
Pymem Documentation

Release 1.0

Fabien Reboia

Jun 02, 2018

Contents

1	User's Guide	3
1.1	Foreword	3
1.2	Installation	4
2	Api Reference	5
2.1	API	5
2.2	Ressources	25
3	Additional Notes	37
3.1	License	37

Welcome to Pymem's documentation. This documentation is divided into different parts. You should start by reading *Installation*, and then head over to the quickstart. Still, you can have a look to the tutorial section that shows different usage of Pymem.

Pymem depends on [pyfasm](#) and in its current version does not support x64.

This part of the documentation, begins with some informations about Pymem, then focuses on step-by-step instructions.

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, write and even assembly injection (thanks to [pyfasm](#)).

1.1.3 Why and when using Pymem

Pymem has been built to reverse games such as *World 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 and see what Pymem is capable of!

Continue to [Installation](#) , the quickstart or tutorial/index.

1.2 Installation

Pymem depends on some external libraries, like [pyfasm](#). Pyfasm is a wrapper around [Flat Assembler](#) and in its current state only works with x86.

Pyfasm is available on [pypi](#) and is part of Pymem requirements.txt. The most straightforward method to start working with Pymem is to use a virtualenv.

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

1.2.1 Virtualenv

Virtualenv is probably what you want to use during development, and if you have shell access to your production machines, you'll probably want to use it there, too.

Virtualenv enables multiple side-by-side installations of Python, one for each project. It doesn't actually install separate copies of Python, but it does provide a clever way to keep different project environments isolated.

We will not cover the installation of neither pip or virtualenv here, so install them first.

Once you have virtualenv installed, just fire up a shell and create your own environment:

```
$ mkdir myproject
$ cd myproject
$ virtualenv pymem
New python executable in pymem/bin/python
Installing setuptools, pip.....done.
```

Now, whenever you want to work on a project, you only have to activate the corresponding environment:

```
$ pymem\scripts\activate.bat
```

And if you want to go back to the real world, use the following command:

```
$ deactivate
```

After doing this, the prompt of your shell should be as familiar as before.

Now, let's move on. Enter the following command to get Pymem activated in your virtualenv:

```
$ pip install pymem
```

A few seconds later and you are good to go.

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 interfaces of Pymem. For parts where Pymem depends on external libraries, we document the most important right here.

2.1.1 Pymem

class Pymem

__init__ (*self*, *process_name=None*)

Initialize the Pymem class.

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

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

set_debug_privilege (*self*, *process_name*)

Leverage current process privileges.

Parameters

- **hToken** (*HANDLE*) – Current process handle
- **lpzPrivilege** (*str*) – privilege name
- **bEnablePrivilege** (*bool*) – Enable privilege

Returns True if privileges have been leveraged.

Return type bool

open_process_from_name (*self*, *process_name*)

Open process given it's name and stores the handle into *self.process_handle*.

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

Raises

- **TypeError** – if *process_name* is not valid
- **pymem.exception.ProcessNotFound** – if process is not found
- **pymem.exception.CouldNotOpenProcess** – if process cannot be opened

open_process_from_id (*self*, *process_id*)

Open process given it's name and stores the handle into *self.process_handle*.

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

Raises

- **TypeError** – if *process_id* is not an integer
- **pymem.exception.CouldNotOpenProcess** – if process cannot be opened

process_base_address

Lookup process base address.

Returns The base address of the current process.

Return type `ctypes.wintypes.HANDLE`

Raises

- **TypeError** – if *process_id* is not an integer
- **pymem.exception.ProcessError** – if could not find process first module address

open_main_thread (*self*)

Open process main thread name and stores the handle into *self.thread_handle* the *thread_id* is also stored into *self.main_thread_id*.

Raises

- **pymem.exception.ProcessError** – if there is no process opened
- **pymem.exception.ProcessError** – if could not list process thread

close_process (*self*)

Close the current opened process

Raises **pymem.exception.ProcessError** – if there is no process opened

allocate (*self*, *size*)

Allocate memory into the current opened process.

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

Returns The base address of the current process.

Return type `ctypes.wintypes.HANDLE`

Raises

- **pymem.exception.ProcessError** – if there is no process opened
- **TypeError** – if *size* is not an integer

free (*self*, *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

- `pymem.exception.ProcessError` – if there is no process opened
- `TypeError` – if address is not an integer

assemble (*self*, *address=None*, *mnemonics=None*)

Assemble mnemonics to bytes using *pyfasm*.

If *address* is given then the origin *org* will be set to the address.

Parameters

- **address** (*int*) – An address of the region of memory to be freed.
- **mnemonics** (*str*) – fasm syntax mnemonics

Returns The assembled mnemonics

Return type bytes

close_main_thread (*self*)

Close the opened main thread

Raises `pymem.exception.ProcessError` – if main thread is not opened

read_bytes (*self*, *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

Returns returns the raw value read

Return type bytes

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

read_char (*self*, *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.

Returns returns the value read

Return type string

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_uchar (*self*, *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.

Returns returns the value read

Return type string

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_int (*self*, *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.

Returns returns the value read

Return type `int`

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_uint (*self*, *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.

Returns returns the value read

Return type `int`

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_short (*self*, *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.

Returns returns the value read

Return type `int`

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_ushort (*self*, *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.

Returns returns the value read

Return type `int`

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_float (*self*, *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.

Returns returns the value read

Return type float

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_long (*self*, *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.

Returns returns the value read

Return type int

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_ulong (*self*, *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.

Returns returns the value read

Return type int

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_longlong (*self*, *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.

Returns returns the value read

Return type int

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_ulonglong (*self*, *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.

Returns returns the value read

Return type int

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_double (*self*, *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.

Returns returns the value read

Return type int

Raises `pymem.exception.ProcessError` – if there id no opened process

Raise `TypeError` if address is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

read_string (*self, 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*) – number of bytes to read

Returns returns the value read

Return type str

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if byte is not a valid integer

Raise `pymem.exception.MemoryReadError` if `ReadProcessMemory` failed

write_int (*self, 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 read.

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

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid integer

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_uint (*self, 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 read.

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

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid integer

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_short (*self, 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 read.

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

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid integer

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_ushort (*self*, *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 read.
- **value** (*int*) – the value to be written

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid integer

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_float (*self*, *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 read.
- **value** (*float*) – the value to be written

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid float

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_long (*self*, *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 read.
- **value** (*float*) – the value to be written

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid int

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_ulong (*self*, *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 read.
- **value** (*float*) – the value to be written

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid int

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_longlong (*self*, *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 read.
- **value** (*float*) – the value to be written

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid int

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_ulonglong (*self, 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 read.
- **value** (*float*) – the value to be written

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid int

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_double (*self, 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 read.
- **value** (*float*) – the value to be written

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a valid int

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_string (*self, 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 read.
- **value** (*bytes*) – the value to be written

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not bytes

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

write_char (*self, 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 read.
- **value** (*float*) – the value to be written

Raises `pymem.exception.ProcessError` – if there id no opened process

Raises `TypeError` if value is not a string

Raise `pymem.exception.MemoryWriteError` if `WriteProcessMemory` failed

2.1.2 Exception's

exception WinAPIError

Global handler for win32api errors

exception PymemError

Custom Pymem exception class.

Except on this class to catch all Pymem specific Exception's

exception ProcessError (*PymemError*)

Raised when something required by a process handle went wrong

exception ProcessNotFound (*ProcessError*)

Raised when process not found

exception CouldNotOpenProcess (*ProcessError*)

Raised when process could not be opened

exception PymemMemoryError (*PymemError*)

Raised when a memory error occurred

exception MemoryReadError (*PymemMemoryError*)

Raised when a memory read error occurred

exception MemoryWriteError (*PymemMemoryError*)

Raised when a memory write error occurred

2.1.3 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** (*ctypes.wintypes.HANDLE*) – 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** (*pymem.ressources.structure.MemoryAllocation*) – The type of memory allocation.
- **protection_type** (*pymem.ressources.structure.MemoryProtection*) – The memory protection for the region of pages to be allocated.

Returns return the base address of the allocated region of pages.

Return type ctypes.wintypes.HANDLE

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** (*ctypes.wintypes.HANDLE*) – A handle to a process. The function frees memory within the virtual address space of the 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** (*pymem.ressources.structure.MemoryProtection*) – The type of free operation.

Returns If the function succeeds, the return value is a nonzero value.

Return type `ctypes.wintypes.BOOL`

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.
- **byte** (*int*) – number of bytes to be read

Returns If the function succeeds, returns the raw value read

Return type `bytes`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `string` of length 1

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `int`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `int`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `int`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `int`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

read_uint (*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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `int`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `float`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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('<I')`

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `int`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `int`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `int`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `int`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

`bytes = read_bytes(handle, address, struct.calcsize('Q'))` `bytes = struct.unpack('<Q', bytes)[0]` return bytes

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `float`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

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** (*ctypes.wintypes.HANDLE*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **address** (*int*) – An address of the region of memory to be freed.

Returns If the function succeeds, returns the value read

Return type `str`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `ReadProcessMemory` failed

write_bytes (*handle, address, src, 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** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have `PROCESS_VM_WRITE` and `PROCESS_VM_OPERATION` access to the process.
- **address** (*int*) – An address in the specified process to which data is written.
- **src** (*int*) – A buffer that contains data to be written in the address space of the specified process.
- **length** (*int*) – The number of bytes to be written to the specified process.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms value using: `ctypes.c_char(value)`.

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have `PROCESS_VM_WRITE` and `PROCESS_VM_OPERATION` access to the process.
- **address** (*int*) – An address in the specified process to which data is written.
- **value** (*int*) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms value using: `ctypes.c_short(value)`.

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have `PROCESS_VM_WRITE` and `PROCESS_VM_OPERATION` access to the process.
- **address** (*int*) – An address in the specified process to which data is written.
- **value** (*int*) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms value using: `ctypes.c_ushort(value)`.

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have PROCESS_VM_WRITE and PROCESS_VM_OPERATION access to the process.
- **address** (*int*) – An address in the specified process to which data is written.
- **value** (*int*) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms value using: `ctypes.c_int(value)`.

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have PROCESS_VM_WRITE and PROCESS_VM_OPERATION access to the process.
- **address** (*int*) – An address in the specified process to which data is written.
- **value** (*int*) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms value using: `ctypes.c_uint(value)`.

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have PROCESS_VM_WRITE and PROCESS_VM_OPERATION access to the process.

- **address** (*int*) – An address in the specified process to which data is written.
- **value** (*int*) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms value using: `ctypes.c_float(value)`.

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have PROCESS_VM_WRITE and PROCESS_VM_OPERATION access to the process.
- **address** (*int*) – An address in the specified process to which data is written.
- **value** (*float*) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms value using: `ctypes.c_long(value)`.

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have PROCESS_VM_WRITE and PROCESS_VM_OPERATION access to the process.
- **address** (*int*) – An address in the specified process to which data is written.
- **value** (*int*) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms value using: `ctypes.c_ulong(value)`.

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

Parameters

- **handle** (`ctypes.wintypes.HANDLE`) – A handle to the process memory to be modified. The handle must have `PROCESS_VM_WRITE` and `PROCESS_VM_OPERATION` access to the process.
- **address** (`int`) – An address in the specified process to which data is written.
- **value** (`int`) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type `bool`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `WriteProcessMemory` failed

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.

Transforms value using: `ctypes.c_longlong(value)`.

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

Parameters

- **handle** (`ctypes.wintypes.HANDLE`) – A handle to the process memory to be modified. The handle must have `PROCESS_VM_WRITE` and `PROCESS_VM_OPERATION` access to the process.
- **address** (`int`) – An address in the specified process to which data is written.
- **value** (`int`) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type `bool`

Raise `TypeError` if address is not a valid integer

Raise `WinAPIError` if `WriteProcessMemory` failed

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.

Transforms value using: `ctypes.c_ulonglong(value)`.

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

Parameters

- **handle** (`ctypes.wintypes.HANDLE`) – A handle to the process memory to be modified. The handle must have `PROCESS_VM_WRITE` and `PROCESS_VM_OPERATION` access to the process.
- **address** (`int`) – An address in the specified process to which data is written.
- **value** (`int`) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms value using: `ctypes.c_double(value)`.

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have PROCESS_VM_WRITE and PROCESS_VM_OPERATION access to the process.
- **address** (*int*) – An address in the specified process to which data is written.
- **value** (*int*) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

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.

Transforms bytecode using: `ctypes.c_char_p(bytecode)`.

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

Parameters

- **handle** (*ctypes.wintypes.HANDLE*) – A handle to the process memory to be modified. The handle must have PROCESS_VM_WRITE and PROCESS_VM_OPERATION access to the process.
- **address** (*int*) – An address in the specified process to which data is written.
- **bytecode** (*str*) – The data to be written.

Returns If the function succeeds, the return value is nonzero.

Return type bool

Raise TypeError if address is not a valid integer

Raise WinAPIError if WriteProcessMemory failed

2.1.4 Process

base_address (*process_id*)

Returns process base address, looking at its modules.

Parameters **process_id** (*ctypes.wintypes.HANDLE*) – The identifier of the process.

Returns The base address of the current process.

Return type `ctypes.wintypes.HANDLE`

open (*process_id*, *debug=None*, *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** (`ctypes.wintypes.HANDLE`) – The identifier of the process to be opened
- **debug** (`bool`) – open process in debug mode
- **process_access** (`pymem.ressources.structure`) – desired access level

Returns A handle of the given `process_id`

Return type `ctypes.wintypes.HANDLE`

open_main_thread (*process_id*)

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

Parameters **process_id** (`ctypes.wintypes.HANDLE`) – The identifier of the process

Returns A handle to the first thread of the given `process_id`

Return type `ctypes.wintypes.HANDLE`

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** (`ctypes.wintypes.HANDLE`) – The identifier of the thread to be opened.

Returns A handle to the first thread of the given `process_id`

Return type `ctypes.wintypes.HANDLE`

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** (`ctypes.wintypes.HANDLE`) – A valid handle to an open object.

Returns If the function succeeds, the return value is nonzero.

Return type `bool`

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 process entry 32.

Return type `list(pymem.ressources.structure.ProcessEntry32)`

process_from_name (*name*)

Open a process given its name.

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

Returns The ProcessEntry32 structure of the given process.

Return type ctypes.wintypes.HANDLE

process_from_id (*process_id*)

Open a process given its name.

Parameters **process_id** (*ctypes.wintypes.HANDLE*) – The identifier of the process

Returns The ProcessEntry32 structure of the given process.

Return type ctypes.wintypes.HANDLE

list_process_thread (*process_id*)

List all threads of given processes_id

Parameters **process_id** (*ctypes.wintypes.HANDLE*) – The identifier of the process

Returns a list of thread entry 32.

Return type list(pymem.ressources.structure.ThreadEntry32)

module_from_name (*process_id, module_name*)

Retrieve a module loaded by given *process_id*.

```
d3d9 = module_from_name(1234, 'd3d9')
```

Parameters

- **process_id** (*ctypes.wintypes.HANDLE*) – The identifier of the process
- **module_name** (*str*) – The module name

Returns ModuleEntry32

list_process_modules (*process_id*)

List all modules of a given processes by its *process_id*

Parameters **process_id** (*ctypes.wintypes.HANDLE*) – The identifier of the process

Returns a list of module entry 32.

Return type list(pymem.ressources.structure.ModuleEntry32)

2.2 Ressources

Placeholder for windows structures, and ctypes definitions around dll functions.

2.2.1 Kernel32

OpenProcess ()

Opens an existing local process object.

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

Parameters

- **dwDesiredAccess** (*DWORD*) – The access to the process object. This access right is checked against the security descriptor for the process. This parameter can be one or more of the process access rights.

- **bInheritHandle** (*BOOL*) – If this value is TRUE, processes created by this process will inherit the handle. Otherwise, the processes do not inherit this handle.
- **dwProcessId** (*DWORD*) – The identifier of the local process to be opened.

Return type `ctypes.c_ulong`

TerminateProcess ()

Terminates the specified process and all of its threads.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms686714%28v=vs.85%29.aspx>

Parameters

- **hProcess** (*HANDLE*) – A handle to the process to be terminated.
- **uExitCode** (*UINT*) – The exit code to be used by the process and threads terminated as a result of this call.

Return type `ctypes.c_ulong`

CloseHandle ()

Closes an open object handle.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms724211%28v=vs.85%29.aspx>

Parameters **hObject** (*HANDLE*) – A valid handle to an open object.

Return type `ctypes.c_long`

GetLastError ()

Retrieves the calling thread's last-error code value. The last-error code is maintained on a per-thread basis. Multiple threads do not overwrite each other's last-error code.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms679360%28v=vs.85%29.aspx>

Return type `ctypes.c_ulong`

GetCurrentProcess ()

Retrieves a pseudo handle for the current process.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms683179%28v=vs.85%29.aspx>

Return type `ctypes.c_ulong`

ReadProcessMemory ()

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

- **hProcess** – A handle to the process with memory that is being read. The handle must have PROCESS_VM_READ access to the process.
- **lpBaseAddress** – A pointer to the base address in the specified process from which to read.
- **lpBuffer** – A pointer to a buffer that receives the contents from the address space of the specified process.
- **nSize** – The number of bytes to be read from the specified process.
- **lpNumberOfBytesRead** – A pointer to a variable that receives the number of bytes transferred into the specified buffer.

Return type ctypes.c_long

WriteProcessMemory ()

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.

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

Parameters

- **dwDesiredAccess** (*DWORD*) – A handle to the process with memory that is being read. The handle must have `PROCESS_VM_READ` access to the process.
- **bInheritHandle** (*BOOL*) – A pointer to the base address in the specified process from which to read.
- **dwProcessId** (*DWORD*) – A pointer to a buffer that receives the contents from the address space of the specified process.

Return type ctypes.c_long

DebugActiveProcess ()

Enables a debugger to attach to an active process and debug it.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms679295%28v=vs.85%29.aspx>

Parameters **dwProcessId** (*DWORD*) – The identifier for the process to be debugged. The debugger is granted debugging access to the process as if it created the process with the `DEBUG_ONLY_THIS_PROCESS` flag. For more information, see the Remarks section of this topic.

Return type ctypes.c_long

VirtualAllocEx ()

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

- **hProcess** (*HANDLE*) – The handle to a process. The function allocates memory within the virtual address space of this process.
- **lpAddress** (*LPVOID*) – The pointer that specifies a desired starting address for the region of pages that you want to allocate.
- **dwSize** (*SIZE_T*) – The size of the region of memory to allocate, in bytes.
- **flAllocationType** (*DWORD*) – The type of memory allocation.
- **flProtect** (*DWORD*) – The identifier for the process to be debugged. The debugger is granted debugging access to the process as if it created the process with the `DEBUG_ONLY_THIS_PROCESS` flag.

Return type ctypes.c_ulong

VirtualProtectEx ()

Changes the protection on a region of committed pages in the virtual address space of a specified process.

<https://msdn.microsoft.com/en-us/library/windows/desktop/aa366899%28v=vs.85%29.aspx>

Parameters

- **hProcess** – A handle to the process whose memory protection is to be changed. The handle must have the `PROCESS_VM_OPERATION` access right.

- **lpAddress** (*LPVOID*) – A pointer to the base address of the region of pages whose access protection attributes are to be changed.
- **dwSize** (*SIZE_T*) – The size of the region whose access protection attributes are changed, in bytes.
- **flNewProtect** (*DWORD*) – The memory protection option. This parameter can be one of the memory protection constants.
- **lpflOldProtect** (*PDWORD*) – The handle to a process. The function allocates memory within the virtual address space of this process.

Return type ctypes.c_long

CreateToolhelp32Snapshot ()

Takes a snapshot of the specified processes, as well as the heaps, modules, and threads used by these processes.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms682489%28v=vs.85%29.aspx>

Parameters

- **dwFlags** (*DWORD*) – The portions of the system to be included in the snapshot.
- **th32ProcessID** (*DWORD*) – The process identifier of the process to be included in the snapshot. This parameter can be zero to indicate the current process. This parameter is used when the TH32CS_SNAPHEAPLIST, TH32CS_SNAPMODULE, TH32CS_SNAPMODULE32, or TH32CS_SNAPALL value is specified. Otherwise, it is ignored and all processes are included in the snapshot.

Return type ctypes.c_ulong

Module32First ()

Retrieves information about the first module associated with a process.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms684218%28v=vs.85%29.aspx>

Parameters

- **hSnapshot** (*HANDLE*) – A handle to the snapshot returned from a previous call to the CreateToolhelp32Snapshot function.
- **lpme** (*LPMODULEENTRY32*) – A pointer to a MODULEENTRY32 structure.

Return type ctypes.c_long

Module32Next ()

Retrieves information about the next module associated with a process or thread.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms684221%28v=vs.85%29.aspx>

Parameters

- **hSnapshot** (*HANDLE*) – A handle to the snapshot returned from a previous call to the CreateToolhelp32Snapshot function.
- **lpme** (*LPMODULEENTRY32*) – A pointer to a MODULEENTRY32 structure.

Return type ctypes.c_long

Process32First ()

Retrieves information about the first process encountered in a system snapshot.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms684834%28v=vs.85%29.aspx>

Parameters

- **hSnapshot** (*HANDLE*) – A handle to the snapshot returned from a previous call to the `CreateToolhelp32Snapshot` function.
- **lppe** (*LPPROCESSENTRY32*) – A pointer to a `PROCESSENTRY32` structure. It contains process information such as the name of the executable file, the process identifier, and the process identifier of the parent process.

Return type `ctypes.c_long`

Process32Next ()

Retrieves information about the next process recorded in a system snapshot.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms684836%28v=vs.85%29.aspx>

Parameters

- **hSnapshot** (*HANDLE*) – A handle to the snapshot returned from a previous call to the `CreateToolhelp32Snapshot` function.
- **lppe** (*LPPROCESSENTRY32*) – A pointer to a `PROCESSENTRY32` structure.

Return type `ctypes.c_long`

Thread32First ()

Retrieves information about the first thread of any process encountered in a system snapshot.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms686728%28v=vs.85%29.aspx>

Parameters

- **hSnapshot** (*HANDLE*) – A handle to the snapshot returned from a previous call to the `CreateToolhelp32Snapshot` function.
- **lppte** (*LPTHREADENTRY32*) – A pointer to a `THREADENTRY32` structure.

Return type `ctypes.c_long`

Thread32Next ()

Retrieves information about the next thread of any process encountered in the system memory snapshot.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms686731%28v=vs.85%29.aspx>

Parameters

- **hSnapshot** (*HANDLE*) – A handle to the snapshot returned from a previous call to the `CreateToolhelp32Snapshot` function.
- **lppte** (*LPTHREADENTRY32*) – A pointer to a `THREADENTRY32` structure.

Return type `ctypes.c_long`

OpenThread ()

Opens an existing thread object.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms684335%28v=vs.85%29.aspx>

Parameters

- **dwDesiredAccess** (*DWORD*) – The access to the thread object. This access right is checked against the security descriptor for the thread. This parameter can be one or more of the thread access rights.
- **bInheritHandle** (*BOOL*) – If this value is `TRUE`, processes created by this process will inherit the handle. Otherwise, the processes do not inherit this handle.
- **dwThreadId** (*DWORD*) – The identifier of the thread to be opened.

Return type ctypes.c_ulong

SuspendThread ()

Suspends the specified thread.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms686345%28v=vs.85%29.aspx>

Parameters **hThread** (*HANDLE*) – A handle to the thread that is to be suspended.

Return type ctypes.c_ulong

ResumeThread ()

Decrements a thread's suspend count. When the suspend count is decremented to zero, the execution of the thread is resumed.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms685086%28v=vs.85%29.aspx>

Parameters **hThread** (*HANDLE*) – A handle to the thread that is to be suspended.

Return type ctypes.c_ulong

GetThreadContext ()

Retrieves the context of the specified thread.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms679362%28v=vs.85%29.aspx>

Parameters

- **hThread** (*HANDLE*) – A handle to the thread whose context is to be retrieved. The handle must have `THREAD_GET_CONTEXT` access to the thread.
- **lpContext** (*LPCONTEXT*) – A pointer to a `CONTEXT` structure that receives the appropriate context of the specified thread.

Return type ctypes.c_ulong

SetThreadContext ()

Sets the context for the specified thread.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms680632%28v=vs.85%29.aspx>

Parameters

- **hThread** (*HANDLE*) – A handle to the thread whose context is to be set. The handle must have the `THREAD_SET_CONTEXT` access right to the thread.
- **lpContext** (*CONTEXT*) – A pointer to a `CONTEXT` structure that contains the context to be set in the specified thread.

Return type ctypes.c_ulong

VirtualFreeEx ()

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

- **hProcess** (*HANDLE*) – A handle to a process. The function frees memory within the virtual address space of the process.
- **lpAddress** (*LPVOID*) – A pointer to the starting address of the region of memory to be freed.
- **dwSize** (*SIZE_T*) – The size of the region of memory to free, in bytes.

- **dwFreeType** (*DWORD*) – The type of free operation.

Return type ctypes.c_long

2.2.2 Structure

Placeholder for windows structures and constants.

class ModuleEntry32 (*ctypes.Structure*)

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>

```
_fields_ = [
    ('dwSize' , ctypes.c_ulong ) ,
    ('th32ModuleID' , ctypes.c_ulong ) ,
    ('th32ProcessID' , ctypes.c_ulong ) ,
    ('GblcntUsage' , ctypes.c_ulong ) ,
    ('ProccntUsage' , ctypes.c_ulong ) ,
    ('modBaseAddr' , ctypes.POINTER(ctypes.c_byte)),
    ('modBaseSize' , ctypes.c_ulong ) ,
    ('hModule' , ctypes.c_ulong ) ,
    ('szModule' , ctypes.c_char * 256 ) ,
    ('szExePath' , ctypes.c_char * 260 )
]
```

class ProcessEntry32 (*ctypes.Structure*)

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

```
_fields_ = [
    ('dwSize' , ctypes.c_ulong ) ,
    ('cntUsage' , ctypes.c_ulong) ,
    ('th32ProcessID' , ctypes.c_ulong) ,
    ('th32DefaultHeapID' , ctypes.POINTER(ctypes.c_ulong) ) ,
    ('th32ModuleID' , ctypes.c_ulong) ,
    ('cntThreads' , ctypes.c_ulong) ,
    ('th32ParentProcessID' , ctypes.c_ulong) ,
    ('pcPriClassBase' , ctypes.c_long) ,
    ('dwFlags' , ctypes.c_ulong) ,
    ('szExeFile' , ctypes.c_char * 260 )
]
```

szExeFile

Returns The szExeFile as a decoded utf-8 string

Return type string

class ThreadEntry32 (*ctypes.Structure*)

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

```
_fields_ = [
    ('dwSize' , ctypes.c_ulong),
    ('cntUsage' , ctypes.c_ulong),
    ('th32ThreadID' , ctypes.c_ulong),
```

(continues on next page)

(continued from previous page)

```

("th32OwnerProcessID", ctypes.c_ulong),
("tpBasePri", ctypes.c_ulong),
("tpDeltaPri", ctypes.c_ulong),
("dwFlags", ctypes.c_ulong)
]

```

PROCESS (object) :

Process manipulation flags

PROCESS_CREATE_PROCESS = 0x0080

Required to create a process.

PROCESS_CREATE_THREAD = 0x0002

Required to create a thread.

PROCESS_DUP_HANDLE = 0x0040

Required to duplicate a handle using DuplicateHandle.

PROCESS_QUERY_INFORMATION = 0x0400

Required to retrieve certain information about a process, such as its token, exit code, and priority class (see OpenProcessToken).

PROCESS_QUERY_LIMITED_INFORMATION = 0x1000

Required to retrieve certain information about a process (see GetExitCodeProcess, GetPriorityClass, IsProcessInJob, QueryFullProcessImageName).

PROCESS_SET_INFORMATION = 0x0200

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

PROCESS_SET_QUOTA = 0x0100

Required to set memory limits using SetProcessWorkingSetSize.

PROCESS_SUSPEND_RESUME = 0x0800

Required to suspend or resume a process.

PROCESS_TERMINATE = 0x0001

Required to terminate a process using TerminateProcess.

PROCESS_VM_OPERATION = 0x0008

Required to perform an operation on the address space of a process (see VirtualProtectEx and WriteProcessMemory).

PROCESS_VM_READ = 0x0010

Required to read memory in a process using ReadProcessMemory.

PROCESS_VM_WRITE = 0x0020

Required to write to memory in a process using WriteProcessMemory.

SYNCHRONIZE = 0x00100000

Required to wait for the process to terminate using the wait functions.

PROCESS_ALL_ACCESS = (0x000F0000 | 0x00100000 | 0xFFF)

All possible access rights for a process object.

DELETE = 0x00010000

Required to delete the object.

READ_CONTROL = 0x00020000

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.

WRITE_DAC = 0x00040000

Required to modify the DACL in the security descriptor for the object.

WRITE_OWNER = 0x00080000

Required to change the owner in the security descriptor for the object.

class MemoryAllocation (*object*)

The type of memory allocation <https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890%28v=vs.85%29.aspx>

MEM_COMMIT = 0x00001000

Allocates memory charges (from the overall size of memory and the paging files on disk) for the specified reserved memory pages. The function also guarantees that when the caller later initially accesses the memory, the contents will be zero. Actual physical pages are not allocated unless/until the virtual addresses are actually accessed.

MEM_RESERVE = 0x00002000

Reserves a range of the process's virtual address space without allocating any actual physical storage in memory or in the paging file on disk.

MEM_RESET = 0x00080000

Indicates that data in the memory range specified by lpAddress and dwSize is no longer of interest. The pages should not be read from or written to the paging file. However, the memory block will be used again later, so it should not be decommitted. This value cannot be used with any other value.

MEM_RESET_UNDO = 0x1000000

MEM_RESET_UNDO should only be called on an address range to which MEM_RESET was successfully applied earlier. It indicates that the data in the specified memory range specified by lpAddress and dwSize is of interest to the caller and attempts to reverse the effects of MEM_RESET. If the function succeeds, that means all data in the specified address range is intact. If the function fails, at least some of the data in the address range has been replaced with zeroes.

MEM_LARGE_PAGES = 0x20000000

Allocates memory using large page support.

MEM_PHYSICAL = 0x00400000

Reserves an address range that can be used to map Address Windowing Extensions (AWE) pages.

MEM_TOP_DOWN = 0x00100000

Allocates memory at the highest possible address. This can be slower than regular allocations, especially when there are many allocations.

MEM_DECOMMIT = 0x4000

Decommits the specified region of committed pages. After the operation, the pages are in the reserved state.

MEM_RELEASE = 0x8000

Releases the specified region of pages. After this operation, the pages are in the free state.

class MemoryProtection (*object*)

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](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366786(v=vs.85).aspx)

PAGE_EXECUTE = 0x10

Enables execute access to the committed region of pages. An attempt to write to the committed region results in an access violation.

PAGE_EXECUTE_READ = 0x20

Enables execute or read-only access to the committed region of pages. An attempt to write to the committed

region results in an access violation.

PAGE_EXECUTE_READWRITE = 0x40

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

PAGE_EXECUTE_WRITECOPY = 0x80

Enables execute, read-only, or copy-on-write access to a mapped view of a file mapping object. An attempt to write to a committed copy-on-write page results in a private copy of the page being made for the process. The private page is marked as PAGE_EXECUTE_READWRITE, and the change is written to the new page.

PAGE_NOACCESS = 0x01

Disables all access to the committed region of pages. An attempt to read from, write to, or execute the committed region results in an access violation.

PAGE_READONLY = 0x02

Enables read-only access to the committed region of pages. An attempt to write to the committed region results in an access violation. If Data Execution Prevention is enabled, an attempt to execute code in the committed region results in an access violation.

PAGE_READWRITE = 0x04

Enables read-only or read/write access to the committed region of pages. If Data Execution Prevention is enabled, attempting to execute code in the committed region results in an access violation.

PAGE_WRITECOPY = 0x08

Enables read-only or copy-on-write access to a mapped view of a file mapping object. An attempt to write to a committed copy-on-write page results in a private copy of the page being made for the process. The private page is marked as PAGE_READWRITE, and the change is written to the new page. If Data Execution Prevention is enabled, attempting to execute code in the committed region results in an access violation.

PAGE_GUARD = 0x100

Pages in the region become guard pages. Any attempt to access a guard page causes the system to raise a STATUS_GUARD_PAGE_VIOLATION exception and turn off the guard page status. Guard pages thus act as a one-time access alarm. For more information, see Creating Guard Pages.

PAGE_NOCACHE = 0x200

Sets all pages to be non-cachable. Applications should not use this attribute except when explicitly required for a device. Using the interlocked functions with memory that is mapped with SEC_NOCACHE can result in an EXCEPTION_ILLEGAL_INSTRUCTION exception.

PAGE_WRITECOMBINE = 0x400

Sets all pages to be write-combined. Applications should not use this attribute except when explicitly required for a device. Using the interlocked functions with memory that is mapped as write-combined can result in an EXCEPTION_ILLEGAL_INSTRUCTION exception.

SIZE_OF_80387_REGISTERS = 80

class FLOATING_SAVE_AREA (*ctypes.Structure*)

Undocumented ctypes.Structure used for ThreadContext.

```

_fields_ = [
    ('ControlWord', ctypes.c_uint),
    ('StatusWord', ctypes.c_uint),
    ('TagWord', ctypes.c_uint),
    ('ErrorOffset', ctypes.c_uint),
    ('ErrorSelector', ctypes.c_uint),
    ('DataOffset', ctypes.c_uint),
    ('DataSelector', ctypes.c_uint),
    ('RegisterArea', ctypes.c_byte * SIZE_OF_80387_REGISTERS),

```

(continues on next page)

(continued from previous page)

```
    ('Cr0NpxState', ctypes.c_uint)
]
```

MAXIMUM_SUPPORTED_EXTENSION = 512**class ThreadContext** (*ctypes.Structure*)

Represents a thread context

```
_fields_ = [
    ('ContextFlags', ctypes.c_uint),
    ('Dr0', ctypes.c_uint),
    ('Dr1', ctypes.c_uint),
    ('Dr2', ctypes.c_uint),
    ('Dr3', ctypes.c_uint),
    ('Dr6', ctypes.c_uint),
    ('Dr7', ctypes.c_uint),
    ('FloatSave', FLOATING_SAVE_AREA),
    ('SegGs', ctypes.c_uint),
    ('SegFs', ctypes.c_uint),
    ('SegEs', ctypes.c_uint),
    ('SegDs', ctypes.c_uint),
    ('Edi', ctypes.c_uint),
    ('Esi', ctypes.c_uint),
    ('Ebx', ctypes.c_uint),
    ('Edx', ctypes.c_uint),
    ('Ecx', ctypes.c_uint),
    ('Eax', ctypes.c_uint),
    ('Ebp', ctypes.c_uint),
    ('Eip', ctypes.c_uint),
    ('SegCs', ctypes.c_uint),
    ('EFlags', ctypes.c_uint),
    ('Esp', ctypes.c_uint),
    ('SegSs', ctypes.c_uint),
    ('ExtendedRegisters', ctypes.c_byte * MAXIMUM_SUPPORTED_EXTENSION)
]
```


Legal information, changelog are here for the interested.

3.1 License

Pymem is licensed under a two clause BSD License. It basically means: do whatever you want with it as long as the copyright in Pymem sticks around, the conditions are not modified and the disclaimer is present. Furthermore you must not use the names of the authors to promote derivatives of the software without written consent.

3.1.1 Authors

Pymem is written and maintained by Fabien Reboia:

Development Lead

- Fabien Reboia <srounet@gmail.com>

3.1.2 General License Definitions

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3.1.3 Fasm License

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps:

(1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents.

States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and

modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all

the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent

works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports

equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option

remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license,

and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version

published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

{one line to give the program's name and a brief idea of what it does.} Copyright (C) {year} {name of author}

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
{project} Copyright (C) {year} {fullname} This program comes with ABSOLUTELY NO WARRANTY;  
for details type 'show w'. This is free software, and you are welcome to redistribute it under certain  
conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

Symbols

`__init__()` (Pymem method), 5

A

`allocate()` (Pymem method), 6
`allocate_memory()` (built-in function), 13
`assemble()` (Pymem method), 7

B

`base_address()` (built-in function), 23

C

`close_handle()` (built-in function), 24
`close_main_thread()` (Pymem method), 7
`close_process()` (Pymem method), 6
`CloseHandle()` (built-in function), 26
`CouldNotOpenProcess`, 13
`CreateToolhelp32Snapshot()` (built-in function), 28

D

`DebugActiveProcess()` (built-in function), 27

F

`FLOATING_SAVE_AREA` (built-in class), 34
`free()` (Pymem method), 6
`free_memory()` (built-in function), 13

G

`GetCurrentProcess()` (built-in function), 26
`GetLastError()` (built-in function), 26
`GetThreadContext()` (built-in function), 30

L

`list_process_modules()` (built-in function), 25
`list_process_thread()` (built-in function), 25
`list_processes()` (built-in function), 24

M

`MemoryAllocation` (built-in class), 33

`MemoryProtection` (built-in class), 33

`MemoryReadError`, 13

`MemoryWriteError`, 13

`Module32First()` (built-in function), 28

`Module32Next()` (built-in function), 28

`module_from_name()` (built-in function), 25

`ModuleEntry32` (built-in class), 31

O

`open()` (built-in function), 24
`open_main_thread()` (built-in function), 24
`open_main_thread()` (Pymem method), 6
`open_process_from_id()` (Pymem method), 6
`open_process_from_name()` (Pymem method), 5
`open_thread()` (built-in function), 24
`OpenProcess()` (built-in function), 25
`OpenThread()` (built-in function), 29

P

`Process32First()` (built-in function), 28
`Process32Next()` (built-in function), 29
`process_base_address` (Pymem attribute), 6
`process_from_id()` (built-in function), 25
`process_from_name()` (built-in function), 24
`ProcessEntry32` (built-in class), 31
`ProcessError`, 13
`ProcessNotFound`, 13
`Pymem` (built-in class), 5
`PymemError`, 13
`PymemMemoryError`, 13

R

`read_bytes()` (built-in function), 14
`read_bytes()` (Pymem method), 7
`read_char()` (built-in function), 14
`read_char()` (Pymem method), 7
`read_double()` (built-in function), 18
`read_double()` (Pymem method), 9
`read_float()` (built-in function), 16

read_float() (Pymem method), 8
read_int() (built-in function), 15
read_int() (Pymem method), 8
read_long() (built-in function), 16
read_long() (Pymem method), 9
read_longlong() (built-in function), 17
read_longlong() (Pymem method), 9
read_short() (built-in function), 15
read_short() (Pymem method), 8
read_string() (built-in function), 18
read_string() (Pymem method), 10
read_uchar() (built-in function), 14
read_uchar() (Pymem method), 7
read_uint() (built-in function), 16
read_uint() (Pymem method), 8
read_ulong() (built-in function), 17
read_ulong() (Pymem method), 9
read_ulonglong() (built-in function), 17
read_ulonglong() (Pymem method), 9
read_ushort() (built-in function), 15
read_ushort() (Pymem method), 8
ReadProcessMemory() (built-in function), 26
ResumeThread() (built-in function), 30

S

set_debug_privilege() (Pymem method), 5
SetThreadContext() (built-in function), 30
SuspendThread() (built-in function), 30
szExeFile (ProcessEntry32 attribute), 31

T

TerminateProcess() (built-in function), 26
Thread32First() (built-in function), 29
Thread32Next() (built-in function), 29
ThreadContext (built-in class), 35
ThreadEntry32 (built-in class), 31

V

VirtualAllocEx() (built-in function), 27
VirtualFreeEx() (built-in function), 30
VirtualProtectEx() (built-in function), 27

W

WinAPIError, 13
write_bytes() (built-in function), 18
write_char() (built-in function), 19
write_char() (Pymem method), 12
write_double() (built-in function), 23
write_double() (Pymem method), 12
write_float() (built-in function), 21
write_float() (Pymem method), 11
write_int() (built-in function), 20
write_int() (Pymem method), 10

write_long() (built-in function), 21
write_long() (Pymem method), 11
write_longlong() (built-in function), 22
write_longlong() (Pymem method), 11
write_short() (built-in function), 19
write_short() (Pymem method), 10
write_string() (built-in function), 23
write_string() (Pymem method), 12
write_uint() (built-in function), 20
write_uint() (Pymem method), 10
write_ulong() (built-in function), 21
write_ulong() (Pymem method), 11
write_ulonglong() (built-in function), 22
write_ulonglong() (Pymem method), 12
write_ushort() (built-in function), 20
write_ushort() (Pymem method), 11
WriteProcessMemory() (built-in function), 27