pymem Documentation

Release alpha

Author

Mar 12, 2022

CONTENTS

1 User’s Guide 3

2 API Reference 15

3 Additional Notes 47

Python Module Index 51

Index 53

Welcome to Pymem's documentation. Get started with *Installation* and then get an overview with the *Quickstart*. There is also a more detailed *Tutorials* section that shows how to write small software with Pymem. The rest of the docs describe each component of Pymem in detail, with a full reference in the *API* section.

Except for running tests or buliding the documentation, Pymem does not require any library it only manipulate ctypes and more precisely WinDLL.

The structure of this documentation is based on Flask.

CHAPTER

ONE

USER'S GUIDE

This part of the documentation, which is mostly prose, begins with some background information about Pymem, then focuses on step-by-step instructions for reversing with Pymem.

1.1 Foreword

Read this before you get started with Pymem. This hopefully answers some questions about the purpose and goals of the project, and then why you should and should not be using it.

1.1.1 Why Pymem ?

I decided to build pymem after some reading of the wonderfull book Gray Hat Python by Justin Seitz, which I recommend as a first reading before even starting using Pymem. The book covers the win32api and important aspects of debuggers. As I wanted to learn more on debugging, hooking and the windows API, I figured out that writing a library was the perfect project.

1.1.2 Pymem history

So back in 2010, with my little knowledge of Python I wrote the first version of this library (which has been entirely rewritten since). I figured out that most of the resources you can find covering C, C++, C# of the windows API works "as it" using python ctypes without any effort, so I decided to wrap some of them into Pymem.

In 2015, I decided to rebirth the library, and to rewrite it using python3. The library is a toolbox for process memory manipulations, it supports memory reads, writes and even assembly injection (thanks to pyfasm).

In 2020, the support for pyfasm was dropped because of its incompatibility with x64 processes. It now includes testing, and the documentation as been totally rewritten with tutorials.

1.1.3 Why and when using Pymem

Pymem has been built to reverse games such as Worlf of Warcraft, so if you plan to write a bot for this kind of game, you're in the right place. You can also use pymem to do injections, assembly, memory pattern search and a lot more.

You should head over the *Tutorials* section and see what Pymem is capable of!

Continue to *Installation*, the *Quickstart* or *Tutorials*.

1.2 Installation

Pymem has no dependencies and works on both x86 and x64 architecture.

You will need Python 3 or newer to get started, so be sure to have an up-to-date Python 3.x installation.

If you are familiar with pyenv, it is highly recommended to sandbox pymem installation within a custom virtualenv.

1.2.1 Path

In order to use all pymem fonctionalities you have to first make sure that system python directory is configured within windows system PATH.

In a PowerShell window type:

```
$env:PATH
```

This PATH should contain the directory where python is installed system wide or at least have access to pythonXX.dll If you don't find python in your PATH, then it is recommended to add it.

```
- Open the Start Search, type in "env", and choose "Edit the system environment variables
↪"
- Click the "Environment Variables..." button
- Under the "System Variables" section (the lower half), find the row with "Path" in the␣
↪first column, and click edit.
- The "Edit environment variable" UI will appear. Here, you can click "New" and type in␣
↪the new path you want to add.
- Add your python path and close the windows (something like: C:\Users\xxx\AppData\Local\
↪Programs\Python\Python38)
```

1.2.2 Virtual environments

Use a virtual environment to manage the dependencies for your project, both in development and in production.

What problem does a virtual environment solve? The more Python projects you have, the more likely it is that you need to work with different versions of Python libraries, or even Python itself. Newer versions of libraries for one project can break compatibility in another project.

Virtual environments are independent groups of Python libraries, one for each project. Packages installed for one project will not affect other projects or the operating system's packages.

Python comes bundled with the `venv` module to create virtual environments.

Create an environment

Create a project folder and a `venv` folder within:

```
$ mkdir myproject
$ cd myproject
$ python3 -m venv venv
```

On Windows:

```
$ py -3 -m venv venv
```

Activate the environment

Before you work on your project, activate the corresponding environment:

```
$ . venv/bin/activate
```

On Windows:

```
> venv\Scripts\activate
```

Your shell prompt will change to show the name of the activated environment.

1.2.3 Install Pymem

Within the activated environment, use the following command to install Pymem:

```
$ pip install pymem
```

Pymem is now installed. Check out the *Quickstart* or go to the *Documentation Overview*.

1.3 Quickstart

Eager to get started? This page gives a good introduction to Pymem. Follow *Installation* to set up a project and install Pymem first.

1.3.1 A Minimal Application

A minimal Pymem application looks something like this:

```python
from pymem import Pymem

pm = Pymem('notepad.exe')
print('Process id: %s' % pm.process_id)
address = pm.allocate(10)
print('Allocated address: %s' % address)
pm.write_int(address, 1337)
value = pm.read_int(address)
print('Allocated value: %s' % value)
pm.free(address)
```

So what did that code do?

1. First we imported the `Pymem` class. An instance of this class will be our win32api wrapper
2. Next we create an instance of this class. The first argument is the name of the windows process we want to hook into.

   Be aware that after creating an instance of Pymem with the process name as an argument, the process will be opened with debug mode flags.

3. We then allocate 10 bytes into given _notepad.exe_ process with `allocate()`.
4. For the example we then write an integer with `write_int()` and read it with `read_int()`.
5. We then free memory from the current opened process at the given address with `free()`.

Save it as `hello.py` or something similar. Make sure to not call your application `pymem.py` because this would conflict with Pymem itself.

To run the application, first start *notepad.exe* be sure to have pymem installed within your current python environment and simply execute your script.

```
$ python hello.py
  Process id: 2345
  Allocated address: 123456789
  Allocated value: 1337
```

1.4 Tutorials

1.4.1 Listing process modules

Pymem comes with somes process utilities like listing loaded modules.

Here is a snippet that will list loaded process modules

```
import pymem

pm = pymem.Pymem('python.exe')
modules = list(pm.list_modules())
for module in modules:
    print(module.name)
```

So what did that code do?

1. we hook pymem with python.exe process
2. we retrieve the list of loaded modules
3. for every module listed, we display its name

note: every *module* is an instance of `MODULEINFO()`

1.4.2 Injecting a python interpreter into any process

Pymem allow you to inject *python.dll* into a target process and then map *py_run_simple_string* with a single call to `inject_python_interpreter()`.

```
from pymem import Pymem
import os
import subprocess

notepad = subprocess.Popen(['notepad.exe'])

pm = Pymem('notepad.exe')
pm.inject_python_interpreter()
```

(continues on next page)

(continued from previous page)

```
filepath = os.path.join(os.path.abspath('.'), 'pymem_injection.txt')
filepath = filepath.replace("\\", "\\\\")
shellcode = """
f = open("{}", "w+")
f.write("pymem_injection")
f.close()
""".format(filepath)
pm.inject_python_shellcode(shellcode)
notepad.kill()
```

So what did that code do?

1. we start notepad process and get its handle
2. we hook pymem with notepad process
3. we call *inject_python_interpreter()* which will:

- dynamically finds the correct python dll and inject it
- register **py_run_simple_string**

4. then we inject some python code with *inject_python_shellcode()* which will:

- **VirtualAllocEx** some space for the code to be written
- write the actual payload into allocated space
- execute **py_run_simple_string** so the python code gets interpreted within the notepad process

5. finally we get rid of notepad process

1.5 Examples from the community

No support will be provided for any community related examples.

Here is a list of programs / scripts made by the community:

1.5.1 External glow ESP for CS:GO

No support will be provided for any community related examples.

Credits goes to Snacky.

Original source code: github

Warning

This comes, "as it" with no guarantees regarding its standing with VAC.

Use this code at your own risk and be aware that **using any sort of hack will resolve in having your steam_id banned**

Snippet

```
import pymem
import pymem.process

dwEntityList = (0x4D4B104)
dwGlowObjectManager = (0x5292F20)
m_iGlowIndex = (0xA428)
m_iTeamNum = (0xF4)


def main():
    print("Diamond has launched.")
    pm = pymem.Pymem("csgo.exe")
    client = pymem.process.module_from_name(pm.process_handle, "client.dll").lpBaseOfDll

    while True:
        glow_manager = pm.read_int(client + dwGlowObjectManager)

        for i in range(1, 32):  # Entities 1-32 are reserved for players.
            entity = pm.read_int(client + dwEntityList + i * 0x10)

            if entity:
                entity_team_id = pm.read_int(entity + m_iTeamNum)
                entity_glow = pm.read_int(entity + m_iGlowIndex)

                if entity_team_id == 2:  # Terrorist
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x4, float(1))
↪# R
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x8, float(0))
↪# G
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0xC, float(0))
↪# B
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x10, float(1))
↪# Alpha
                    pm.write_int(glow_manager + entity_glow * 0x38 + 0x24, 1)
↪# Enable glow

                elif entity_team_id == 3:  # Counter-terrorist
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x4, float(0))
↪# R
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x8, float(0))
↪# G
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0xC, float(1))
↪# B
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x10, float(1))
↪# Alpha
                    pm.write_int(glow_manager + entity_glow * 0x38 + 0x24, 1)
↪# Enable glow


if __name__ == '__main__':
    main()
```

1.5.2 AssaultCube External ESP (Pygame, Pymem)

No support will be provided for any community related examples.

Meow, feel free to contact pymem author if you want this example to be removed as we didn't reached you to have your authorisation to post this

Credits goes to Meow.

Original source code: link

Warning

This comes, "as it" with no guarantees regarding its standing with any anti-cheat related.

Use this code at your own risk and be aware that **using any sort of hack may resolve in having your account banned**

Snippet

See **Original source code** above to see the whole source code.

Video

1.5.3 CS:GO Esp

No support will be provided for any community related examples.

pSilent, feel free to contact pymem author if you want this example to be removed as we didn't reached you to have your authorisation to post this

Credits goes to pSilent

Original post: link

Original source code: github

Warning

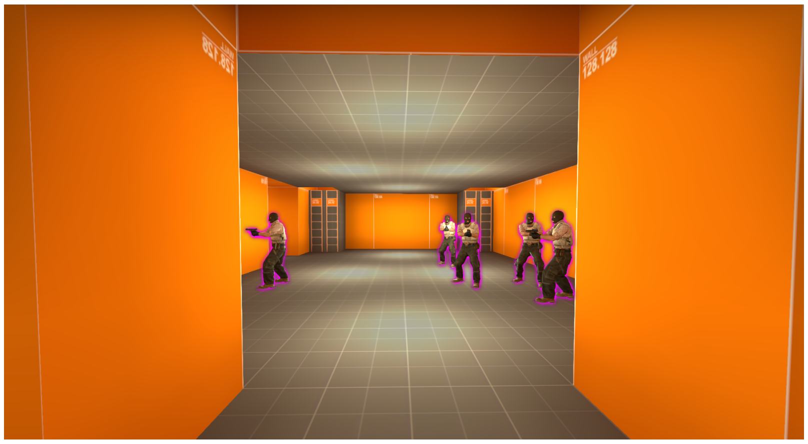
This comes, "as it" with no guarantees regarding its standing with any anti-cheat related.

Use this code at your own risk and be aware that **using any sort of hack may resolve in having your account banned**

Snippet

See **Original source code** above to see the whole source code.

Screenshot



1.5.4 No flash cheat for CS:GO

No support will be provided for any community related examples.

Credits goes to Snacky.

Original source code: github

Warning

This comes, "as it" with no guarantees regarding its standing with VAC.

Use this code at your own risk and be aware that **using any sort of hack will resolve in having your steam_id banned**

Snippet

```
import pymem
import pymem.process
import time

dwLocalPlayer = (0xD36B94)
m_flFlashMaxAlpha = (0xA40C)


def main():
    print("Emerald has launched.")
    pm = pymem.Pymem("csgo.exe")
    client = pymem.process.module_from_name(pm.process_handle, "client.dll").lpBaseOfDll

    while True:
        player = pm.read_int(client + dwLocalPlayer)
        if player:
            flash_value = player + m_flFlashMaxAlpha
            if flash_value:
                pm.write_float(flash_value, float(0))
```

(continues on next page)

(continued from previous page)

```
        time.sleep(1)


if __name__ == '__main__':
    main()
```

1.5.5 Auto bunny hopper for CS:GO

No support will be provided for any community related examples.

Credits goes to Snacky.

Original source code: github

Warning

This comes, "as it" with no guarantees regarding its standing with VAC.

Use this code at your own risk and be aware that **using any sort of hack will resolve in having your steam_id banned**

Snippet

```
import keyboard
import pymem
import pymem.process
import time
from win32gui import GetWindowText, GetForegroundWindow

dwForceJump = (0x51F4D88)
dwLocalPlayer = (0xD36B94)
m_fFlags = (0x104)


def main():
    print("Ruby has launched.")
    pm = pymem.Pymem("csgo.exe")
    client = pymem.process.module_from_name(pm.process_handle, "client.dll").lpBaseOfDll

    while True:
        if not GetWindowText(GetForegroundWindow()) == "Counter-Strike: Global Offensive
↪":
            continue

        if keyboard.is_pressed("space"):
            force_jump = client + dwForceJump
            player = pm.read_int(client + dwLocalPlayer)
            if player:
                on_ground = pm.read_int(player + m_fFlags)
                if on_ground and on_ground == 257:
                    pm.write_int(force_jump, 5)
```

(continues on next page)

(continued from previous page)

```
                    time.sleep(0.08)
                    pm.write_int(force_jump, 4)

        time.sleep(0.002)


if __name__ == '__main__':
    main()
```

1.5.6 Trigger bot for CS:GO

No support will be provided for any community related examples.

Credits goes to Snacky.

Original source code: github

Warning

This comes, "as it" with no guarantees regarding its standing with VAC.

Use this code at your own risk and be aware that **using any sort of hack will resolve in having your steam_id banned**

Snippet

```
import keyboard
import pymem
import pymem.process
import time
from win32gui import GetWindowText, GetForegroundWindow

dwEntityList = (0x4D4B104)
dwForceAttack = (0x317C6EC)
dwLocalPlayer = (0xD36B94)
m_fFlags = (0x104)
m_iCrosshairId = (0xB3D4)
m_iTeamNum = (0xF4)

trigger_key = "shift"


def main():
    print("Sapphire has launched.")
    pm = pymem.Pymem("csgo.exe")
    client = pymem.process.module_from_name(pm.process_handle, "client.dll").lpBaseOfDll

    while True:
        if not keyboard.is_pressed(trigger_key):
            time.sleep(0.1)
```

(continues on next page)

(continued from previous page)

```
        if not GetWindowText(GetForegroundWindow()) == "Counter-Strike: Global Offensive
↪":
            continue

        if keyboard.is_pressed(trigger_key):
            player = pm.read_int(client + dwLocalPlayer)
            entity_id = pm.read_int(player + m_iCrosshairId)
            entity = pm.read_int(client + dwEntityList + (entity_id - 1) * 0x10)

            entity_team = pm.read_int(entity + m_iTeamNum)
            player_team = pm.read_int(player + m_iTeamNum)

            if entity_id > 0 and entity_id <= 64 and player_team != entity_team:
                pm.write_int(client + dwForceAttack, 6)

            time.sleep(0.006)


if __name__ == '__main__':
    main()
```

1.6 Common issues

Here is a non-exhaustive list of common issues related to pymem usage:

- The token does not have the specified privilege

Pymem requires that the terminal or interpreter be ran with administrator privileges

- AttributeError: ‘NoneType’ object has no attribute or any other Python error

Make sure you are running at least Python 3.5 (earlier versions are unsupported, future versions may have breaking issues)

CHAPTER

TWO

API REFERENCE

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API

This part of the documentation covers all the methods of Pymem. For parts where Pymem depends on external dlls, we document the most important right here and provide links to the canonical documentation.

2.1.1 Pymem

`class pymem.Pymem(process_name=None)`

Initialize the Pymem class. If process_name is given, will open the process and retrieve a handle over it.

Parameters **`process_name`** (*`str`*) – The name of the process to be opened

`allocate(size)`

Allocate memory into the current opened process.

Parameters **`size`** (*`int`*) – The size of the region of memory to allocate, in bytes.

Raises

- *`ProcessError`* – If there is no process opened
- **`TypeError`** – If size is not an integer

Returns The base address of the current process.

Return type int

`property base_address`

Gets the memory address where the main module was loaded (ie address of exe file in memory)

Raises

- **`TypeError`** – If process_id is not an integer
- *`ProcessError`* – Could not find process first module address

Returns Address of main module

Return type int

`check_wow64()`

Check if a process is running under WoW64.

close_process()
Close the current opened process

Raises *ProcessError* – If there is no process opened

free(*address*)
Free memory from the current opened process given an address.

Parameters **address** (*int*) – An address of the region of memory to be freed.

Raises

- *ProcessError* – If there is no process opened
- **TypeError** – If address is not an integer

inject_python_interpreter(*initsigs=1*)
Inject python interpreter into target process and call Py_InitializeEx.

inject_python_shellcode(*shellcode*)
Inject a python shellcode into memory and execute it.

Parameters **shellcode** (*str*) – A string with python instructions.

list_modules()
List a process loaded modules.

Returns List of process loaded modules

Return type list(*MODULEINFO*)

property main_thread
Retrieve ThreadEntry32 of main thread given its creation time.

Raises *ProcessError* – If there is no process opened or could not list process thread

Returns Process main thread

Return type *Thread*

property main_thread_id
Retrieve th32ThreadID from main thread

Raises *ProcessError* – If there is no process opened or could not list process thread

Returns Main thread identifier

Return type int

open_process_from_id(*process_id*)
Open process given its name and stores the handle into *self.process_handle*.

Parameters **process_id** (*int*) – The unique process identifier

Raises

- **TypeError** – If process identifier is not an integer
- *CouldNotOpenProcess* – If process cannot be opened

open_process_from_name(*process_name*)
Open process given its name and stores the handle into process_handle

Parameters **process_name** (*str*) – The name of the process to be opened

Raises

- **TypeError** – If process name is not valid

- *ProcessNotFound* – If process name is not found
- *CouldNotOpenProcess* – If process cannot be opened

property process_base

Lookup process base Module.

Raises

- **TypeError** – process_id is not an integer
- *ProcessError* – Could not find process first module address

Returns Base module information

Return type *MODULEINFO*

read_bool(*address*)

Reads 1 byte from an area of memory in a specified process.

Parameters **address** (*int*) – An address of the region of memory to be read.

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type bool

read_bytes(*address*, *length*)

Reads bytes from an area of memory in a specified process.

Parameters

- **address** (*int*) – An address of the region of memory to be read.
- **length** (*int*) – Number of bytes to be read

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed

Returns the raw value read

Return type bytes

read_char(*address*)

Reads 1 byte from an area of memory in a specified process.

Parameters **address** (*int*) – An address of the region of memory to be read.

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type str

read_double(*address*)
Reads 8 byte from an area of memory in a specified process.

Parameters **address** (*int*) – An address of the region of memory to be read.

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type int

read_float(*address*)
Reads 4 byte from an area of memory in a specified process.

Parameters **address** (*int*) – An address of the region of memory to be read.

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type float

read_int(*address*)
Reads 4 byte from an area of memory in a specified process.

Parameters **address** (*int*) – An address of the region of memory to be read.

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type int

read_long(*address*)
Reads 4 byte from an area of memory in a specified process.

Parameters **address** (*int*) – An address of the region of memory to be read.

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type int

read_longlong(*address*)
Reads 8 byte from an area of memory in a specified process.

Parameters **address** (*int*) – An address of the region of memory to be read.

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type int

read_short(*address*)

Reads 2 byte from an area of memory in a specified process.

Parameters **address** (*int*) – An address of the region of memory to be read.

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type int

read_string(*address*, *byte=50*)

Reads n *byte* from an area of memory in a specified process.

Parameters

- **address** (*int*) – An address of the region of memory to be read.
- **byte** (*int*) – Amount of bytes to be read

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type str

read_uchar(*address*)

Reads 1 byte from an area of memory in a specified process.

Parameters **address** (*int*) – An address of the region of memory to be read.

Raises

- *ProcessError* – If there is no opened process
- *MemoryReadError* – If ReadProcessMemory failed
- **TypeError** – If address is not a valid integer

Returns returns the value read

Return type str

`read_uint`(*address*)
Reads 4 byte from an area of memory in a specified process.

Parameters **`address`** (*`int`*) – An address of the region of memory to be read.

Raises

- *`ProcessError`* – If there is no opened process
- *`MemoryReadError`* – If ReadProcessMemory failed
- **`TypeError`** – If address is not a valid integer

Returns returns the value read

Return type int

`read_ulong`(*address*)
Reads 4 byte from an area of memory in a specified process.

Parameters **`address`** (*`int`*) – An address of the region of memory to be read.

Raises

- *`ProcessError`* – If there is no opened process
- *`MemoryReadError`* – If ReadProcessMemory failed
- **`TypeError`** – If address is not a valid integer

Returns returns the value read

Return type int

`read_ulonglong`(*address*)
Reads 8 byte from an area of memory in a specified process.

Parameters **`address`** (*`int`*) – An address of the region of memory to be read.

Raises

- *`ProcessError`* – If there is no opened process
- *`MemoryReadError`* – If ReadProcessMemory failed
- **`TypeError`** – If address is not a valid integer

Returns returns the value read

Return type int

`read_ushort`(*address*)
Reads 2 byte from an area of memory in a specified process.

Parameters **`address`** (*`int`*) – An address of the region of memory to be read.

Raises

- *`ProcessError`* – If there is no opened process
- *`MemoryReadError`* – If ReadProcessMemory failed
- **`TypeError`** – If address is not a valid integer

Returns returns the value read

Return type int

`start_thread`(*address*, *params=None*)
Create a new thread within the current debugged process.

Parameters

- **`address`** (*`int`*) – An address from where the thread starts
- **`params`** (*`int`*) – An optional address with thread parameters

Returns The new thread identifier

Return type int

`write_bool`(*address*, *value*)
Write *value* to the given *address* into the current opened process.

Parameters

- **`address`** (*`int`*) – An address of the region of memory to be written.
- **`value`** (*`bool`*) – the value to be written

Raises

- *`ProcessError`* – If there is no opened process
- *`MemoryWriteError`* – If WriteProcessMemory failed
- **`TypeError`** – If address is not a valid integer

`write_bytes`(*address*, *value*, *length*)
Write *value* to the given *address* into the current opened process.

Parameters

- **`address`** (*`int`*) – An address of the region of memory to be written.
- **`value`** (*`bytes`*) – the value to be written
- **`length`** (*`int`*) – Number of bytes to be written

Raises

- *`ProcessError`* – If there is no opened process
- *`MemoryWriteError`* – If WriteProcessMemory failed
- **`TypeError`** – If address is not a valid integer

`write_char`(*address*, *value*)
Write *value* to the given *address* into the current opened process.

Parameters

- **`address`** (*`int`*) – An address of the region of memory to be written.
- **`value`** (*`str`*) – the value to be written

Raises

- *`ProcessError`* – If there is no opened process
- *`MemoryWriteError`* – If WriteProcessMemory failed
- **`TypeError`** – If address is not a valid integer

`write_double`(*address*, *value*)
Write *value* to the given *address* into the current opened process.

Parameters

- **`address`** (*`int`*) – An address of the region of memory to be written.

- **value** (*float*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_float(*address*, *value*)
Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*float*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_int(*address*, *value*)
Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_long(*address*, *value*)
Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_longlong(*address*, *value*)
Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_short(*address*, *value*)

Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_string(*address*, *value*)

Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*str*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_uchar(*address*, *value*)

Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_uint(*address*, *value*)

Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_ulong(*address*, *value*)

Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_ulonglong(*address*, *value*)

Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

write_ushort(*address*, *value*)

Write *value* to the given *address* into the current opened process.

Parameters

- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – the value to be written

Raises

- *ProcessError* – If there is no opened process
- *MemoryWriteError* – If WriteProcessMemory failed
- **TypeError** – If address is not a valid integer

2.1.2 Structures

class `pymem.ressources.structure.`**`CLIENT_ID`**

class `pymem.ressources.structure.`**`EnumProcessModuleEX`**

The following are the EnumProcessModuleEX flags

https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682633(v=vs.85).aspx

`LIST_MODULES_32BIT = 1`

List the 32-bit modules

`LIST_MODULES_64BIT = 2`

List the 64-bit modules.

`LIST_MODULES_ALL = 3`

List all modules.

`LIST_MODULES_DEFAULT = 0`

Use the default behavior.

class `pymem.ressources.structure.`**`FILETIME`**

class `pymem.ressources.structure.`**`FLOATING_SAVE_AREA`**

Undocumented ctypes.Structure used for ThreadContext.

`pymem.ressources.structure.`**`LPMODULEENTRY32`**

alias of `pymem.ressources.structure.LP_ModuleEntry32`

`pymem.ressources.structure.`**`LPSECURITY_ATTRIBUTES`**

alias of `pymem.ressources.structure.LP_SECURITY_ATTRIBUTES`

class `pymem.ressources.structure.`**`LUID`**

class `pymem.ressources.structure.`**`LUID_AND_ATTRIBUTES`**

`pymem.ressources.structure.`**`MEMORY_BASIC_INFORMATION`**

alias of *`pymem.ressources.structure.MEMORY_BASIC_INFORMATION64`*

class `pymem.ressources.structure.`**`MEMORY_BASIC_INFORMATION32`**

Contains information about a range of pages in the virtual address space of a process. The VirtualQuery and VirtualQueryEx functions use this structure.

https://msdn.microsoft.com/en-us/library/windows/desktop/aa366775(v=vs.85).aspx

class `pymem.ressources.structure.`**`MEMORY_BASIC_INFORMATION64`**

class `pymem.ressources.structure.`**`MEMORY_PROTECTION`**(*value*)

The following are the memory-protection options; you must specify one of the following values when allocating or protecting a page in memory https://msdn.microsoft.com/en-us/library/windows/desktop/aa366786(v=vs.85).aspx

`PAGE_EXECUTE_READWRITE = 64`

Enables execute, read-only, or read/write access to the committed region of pages.

class `pymem.ressources.structure.`**`MEMORY_STATE`**(*value*)

The type of memory allocation

`MEM_DECOMMIT = 16384`

Decommits the specified region of committed pages. After the operation, the pages are in the reserved state. https://msdn.microsoft.com/en-us/library/windows/desktop/aa366894(v=vs.85).aspx

`MEM_FREE = 65536`

XXX

`MEM_RELEASE = 32768`
Releases the specified region of pages. After the operation, the pages are in the free state. https://msdn.microsoft.com/en-us/library/windows/desktop/aa366894(v=vs.85).aspx

`MEM_RESERVE = 8192`
XXX

`class pymem.ressources.structure.MEMORY_TYPES(`*value*`)`
An enumeration.

`MEM_IMAGE = 16777216`
XXX

`MEM_MAPPED = 262144`
XXX

`MEM_PRIVATE = 131072`
XXX

`class pymem.ressources.structure.MODULEINFO(`*handle*`)`
Contains the module load address, size, and entry point.

`lpBaseOfDll`

`SizeOfImage`

`EntryPoint`

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684229(v=vs.85).aspx

`class pymem.ressources.structure.ModuleEntry32(`**args*, ***kwds*`)`
Describes an entry from a list of the modules belonging to the specified process.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684225%28v=vs.85%29.aspx

`class pymem.ressources.structure.NT_TIB`

`class pymem.ressources.structure.PROCESS(`*value*`)`
Process manipulation flags

`DELETE = 65536`
Required to delete the object.

`PROCESS_ALL_ACCESS = 2035711`
All possible access rights for a process object.

`PROCESS_CREATE_PROCESS = 128`
Required to create a process.

`PROCESS_CREATE_THREAD = 2`
Required to create a thread.

`PROCESS_DUP_HANDLE = 64`
PROCESS_DUP_HANDLE

`PROCESS_SET_INFORMATION = 512`
Required to set certain information about a process, such as its priority class (see SetPriorityClass).

`PROCESS_SET_QUOTA = 256`
Required to set memory limits using SetProcessWorkingSetSize.

`PROCESS_SUSPEND_RESUME = 2048`
Required to suspend or resume a process.

`PROCESS_TERMINATE = 1`
: Required to terminate a process using TerminateProcess.

`PROCESS_VM_OPERATION = 8`
: Required to perform an operation on the address space of a process (see VirtualProtectEx and WriteProcessMemory).

`PROCESS_VM_READ = 16`
: Required to read memory in a process using ReadProcessMemory.

`PROCESS_VM_WRITE = 32`
: Required to write to memory in a process using WriteProcessMemory.

`READ_CONTROL = 131072`
: Required to read information in the security descriptor for the object, not including the information in the SACL. To read or write the SACL, you must request the ACCESS_SYSTEM_SECURITY access right. For more information see SACL Access Right.

`STANDARD_RIGHTS_REQUIRED = 983040`
: Combines DELETE, READ_CONTROL, WRITE_DAC, and WRITE_OWNER access.

`SYNCHRONIZE = 1048576`
: Required to wait for the process to terminate using the wait functions.

`WRITE_DAC = 262144`
: Required to modify the DACL in the security descriptor for the object.

`WRITE_OWNER = 524288`
: Required to change the owner in the security descriptor for the object.

`pymem.ressources.structure.PTOKEN_PRIVILEGES`
: alias of `pymem.ressources.structure.LP_TOKEN_PRIVILEGES`

`class pymem.ressources.structure.ProcessEntry32(*args, **kwds)`
: Describes an entry from a list of the processes residing in the system address space when a snapshot was taken.

  https://msdn.microsoft.com/en-us/library/windows/desktop/ms684839(v=vs.85).aspx

`class pymem.ressources.structure.SECURITY_ATTRIBUTES`
: The SECURITY_ATTRIBUTES structure contains the security descriptor for an object and specifies whether the handle retrieved by specifying this structure is inheritable.

  https://msdn.microsoft.com/en-us/library/windows/desktop/aa379560(v=vs.85).aspx

`class pymem.ressources.structure.SE_TOKEN_PRIVILEGE(value)`
: An access token contains the security information for a logon session. The system creates an access token when a user logs on, and every process executed on behalf of the user has a copy of the token.

`class pymem.ressources.structure.SMALL_TEB`

`class pymem.ressources.structure.SYSTEM_INFO`
: Contains information about the current computer system. This includes the architecture and type of the processor, the number of processors in the system, the page size, and other such information.

  https://msdn.microsoft.com/en-us/library/windows/desktop/ms724958(v=vs.85).aspx

`class pymem.ressources.structure.THREAD_BASIC_INFORMATION`

`class pymem.ressources.structure.TIB_UNION`

`class pymem.ressources.structure.TOKEN(value)`
: An enumeration.

`class pymem.ressources.structure.TOKEN_PRIVILEGES`

class pymem.ressources.structure.**ThreadContext**

Represents a thread context

class pymem.ressources.structure.**ThreadEntry32**(**args*, ***kwds*)

Describes an entry from a list of the threads executing in the system when a snapshot was taken.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms686735(v=vs.85).aspx

2.1.3 Pattern

pymem.pattern.**pattern_scan_module**(*handle*, *module*, *pattern*, *, *return_multiple=False*)

Given a handle over an opened process and a module will scan memory after a byte pattern and return its corresponding memory address.

Parameters

- **handle** (*int*) – Handle to an open object
- **module** (MODULEINFO) – An instance of a given module
- **pattern** (*bytes*) – A regex byte pattern to search for
- **return_multiple** (*bool*) – If multiple results should be returned instead of stopping on the first

Returns Memory address of given pattern, or None if one was not found or a list of found addresses in return_multiple is True

Return type int, list, optional

Examples

```
>>> pm = pymem.Pymem("Notepad.exe")
# Here the "." means that the byte can be any byte; a "wildcard"
# also note that this pattern may be outdated
>>> bytes_pattern = b".\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00" \
...                 b"\x00\x00\x00\x00\x00\x00..\x00\x00..\x00\x00\x64\x04"
>>> module_notepad = pymem.process.module_from_name(pm.process_handle, "Notepad.exe
↪")
>>> character_count_address = pymem.pattern.pattern_scan_module(pm.process_handle,␣
↪module_notepad, bytes_pattern)
```

pymem.pattern.**scan_pattern_page**(*handle*, *address*, *pattern*, *, *return_multiple=False*)

Search a byte pattern given a memory location. Will query memory location information and search over until it reaches the length of the memory page. If nothing is found the function returns the next page location.

Parameters

- **handle** (*int*) – Handle to an open object
- **address** (*int*) – An address to search from
- **pattern** (*bytes*) – A regex byte pattern to search for
- **return_multiple** (*bool*) – If multiple results should be returned instead of stopping on the first

Returns

next_region, found address

found address may be None if one was not found, or we didn't have permission to scan the region

if return_multiple is True found address will instead be a list of found addresses or an empty list if no results

Return type tuple

Examples

```
>>> pm = pymem.Pymem("Notepad.exe")
>>> address_reference = 0x7ABC00001
# Here the "." means that the byte can be any byte; a "wildcard"
# also note that this pattern may be outdated
>>> bytes_pattern = b".\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00" \
...                 b"\x00\x00\x00\x00\x00\x00..\x00\x00..\x00\x00\x64\x04"
>>> character_count_address = pymem.pattern.scan_pattern_page(pm.process_handle,␣
↪address_reference, bytes_pattern)
```

2.1.4 Process

pymem.process.**base_module**(*handle*)

Returns process base module

Parameters **handle** (*int*) – A valid handle to an open object

Returns The base module of the process

Return type *MODULEINFO*

pymem.process.**close_handle**(*handle*)

Closes an open object handle. https://msdn.microsoft.com/en-us/library/windows/desktop/ms724211%28v=vs.85%29.aspx

Parameters **handle** (*int*) – A valid handle to an open object

Returns If the closure succeeded

Return type bool

pymem.process.**enum_process_module**(*handle*)

List and retrieves the base names of the specified loaded module within a process https://msdn.microsoft.com/en-us/library/windows/desktop/ms682633(v=vs.85).aspx https://msdn.microsoft.com/en-us/library/windows/desktop/ms683196(v=vs.85).aspx

Parameters **handle** (*int*) – Handle of the process to enum the modules of

Returns The process's modules

Return type list[*MODULEINFO*]

pymem.process.**enum_process_thread**(*process_id*)

List all threads of given processes_id

Parameters **process_id** (*int*) – Identifier of the process to enum the threads of

Returns The process's threads

Return type list[*ThreadEntry32*]

pymem.process.**get_luid**(*name*)
Get the LUID for the SeCreateSymbolicLinkPrivilege

pymem.process.**get_process_token**()
Get the current process token

pymem.process.**get_python_dll**(*version*)
Given a python dll version will find its path using the current process as a placeholder

Parameters **version** (*str*) – A string representation of python version as a dll (python38.dll)

Returns The full path of dll

Return type str

pymem.process.**inject_dll**(*handle*, *filepath*)
Inject a dll into opened process.

Parameters

- **handle** (*int*) – Handle to an open object
- **filepath** (*bytes*) – Dll to be injected filepath

Returns The address of injected dll

Return type DWORD

pymem.process.**is_64_bit**(*handle*)
Determines whether the specified process is running under WOW64 (emulation).

Parameters **handle** (*int*) – Handle of the process to check wow64 status of

Returns If the process is running under wow64

Return type bool

pymem.process.**list_processes**()
List all processes https://msdn.microsoft.com/en-us/library/windows/desktop/ms682489%28v=vs.85%29.aspx https://msdn.microsoft.com/en-us/library/windows/desktop/ms684834%28v=vs.85%29.aspx

Returns A list of open process entries

Return type list[*ProcessEntry32*]

pymem.process.**module_from_name**(*process_handle*, *module_name*)
Retrieve a module loaded by given process.

Parameters

- **process_handle** (*int*) – Handle to the process to get the module from
- **module_name** (*str*) – Name of the module to get

Returns The retrieved module

Return type *MODULEINFO*

Examples

```
>>> d3d9 = module_from_name(process_handle, 'd3d9')
```

`pymem.process.open`(*process_id*, *debug=True*, *process_access=None*)

Open a process given its process_id. By default, the process is opened with full access and in debug mode.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684320%28v=vs.85%29.aspx https://msdn.microsoft.com/en-us/library/windows/desktop/aa379588%28v=vs.85%29.aspx

Parameters

- **process_id** (*int*) – The identifier of the process to be opened
- **debug** (*bool*) – If the process should be opened in debug mode
- **process_access** (`pymem.ressources.structure.PROCESS`) – Desired access level, defaulting to all access

Returns A handle to the opened process

Return type int

`pymem.process.open_main_thread`(*process_id*)

List given process threads and return a handle to first created one.

Parameters **process_id** (*int*) – The identifier of the process

Returns A handle to the main thread

Return type int

`pymem.process.open_thread`(*thread_id*, *thread_access=None*)

Opens an existing thread object. https://msdn.microsoft.com/en-us/library/windows/desktop/ms684335%28v=vs.85%29.aspx

Parameters

- **thread_id** (*int*) – The identifier of the thread to be opened
- **thread_access** (*int*) – Desired access level, defaulting to all access

Returns A handle to the opened thread

Return type int

`pymem.process.process_from_id`(*process_id*)

Open a process given its name.

Parameters **process_id** (*int*) – The identifier of the process to be opened

Returns The process entry of the opened process

Return type *ProcessEntry32*

`pymem.process.process_from_name`(*name*)

Open a process given its name.

Parameters **name** (*str*) – The name of the process to be opened

Returns The process entry of the opened process

Return type *ProcessEntry32*

`pymem.process.set_debug_privilege`(*lpszPrivilege*, *bEnablePrivilege*)

Leverage current process privileges.

Parameters

- **lpszPrivilege** (*str*) – Privilege name
- **bEnablePrivilege** (*bool*) – Enable privilege

Returns If privileges have been leveraged

Return type bool

2.1.5 Ptypes

class pymem.ptypes.**RemotePointer**(*handle*, *v*, *endianess='little-endian'*)

Pointer capable of reading the value mapped into another process memory.

Parameters

- **handle** (*int*) – Handle to the process
- **v** (*int, RemotePointer, any ctypes type*) – The address value
- **endianess** (*str*) – The endianess of the remote pointer, defaulting to little-endian

Raises ***PymemAlignmentError*** – If endianess is not a valid alignment

Notes

The bool of RemotePointer checks if the internal value is 0

property **cvalue**

Reads targeted process memory and returns the value pointed by the given address.

Returns The value pointed at by this remote pointer as a ctypes type instance

Return type a ctypes type

property **value**

Reads targeted process memory and returns the value pointed by the given address.

Returns The value pointed at by this remote pointer

Return type int

2.1.6 Thread

class pymem.thread.**Thread**(*process_handle*, *th_entry_32*)

Provides basic thread information such as TEB.

Parameters

- **process_handle** (*int*) – A handle to an opened process
- **th_entry_32** (ThreadEntry32) – Target thread's entry object

2.1.7 Memory

pymem.memory.**allocate_memory**(*handle*, *size*, *allocation_type=None*, *protection_type=None*)

Reserves or commits a region of memory within the virtual address space of a specified process. The function initializes the memory it allocates to zero, unless MEM_RESET is used.

https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **size** (*int*) – The size of the region of memory to allocate, in bytes.
- **allocation_type** (MEMORY_STATE) – The type of memory allocation.
- **protection_type** (MEMORY_PROTECTION) – The memory protection for the region of pages to be allocated.

Returns The address of the allocated region of pages.

Return type int

pymem.memory.**free_memory**(*handle*, *address*, *free_type=None*)

Releases, decommits, or releases and decommits a region of memory within the virtual address space of a specified process.

https://msdn.microsoft.com/en-us/library/windows/desktop/aa366894%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be freed.
- **free_type** (MEMORY_PROTECTION) – The type of free operation.

Returns A boolean indicating if the call was a success.

Return type int

pymem.memory.**read_bool**(*handle*, *address*)

Reads 1 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('?')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – If ReadProcessMemory failed

Returns The raw value read as a string

Return type bool

`pymem.memory.`**`read_bytes`**(*handle*, *address*, *byte*)

Reads data from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **`handle`** (*`int`*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **`address`** (*`int`*) – An address of the region of memory to be read.
- **`byte`** (*`int`*) – Number of bytes to be read

Raises

- **`TypeError`** – If address is not a valid integer
- ***`WinAPIError`*** – If ReadProcessMemory failed

Returns The raw value read as bytes

Return type bytes

`pymem.memory.`**`read_char`**(*handle*, *address*)

Reads 1 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<b')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **`handle`** (*`int`*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **`address`** (*`int`*) – An address of the region of memory to be read.

Raises

- **`TypeError`** – If address is not a valid integer
- ***`WinAPIError`*** – If ReadProcessMemory failed

Returns The raw value read as a string

Return type str

`pymem.memory.`**`read_double`**(*handle*, *address*)

Reads 8 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<d')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – If ReadProcessMemory failed

Returns The raw value read as a float

Return type float

`pymem.memory.`**`read_float`**`(`*handle*, *address*`)`

Reads 4 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<f')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – If ReadProcessMemory failed

Returns The raw value read as a float

Return type float

`pymem.memory.`**`read_int`**`(`*handle*, *address*`)`

Reads 4 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<i')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – If ReadProcessMemory failed

Returns The raw value read as an int

Return type int

`pymem.memory.read_long(handle, address)`

Reads 4 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<l')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – If ReadProcessMemory failed

Returns The raw value read as an int

Return type int

`pymem.memory.read_longlong(handle, address)`

Reads 8 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<q')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – If ReadProcessMemory failed

Returns The raw value read as an int

Return type int

`pymem.memory.read_short(handle, address)`

Reads 2 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<h')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- `TypeError` – If address is not a valid integer
- *`WinAPIError`* – If ReadProcessMemory failed

Returns The raw value read as an int

Return type int

`pymem.memory.`**`read_string`**(*handle*, *address*, *byte=50*)

Reads n *byte* from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **`handle`** (*`int`*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **`address`** (*`int`*) – An address of the region of memory to be read.
- **`byte`** (*`int, default=50`*) – max number of bytes to check for null terminator, defaults to 50

Raises

- `TypeError` – If address is not a valid integer
- *`WinAPIError`* – If ReadProcessMemory failed

Returns The raw value read as a string

Return type str

`pymem.memory.`**`read_uchar`**(*handle*, *address*)

Reads 1 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<B')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **`handle`** (*`int`*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **`address`** (*`int`*) – An address of the region of memory to be read.

Raises

- `TypeError` – If address is not a valid integer
- *`WinAPIError`* – If ReadProcessMemory failed

Returns The raw value read as an int

Return type int

`pymem.memory.`**`read_uint`**(*handle*, *address*, *is_64=False*)

Reads 4 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<I')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.
- **is_64** (*bool*) – Should we unpack as big-endian

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – If ReadProcessMemory failed

Returns The raw value read as an int

Return type int

`pymem.memory.`**`read_ulong`**(*handle*, *address*)

Reads 4 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<L')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – If ReadProcessMemory failed

Returns The raw value read as an int

Return type int

`pymem.memory.`**`read_ulonglong`**(*handle*, *address*)

Reads 8 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<Q')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- **TypeError** – If address is not a valid integer

- `WinAPIError` – If ReadProcessMemory failed

Returns The raw value read as an int

Return type int

`pymem.memory.`**`read_ushort`**`(`*handle*, *address*`)`

Reads 2 byte from an area of memory in a specified process. The entire area to be read must be accessible or the operation fails.

Unpack the value using struct.unpack('<H')

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be read.

Raises

- **TypeError** – If address is not a valid integer
- `WinAPIError` – If ReadProcessMemory failed

Returns The raw value read as an int

Return type int

`pymem.memory.`**`virtual_query`**`(`*handle*, *address*`)`

Retrieves information about a range of pages within the virtual address space of a specified process.

https://msdn.microsoft.com/en-us/library/windows/desktop/aa366775(v=vs.85).aspx https://msdn.microsoft.com/en-us/library/windows/desktop/aa366907(v=vs.85).aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of to be read.

Returns A memory basic information object

Return type MEMORY_BASIC_INFORMATION

`pymem.memory.`**`write_bool`**`(`*handle*, *address*, *value*`)`

Writes 1 byte to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*bool*) – A boolean representing the value to be written

Raises

- **TypeError** – If address is not a valid integer
- *WinAPIError* – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_bytes`**(*handle*, *address*, *data*, *length*)

Writes data to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

Casts address using ctypes.c_char_p.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **data** (*void*) – A buffer that contains data to be written
- **length** (*int*) – Number of bytes to be written.

Raises

- **TypeError** – If address is not a valid integer
- *WinAPIError* – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_char`**(*handle*, *address*, *value*)

Writes 1 byte to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*str*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- *WinAPIError* – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_double`**(*handle*, *address*, *value*)

Writes 8 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*float*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_float`**(*handle*, *address*, *value*)

Writes 4 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*float*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_int`**(*handle*, *address*, *value*)

Writes 4 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer

- *WinAPIError* – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_long`**`(`*handle*, *address*, *value*`)`

Writes 4 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- *WinAPIError* – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_longlong`**`(`*handle*, *address*, *value*`)`

Writes 8 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- *WinAPIError* – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_short`**`(`*handle*, *address*, *value*`)`

Writes 2 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_string`**(*handle*, *address*, *bytecode*)

Writes n *bytes* of len(*bytecode*) to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **bytecode** (*str, bytes*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.`**`write_uchar`**(*handle*, *address*, *value*)

Writes 1 byte to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*str*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.write_uint`(*handle*, *address*, *value*)

Writes 4 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.write_ulong`(*handle*, *address*, *value*)

Writes 4 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

`pymem.memory.write_ulonglong`(*handle*, *address*, *value*)

Writes 8 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.

- **value** (*int*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

pymem.memory.**write_ushort**(*handle*, *address*, *value*)

Writes 2 bytes to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674%28v=vs.85%29.aspx

Parameters

- **handle** (*int*) – The handle to a process. The function allocates memory within the virtual address space of this process. The handle must have the PROCESS_VM_OPERATION access right.
- **address** (*int*) – An address of the region of memory to be written.
- **value** (*int*) – A buffer that contains data to be written

Raises

- **TypeError** – If address is not a valid integer
- ***WinAPIError*** – if WriteProcessMemory failed

Returns A boolean indicating a successful write.

Return type bool

2.1.8 Exceptions

exception pymem.exception.**CouldNotOpenProcess**(*process_id*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pymem.exception.**MemoryReadError**(*address*, *length*, *error_code=None*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pymem.exception.**MemoryWriteError**(*address*, *value*, *error_code=None*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pymem.exception.**ProcessError**(*message*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pymem.exception.**ProcessNotFound**(*process_name*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pymem.exception.**PymemAlignmentError**(*message*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pymem.exception.**PymemError**(*message*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pymem.exception.**PymemMemoryError**(*message*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pymem.exception.**PymemTypeError**(*message*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pymem.exception.**WinAPIError**(*error_code*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

CHAPTER

THREE

ADDITIONAL NOTES

Design notes, legal information and changelog are here for the interested.

3.1 License

3.2 How to contribute to Pymem

Thank you for considering contributing to Pymem!

3.2.1 Support questions

Please, don't use the issue tracker for this. The issue tracker is a tool to address bugs and feature requests in Pymem itself. Use one of the following resources for questions about using Pymem or issues with your own code:

- The `#general` channel on our Discord chat: https://discord.gg/xaWNac8
- Ask on Stack Overflow. Search with Google first using: `site:stackoverflow.com python pymem {search term, exception message, etc.}`

3.2.2 Reporting issues

Include the following information in your post:

- Describe what you expected to happen.
- If possible, include a minimal reproducible example to help us identify the issue. This also helps check that the issue is not with your own code.
- Describe what actually happened. Include the full traceback if there was an exception.
- List your Python, Pymem versions. If possible, check if this issue is already fixed in the latest releases or the latest code in the repository.

3.2.3 Submitting patches

If there is not an open issue for what you want to submit, prefer opening one for discussion before working on a PR. You can work on any issue that doesn't have an open PR linked to it or a maintainer assigned to it. These show up in the sidebar. No need to ask if you can work on an issue that interests you.

Include the following in your patch:

- Include tests if your patch adds or changes code. Make sure the test fails without your patch.
- Update any relevant docs pages and docstrings.

First time setup

- Download and install the latest version of git.
- Configure git with your username and email.

```
$ git config --global user.name 'your name'
$ git config --global user.email 'your email'
```

- Make sure you have a GitHub account.
- Fork Pymem to your GitHub account by clicking the Fork button.
- Clone the main repository locally.

```
$ git clone https://github.com/srounet/pymem
$ cd pymem
```

- Add your fork as a remote to push your work to. Replace `{username}` with your username. This names the remote "fork", the default Pymem remote is "origin".

```
git remote add fork https://github.com/{username}/pymem
```

- Create a virtualenv.

```
$ python3 -m venv env
$ . env/bin/activate
```

  On Windows, activating is different.

```
> env\Scripts\activate
```

- Install Pymem in editable mode with development dependencies.

  ```
  $ pip install -e .
  ```

Start coding

- Create a branch to identify the issue you would like to work on. If you're submitting a bug or documentation fix, branch off of the latest ".x" branch.

  ```
  $ git fetch origin
  $ git checkout -b your-branch-name origin/1.1.x
  ```

  If you're submitting a feature addition or change, branch off of the "master" branch.

  ```
  $ git fetch origin
  $ git checkout -b your-branch-name origin/master
  ```

- Using your favorite editor, make your changes, committing as you go.
- Include tests that cover any code changes you make. Make sure the test fails without your patch. Run the tests as described below.
- Push your commits to your fork on GitHub and create a pull request. Link to the issue being addressed with `fixes #123` in the pull request.

  ```
  $ git push --set-upstream fork your-branch-name
  ```

Running the tests

Run the basic test suite with pytest.

```
$ python -m pytest
```

This runs the tests for the current environment, which is usually sufficient. CI will run the full suite when you submit your pull request.

Running test coverage

Generating a report of lines that do not have test coverage can indicate where to start contributing. Run `pytest` using `coverage` and generate a report.

```
$ pip install -r requirements-test.txt
$ python -m pytest --cov=pymem
```

Building the docs

Build the docs in the `docs` directory using Sphinx.

```
$ cd docs/source
$ make clean
$ make html
```

Open `_build/html/index.html` in your browser to view the docs.

Read more about Sphinx.

PYTHON MODULE INDEX

p

`pymem`, 15
`pymem.exception`, 45
`pymem.memory`, 33
`pymem.pattern`, 28
`pymem.process`, 29
`pymem.ptypes`, 32
`pymem.ressources.structure`, 25
`pymem.thread`, 32

INDEX

A

allocate() (*pymem.Pymem method*), 15
allocate_memory() (*in module pymem.memory*), 33

B

base_address (*pymem.Pymem property*), 15
base_module() (*in module pymem.process*), 29

C

check_wow64() (*pymem.Pymem method*), 15
CLIENT_ID (*class in pymem.ressources.structure*), 25
close_handle() (*in module pymem.process*), 29
close_process() (*pymem.Pymem method*), 15
CouldNotOpenProcess, 45
cvalue (*pymem.ptypes.RemotePointer property*), 32

D

DELETE (*pymem.ressources.structure.PROCESS attribute*), 26

E

EntryPoint (*pymem.ressources.structure.MODULEINFO attribute*), 26
enum_process_module() (*in module pymem.process*), 29
enum_process_thread() (*in module pymem.process*), 29
EnumProcessModuleEX (*class in pymem.ressources.structure*), 25

F

FILETIME (*class in pymem.ressources.structure*), 25
FLOATING_SAVE_AREA (*class in pymem.ressources.structure*), 25
free() (*pymem.Pymem method*), 16
free_memory() (*in module pymem.memory*), 33

G

get_luid() (*in module pymem.process*), 29
get_process_token() (*in module pymem.process*), 30
get_python_dll() (*in module pymem.process*), 30

I

inject_dll() (*in module pymem.process*), 30
inject_python_interpreter() (*pymem.Pymem method*), 16
inject_python_shellcode() (*pymem.Pymem method*), 16
is_64_bit() (*in module pymem.process*), 30

L

list_modules() (*pymem.Pymem method*), 16
LIST_MODULES_32BIT (*pymem.ressources.structure.EnumProcessModuleE attribute*), 25
LIST_MODULES_64BIT (*pymem.ressources.structure.EnumProcessModuleE attribute*), 25
LIST_MODULES_ALL (*pymem.ressources.structure.EnumProcessModuleEX attribute*), 25
LIST_MODULES_DEFAULT (*pymem.ressources.structure.EnumProcessModuleEX attribute*), 25
list_processes() (*in module pymem.process*), 30
lpBaseOfDll (*pymem.ressources.structure.MODULEINFO attribute*), 26
LPMODULEENTRY32 (*in module pymem.ressources.structure*), 25
LPSECURITY_ATTRIBUTES (*in module pymem.ressources.structure*), 25
LUID (*class in pymem.ressources.structure*), 25
LUID_AND_ATTRIBUTES (*class in pymem.ressources.structure*), 25

M

main_thread (*pymem.Pymem property*), 16
main_thread_id (*pymem.Pymem property*), 16
MEM_DECOMMIT (*pymem.ressources.structure.MEMORY_STATE attribute*), 25
MEM_FREE (*pymem.ressources.structure.MEMORY_STATE attribute*), 25
MEM_IMAGE (*pymem.ressources.structure.MEMORY_TYPES attribute*), 26
MEM_MAPPED (*pymem.ressources.structure.MEMORY_TYPES attribute*), 26

MEM_PRIVATE (*pymem.ressources.structure.MEMORY_TYPES attribute*), 26
MEM_RELEASE (*pymem.ressources.structure.MEMORY_STATE attribute*), 25
MEM_RESERVE (*pymem.ressources.structure.MEMORY_STATE attribute*), 26
MEMORY_BASIC_INFORMATION (*in module pymem.ressources.structure*), 25
MEMORY_BASIC_INFORMATION32 (*class in pymem.ressources.structure*), 25
MEMORY_BASIC_INFORMATION64 (*class in pymem.ressources.structure*), 25
MEMORY_PROTECTION (*class in pymem.ressources.structure*), 25
MEMORY_STATE (*class in pymem.ressources.structure*), 25
MEMORY_TYPES (*class in pymem.ressources.structure*), 26
MemoryReadError, 45
MemoryWriteError, 45
module
    pymem, 15
    pymem.exception, 45
    pymem.memory, 33
    pymem.pattern, 28
    pymem.process, 29
    pymem.ptypes, 32
    pymem.ressources.structure, 25
    pymem.thread, 32
module_from_name() (*in module pymem.process*), 30
ModuleEntry32 (*class in pymem.ressources.structure*), 26
MODULEINFO (*class in pymem.ressources.structure*), 26

N

NT_TIB (*class in pymem.ressources.structure*), 26

O

open() (*in module pymem.process*), 31
open_main_thread() (*in module pymem.process*), 31
open_process_from_id() (*pymem.Pymem method*), 16
open_process_from_name() (*pymem.Pymem method*), 16
open_thread() (*in module pymem.process*), 31

P

PAGE_EXECUTE_READWRITE (*pymem.ressources.structure.MEMORY_PROTECTION attribute*), 25
pattern_scan_module() (*in module pymem.pattern*), 28
PROCESS (*class in pymem.ressources.structure*), 26
PROCESS_ALL_ACCESS (*pymem.ressources.structure.PROCESS attribute*), 26
process_base (*pymem.Pymem property*), 17
PROCESS_CREATE_PROCESS (*pymem.ressources.structure.PROCESS attribute*), 26
PROCESS_CREATE_THREAD (*pymem.ressources.structure.PROCESS attribute*), 26
PROCESS_DUP_HANDLE (*pymem.ressources.structure.PROCESS attribute*), 26
process_from_id() (*in module pymem.process*), 31
process_from_name() (*in module pymem.process*), 31
PROCESS_SET_INFORMATION (*pymem.ressources.structure.PROCESS attribute*), 26
PROCESS_SET_QUOTA (*pymem.ressources.structure.PROCESS attribute*), 26
PROCESS_SUSPEND_RESUME (*pymem.ressources.structure.PROCESS attribute*), 26
PROCESS_TERMINATE (*pymem.ressources.structure.PROCESS attribute*), 26
PROCESS_VM_OPERATION (*pymem.ressources.structure.PROCESS attribute*), 27
PROCESS_VM_READ (*pymem.ressources.structure.PROCESS attribute*), 27
PROCESS_VM_WRITE (*pymem.ressources.structure.PROCESS attribute*), 27
ProcessEntry32 (*class in pymem.ressources.structure*), 27
ProcessError, 45
ProcessNotFound, 45
PTOKEN_PRIVILEGES (*in module pymem.ressources.structure*), 27
pymem
    module, 15
Pymem (*class in pymem*), 15
pymem.exception
    module, 45
pymem.memory
    module, 33
pymem.pattern
    module, 28
pymem.process
    module, 29
pymem.ptypes
    module, 32
pymem.ressources.structure
    module, 25
pymem.thread
    module, 32
PymemAlignmentError, 45
PymemError, 45
PymemMemoryError, 46
PymemTypeError, 46

R

read_bool() (*in module pymem.memory*), 33
read_bool() (*pymem.Pymem method*), 17
read_bytes() (*in module pymem.memory*), 34
read_bytes() (*pymem.Pymem method*), 17
read_char() (*in module pymem.memory*), 34
read_char() (*pymem.Pymem method*), 17
READ_CONTROL (*pymem.ressources.structure.PROCESS attribute*), 27
read_double() (*in module pymem.memory*), 34
read_double() (*pymem.Pymem method*), 17
read_float() (*in module pymem.memory*), 35
read_float() (*pymem.Pymem method*), 18
read_int() (*in module pymem.memory*), 35
read_int() (*pymem.Pymem method*), 18
read_long() (*in module pymem.memory*), 35
read_long() (*pymem.Pymem method*), 18
read_longlong() (*in module pymem.memory*), 36
read_longlong() (*pymem.Pymem method*), 18
read_short() (*in module pymem.memory*), 36
read_short() (*pymem.Pymem method*), 19
read_string() (*in module pymem.memory*), 37
read_string() (*pymem.Pymem method*), 19
read_uchar() (*in module pymem.memory*), 37
read_uchar() (*pymem.Pymem method*), 19
read_uint() (*in module pymem.memory*), 37
read_uint() (*pymem.Pymem method*), 19
read_ulong() (*in module pymem.memory*), 38
read_ulong() (*pymem.Pymem method*), 20
read_ulonglong() (*in module pymem.memory*), 38
read_ulonglong() (*pymem.Pymem method*), 20
read_ushort() (*in module pymem.memory*), 39
read_ushort() (*pymem.Pymem method*), 20
RemotePointer (*class in pymem.ptypes*), 32

S

scan_pattern_page() (*in module pymem.pattern*), 28
SE_TOKEN_PRIVILEGE (*class in pymem.ressources.structure*), 27
SECURITY_ATTRIBUTES (*class in pymem.ressources.structure*), 27
set_debug_privilege() (*in module pymem.process*), 31
SizeOfImage (*pymem.ressources.structure.MODULEINFO attribute*), 26
SMALL_TEB (*class in pymem.ressources.structure*), 27
STANDARD_RIGHTS_REQUIRED (*pymem.ressources.structure.PROCESS attribute*), 27
start_thread() (*pymem.Pymem method*), 20
SYNCHRONIZE (*pymem.ressources.structure.PROCESS attribute*), 27
SYSTEM_INFO (*class in pymem.ressources.structure*), 27

T

Thread (*class in pymem.thread*), 32
THREAD_BASIC_INFORMATION (*class in pymem.ressources.structure*), 27
ThreadContext (*class in pymem.ressources.structure*), 27
ThreadEntry32 (*class in pymem.ressources.structure*), 28
TIB_UNION (*class in pymem.ressources.structure*), 27
TOKEN (*class in pymem.ressources.structure*), 27
TOKEN_PRIVILEGES (*class in pymem.ressources.structure*), 27

V

value (*pymem.ptypes.RemotePointer property*), 32
virtual_query() (*in module pymem.memory*), 39

W

WinAPIError, 46
with_traceback() (*pymem.exception.CouldNotOpenProcess method*), 45
with_traceback() (*pymem.exception.MemoryReadError method*), 45
with_traceback() (*pymem.exception.MemoryWriteError method*), 45
with_traceback() (*pymem.exception.ProcessError method*), 45
with_traceback() (*pymem.exception.ProcessNotFound method*), 45
with_traceback() (*pymem.exception.PymemAlignmentError method*), 45
with_traceback() (*pymem.exception.PymemError method*), 46
with_traceback() (*pymem.exception.PymemMemoryError method*), 46
with_traceback() (*pymem.exception.PymemTypeError method*), 46
with_traceback() (*pymem.exception.WinAPIError method*), 46
write_bool() (*in module pymem.memory*), 39
write_bool() (*pymem.Pymem method*), 21
write_bytes() (*in module pymem.memory*), 40
write_bytes() (*pymem.Pymem method*), 21
write_char() (*in module pymem.memory*), 40
write_char() (*pymem.Pymem method*), 21
WRITE_DAC (*pymem.ressources.structure.PROCESS attribute*), 27
write_double() (*in module pymem.memory*), 40
write_double() (*pymem.Pymem method*), 21
write_float() (*in module pymem.memory*), 41
write_float() (*pymem.Pymem method*), 22
write_int() (*in module pymem.memory*), 41
write_int() (*pymem.Pymem method*), 22

write_long() (*in module pymem.memory*), 42
write_long() (*pymem.Pymem method*), 22
write_longlong() (*in module pymem.memory*), 42
write_longlong() (*pymem.Pymem method*), 22
WRITE_OWNER (*pymem.ressources.structure.PROCESS attribute*), 27
write_short() (*in module pymem.memory*), 42
write_short() (*pymem.Pymem method*), 23
write_string() (*in module pymem.memory*), 43
write_string() (*pymem.Pymem method*), 23
write_uchar() (*in module pymem.memory*), 43
write_uchar() (*pymem.Pymem method*), 23
write_uint() (*in module pymem.memory*), 44
write_uint() (*pymem.Pymem method*), 23
write_ulong() (*in module pymem.memory*), 44
write_ulong() (*pymem.Pymem method*), 24
write_ulonglong() (*in module pymem.memory*), 44
write_ulonglong() (*pymem.Pymem method*), 24
write_ushort() (*in module pymem.memory*), 45
write_ushort() (*pymem.Pymem method*), 24