

---

**vmelib**  
*Release 1.4*

**Viacheslav V. Kaminskiy, Stepan A. Zakharov**

**Apr 28, 2022**

# CONTENTS

<b>1</b>	<b>General description</b>	<b>1</b>
<b>2</b>	<b>Requirements</b>	<b>2</b>
<b>3</b>	<b>Installation</b>	<b>3</b>
3.1	<i>CAENVMElib</i> . . . . .	3
3.2	Compiling and installation . . . . .	3
<b>4</b>	<b>Usage</b>	<b>4</b>
<b>5</b>	<b>Examples</b>	<b>11</b>
5.1	Accessing similar registers of one VME module . . . . .	11
5.2	Accessing heterogeneous registers of many VME modules of one crate . . . . .	11
5.3	Accessing heterogeneous registers of many VME modules of few crates . . . . .	12
<b>6</b>	<b>Changes</b>	<b>13</b>
	<b>Index</b>	<b>15</b>

## GENERAL DESCRIPTION

**vmelib** is a Python library which provides functionality of the official C++ library *CAENVMelib* interacting with CAEN VME bus blocks. It wraps most general and most useful functions in Python-way.

All interaction with devices are done via object of Python class *CAENVMEDevice*. It provides functions for VME devices:

- read single registers, many contiguous (with or without BLT) or non-contiguous registers;
- write data to single registers, many contiguous (with or without BLT) or non-contiguous registers;
- read-write single registers;
- set interrupt request mask and check/wait for interrupt request;
- get/set VMEbus arbiter type, requester type and requester priority level;
- get/set VMEbus timeout;
- auxiliary functions.

### Authors

- **Viacheslav Kaminskiy** - *Main work, interface, docs* - [kaminsky](<https://git.inp.nsk.su/kaminsky>)
- **Stepan Zakharov** - *Code wrapping* - [zakharov](<https://git.inp.nsk.su/zakharov>)

## REQUIREMENTS

- C++ ver. > 11
- Python > 3.5
- *Python.h* and *libpython3[.x].so* (e.g, *python3-dev* package in Debian-based OSes)
- *numpy* (*numpy.h*)
- *setuptools*

To install them in modern Debian-based OSes:

```
sudo apt install gcc python3 python3-dev  
sudo apt install python3-numpy python3-setuptools
```

or use *pip install numpy setuptools*, etc. to install Python packages in virtual environment or in user directory.

Download and unpack somewhere *CAENVMELib* Library from CAEN official site: <https://www.caen.it/download/?filter=CAENVMELib%20Library>. Unpack in Debian-based OSes:

```
tar -zvf CAENVMELib-x.y.z.tgz
```

## INSTALLATION

### 3.1 CAENVMElib

There are two installation ways:

1. **System-wide** (*/usr/lib* and */usr/include*), superuser rights are required. Follow the instructions at *CAENVMElibReadme.txt* or <https://www.caen.it/download/?filter=CAENVMElib%20Library>. Basically, run *install* script in *CAENVMElib-x.y.z/lib*.
2. **Local**. Create link or copy *libCAENVME.so* in *CAENVMElib-x.y.z/lib/x64* (or *\*x86*) directory:

```
ln -s libCAENVME.so.x.y.z libCAENVME.so
```

### 3.2 Compiling and installation

- If *CAENVMElib* was installed **system-wide** (with superuser rights), no additional actions are required.
- If *CAENVMElib* was installed **local**, there are two options:
  1. Set location of *CAENVMElib* library at install-time:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/some/location/CAENVMElib-x.y.z/lib/x64/
```

Also you have to set *LD\_LIBRARY\_PATH* at run-time, e.g, in *~/.bashrc*.

2. Otherwise you can hard-link *vmelib* to *libCAENVME* at installation time. In this case you have to re-install *vmelib* if Python/ numpy version or location of *CAENVMElib* change.

Compile the library by typing

```
sudo python3 setup.py install
```

at *vmelib* directory to install the library **system-wide**. Or

```
python3 setup.py install --user
```

to install **in user directory**. Or

```
python setup.py install
```

to install **in virtual environment** (e.g, *venv*).

*setup.py* will try to search all needed components, if fails, it will ask for paths or give recommendations.

## USAGE

**class** `vmelib.CAENVMEDevice`(*bd\_type=1, link=0, bd\_num=0, addr\_mode=13, data\_width=4, verbosity=0, base\_addr=0*)

An object representing CAEN VME crate with communication bridge. It allows to perform read/write/irq operations of VME devices (modules).

**\_\_init\_\_**(*bd\_type=1, link=0, bd\_num=0, addr\_mode=13, data\_width=4, verbosity=0, base\_addr=0*)  
Opens communication board and sets base address and default communication parameters.

If you use one VME module, it is convenient to set all the parameters in `__init__`, including base address. Otherwise, if using few links/boards, you should init the device without module-specific parameters and the set link/board with `open_board`.

### Parameters

- **bd\_type** (*int, optional*) – Model of the bridge for further data transfers. Can be modified further in `open_board`. Default: `vmelib.V2718`.
- **link** (*int, optional*) – Link number (don't care for V1718) for further data transfers, e.g, number of PCI card. Can be modified further in `open_board`. Default: 0.
- **bd\_num** (*int, optional*) – Board number in the link for further data transfers, e.g, number of slot in PCI card. Can be modified further in `open_board`. Default: 0.
- **addr\_mode** (*int, optional*) – Address modifier. This value is set as default in read/write functions and can be modified in `open_device`. Default: `vmelib.A32_S_DATA`.
- **data\_width** (*int, optional*) – Data chunk size, in bytes. This value is set as default in read/write functions and can be modified in `open_device`. Default: `vmelib.D32`.
- **verbosity** (*int, optional*) – 0 - debug messages are not printed. 1 - some debug messages are printed in stdout. 2 - all debug messages are printed in stdout. Default: 0.
- **base\_addr** (*int, optional*) – VME base address in format 0XXXXX0000. This value is set as default in read/write functions and can be modified in `open_device`. Default: 0xFFFFFFFF.

**Returns** `CAENVMEDevice` – An instance of “VME crate”.

**Raises** `VMElibError` – If error occurs.

**\_\_del\_\_**()  
`CAENVMEDevice` destructor.

**open\_board**(*bd\_type, link, bd\_num*)  
Opens communication board. It can useful if you want open another board in the same `CAENVMEDevice` instance.

### Parameters

- **bd\_type** (*int*) – model of the bridge, e.g. *vmlib.V2718*.
- **link** (*int*) – link number (don't care for V1718), e.g. number of PCI card, default: 0.
- **bd\_num** (*int*) – board number in the link, e.g. number of slot in PCI card, default: 0.

**Returns** *None*

**Raises** **VMLibError** – If error occurs.

**open\_device**(*base\_addr, addr\_mode, data\_width*)

Sets default communication parameters for VME device (module). Does not make any data transfer. It is convenient when you work with one module.

**Parameters**

- **base\_addr** (*int*) – VME base address in format 0XXXXX0000, arbitrary. This value is set as default in proceeding read/write functions.
- **addr\_mode** (*int*) – Address modifier (e.g. *vmlib.A32\_S\_DATA*). This value is set as default in read/write functions.
- **data\_width** (*int*) – Data chunk size, in bytes (e.g. *vmlib.D32*), arbitrary. This value is set as default in read/write functions.

**Returns** *None*

**read\_reg**(*reg\_addr, base\_addr=4294967295, addr\_mode=255, data\_width=255*)

Reads single register from VME device (module).

**Parameters**

- **reg\_addr** (*int*) – Address of the register in the device in format 0XXXXX.
- **base\_addr** (*int, optional*) – VME base address in format 0XXXXX0000. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **addr\_mode** (*int, optional*) – Address modifier. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **data\_width** (*int, optional*) – Data chunk size, in bytes. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).

**Returns** *int* – Value read.

**Raises** **VMLibError** – If error occurs.

**write\_reg**(*data, reg\_addr, base\_addr=4294967295, addr\_mode=255, data\_width=255*)

Writes single register to VME device (module).

**Parameters**

- **data** (*int*) – Data to write.
- **reg\_addr** (*int*) – Address of the register in the device in format 0XXXXX.
- **base\_addr** (*int, optional*) – VME base address in format 0XXXXX0000. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **addr\_mode** (*int, optional*) – Address modifier. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **data\_width** (*int, optional*) – Data chunk size, in bytes. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).

**Returns** *None*

**Raises `VMelibError`** – If error occurs.

**`read_write(data, reg_addr, base_addr=4294967295, addr_mode=255, data_width=255)`**

Reads single register to VME device (module) and immediately the writes data into it.

**Parameters**

- **`data (int)`** – Data to write.
- **`reg_addr (int)`** – Address of the register in the device in format 0XXXXX.
- **`base_addr (int, optional)`** – VME base address in format 0XXXXX0000. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **`addr_mode (int, optional)`** – Address modifier. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **`data_width (int, optional)`** – Data chunk size, in bytes. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).

**Returns** *int* – Value read.

**Raises `VMelibError`** – If error occurs.

**`read_page(size, reg_addr, base_addr=4294967295, addr_mode=255, data_width=255, blt=True)`**

Reads many contiguous registers (page) from VME device (module). It can perform both block transfer (BLT) read and single read of contiguous registers in single data transfer cycle.

**Parameters**

- **`size (int)`** – Size of array to read, i.e, number of values (not bytes!).
- **`reg_addr (int)`** – Address of the first register to read in the device in format 0XXXXX.
- **`base_addr (int, optional)`** – VME base address in format 0XXXXX0000. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **`addr_mode (int, optional)`** – Address modifier. Address modifier can be set for single register access (i.e, `vmelib.A32_U_DATA` instead of `vmelib.A32_U_BLT`), it is automatically corrected. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **`data_width (int, optional)`** – Data chunk size, in bytes. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **`blt (bool, optional)`** – If *True* performs real BLT read cycle (faster), if *False* performs many single reads in single data transfer cycle (slower), default: *True* (do BLT).

**Returns** *obj: numpy.array* – Values read.

**Raises `VMelibError`** – If error occurs.

**See also:**

*`read_multi`* to read non-contiguous registers, it is slower. *`read_page`* do the same type of data access if *blt* is *False*.

**`write_page(data, reg_addr, base_addr=4294967295, addr_mode=255, data_width=255, blt=True)`**

Write many contiguous registers (page) from VME device (module). It can perform both block transfer (BLT) write and single write of contiguous registers in single data transfer cycle.

**Parameters**

- **`data (list of int)`** – Data to write.



- **reg\_addr** (*int*) – Address of the first register to write in the device in format 0xXXXX.
- **base\_addr** (*int*, *optional*) – VME base address in format 0xXXXX0000. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **addr\_mode** (*int*, *optional*) – Address modifier. Address modifier can be set for single register access (i.e, `vmelib.A32_U_DATA` instead of `vmelib.A32_U_BLT`), it is automatically corrected. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **data\_width** (*int*, *optional*) – Data chunk size, in bytes. If not set, the default value is used (which was set earlier by `__init__` or `open_device`).
- **blt** (*bool*, *optional*) – If *True* performs real BLT write cycle (faster), if *False* performs many single writes in single data transfer cycle (slower), default: *True* (BLT).

**Returns** *None*

**Raises** **VMElibError** – If error occurs.

**See also:**

`write_multi` to write non-contiguous registers, it is slower. `write_page` do the same type of data access if `blt` is *False*.

**read\_multi**(*regs*, *addr\_modes*=[], *data\_widths*=[])

Reads many arbitrary registers from VME device (module), i.e, perform many single reads.

#### Parameters

- **regs** (*list of int*) – Addresses of the registers in format 0xYYYYXXXX (register address + base address) or 0xXXXX (register address only, default base address is used).
- **addr\_modes** (*list of int*, *optional*) – Address modifiers. If the list is empty, the default values are used (which was set earlier in `__init__` or `open_device`). Default: empty list.
- **data\_widths** (*list of int*, *optional*) – Data chunk sizes, in bytes. If the list is empty, the default values are used (which was set earlier in `__init__` or `open_device`). Default: empty list.

**Returns** *obj: numpy.array* – Values read.

**Raises**

- **VMElibError** – If error occurs.
- **IndexError** – If `addr_mode` or `data_width` have wrong length.

**See also:**

`read_page` to read contiguous registers, it is faster in BLT mode.

**write\_multi**(*data*, *regs*, *addr\_modes*=[], *data\_widths*=[])

Write many arbitrary registers from VME device (module), i.e, perform many single writes.

#### Parameters

- **data** (*list of int*) – Data to write.
- **regs** (*list of int*) – Addresses of the registers in format 0xYYYYXXXX (register address + base address) or 0xXXXX (register address only, default base address is used).

- **addr\_modes** (*list of int, optional*) – Address modifiers. If the list is empty, the default values are used (which was set earlier in `__init__` or `open_device`). Default: empty list.
- **data\_widths** (*list of int, optional*) – Data chunk sizes, in bytes. If the list is empty, the default values are used (which was set earlier in `__init__` or `open_device`). Default: empty list.

**Returns** *None*

**Raises**

- **VMelibError** – If error occurs.
- **IndexError** – If *data*, *regs*, *addr\_mode* or *data\_width* have wrong lengths.

**See also:**

[\*write\\_page\*](#) to write contiguous registers, it is faster in BLT mode.

**irq\_enable**(*mask*)

Enables the IRQ lines specified by mask.

**Parameters** **mask** (*int*) – Bit mask: interrupt levels from 1 to 7 (from IRQ1 to IRQ7).

**Returns** *None*

**Raises** **VMelibError** – If error occurs.

**irq\_disable**(*mask*)

Disables the IRQ lines specified by mask.

**Parameters** **mask** (*int*) – Bit mask: interrupt levels from 1 to 7 (from IRQ1 to IRQ7).

**Returns** *None*

**Raises** **VMelibError** – If error occurs.

**irq\_wait**(*mask, timeout*)

Waits the IRQ lines specified by mask during timeout.

**Parameters**

- **mask** (*int*) – Bit mask: interrupt levels from 1 to 7 (from IRQ1 to IRQ7).
- **timeout** (*int*) – Timeout in milliseconds.

**Returns** *None*

**Raises**

- **VMelibError("TimeoutError")** – If timeout expires.
- **VMelibError** – If other error occurs.

**irq\_check**()

Check the IRQ lines.

**Parameters** **None** –

**Returns** *list of int* – Bit mask: interrupt levels from 1 to 7 (from IRQ1 to IRQ7).

**Raises** **VMelibError** – If error occurs.

**See also:**

[\*irq\\_wait\*](#) faster and more convenient way to follow IRQ state.

**get\_timeout()**

Get VME bus timeout.

**Parameters** *None* –

**Returns** *int* – VME bus timeout: 50 us or 400 us.

**Raises** **VMElibError** – If error occurs.

**set\_timeout(timeout)**

Set VME bus timeout.

**Parameters** **timeout** (*int*) – Timeout in microseconds, 50 or 400.

**Returns** *None*

**Raises** **VMElibError** – If error occurs.

**get\_arbiter\_type()**

Get the type of VME bus arbiter implemented on the module.

**Parameters** *None* –

**Returns** *str* – Arbiter type: “priority arbiter” or “round-robin arbiter”.

**Raises** **VMElibError** – If error occurs.

**set\_arbiter\_type(arbiter\_type)**

Set the type of VME bus arbiter implemented on the module.

**Parameters** **arbiter\_type** (*int* or *str*) – Arbiter type: 0 (“priority arbiter”) or 1 (“round-robin arbiter”).

**Returns** *None*

**Raises**

- **ValueError** – If *arbiter\_type* is not integer or string in [“priority arbiter”, “round-robin arbiter”].
- **VMElibError** – If error occurs.

**get\_requester\_type()**

Get VME requester type.

**Parameters** *None* –

**Returns** *str* – VME bus requester type: “fair” (fair bus requester) or “demand” (on demand bus requester).

**Raises** **VMElibError** – If error occurs.

**set\_requester\_type(requester\_type)**

Set VME requester type.

**Parameters** **requester\_type** (*int* or *str*) – VME bus requester type: 0 (“fair”, fair bus requester) or 1 (“demand”, on demand bus requester).

**Returns** *None*

**Raises**

- **ValueError** – If *requester\_type* not in [0, 1, “fair”, “demand”].
- **VMElibError** – If error occurs.

**get\_requester\_level()**

Get VME requester priority level.

**Parameters** *None* –

**Returns** *int* – VME bus requester priority level: 0, 1, 2, 3.

**Raises** **VMElibError** – If error occurs.

**set\_requester\_level**(*requester\_level*)

Set VME requester priority level.

**Parameters** **requester\_level** (*int*) – VME bus requester priority level: 0, 1, 2, 3.

**Returns** *None*

**Raises**

- **ValueError** – If *requester\_level* not in [0, 1, 2, 3].
- **VMElibError** – If error occurs.

**get\_release\_type**()

Get VME bus release type.

**Parameters** *None* –

**Returns** *str* – VME bus release type: “RWD” (Release When Done) or “ROR” (Release On Request).

**Raises** **VMElibError** – If error occurs.

**set\_release\_type**(*release\_type*)

Set VME bus release type.

**Parameters** **release\_type** (*int or str*) – VME bus release type: 0 (or “RWD”, Release When Done) or 1 (or “ROR”, Release On Request).

**Returns** *None*

**Raises**

- **ValueError** – If *release\_type* not in [0, 1, “RWD”, “ROR”].
- **VMElibError** – If error occurs.

**get\_sw\_info**()

Get software and firmware versions.

**Parameters** *None* –

**Returns** *dict of str* – CAENVMElib version, board (bridge) firmware version.

**Raises** **VMElibError** – If error occurs.

## EXAMPLES

## 5.1 Accessing similar registers of one VME module

```

import vmelib
ba = 0x32100000
addr = 0x1800 # some register
device = vmelib.CAENVMEDevice(bd_type=vmelib.V2718, link=0, bd_num=0, addr_
↪mode=vmelib.A32_S_DATA, data_width=vmelib.D32, base_addr=ba)
print("reg =", device.read_reg(addr))      # read some data
device.write_reg(0x1234, addr)             # write some data
many_cont_regs = device.read_page(10, addr) # read 10 contiguous registers via_
↪BLT (not single reads)
addrs = list(range(addr, addr + 10*4, 4))
many_regs = device.read_multi(addrs)       # read not-contiguous data via_
↪single reads with default parameters

```

## 5.2 Accessing heterogeneous registers of many VME modules of one crate

```

import vmelib
ba1 = 0x32100000
ba2 = 0x43210000
addr1 = 0x1800
addr2 = 0x1000
dw1 = vmelib.D32
dw2 = vmelib.D16
device = vmelib.CAENVMEDevice(bd_type=vmelib.V2718, link=0, bd_num=0, addr_
↪mode=vmelib.A32_S_DATA)
# read some data
print("reg1 =", device.read_reg(addr1, base_addr=ba1, data_width=dw1))
print("reg2 =", device.read_reg(addr2, ba2, dw2))
device.write_reg(0x1234, addr1, ba1, dw1) # write some data
# set module with ba1 as default
device.open_device(ba1, vmelib.A32_S_DATA, dw1)
# no need to set full address for ba1 because it is a "default device"
regs = list(range(addr1, addr1 + 10*4, 4))
# for other base addresses full register addresses are needed

```

(continues on next page)

(continued from previous page)

```
regs.extend(list(range(addr2 + ba2, addr2 + ba2 + 10*2, 2)))  
# need to set data width because they are different for two modules  
dws = [dw1] * 4 + [dw2] * 2  
# read not-contiguous data via single reads with default parameters  
many_regs = device.read_multi(regs, addr_modes=[], data_widths=dws)
```

## 5.3 Accessing heterogeneous registers of many VME modules of few crates

(in progress)

## CHANGES

### Version 1.0

Basic functionality, *CAENVMElib* version = 2.56

### Version 1.1

*CAENVMElib* version <= 3.1.0

### Version 1.2

Support for *CAENVMElib* version 3.3.0 only. Updated setup system: now *vmelib* can be installed in virtual environments.

### Version 1.3 (>15.03.2022)

*CAENVMElib* version = 3.3.0. Updated setup system: automatic configuration. Updated autodoc.

*read\_multi* -> *read\_page*: read contiguous data, starting from some register. It can be read using a block transfer read cycle (faster, but not all devices support this) or a block of single read cycles (slower than BLT cycle but faster than discrete single reads, e.g, sequence of *read\_reg*).

Added *read\_multi*: read many arbitrary non-contiguous data passed via Python lists.

Added *write\_page*: write many contiguous data, starting from some register.

Added *write\_multi*: write many arbitrary non-contiguous data passed via Python lists.

Added *irq\_check*, *set\_timeout*, *get\_timeout*, *get\_sw\_info*, *get\_arbiter\_type*, *set\_arbiter\_type*, *read\_write*.

Added *get\_requester\_type*, *set\_requester\_type*, *get\_requester\_level*, *set\_requester\_level*.

Added *get\_release\_type*, *set\_release\_type*.

**Version 1.4 (>27.04.2022)**

*CAENVMElib* version = 3.3.0. Cumulative update, stable version.



## Symbols

`__del__()` (*vmelib.CAENVMEDevice method*), 4  
`__init__()` (*vmelib.CAENVMEDevice method*), 4

## C

`CAENVMEDevice` (*class in vmlib*), 4

## G

`get_arbiter_type()` (*vmelib.CAENVMEDevice method*), 9  
`get_release_type()` (*vmelib.CAENVMEDevice method*), 10  
`get_requester_level()` (*vmelib.CAENVMEDevice method*), 9  
`get_requester_type()` (*vmelib.CAENVMEDevice method*), 9  
`get_sw_info()` (*vmelib.CAENVMEDevice method*), 10  
`get_timeout()` (*vmelib.CAENVMEDevice method*), 9

## I

`irq_check()` (*vmelib.CAENVMEDevice method*), 8  
`irq_disable()` (*vmelib.CAENVMEDevice method*), 8  
`irq_enable()` (*vmelib.CAENVMEDevice method*), 8  
`irq_wait()` (*vmelib.CAENVMEDevice method*), 8

## O

`open_board()` (*vmelib.CAENVMEDevice method*), 4  
`open_device()` (*vmelib.CAENVMEDevice method*), 5

## R

`read_multi()` (*vmelib.CAENVMEDevice method*), 7  
`read_page()` (*vmelib.CAENVMEDevice method*), 6  
`read_reg()` (*vmelib.CAENVMEDevice method*), 5  
`read_write()` (*vmelib.CAENVMEDevice method*), 6

## S

`set_arbiter_type()` (*vmelib.CAENVMEDevice method*), 9  
`set_release_type()` (*vmelib.CAENVMEDevice method*), 10

`set_requester_level()` (*vmelib.CAENVMEDevice method*), 10  
`set_requester_type()` (*vmelib.CAENVMEDevice method*), 9  
`set_timeout()` (*vmelib.CAENVMEDevice method*), 9

## W

`write_multi()` (*vmelib.CAENVMEDevice method*), 7  
`write_page()` (*vmelib.CAENVMEDevice method*), 6  
`write_reg()` (*vmelib.CAENVMEDevice method*), 5