to:

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

If you are doing:
```n
```python
>>> print(visa.get_instruments_list())

change it to:
```n
```python
>>> print(rm.list_resources())

If you are doing:
```n
```python
>>> import pyvisa.vpp43 as vpp43
>>> vpp43.visa_library.load_library("/path/to/my/libvisa.so.7")

change it to:
```n
```python
>>> import visa
>>> rm = visa.ResourceManager("/path/to/my/libvisa.so.7")
>>> lib = rm.visalib

If you are doing::
```n
```python
>>> vpp43.lock(session)

change it to:
```n
```python
>>> lib.lock(session)

or better:
```n
```python
>>> resource.lock()

If you are doing::
```n
```python
>>> inst.term_chars = '\r'

change it to:
```n
```python
>>> inst.read_termination = '\r'
>>> inst.write_termination = '\r'

If you are doing::
```n
```python
>>> print(lib.status)

change it to:
```n
```python
>>> print(lib.last_status)

or even better, do it per resource:
```n
```python
>>> print(rm.last_status) # for the resource manager
>>> print(inst.last_status) # for a specific instrument
```
If you are doing::

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

change it to:

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

As you see, most of the code shown above is making a few things explicit. 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`

### 3.2.2 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 instatiated 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:

```python
>>> lib1 = visa.VisaLibrary("/path/to/my/libvisa.so.7")
>>> lib2 = visa.VisaLibrary("/path/to/my/libvisa.so.7")
>>> lib3 = visa.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:

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

or equivalently:

```python
>>> rm = visa.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 visa
>>> rm = visa.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:

```python
>>> 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.
3.3 Contributing to PyVISA

You can contribute in different ways:

3.3.1 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:

```
python -m visa info
```

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

```
import visa
visa.log_to_screen()
```

3.3.2 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 CE. It is a free Python distribution by Continuum Analytics that includes many scientific packages.

3.3.3 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.
3.3.4 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.

3.4 Frequently asked questions

3.4.1 Is PyVISA endorsed by National Instruments?

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

3.4.2 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 is currently maintained by Hernan E. Grecco <hernan.grecco@gmail.com>.

Take a look at AUTHORS for more information

3.4.3 Is PyVISA thread-safe?

Yes, PyVISA is thread safe starting from version 1.6.

3.4.4 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:

```python
import visa
visa.log_to_screen()
```

3.4.5 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:

```bash
python -m visa info
```

3.4.6 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
3.4.7 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:

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

or:

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

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

3.4.8 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:

```
OS ERR 0: 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:

```
OS ERR 0: 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:

```
python -m visa 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"
```
to 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.

### 3.4.9 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 help file in HTML

### 3.5 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):

```
python -m visa info
```
or equivalently:

```
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.*
3.5.1 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.

3.5.2 Windows

Download NI-VISA for Windows

Supports:

- 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).

3.5.3 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.
3.6 API

3.6.1 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.NIVisaLibrary, which implements a ctypes wrapper around the NI-VISA library.

    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 visa
    >>> rm = visa.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. (Constants.READ*, .WRITE*, .IO*)

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`

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

```python
def get_last_status_in_session(session)
Last status in session.
Helper function to be called by resources properties.
```

```python
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 to 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`

```python
in_64(session, space, offset, extended=False)
```

Reads in a 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`

```python
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`

```python
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.

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`

```python
def 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`

```python
def 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`

```python
def 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

```
move_out_32(session, space, offset, length, data, extended=False)
```

Moves a 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

```
move_out_64(session, space, offset, length, data, extended=False)
```

Moves a 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

```
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 – Specifies the maximum time period (in milliseconds) that this operation waits before returning 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 a 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.


**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.


**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.
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.

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

Corresponds to viUnlock function of the VISA library.

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

Corresponds to viUnmapAddress function of the VISA library.
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

3.6.2 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Matches any one character.</td>
</tr>
<tr>
<td>[list]</td>
<td>Matches any one character from the enclosed list. You can use a hyphen to match a range of characters.</td>
</tr>
<tr>
<td>[^list]</td>
<td>Matches any character not in the enclosed list. You can use a hyphen to match a range of characters.</td>
</tr>
<tr>
<td>•</td>
<td>Matches 0 or more occurrences of the preceding character or expression.</td>
</tr>
<tr>
<td>•</td>
<td>Matches 1 or more occurrences of the preceding character or expression.</td>
</tr>
<tr>
<td>Exp</td>
<td>exp</td>
</tr>
</tbody>
</table>

(exp) Grouping characters or expressions.

Thus the default query, ‘?*::INSTR’, matches any sequences of characters ending ending with ‘::IN-STR’.

    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) – access mode.
• open_timeout – time out to open.

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) – access mode.
• open_timeout – time out to open.
• 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.

3.6.3 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 discarded. (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*) – Milliseconds before the open operation times out.

**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:

```python  
>>> 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_MANF_NAME'], \class 'pyvisa.attributes.AttrVI_ATTR_MAX_QUEUE_LENGTH'], \class 'pyvisa.attributes.AttrVI_ATTR_TMO_VALUE']

**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.
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
            discarded. (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.

clear
encoding
    Encoding used for read and write operations.

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** (:class:`pyvisa.constants.AccessModes`) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** (:class:`int`) – Milliseconds before the open operation times out.

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

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

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'>, delay=None)
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)*

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)

**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` – 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:

```python
>>> 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**

```python
visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_MANF_NAME'>, <class 'pyvisa.attributes.AttrVI_ATTR_RSRC_SPEC_VERSION'>, ...
```
• **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=None, 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** *(None)* – data to be written to the device.
- **encoding** *(None)* – function used to convert each value. String formatting codes are also accepted. Defaults to str.
- **separator** *(callable | str)* – 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_ascii_values** *(message=None, values=None, 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 written to the device.
- **converter** *(callable | str)* – function used to convert each value. String formatting codes are also accepted. Defaults to str.
- **separator** *(callable | str)* – 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=None, values=None, datatype='f', is_big_endian=False, termination=None, encoding=None)*

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 written to the device.
- **datatype** *(str)* – the format string for a single element. See struct module.
- **is_big_endian** *(False)* – boolean indicating endianess.

**Returns** number of bytes written.

**Return type** int

**write_raw** *(message=None)*

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 discarded. (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**  
**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**  
**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**  
**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**  
**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) – Milliseconds before the open operation times out.
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** \(0 \leq \text{value} \leq 4294967295\)

### 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:

```python
>>> 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

```python
visa_attributes_classes = [AttrVI_ATTR_RSRC_MANF_NAME>, 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.

```python
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 discarded. (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 and cause the buffer data to be written to the device.

**Parameters**  
• `mask` – Specifies the action to be taken with flushing the buffer. (Constants.READ*, .WRITE*, .IO*)
**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, TCP/IP, 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 \( 0 \leq \text{value} \leq 65535 \)

**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)* – Milliseconds before the open operation times out.

**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`

```python
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`

```python
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`

```python
query_binary_values(message, datatype='f', is_big_endian=False, container=<class 'list'>, delay=None, header_fmt='ieee', expect_termination=True)
```

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)

**Returns**
the answer from the device.

**Return type**
`list`

```python
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'>, delay=None)*

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)*

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 endianness. 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)

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* – 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)

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)

VI_ATTR_RSRC_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_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** $0 \leq \text{value} \leq 4294967295$

**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): \text{float}('+\text{inf}')
  (for convenience, None is considered as \text{float}('+\text{inf}')

To set an infinite timeout, you can also use:

```python
>>> 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**

```python
visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_MANF_NAME'>, <class 'pyvisa.attributes.AttrVI_ATTR_RSRC_TYPE'>, ...
```
• **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.

```python
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` *(optional)*
- `encoding` *(optional)*

**Returns** number of bytes written.

**Return type** *int*

```python
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 written to the device.
- `converter` *(callable | str) – function used to convert each value. String formatting codes are also accepted. Defaults to str.*
- `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*

```python
write_binary_values(message, values, datatype='f', is_big_endian=False, termination=None, encoding=None)
```
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 written to the device.
- `datatype` – the format string for a single element. See struct module.
- `is_big_endian` – boolean indicating endianess.

**Returns** number of bytes written.

**Return type** *int*

```python
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

3.6. API 91
**clear**

Cleared the 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 discarded. (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.

**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

3.6. API 93
• **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.

```python
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)

```python
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

```python
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**) – Milliseconds before the open operation times out.

```python
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**

```python
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)
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)

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'>, delay=None)
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*)

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)

**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** – 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.

3.6. API
**PyVISA Documentation, Release 1.10.0.dev0**

Raises \texttt{pyvisa.errors.InvalidSession} if session is closed.

**\texttt{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** \texttt{pyvisa.constants.StatusCode}

**\texttt{spec_version}**

\texttt{VIATTR_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** \texttt{VIATTR_RSRC_SPEC_VERSION (1073676656)}

**Type** int

**Range** $0 \leq \text{value} \leq 4294967295$

**stb**

Service request status register.

**\texttt{timeout}**

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

Special values: - \texttt{immediate} \texttt{(VI\_TMO\_IMMEDIATE): 0}

(for convenience, any value smaller than 1 is considered as 0)

- \texttt{infinite} \texttt{(VI\_TMO\_INFINITE): float('inf')} (for convenience, None is considered as float('inf'))

To set an infinite timeout, you can also use:

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

**\texttt{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.

**\texttt{unlock}()**

Relinquishes a lock for the specified resource.

**\texttt{values_format}**
**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.*
- **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 written to the device.
- **converter** *(callable | str) – function used to convert each value. String formatting codes are also accepted. Defaults to str.*
- **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*)

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 written to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.

**Returns** number of bytes written.
write_raw

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

Returns
number of bytes written.

Return type
int

write_termination

write_values

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().

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 dicarden. (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.

**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](https://pyvisa.readthedocs.io/en/stable/api.html#access-mode)) – Specifies the mode by which the resource is to be accessed.

• **open_timeout** *(int)* – Milliseconds before the open operation times out.

**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='\n', 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)

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)

**Returns** the answer from the device.

**Return type** list

```
query_delay = 0.0
```

3.6. API
**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=list, delay=None)*

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=list, header_fmt='ieee', expect_termination=True)*

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)

**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` – 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:

```python
>>> 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_MANF_NAME'>, <class 'pyvisa.attributes.AttrVI_ATTR_RSRC_MFG_MODEL'>, <class 'pyvisa.attributes.AttrVI_ATTR_RSRC_MFG_SERIAL'>, <class 'pyvisa.attributes.AttrVI_ATTR_MAX_QUEUE_LENGTH'>, <class 'pyvisa.attributes.AttrVI_ATTR_TMO_VALUE'>]`

`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.

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 written to the device.
• **converter** *(callable | str)* – function used to convert each value. String formatting codes are also accepted. Defaults to str.

• **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)*

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 written to the device.

• **datatype** – the format string for a single element. See struct module.

• **is_big_endian** – boolean indicating endianess.

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 = ''
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 discarded. (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.

**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*) – Milliseconds before the open operation times out.

**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=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=list, delay=None, header_fmt='ieee', expect_termination=True)*
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)

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=[list], delay=None)

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)

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)
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** – 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**

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:

```python
>>> 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**
PyVISA Documentation, Release 1.10.0.dev0

visa_attributes_classes = [
    <class 'pyvisa.attributes.AttrVI_ATTR_RSRC_MANF_NAME'>,
    <class 'pyvisa.attributes.AttrVI_ATTR_RSRCבני_ベン_NAME'>,
    <class 'pyvisa.attributes.AttrVI_ATTR_MAX_QUEUE_LENGTH'>,
    <class 'pyvisa.attributes.AttrVI_ATTR_TMO_VALUE'>
]

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.

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 written to the device.

• converter (callable | str) – function used to convert each value. String formatting codes are also accepted. Defaults to str.

• 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)

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 written to the device.

• datatype – the format string for a single element. See struct module.

• is_big_endian – boolean indicating endianess.
**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.

**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, TCP/IP, 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**  
  \(\text{pyvisa.constants.AccessModes}\) – Specifies the mode by which the resource is to be accessed.

- **open_timeout**  
  \(\text{int}\) – Milliseconds before the open operation times out.

**query**  
\((message, delay=None)\)

A combination of write(message) and read()

**Parameters**

- **message**  
  \(\text{str}\) – 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.

<table>
<thead>
<tr>
<th>Type separator: (str) -&gt; collections.Iterable[int]</th>
<th>str</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Returns the answer from the device.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Return type list</th>
</tr>
</thead>
</table>

**query_binary_values** (*message*, *datatype*='f', *is_big_endian*=False, *container*=<class 'list'>, *delay*=None, *header_fmt*='ieee', *expect_termination*=True)

Query the device for values in binary format returning an iterable of values.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
</table>

• **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)

<table>
<thead>
<tr>
<th>Returns the answer from the device.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Return type list</th>
</tr>
</thead>
</table>

**query_delay** = 0.0

**query_values** (*message*, *delay*=None)

Query the device for values returning an iterable of values.

<table>
<thead>
<tr>
<th>The datatype expected is obtained from values_format</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Returns the answer from the device.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Return type list</th>
</tr>
</thead>
</table>

**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'>, delay=None)*

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)*

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)

**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` – 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** (3221160308)

**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.

specification 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:

```python
>>> 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_MANF_NAME'>, <class 'pyvisa.attributes_ATTRVI_ATTR_MAX_QUEUE_LENGTH'>, <class 'pyvisa.attributes_ATTRVI_ATTR_TMO_VALUE'>]

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.
    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 written to the device.
    • converter (callable | str) – function used to convert each value. String formatting codes are also accepted. Defaults to str.
    • 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

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

Parameters

- **message** *(unicode (Py2) or str (Py3)) – the message to be sent.*
- **values** – data to be written to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.

Returns number of bytes written.

Return type int

write_raw

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

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 discarded. (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)
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.

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  $0 \leq \text{value} \leq 65535$

**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**  $0 \leq \text{value} \leq 65535$

**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)
**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`) – Milliseconds before the open operation times out.

**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
• **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)*

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)

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'>, delay=None)*

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)*

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)

**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** – 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**

**VIATTR_GPIB_REN_STATE** returns the current state of the GPIB REN (Remote ENable) interface line.

**VISA Attribute** VIATTR_GPIB_REN_STATE (1073676673)

**Type** class:pyvisa.constants.LineState

**resource_class**

**VIATTR_RSRC_CLASS** specifies the resource class (for example, “INSTR”) as defined by the canonical resource name.

**VISA Attribute** VIATTR_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**

**VIATTR_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** VIATTR_RSRC_MANF_NAME (3221160308)

**resource_name**

**VIATTR_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_GPIBSECONDARY_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_GPIBSECONDARY_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 to 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_MANF_NAME'>, ...

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. Default: 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.

```python
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.
- **Returns** number of bytes written.
- **Return type** int

```python
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 written to the device.
- **converter** *(callable | str)* – function used to convert each value. String formatting codes are also accepted. Defaults to str.
- **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

```python
write_binary_values(message, values, datatype='f', is_big_endian=False, termination=None, encoding=None)
```
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 written to the device.
- **datatype** – the format string for a single element. See struct module.
- **is_big_endian** – boolean indicating endianess.

**Returns** number of bytes written.

**Return type** int

```python
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 discarded. (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

**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 appli-
cation.
•  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,
TCP/IP, 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) – Milliseconds before the open operation times out.

**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 $0 \leq \text{value} \leq 30$ 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 to 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:

```python
>>> 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_MANF_NAME'>, ...

**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.

```python
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()
```

```python
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 discarded. (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 \( (\text{access\_mode}=\langle \text{AccessModes.no\_lock: 0} \rangle, \text{open\_timeout}=5000) \)

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) – Milliseconds before the open operation times out.

read\_memory \( (\text{space, offset, width, extended}=\text{False}) \)

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 \( (\text{interface\_type, resource\_class}) \)

**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

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_MANF_NAME'], <class 'pyvisa.attributes.AttrVI_ATTR_MAX_QUEUE_LENGTH'>, <class 'pyvisa.attributes.AttrVI_ATTR_TMO_VALUE'>]
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][CHASSIS:chassis number][SLOT:slot number][function][::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 discarded. (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)

<table>
<thead>
<tr>
<th>Type</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>0 &lt;= value &lt;= 4294967295</td>
</tr>
</tbody>
</table>

**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)

<table>
<thead>
<tr>
<th>Type</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>0 &lt;= value &lt;= 65535</td>
</tr>
</tbody>
</table>

**interface_type**
The interface type of the resource as a number.

**last_status**
Last status code for this session.

**Returns** 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) – Milliseconds before the open operation times out.

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 viln* 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_MANF_NAME'>, ...

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 discarded. (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) – Milliseconds before the open operation times out.

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:

```python
>>> 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.

```python
type(visa_attributes_classes)  # visa_attributes_classes = [<class 'pyvisa.attributes.AttrVI_ATTR_RSRC_MANF_NAME'>, <class ... <class 'pyvisa.attributes.AttrVI_ATTR_MAX_QUEUE_LENGTH'>, <class 'pyvisa.attributes.AttrVI_ATTR_TMO_VALUE'>]
```

**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.

```python
type(write_memory)  # 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.

```python
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 discarded. (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 \texttt{warnings\_constants} – constants identifying the warnings to ignore.

implementation_version

\texttt{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.

\textbf{VISA Attribute} \texttt{VI\_ATTR\_RSRC\_IMPL\_VERSION} (1073676291)

\textbf{Type} int

\textbf{Range} 0 <= value <= 4294967295

install_handler (\texttt{event\_type}, \texttt{handler}, \texttt{user\_handle}=None)

Installs handlers for event callbacks in this resource.

Parameters

• \texttt{event\_type} – Logical event identifier.

• \texttt{handler} – Interpreted as a valid reference to a handler to be installed by a client application.

• \texttt{user\_handle} – A value specified by an application that can be used for identifying handlers uniquely for an event type.

Returns \texttt{user\_handle} (a ctypes object)

interface_number

\texttt{VI\_ATTR\_INTF\_NUM} specifies the board number for the given interface.

\textbf{VISA Attribute} \texttt{VI\_ATTR\_INTF\_NUM} (1073676662)

\textbf{Type} int

\textbf{Range} 0 <= value <= 65535

interface_type

The interface type of the resource as a number.

io_protocol

\texttt{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, TCP/IP, or USB RAW, you can choose normal transfers or 488.2-defined strings. In USB INSTR, you can choose normal or vendor-specific transfers.

\textbf{VISA Attribute} \texttt{VI\_ATTR\_IO\_PROT} (1073676316)

\textbf{Type} int

\textbf{Range} 0 <= value <= 65535

is_4882_compliant

\texttt{VI\_ATTR\_4882\_COMPLIANT} specifies whether the device is 488.2 compliant.

\textbf{VISA Attribute} \texttt{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) – Milliseconds before the open operation times out.

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:

```python
>>> 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_MANF_NAME'>, <class 'pyvisa.attributes.AttrVI_ATTR_MAX_QUEUE_LENGTH'>, <class 'pyvisa.attributes.AttrVI_ATTR_TMO_VALUE'>]`

`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

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** \(0 \leq value \leq 1\)

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 discarded. (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** (:class:pyvisa.constants.AccessModes) – Specifies the mode by which the resource is to be accessed.
- **open_timeout** *(int)* – Milliseconds before the open operation times out.
**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:

```python
>>> 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_MANF_NAME'>, <class 'pyvisa.attributes.AttrVI_ATTR_RSRC_DISPLAY_STRING'>, ... <class 'pyvisa.attributes.AttrVI_ATTR_MAX_QUEUE_LENGTH'>, <class 'pyvisa.attributes.AttrVI_ATTR_TMO_VALUE'>]

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 discarded. (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** (:class:`pyvisa.constants.AccessModes`) – Specifies the mode by which the resource is to be accessed.

- **open_timeout (int)** – Milliseconds before the open operation times out.
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:

```python
>>> 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_MANF_NAME'>, <class ...

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.

3.6.4 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 resource 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 terminates 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
TCP/IP 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.

listener = 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.
p

pyvisa.constants, 179
Index

A
AccessModes (class in pyvisa.constants), 179
address_state (pyvisa.resources.GPIBInterface attribute), 139
AddressState (class in pyvisa.constants), 181
allow_dma (pyvisa.resources.GPIBInstrument attribute), 128
allow_dma (pyvisa.resources.GPIBInterface attribute), 139
allow_dma (pyvisa.resources.PXIInstrument attribute), 151
allow_dma (pyvisa.resources.SerialInstrument attribute), 80
allow_dma (pyvisa.resources.TCPIPInstrument attribute), 91
allow_dma (pyvisa.resources.VXIInstrument attribute), 163
allow_dma (pyvisa.resources.VXIMemory attribute), 169
allow_transmit (pyvisa.resources.SerialInstrument attribute), 80
asrl (pyvisa.constants.InterfaceType attribute), 180
assert_interrupt_signal() (pyvisa.highlevel.VisaLibraryBase method), 37
assert_trigger() (pyvisa.highlevel.VisaLibraryBase method), 37
assert_trigger() (pyvisa.resources.GPIBInstrument method), 128
assert_trigger() (pyvisa.resources.MessageBasedResource method), 67
assert_trigger() (pyvisa.resources.SerialInstrument method), 80
assert_trigger() (pyvisa.resources.TCPIPInstrument method), 91
assert_trigger() (pyvisa.resources.TCPIP Socket method), 100
assert_trigger() (pyvisa.resources.USBInstrument method), 108
assert_trigger() (pyvisa.resources.USBRaw method), 119
assert_utility_signal() (pyvisa.highlevel.VisaLibraryBase method), 37
asserted (pyvisa.constants.LineState attribute), 181
atn_state (pyvisa.resources.GPIBInterface attribute), 139

B
baud_rate (pyvisa.resources.SerialInstrument attribute), 80
before_close() (pyvisa.resources.FirewireInstrument method), 146
before_close() (pyvisa.resources.GPIBInstrument method), 128
before_close() (pyvisa.resources.GPIBInterface method), 139
before_close() (pyvisa.resources.MessageBasedResource method), 67
before_close() (pyvisa.resources.PXIInstrument method), 151
before_close() (pyvisa.resources.PXIMemory method), 158
before_close() (pyvisa.resources.RegisterBasedResource method), 75
before_close() (pyvisa.resources.Resource method), 62
before_close() (pyvisa.resources.SerialInstrument method), 81
before_close() (pyvisa.resources.TCPIPInstrument method), 91
before_close() (pyvisa.resources.TCPIP Socket method), 100
before_close() (pyvisa.resources.USBInstrument method), 108
before_close() (pyvisa.resources.USBRaw method), 119
before_close() (pyvisa.resources.VXIBackplane method), 175
before_close() (pyvisa.resources.VXIInstrument method), 163
before_close() (pyvisa.resources.VXIMemory method), 169
break_length (pyvisa.resources.SerialInstrument attribute), 81
<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>break_state</td>
<td>pyvisa.resources.SerialInstrument attribute</td>
<td>81</td>
</tr>
<tr>
<td>buffer_read()</td>
<td>pyvisa.highlevel.VisaLibraryBase method</td>
<td>38</td>
</tr>
<tr>
<td>buffer_write()</td>
<td>pyvisa.highlevel.VisaLibraryBase method</td>
<td>38</td>
</tr>
<tr>
<td>bytes_in_buffer</td>
<td>pyvisa.resources.SerialInstrument attribute</td>
<td>81</td>
</tr>
<tr>
<td>chunk_size</td>
<td>pyvisa.resources.GPIBInstrument attribute</td>
<td>128</td>
</tr>
<tr>
<td>chunk_size</td>
<td>pyvisa.resources.MessageBasedResource attribute</td>
<td>67</td>
</tr>
<tr>
<td>chunk_size</td>
<td>pyvisa.resources.SerialInstrument attribute</td>
<td>81</td>
</tr>
<tr>
<td>chunk_size</td>
<td>pyvisa.resources.TCPIPInstrument attribute</td>
<td>91</td>
</tr>
<tr>
<td>chunk_size</td>
<td>pyvisa.resources.TCPIPSocket attribute</td>
<td>100</td>
</tr>
<tr>
<td>chunk_size</td>
<td>pyvisa.resources.USBI nstrument attribute</td>
<td>108</td>
</tr>
<tr>
<td>chunk_size</td>
<td>pyvisa.resources.USBRaw attribute</td>
<td>119</td>
</tr>
<tr>
<td>clear()</td>
<td>pyvisa.highlevel.VisaLibraryBase method</td>
<td>38</td>
</tr>
<tr>
<td>clear()</td>
<td>pyvisa.resources.FirewireInstrument method</td>
<td>146</td>
</tr>
<tr>
<td>clear()</td>
<td>pyvisa.resources.GPIBInstrument method</td>
<td>128</td>
</tr>
<tr>
<td>clear()</td>
<td>pyvisa.resources.GPIBInterface method</td>
<td>139</td>
</tr>
<tr>
<td>clear()</td>
<td>pyvisa.resources.MessageBasedResource method</td>
<td>67</td>
</tr>
<tr>
<td>clear()</td>
<td>pyvisa.resources.PXIInstrument method</td>
<td>151</td>
</tr>
<tr>
<td>clear()</td>
<td>pyvisa.resources.PXIMemory method</td>
<td>158</td>
</tr>
<tr>
<td>clear()</td>
<td>pyvisa.resources.RegisterBasedResource method</td>
<td>75</td>
</tr>
<tr>
<td>close()</td>
<td>pyvisa.resources.RegisterBasedResource method</td>
<td>75</td>
</tr>
<tr>
<td>close()</td>
<td>pyvisa.resources.Resource method</td>
<td>62</td>
</tr>
<tr>
<td>close()</td>
<td>pyvisa.resources.TCPIPInstrument method</td>
<td>81</td>
</tr>
<tr>
<td>close()</td>
<td>pyvisa.resources.TCPIPSocket method</td>
<td>100</td>
</tr>
<tr>
<td>control_atn()</td>
<td>pyvisa.resources.GPIBInstrument method</td>
<td>129</td>
</tr>
<tr>
<td>control_atn()</td>
<td>pyvisa.resources.GPIBInterface method</td>
<td>139</td>
</tr>
<tr>
<td>control_in()</td>
<td>pyvisa.resources.USBI nstrument method</td>
<td>109</td>
</tr>
<tr>
<td>control_out()</td>
<td>pyvisa.resources.USBI nstrument method</td>
<td>109</td>
</tr>
<tr>
<td>control_ren()</td>
<td>pyvisa.resources.GPIBInstrument method</td>
<td>129</td>
</tr>
<tr>
<td>control_ren()</td>
<td>pyvisa.resources.GPIBInterface method</td>
<td>139</td>
</tr>
<tr>
<td>control_ren()</td>
<td>pyvisa.resources.TCPIPInstrument method</td>
<td>92</td>
</tr>
<tr>
<td>control_ren()</td>
<td>pyvisa.resources.USBI nstrument method</td>
<td>109</td>
</tr>
<tr>
<td>CR</td>
<td>pyvisa.resources.GPIBInstrument attribute</td>
<td>128</td>
</tr>
<tr>
<td>CR</td>
<td>pyvisa.resources.MessageBasedResource attribute</td>
<td>66</td>
</tr>
<tr>
<td>CR</td>
<td>pyvisa.resources.SerialInstrument attribute</td>
<td>80</td>
</tr>
<tr>
<td>CR</td>
<td>pyvisa.resources.TCPIPInstrument attribute</td>
<td>91</td>
</tr>
<tr>
<td>CR</td>
<td>pyvisa.resources.TCPIPSocket attribute</td>
<td>100</td>
</tr>
<tr>
<td>CR</td>
<td>pyvisa.resources.USBI nstrument attribute</td>
<td>108</td>
</tr>
<tr>
<td>CR</td>
<td>pyvisa.resources.USBRaw attribute</td>
<td>119</td>
</tr>
<tr>
<td>data_bits</td>
<td>pyvisa.resources.SerialInstrument attribute</td>
<td>81</td>
</tr>
<tr>
<td>destination_increment</td>
<td>pyvisa.resources.PXIInstrument attribute</td>
<td>151</td>
</tr>
<tr>
<td>destination_increment</td>
<td>pyvisa.resources.PXIMemory attribute</td>
<td>158</td>
</tr>
<tr>
<td>destination_increment</td>
<td>pyvisa.resources.VXIMemory attribute</td>
<td>164</td>
</tr>
<tr>
<td>disable_event()</td>
<td>pyvisa.highlevel.VisaLibraryBase method</td>
<td>169</td>
</tr>
<tr>
<td>disable_event()</td>
<td>pyvisa.resources.FirewireInstrument method</td>
<td>146</td>
</tr>
<tr>
<td>disable_event()</td>
<td>pyvisa.resources.GPIBInstrument method</td>
<td>129</td>
</tr>
<tr>
<td>disable_event()</td>
<td>pyvisa.resources.GPIBInterface method</td>
<td>140</td>
</tr>
</tbody>
</table>
Index 191
encoding (pyvisa.resources.USBInstrument attribute), 110
encoding (pyvisa.resources.USBRaw attribute), 120
end_input (pyvisa.resources.SerialInstrument attribute), 82
EOI line, 15
error_abort (pyvisa.constants.StatusCode attribute), 181
error_allocation (pyvisa.constants.StatusCode attribute), 181
error_attribute_read_only (pyvisa.constants.StatusCode attribute), 181
error_bus_error (pyvisa.constants.StatusCode attribute), 181
error_closing_failed (pyvisa.constants.StatusCode attribute), 181
error_connection_lost (pyvisa.constants.StatusCode attribute), 181
error_file_access (pyvisa.constants.StatusCode attribute), 182
error_file_i_o (pyvisa.constants.StatusCode attribute), 182
error_handler_not_installed (pyvisa.constants.StatusCode attribute), 182
error_in_progress (pyvisa.constants.StatusCode attribute), 182
error_input_protocol_violation (pyvisa.constants.StatusCode attribute), 182
error_interface_number_not_configured (pyvisa.constants.StatusCode attribute), 182
error_interrupt_pending (pyvisa.constants.StatusCode attribute), 182
error_invalid_access_key (pyvisa.constants.StatusCode attribute), 182
error_invalid_access_mode (pyvisa.constants.StatusCode attribute), 182
error_invalid_address_space (pyvisa.constants.StatusCode attribute), 182
error_invalid_context (pyvisa.constants.StatusCode attribute), 182
error_invalid_degree (pyvisa.constants.StatusCode attribute), 182
error_invalid_event (pyvisa.constants.StatusCode attribute), 182
error_invalid_expression (pyvisa.constants.StatusCode attribute), 182
error_invalid_format (pyvisa.constants.StatusCode attribute), 182
error_invalid_handler_reference (pyvisa.constants.StatusCode attribute), 182
error_invalid_job_i_d (pyvisa.constants.StatusCode attribute), 182
error_invalid_length (pyvisa.constants.StatusCode attribute), 182
error_invalid_lock_type (pyvisa.constants.StatusCode attribute), 182
error_invalid_mask (pyvisa.constants.StatusCode attribute), 182
error_invalid_mechanism (pyvisa.constants.StatusCode attribute), 182
error_invalid_mode (pyvisa.constants.StatusCode attribute), 183
error_invalid_object (pyvisa.constants.StatusCode attribute), 183
error_invalid_offset (pyvisa.constants.StatusCode attribute), 183
error_invalid_parameter (pyvisa.constants.StatusCode attribute), 183
error_invalid_protocol (pyvisa.constants.StatusCode attribute), 183
error_invalid_resource_name (pyvisa.constants.StatusCode attribute), 183
error_invalid_setup (pyvisa.constants.StatusCode attribute), 183
error_invalid_size (pyvisa.constants.StatusCode attribute), 183
error_invalid_width (pyvisa.constants.StatusCode attribute), 183
error_io (pyvisa.constants.StatusCode attribute), 183
error_machine_not_available (pyvisa.constants.StatusCode attribute), 183
error_memory_not_shared (pyvisa.constants.StatusCode attribute), 183
error_no_listeners (pyvisa.constants.StatusCode attribute), 183
error_no_permission (pyvisa.constants.StatusCode attribute), 183
error_nonimplemented_operation (pyvisa.constants.StatusCode attribute), 183
error_nonsupported_attribute (pyvisa.constants.StatusCode attribute), 183
error_nonsupported_attribute_state (pyvisa.constants.StatusCode attribute), 183
error_nonsupported_format (pyvisa.constants.StatusCode attribute), 183
error_nonsupported_interrupt (pyvisa.constants.StatusCode attribute), 183
error_nonsupported_line (pyvisa.constants.StatusCode attribute), 184
error_nonsupported_mode (pyvisa.constants.StatusCode attribute), 184

Index
error_nonsupported_offset (pyvisa.constants.StatusCode attribute), 184
error_nonsupported_offset_alignment (pyvisa.constants.StatusCode attribute), 184
ever_nonsupported_operation (pyvisa.constants.StatusCode attribute), 184
error_nonsupported_varying_widths (pyvisa.constants.StatusCode attribute), 184
error_nonsupported_width (pyvisa.constants.StatusCode attribute), 184
ever_not_cic (pyvisa.constants.StatusCode attribute), 184
ever_not_enabled (pyvisa.constants.StatusCode attribute), 184
ever_not_system_controller (pyvisa.constants.StatusCode attribute), 184
ever_output_protocol_violation (pyvisa.constants.StatusCode attribute), 184
ever_queue_error (pyvisa.constants.StatusCode attribute), 184
ever_queue_overflow (pyvisa.constants.StatusCode attribute), 184
ever_raw_read_protocol_violation (pyvisa.constants.StatusCode attribute), 184
ever_raw_write_protocol_violation (pyvisa.constants.StatusCode attribute), 184
ever_resource_busy (pyvisa.constants.StatusCode attribute), 184
ever_resource_locked (pyvisa.constants.StatusCode attribute), 184
ever_resource_not_found (pyvisa.constants.StatusCode attribute), 184
ever_response_pending (pyvisa.constants.StatusCode attribute), 184
ever_serial_framing (pyvisa.constants.StatusCode attribute), 185
ever_serial_overrun (pyvisa.constants.StatusCode attribute), 185
ever_serial_parity (pyvisa.constants.StatusCode attribute), 185
ever_session_not_locked (pyvisa.constants.StatusCode attribute), 185
ever_srq_not_occurred (pyvisa.constants.StatusCode attribute), 185
ever_system_error (pyvisa.constants.StatusCode attribute), 185
ever_timeout (pyvisa.constants.StatusCode attribute), 185
ever_trigger_not_mapped (pyvisa.constants.StatusCode attribute), 185
ever_window_not_mapped (pyvisa.constants.StatusCode attribute), 185
ever_user_buffer (pyvisa.constants.StatusCode attribute), 185

G

get_attribute() (pyvisa.highlevel.VisaLibraryBase method), 39
get_debug_info() (pyvisa.highlevel.VisaLibraryBase static method), 40
get_last_status_in_session() (pyvisa.highlevel.VisaLibraryBase method), 40
get_library_paths() (pyvisa.highlevel.VisaLibraryBase static method), 40
get_visa_attribute() (pyvisa.resources.FirewireInstrument method), 146
get_visa_attribute() (pyvisa.resources.GPIBInstrument method), 130
get_visa_attribute() (pyvisa.resources.GPIBInterface method), 140
get_visa_attribute() (pyvisa.resources.MessageBasedResource method), 67
get_visa_attribute() (pyvisa.resources.PXIInstrument method), 152
get_visa_attribute() (pyvisa.resources.PXIMemory method), 158
get_visa_attribute() (pyvisa.resources.RegisterBasedResource method), 75
get_visa_attribute() (pyvisa.resources.Resource method), 63
get_visa_attribute() (pyvisa.resources.SerialInstrument method), 82
get_visa_attribute() (pyvisa.resources.TCPIPInstrument method), 92
get_visa_attribute() (pyvisa.resources.TCPIPSocket method), 101
get_visa_attribute() (pyvisa.resources.USBIInstrument method), 110
get_visa_attribute() (pyvisa.resources.USBRaw method), 120
get_visa_attribute() (pyvisa.resources.VXIBackplane method), 176
get_visa_attribute() (pyvisa.resources.VXIInstrument method), 164
get_visa_attribute() (pyvisa.resources.VXIMemory
method), 170

gpib (pyvisa.constants.InterfaceType attribute), 180
gpib_command() (pyvisa.highlevel.VisaLibraryBase
method), 40
gpib_control_atn() (pyvisa.highlevel.VisaLibraryBase
method), 40
gpib_control_ren() (pyvisa.highlevel.VisaLibraryBase
method), 40
gpib_pass_control() (pyvisa.highlevel.VisaLibraryBase
method), 40
gpib_send_ifc() (pyvisa.highlevel.VisaLibraryBase
method), 41
gpib_vxi (pyvisa.constants.InterfaceType attribute), 180
GPIBInstrument (class in pyvisa.resources), 128
GPIBInterface (class in pyvisa.resources), 139
group_execute_trigger() (pyvisa.resources.GPIBInterface
method), 140

H

handlers (pyvisa.highlevel.VisaLibraryBase attribute), 41
hs488 (pyvisa.constants.IOProtocol attribute), 181

I

ignore_warning() (pyvisa.highlevel.VisaLibraryBase
method), 41
ignore_warning() (pyvisa.resources.FirewireInstrument
method), 147
ignore_warning() (pyvisa.resources.GPIBInstrument
method), 130
ignore_warning() (pyvisa.resources.GPIBInterface
method), 140
ignore_warning() (pyvisa.resources.MessageBasedResource
method), 67
ignore_warning() (pyvisa.resources.PXIInstrument
method), 152
ignore_warning() (pyvisa.resources.PXIMemory
method), 159
ignore_warning() (pyvisa.resources.RegisterBasedResource
method), 75
ignore_warning() (pyvisa.resources.Resource method),
63
ignore_warning() (pyvisa.resources.SerialInstrument
method), 83
ignore_warning() (pyvisa.resources.TCPIPInstrument
method), 92
ignore_warning() (pyvisa.resources.TCPIPSocket
method), 101
ignore_warning() (pyvisa.resources.USBIInstrument
method), 110
ignore_warning() (pyvisa.resources.USBRaw method),
120
ignore_warning() (pyvisa.resources.VXIBackplane
method), 176

ignore_warning() (pyvisa.resources.VXIInstrument
method), 164
ignore_warning() (pyvisa.resources.VXIMemory
method), 170
implementation_version (pyvisa.resources.FirewireInstrument
attribute), 147
implementation_version (pyvisa.resources.GPIBInstrument
attribute), 130
implementation_version (pyvisa.resources.GPIBInterface
attribute), 140
implementation_version (pyvisa.resources.MessageBasedResource
attribute), 67
implementation_version (pyvisa.resources.PXIInstrument
attribute), 152
implementation_version (pyvisa.resources.PXIMemory
attribute), 159
implementation_version (pyvisa.resources.RegisterBasedResource
attribute), 75
implementation_version (pyvisa.resources.Resource attribute), 63
implementation_version (pyvisa.resources.SerialInstrument
attribute), 83
implementation_version (pyvisa.resources.TCPIPInstrument
attribute), 92
implementation_version (pyvisa.resources.TCPIPSocket
attribute), 101
implementation_version (pyvisa.resources.USBIInstrument
attribute), 110
implementation_version (pyvisa.resources.USBRaw attribute), 120
implementation_version (pyvisa.resources.VXIBackplane
attribute), 176
implementation_version (pyvisa.resources.VXIInstrument
attribute), 165
implementation_version (pyvisa.resources.VXIMemory
attribute), 170
in_16() (pyvisa.highlevel.VisaLibraryBase method), 41
in_32() (pyvisa.highlevel.VisaLibraryBase method), 41
in_64() (pyvisa.highlevel.VisaLibraryBase method), 42
in_8() (pyvisa.highlevel.VisaLibraryBase method), 42
install_handler() (pyvisa.highlevel.VisaLibraryBase
method), 42
install_handler() (pyvisa.resources.FirewireInstrument
method), 147
install_handler() (pyvisa.resources.GPIBInstrument
method), 130
install_handler() (pyvisa.resources.GPIBInterface
method), 141
install_handler() (pyvisa.resources.MessageBasedResource
method), 68
install_handler() (pyvisa.resources.PXIInstrument
method), 153
install_handler() (pyvisa.resources.PXIMemory method),
159
install_handler() (pyvisa.resources.RegisterBasedResource method), 76
install_handler() (pyvisa.resources.Resource method), 63
install_handler() (pyvisa.resources.SerialInstrument method), 83
install_handler() (pyvisa.resources.TCPIPInstrument method), 93
install_handler() (pyvisa.resources.TCPIPSocket method), 101
install_handler() (pyvisa.resources.USBIInterface method), 110
install_handler() (pyvisa.resources.USBRaw method), 120
install_handler() (pyvisa.resources.VXIBackplane method), 176
install_handler() (pyvisa.resources.VXIInstrument method), 165
install_handler() (pyvisa.resources.VXIMemory method), 171
install_visa_handler() (pyvisa.highlevel.VisaLibraryBase method), 43
interface_number (pyvisa.resources.FirewireInstrument attribute), 147
interface_number (pyvisa.resources.GPIBInstrument attribute), 130
interface_number (pyvisa.resources.GPIBInterface attribute), 141
interface_number (pyvisa.resources.MessageBasedResource attribute), 68
interface_number (pyvisa.resources.PXIInstrument attribute), 153
interface_number (pyvisa.resources.PXIMemory attribute), 159
interface_number (pyvisa.resources.USBIInterface attribute), 64
interface_number (pyvisa.resources.TCPIPInstrument attribute), 93
interface_number (pyvisa.resources.TCPIPSocket attribute), 101
interface_number (pyvisa.resources.USBRaw attribute), 120
interface_number (pyvisa.resources.VXIBackplane attribute), 177
interface_number (pyvisa.resources.VXIInstrument attribute), 165
interface_number (pyvisa.resources.VXIMemory attribute), 171
InterfaceType (class in pyvisa.constants), 180
io_protocol (pyvisa.resources.GPIBInstrument attribute), 131
io_protocol (pyvisa.resources.GPIBInterface attribute), 141
io_protocol (pyvisa.resources.SerialInstrument attribute), 83
io_protocol (pyvisa.resources.TCPIPSocket attribute), 101
io_protocol (pyvisa.resources.USBRaw attribute), 111
io_protocol (pyvisa.resources.VXIBackplane attribute), 181
IOProtocol (class in pyvisa.constants), 181
is_4882_compliant (pyvisa.resources.USBIInterface attribute), 111
is_4882_compliant (pyvisa.resources.VXIInstrument attribute), 165
is_controller_in_charge (pyvisa.resources.GPIBInterface attribute), 141
is_system_controller (pyvisa.resources.GPIBInterface attribute), 141
issue_warning_on (pyvisa.highlevel.VisaLibraryBase attribute), 43
last_bit (pyvisa.constants.SerialTermination attribute), 180
last_status (pyvisa.highlevel.ResourceManager attribute), 60
last_status (pyvisa.highlevel.VisaLibraryBase attribute), 43
last_status (pyvisa.resources.FirewireInstrument attribute), 147
last_status (pyvisa.resources.GPIBInstrument attribute), 131
last_status (pyvisa.resources.GPIBInterface attribute), 142
last_status (pyvisa.resources.MessageBasedResource attribute), 68
last_status (pyvisa.resources.PXIInstrument attribute), 153
last_status (pyvisa.resources.PXIMemory attribute), 159
last_status (pyvisa.resources.Resource attribute), 64
last_status (pyvisa.resources.SerialInstrument attribute), 84
last_status (pyvisa.resources.TCPIPInstrument attribute), 93
last_status (pyvisa.resources.TCPIPSocket attribute), 102
last_status (pyvisa.resources.USBRaw attribute), 121
last_status (pyvisa.resources.VXIBackplane attribute), 177
last_status (pyvisa.resources.VXIInstrument attribute), 166
LF (pyvisa.resources.GPIBInstrument attribute), 128
LF (pyvisa.resources.MessageBasedResource attribute), 66
LF (pyvisa.resources.SerialInstrument attribute), 80
LF (pyvisa.resources.TCPIPInstrument attribute), 91
LF (pyvisa.resources.TCPIPSocket attribute), 100
LF (pyvisa.resources.USBRaw attribute), 108
LF (pyvisa.resources.VXIBackplane attribute), 119
LineState (class in pyvisa.constants), 181
list_resources() (pyvisa.highlevel.ResourceManager method), 60
list_resources() (pyvisa.highlevel.VisaLibraryBase method), 43
list_resources_info() (pyvisa.highlevel.ResourceManager method), 60
listener (pyvisa.constants.AddressState attribute), 181
lock() (pyvisa.highlevel.VisaLibraryBase method), 43
lock() (pyvisa.resources.FirewireInstrument method), 147
lock() (pyvisa.resources.GPIBInstrument method), 131
lock() (pyvisa.resources.GPIBInterface method), 142
lock() (pyvisa.resources.MessageBasedResource method), 68
lock() (pyvisa.resources.PXIInstrument method), 153
lock() (pyvisa.resources.PXIMemory method), 159
lock() (pyvisa.resources.RegisterBasedResource method), 76
lock() (pyvisa.resources.Resource method), 64
lock() (pyvisa.resources.SerialInstrument method), 84
lock() (pyvisa.resources.TCPIPInstrument method), 93
lock() (pyvisa.resources.TCPIPSocket method), 102
lock() (pyvisa.resources.USBRaw method), 121
lock() (pyvisa.resources.VXIBackplane method), 177
lock() (pyvisa.resources.VXIInstrument method), 166
lock() (pyvisa.resources.VXIMemory method), 171
lock_context() (pyvisa.resources.FirewireInstrument method), 148
lock_context() (pyvisa.resources.GPIBInstrument method), 131
lock_context() (pyvisa.resources.GPIBInterface method), 131
lock_context() (pyvisa.resources.MessageBasedResource method), 142
lock_context() (pyvisa.resources.PXIInstrument method), 153
lock_context() (pyvisa.resources.PXIMemory method), 160
lock_context() (pyvisa.resources.RegisterBasedResource method), 76
lock_context() (pyvisa.resources.Resource method), 64
lock_context() (pyvisa.resources.SerialInstrument method), 84
lock_context() (pyvisa.resources.TCPIPInstrument method), 93
lock_context() (pyvisa.resources.TCPIPSocket method), 102
lock_context() (pyvisa.resources.USBRaw method), 111
lock_context() (pyvisa.resources.VXIBackplane method), 121
lock_context() (pyvisa.resources.VXIInstrument method), 177
lock_context() (pyvisa.resources.VXIMemory method), 166
lock_context() (pyvisa.resources.VXIMemory method), 171
lock_excl() (pyvisa.resources.FirewireInstrument method), 148
lock_excl() (pyvisa.resources.GPIBInstrument method), 131
lock_excl() (pyvisa.resources.GPIBInterface method), 131
lock_excl() (pyvisa.resources.MessageBasedResource method), 142
lock_excl() (pyvisa.resources.PXIInstrument method), 148
lock_excl() (pyvisa.resources.PXIMemory method), 131
lock_excl() (pyvisa.resources.USBRaw method), 121
lock_excl() (pyvisa.resources.VXIBackplane method), 177
lock_excl() (pyvisa.resources.VXIInstrument method), 166
lock_excl() (pyvisa.resources.VXIMemory method), 171
lock_excl() (pyvisa.resources.VXIMemory method), 171
read_bytes() (pyvisa.resources.SerialInstrument method), 87
read_bytes() (pyvisa.resources.TCPIPInstrument method), 96
read_bytes() (pyvisa.resources.TCPIPSocket method), 105
read_bytes() (pyvisa.resources.USBIinterface method), 115
read_bytes() (pyvisa.resources.USBRaw method), 124
read_memory() (pyvisa.highlevel.VisaLibraryBase method), 54
read_memory() (pyvisa.resources.FirewireInstrument method), 149
read_memory() (pyvisa.resources.PXIInstrument method), 155
read_memory() (pyvisa.resources.PXIMemory method), 161
read_memory() (pyvisa.resources.RegisterBasedResource method), 77
read_memory() (pyvisa.resources.VXIIMemory method), 172
read_raw() (pyvisa.resources.GPIBInstrument method), 134
read_raw() (pyvisa.resources.MessageBasedResource method), 71
read_raw() (pyvisa.resources.SerialInstrument method), 87
read_raw() (pyvisa.resources.TCPIPInstrument method), 96
read_raw() (pyvisa.resources.TCPIPSocket method), 105
read_raw() (pyvisa.resources.USBRaw method), 124
read_stb() (pyvisa.highlevel.VisaLibraryBase method), 54
read_stb() (pyvisa.resources.GPIBInstrument method), 134
read_stb() (pyvisa.resources.MessageBasedResource method), 71
read_stb() (pyvisa.resources.SerialInstrument method), 87
read_stb() (pyvisa.resources.TCPIPInstrument method), 96
read_stb() (pyvisa.resources.TCPIPSocket method), 105
read_stb() (pyvisa.resources.USBIinterface method), 115
read_stb() (pyvisa.resources.USBRaw method), 125
read_termination (pyvisa.resources.TCPIPSocket attribute), 105
read_termination (pyvisa.resources.USBIinterface attribute), 115
read_termination (pyvisa.resources.USBRaw attribute), 125
read_termination_context() (pyvisa.resources.GPIBInstrument method), 135
read_termination_context() (pyvisa.resources.MessageBasedResource method), 71
read_termination_context() (pyvisa.resources.SerialInstrument method), 87
read_termination_context() (pyvisa.resources.TCPIPInstrument method), 96
read_termination_context() (pyvisa.resources.TCPIPSocket method), 105
read_termination_context() (pyvisa.resources.USBRaw method), 125
read_values() (pyvisa.resources.GPIBInstrument method), 135
read_values() (pyvisa.resources.MessageBasedResource method), 71
read_values() (pyvisa.resources.SerialInstrument method), 87
read_values() (pyvisa.resources.TCPIPInstrument method), 96
read_values() (pyvisa.resources.TCPIPSocket method), 105
read_values() (pyvisa.resources.USBRaw method), 125
register() (pyvisa.resources.FirewireInstrument class method), 149
register() (pyvisa.resources.GPIBInstrument class method), 135
register() (pyvisa.resources.GPIBInterface class method), 143
register() (pyvisa.resources.MessageBasedResource class method), 72
register() (pyvisa.resources.PXIInstrument class method), 155
register() (pyvisa.resources.PXIMemory class method), 161
register() (pyvisa.resources.RegisterBasedResource class
Index

202

(resource_name (pyvisa.resources.RegisterBasedResource attribute), 78)

resource_manufacturer_name
(resource_name (pyvisa.resources.Resource attribute), 65)

resource_manufacturer_name
(resource_name (pyvisa.resources.SerialInstrument attribute), 88)

resource_manufacturer_name
(resource_name (pyvisa.resources.TCPIPInstrument attribute), 97)

resource_manufacturer_name
(resource_name (pyvisa.resources.TCPIPSocket attribute), 105)

resource_manufacturer_name
(resource_name (pyvisa.resources.USBInstrument attribute), 115)

resource_manufacturer_name
(resource_name (pyvisa.resources.USBRaw attribute), 125)

resource_manufacturer_name
(resource_name (pyvisa.resources.VXIBackplane attribute), 178)

resource_manufacturer_name
(resource_name (pyvisa.resources.VXIInstrument attribute), 167)

resource_manufacturer_name
(resource_name (pyvisa.resources.VXIMemory attribute), 173)

ResourceInfo (class in pyvisa.highlevel), 59

ResourceManager (class in pyvisa.highlevel), 60

rio (pyvisa.constants.InterfaceType attribute), 180

rsnrp (pyvisa.constants.InterfaceType attribute), 180

S

secondary_address (pyvisa.resources.GPIBInstrument attribute), 136

send_command() (pyvisa.resources.GPIBInstrument method), 144

send_command() (pyvisa.resources.GPIBInterface method), 144

send_end, 14

send_end (pyvisa.resources.GPIBInstrument attribute), 136

send_end (pyvisa.resources.GPIBInterface attribute), 144

send_end (pyvisa.resources.SerialInstrument attribute), 88

send_end (pyvisa.resources.TCPIPInstrument attribute), 97

send_end (pyvisa.resources.USBInstrument attribute), 116

send_end (pyvisa.resources.VXIInstrument attribute), 167

send_ifc() (pyvisa.resources.GPIBInstrument method), 136

send_ifc() (pyvisa.resources.GPIBInterface method), 144

serial_number (pyvisa.resources.USBInstrument attribute), 116

SerialInstrument (class in pyvisa.resources), 80

SerialTermination (class in pyvisa.constants), 180

session (pyvisa.highlevel.ResourceManager attribute), 61

session (pyvisa.resources.FirewireInstrument attribute), 149

session (pyvisa.resources.GPIBInstrument attribute), 135

session (pyvisa.resources.GPIBInterface attribute), 144

session (pyvisa.resources.MessageBasedResource attribute), 72

session (pyvisa.resources.PXIInstrument attribute), 156

session (pyvisa.resources.PXIMemory attribute), 161

session (pyvisa.resources.RegisterBasedResource attribute), 78

session (pyvisa.resources.Resource attribute), 65

session (pyvisa.resources.SerialInstrument attribute), 88

session (pyvisa.resources.TCPIPInstrument attribute), 97

session (pyvisa.resources.TCPIPSocket attribute), 106

session (pyvisa.resources.USBInstrument attribute), 116

session (pyvisa.resources.USBRaw attribute), 125

session (pyvisa.resources.VXIBackplane attribute), 178

resource_name (pyvisa.resources.VXIInstrument attribute), 167

resource_name (pyvisa.resources.VXIMemory attribute), 173

ResourceInfo (class in pyvisa.highlevel), 59

ResourceManager (class in pyvisa.highlevel), 60

rio (pyvisa.constants.InterfaceType attribute), 180

rsnrp (pyvisa.constants.InterfaceType attribute), 180

S
session (pyvisa.resources.USBRaw attribute), 126
session (pyvisa.resources.VXIBackplane attribute), 178
session (pyvisa.resources.VXIInstrument attribute), 168
set_attribute() (pyvisa.highlevel.VisaLibraryBase method), 54
set_buffer() (pyvisa.highlevel.VisaLibraryBase method), 55
set_visa_attribute() (pyvisa.resources.FirewireInstrument method), 150
set_visa_attribute() (pyvisa.resources.GPIBInstrument method), 136
set_visa_attribute() (pyvisa.resources.GPIBInterface method), 145
set_visa_attribute() (pyvisa.resources.MessageBasedResource method), 72
set_visa_attribute() (pyvisa.resources.PXIInstrument method), 156
set_visa_attribute() (pyvisa.resources.PXIMemory method), 162
set_visa_attribute() (pyvisa.resources.RegisterBasedResource method), 78
set_visa_attribute() (pyvisa.resources.Resource method), 65
set_visa_attribute() (pyvisa.resources.SerialInstrument method), 88
set_visa_attribute() (pyvisa.resources.TCPIPInstrument method), 98
set_visa_attribute() (pyvisa.resources.TCPIPSocket method), 106
set_visa_attribute() (pyvisa.resources.USBRaw method), 126
set_visa_attribute() (pyvisa.resources.VXIBackplane method), 178
set_visa_attribute() (pyvisa.resources.VXIInstrument method), 168
set_visa_attribute() (pyvisa.resources.VXIMemory method), 174
shared_lock (pyvisa.constants.AccessModes attribute), 180
source_increment (pyvisa.resources.PXIInstrument attribute), 156
source_increment (pyvisa.resources.PXIMemory attribute), 162
source_increment (pyvisa.resources.VXIInstrument attribute), 168
source_increment (pyvisa.resources.VXIMemory attribute), 174
space (pyvisa.constants.Parity attribute), 180
spec_version (pyvisa.resources.FirewireInstrument attribute), 150
spec_version (pyvisa.resources.GPIBInterface attribute), 145
spec_version (pyvisa.resources.MessageBasedResource attribute), 73
spec_version (pyvisa.resources.PXIInstrument attribute), 156
spec_version (pyvisa.resources.PXIMemory attribute), 162
spec_version (pyvisa.resources.RegisterBasedResource attribute), 79
spec_version (pyvisa.resources.Resource attribute), 65
spec_version (pyvisa.resources.TCPIPInstrument attribute), 88
spec_version (pyvisa.resources.TCPIPSocket attribute), 98
spec_version (pyvisa.resources.USBRaw attribute), 126
spec_version (pyvisa.resources.VXIBackplane attribute), 178
spec_version (pyvisa.resources.VXIInstrument attribute), 168
spec_version (pyvisa.resources.VXIMemory attribute), 174
status_description() (pyvisa.highlevel.VisaLibraryBase method), 55
StatusCode (class in pyvisa.constants), 181
stb (pyvisa.resources.GPIBInstrument attribute), 137
stb (pyvisa.resources.MessageBasedResource attribute), 73
stb (pyvisa.resources.PXIInstrument attribute), 89
stb (pyvisa.resources.TCPIPInstrument attribute), 98
stb (pyvisa.resources.TCPIPSocket attribute), 106
stb (pyvisa.resources.USBRaw attribute), 117
stop_bits (pyvisa.resources.SerialInstrument attribute), 89
StopBits (class in pyvisa.constants), 180
success (pyvisa.constants.StatusCode attribute), 185
success_device_not_present (pyvisa.constants.StatusCode attribute), 185
success_event_already_disabled (pyvisa.constants.StatusCode attribute), 185
success_event_already_enabled (pyvisa.constants.StatusCode attribute), 185
success_max_count_read (pyvisa.constants.StatusCode attribute), 185
success_nested_exclusive (pyvisa.constants.StatusCode attribute), 185
success_nested_shared (pyvisa.constants.StatusCode attribute), 185
success_no_more_handler_calls_in_chain (pyvisa.constants.StatusCode attribute), 185

Index 203
(pyvisa.constants.StatusCode attribute), 185
success_queue_already_empty
(pyvisa.constants.StatusCode attribute), 185
success_queue_not_empty (pyvisa.constants.StatusCode attribute), 185
success_synchronous (pyvisa.constants.StatusCode attribute), 186
success_trigger_already_mapped
(pyvisa.constants.StatusCode attribute), 186
success_syncronous (pyvisa.constants.StatusCode attribute), 186
success_termination_character_read
(pyvisa.constants.StatusCode attribute), 186

T
talker (pyvisa.constants.AddressState attribute), 181
tcpip (pyvisa.constants.InterfaceType attribute), 180
TCPIPInstrument (class in pyvisa.resources), 91
TCPIPSocket (class in pyvisa.resources), 100
terminate() (pyvisa.highlevel.VisaLibraryBase method), 55
termination_break (pyvisa.constants.SerialTermination attribute), 180
termination_char (pyvisa.constants.SerialTermination attribute), 180
timeout (pyvisa.resources.FirewireInstrument attribute), 150
timeout (pyvisa.resources.GPIBInstrument attribute), 137
timeout (pyvisa.resources.GPIBInterface attribute), 145
timeout (pyvisa.resources.MessageBasedResource attribute), 73
timeout (pyvisa.resources.PXIInstrument attribute), 156
timeout (pyvisa.resources.PXIMemory attribute), 162
timeout (pyvisa.resources.RegisterBasedResource attribute), 79
timeout (pyvisa.resources.Resource attribute), 66
timeout (pyvisa.resources.SerialInstrument attribute), 89
timeout (pyvisa.resources.TCPIPInstrument attribute), 98
timeout (pyvisa.resources.TCPIPSocket attribute), 106
timeout (pyvisa.resources.USBInstrument attribute), 117
timeout (pyvisa.resources.USBRaw attribute), 126
timeout (pyvisa.resources.VXIBackplane attribute), 179
timeout (pyvisa.resources.VXIInstrument attribute), 168
timeout (pyvisa.resources.VXIMemory attribute), 174
two (pyvisa.constants.StopBits attribute), 180
uninstall_handler() (pyvisa.resources.GPIBInstrument method), 137
uninstall_handler() (pyvisa.resources.GPIBInterface method), 145
uninstall_handler() (pyvisa.resources.MessageBasedResource method), 73
uninstall_handler() (pyvisa.resources.PXIInstrument method), 157
uninstall_handler() (pyvisa.resources.PXIMemory method), 162
uninstall_handler() (pyvisa.resources.RegisterBasedResource method), 79
uninstall_handler() (pyvisa.resources.Resource method), 66
uninstall_handler() (pyvisa.resources.SerialInstrument method), 89
uninstall_handler() (pyvisa.resources.TCPIPInstrument method), 98
uninstall_handler() (pyvisa.resources.TCPIPSocket method), 107
uninstall_handler() (pyvisa.resources.USBInstrument method), 117
uninstall_handler() (pyvisa.resources.USBRaw method), 126
uninstall_handler() (pyvisa.resources.VXIBackplane method), 179
uninstall_handler() (pyvisa.resources.VXIInstrument method), 169
uninstall_handler() (pyvisa.resources.VXIMemory method), 174

U
unaddressed (pyvisa.constants.AddressState attribute), 181
unasserted (pyvisa.constants.LineState attribute), 181
uninstall_all_visa_handlers()
(pyvisa.highlevel.VisaLibraryBase method), 56
uninstall_handler() (pyvisa.highlevel.VisaLibraryBase method), 56
uninstall_handler() (pyvisa.resources.FirewireInstrument method), 150
unknown (pyvisa.constants.InterfaceType attribute), 181
unknown (pyvisa.constants.LineState attribute), 181
unlock() (pyvisa.highlevel.VisaLibraryBase method), 56
unlock() (pyvisa.resources.FirewireInstrument method), 150
unlock() (pyvisa.resources.GPIBInstrument method), 137
unlock() (pyvisa.resources.GPIBInterface method), 145
unlock() (pyvisa.resources.MessageBasedResource method), 73
unlock() (pyvisa.resources.PXIInstrument method), 157
unlock() (pyvisa.resources.PXIMemory method), 163
unlock() (pyvisa.resources.RegisterBasedResource method), 79
unlock() (pyvisa.resources.Resource method), 66
unlock() (pyvisa.resources.SerialInstrument method), 89
unlock() (pyvisa.resources.TCPIPInstrument method), 98
unlock() (pyvisa.resources.TCPIPSocket method), 107
unlock() (pyvisa.resources.USBInstrument method), 117
unlock() (pyvisa.resources.USBRaw method), 126
unlock() (pyvisa.resources.VXIBackplane method), 179
unlock() (pyvisa.resources.VXIInstrument method), 169
unlock() (pyvisa.resources.VXIMemory method), 174
wait_on_event() (pyvisa.resources.VXIMemory method), 175
warning_configuration_not_loaded (pyvisa.constants.StatusCode attribute), 186
warning_ext_function_not_implemented (pyvisa.constants.StatusCode attribute), 186
warning_nonsupported_attribute_state (pyvisa.constants.StatusCode attribute), 186
warning_nonsupported_buffer (pyvisa.constants.StatusCode attribute), 186
warning_null_object (pyvisa.constants.StatusCode attribute), 186
warning_queue_overflow (pyvisa.constants.StatusCode attribute), 186
warning_unknown_status (pyvisa.constants.StatusCode attribute), 186
write() (pyvisa.highlevel.VisaLibraryBase method), 58
write() (pyvisa.resources.GPIBInstrument method), 138
write() (pyvisa.resources.MessageBasedResource method), 74
write() (pyvisa.resources.SerialInstrument method), 90
write() (pyvisa.resources.TCPIPInstrument method), 99
write() (pyvisa.resources.TCPIPSocket method), 107
write() (pyvisa.resources.USBInstrument method), 118
write_ascii_values() (pyvisa.resources.GPIBInstrument method), 138
write_ascii_values() (pyvisa.resources.MessageBasedResource method), 74
write_ascii_values() (pyvisa.resources.SerialInstrument method), 90
write_ascii_values() (pyvisa.resources.TCPIPInstrument method), 99
write_ascii_values() (pyvisa.resources.TCPIPSocket method), 107
write_ascii_values() (pyvisa.resources.USBInstrument method), 118
write_ascii_values() (pyvisa.resources.USBRaw method), 127
write_binary_values() (pyvisa.resources.GPIBInstrument method), 138
write_binary_values() (pyvisa.resources.MessageBasedResource method), 74
write_binary_values() (pyvisa.resources.SerialInstrument method), 90
write_binary_values() (pyvisa.resources.TCPIPInstrument method), 99
write_binary_values() (pyvisa.resources.TCPIPSocket method), 108
write_binary_values() (pyvisa.resources.USBInstrument method), 118
write_binary_values() (pyvisa.resources.USBRaw method), 127
write_raw() (pyvisa.resources.GPIBInstrument method), 138
write_raw() (pyvisa.resources.MessageBasedResource method), 74
write_raw() (pyvisa.resources.SerialInstrument method), 90
write_raw() (pyvisa.resources.TCPIPInstrument method), 100
write_raw() (pyvisa.resources.TCPIPSocket method), 108
write_raw() (pyvisa.resources.USBInstrument method), 119
write_raw() (pyvisa.resources.USBRaw method), 128
write_termination (pyvisa.resources.GPIBInstrument attribute), 128
write_termination (pyvisa.resources.MessageBasedResource attribute), 139
write_termination (pyvisa.resources.SerialInstrument attribute), 91
write_termination (pyvisa.resources.TCPIPInstrument attribute), 100
write_termination (pyvisa.resources.TCPIPSocket attribute), 108
write_termination (pyvisa.resources.USBInstrument attribute), 119
write_termination (pyvisa.resources.USBRaw attribute), 128
write_values() (pyvisa.resources.GPIBInstrument method), 139
write_values() (pyvisa.resources.MessageBasedResource method), 75
write_values() (pyvisa.resources.SerialInstrument method), 91
write_values() (pyvisa.resources.TCPIPInstrument method), 100
write_values() (pyvisa.resources.TCPIPSocket method), 108
write_values() (pyvisa.resources.USBInstrument method), 119
write_values() (pyvisa.resources.USBRaw method), 128

X

xoff_char (pyvisa.resources.SerialInstrument attribute), 91
xon_char (pyvisa.resources.SerialInstrument attribute), 91