Pymem Documentation

Release 1.0

Fabien Reboia

Jun 02, 2018

Contents

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 it's current version does not supports x64.

CHAPTER 1

User's Guide

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

You should head over the tutorials 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.

CHAPTER 2

Api Reference

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

2.1 API

This part of the documentation covers all the 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
- **`lpszPrivilege`** (*`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 occured

exception MemoryReadError(*PymemMemoryError*)
Raised when a memory read error occured

exception MemoryWriteError(*PymemMemoryError*)
Raised when a memory write error occured

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('<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_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.
- **lpte** (*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.
- **lpte** (*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_long

`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_long

`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 ),
    ( 'GlblcntUsage' , 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

```
_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

```
_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

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)
]
```

CHAPTER 3

Additional Notes

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

> 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

Index