Virtual Memory Validator

by

Software Verify

Software Verify

Troubleshoot your software

Virtual Memory Validator

Virtual Memory Visualization for Windows NT, 2000, XP

by Software Verify Limited

Welcome to the VM Validator software tool. VM Validator is software tool that provides a graphic visualization of an application's virtual memory space and of the application's virtual memory page table. For certain types of memory performance problem VM Validator provides a crucial insight into the workings of your application.

We hope you will find this document useful.

VM Validator Help

Printed: November 2024 in United Kingdom.

Table of Contents

Foreword 1

Part I Overview 2

1 Introducing VM Validator ... 3
2 Why VM Validator? ... 4
3 What do you need to run VM Validator? ... 4
4 How to get VM Validator ... 5
5 How does VM Validator work? ... 5
6 What does VM Validator do? ... 5

Part II The User Interface 6

1 Menu Reference ... 7
2 Toolbar Reference ... 10
3 The main display ... 11
Summary ... 11
Virtual ... 15
Pages ... 16
Paragraphs ... 19
4 Load Session ... 21
5 Save Session ... 22
6 Load Minidump ... 22
7 Save Minidump ... 23
8 Settings ... 23
Symbols ... 24
9 Starting your target program ... 26
Launching the program ... 26
Re-Launching the program ... 28
Injecting into a running program ... 28
10 Stopping your target program ... 30
11 Closing VM Validator ... 30
12 Software Updates ... 30
13 View Memory Dialog ... 34
14 Search Memory Dialog ... 35
15 Help ... 36
Tip of the day ... 36
About VM Validator ... 36
Help Topics ... 36

Part III Command Line Interface 37

1 Alphabetic Reference ... 38

2 Usage Reference 39
3 Command Line Examples 41

Part IV Frequently Asked Questions 44

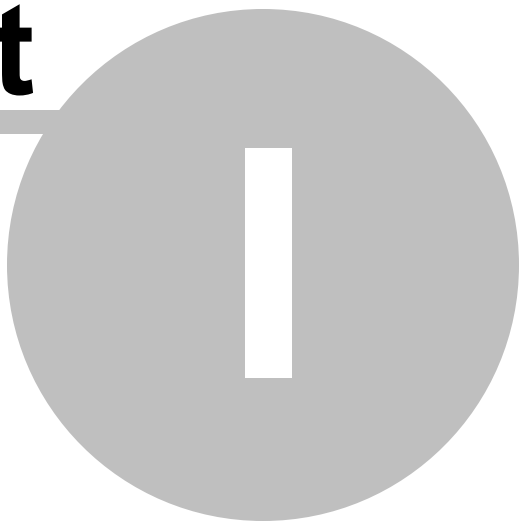
1 I have an idea for a feature, can it be added to VM Validator? 45
2 What file extensions does VM Validator use for itself? 45
3 What is Wasted Memory? 45

Index 47

Foreword

This is just another title page
placed between table of contents
and topics

Part



1 Overview

Welcome to the VM Validator help manual.

This section provides a brief overview of the capabilities of VM Validator.

This help manual is available in Compiled HTML Help (Windows Help files), PDF, and online.

Windows Help	https://www.softwareverify.com/documentation/chm/vmValidator.chm
PDF	https://www.softwareverify.com/documentation/pdfs/vmValidator.pdf
Online	https://www.softwareverify.com/documentation/html/vmValidator/index.html

1.1 Introducing VM Validator

VM Validator is an tool for visualizing the virtual memory of a process for Windows XP and any subsequent version, for both 32 bit (x86) and 64 bit (x64) processors.

VM Validator reads the virtual memory information for a given process on a regular basis and displays the results graphically. This allows you to identify the impact that virtual memory paging is having on your program's performance.

The user interface is split into three separate sections, each section dedicated to a different task. Typically when using VM Validator a user will use one section to analyse a problem that is present in the target program. The four main sections are:

- Summary

An high level overview of the main memory statistics of the program. Additional data relating to DLLs and page faults is also available.

- Virtual

A visual display of the allocation state of each memory page in the program. This is a similar view (that has different functionality) to the Virtual view in Memory Validator.

- Pages

A tabular display of all the virtual memory pages in the program. This is a similar view (that has different functionality) to the Virtual view in Memory Validator.

- Paragraphs

A tabular display of all the virtual memory paragraphs in the program. This is a similar view (that has different functionality) to the Virtual view in Memory Validator.

A memory paragraph is the smallest amount of memory that the VirtualAlloc() function can allocate. This size is defined by the operating system dwAllocationGranularity returned from Win32 API call **GetSystemInfo()**).
For 32 bit x86 Windows Operating Systems a memory paragraph is 64Kb.

1.2 Why VM Validator?

VM Validator allows you to analyse a program's virtual memory performance in a graphical manner. This is much more useful than a binary dump of page addresses that can be found in various books on undocumented secrets about Windows NT® books. As the display changes you can identify areas that are repeatedly paged in and then paged out. When these areas have been identified you can make changes to your program's memory management to try to reduce the page swapping.

Reliable

VM Validator has been created with the following criteria in mind.

1) VM Validator must have no adverse effect on the program's behaviour.

The program must behave in the same way when being inspected by VM Validator as when the program is running without being inspected by VM Validator.

2) VM Validator must be reliable and avoid causing the target program to crash.

3) VM Validator must be capable of having as little impact on the target programs performance as possible.

4) VM Validator's user interface must be independent of the target program.

VM Validator's user interface is independent of the target program.

- If the VM Validator user interface crashes, your target program will not crash.
- If the target program crashes the VM Validator user interface will not crash - you will still have data to work with.
- If the target program is stopped in the debugger, VM Validator's user interface will continue to work.

5) Flexibility

Where there are multiple ways of presenting the data, the user should be given a choice over how that display works. Not all users like the same choices, so providing some choice over the display is always better than forcing all users to use the same settings.

1.3 What do you need to run VM Validator?

- User Privileges

A standard user account is required.

- Operating System

VM Validator requires Windows XP or better.

1.4 How to get VM Validator

VM Validator is free commercial use. VM Validator can be downloaded for Software Verify's website at https://www.softwareverify.com.

Whilst VM Validator is free for commercial use, VM Validator is copyrighted software and is not in the public domain. You are free to use the software at your own risk.
You are not allowed to distribute the software in any form, or to sell the software, or to host the software on a website.

Contact Software Verify at:

Software Verify Limited
Suffolk Business Park
Eldo House
Kempson Way
Bury Saint Edmunds
IP32 7AR
United Kingdom

email sales@softwareverify.com
web https://www.softwareverify.com
blog https://www.softwareverify.com/blog
twitter http://twitter.com/softwareverify

1.5 How does VM Validator work?

VM Validator monitors the target process in a non-invasive manner collecting data about the target process using the Win32 API.

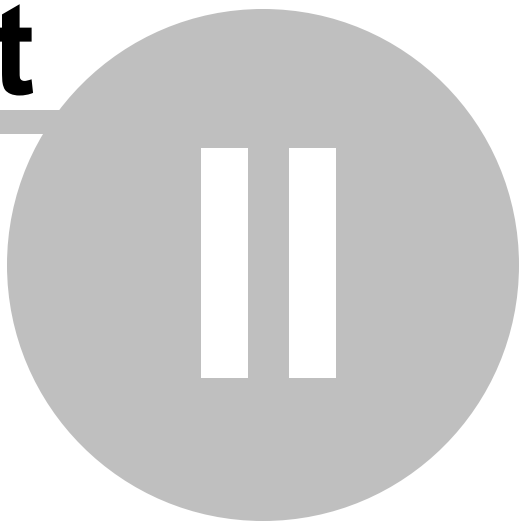
1.6 What does VM Validator do?

VM Validator provides functionality to allow virtual memory information to be inspected so that virtual memory status and swapping can be detected and analysed.

As the display changes you can identify areas that are repeatedly paged in and then paged out. When these areas have been identified you can make changes to your program's memory management to try to reduce the page swapping.

An additional display shows the classification that the Windows NT® memory manager gives to each area of memory.

Part



2 The User Interface

This section describes the various functions of the user interface so that you can get the most from using VM Validator.

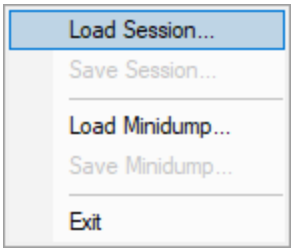
Typical usage of VM Validator is:
- Start the target program.
- Examine the virtual memory paging behaviour of the target program as you use the target program to perform actions of interest.
- Close the program.

However there is much more to VM Validator than the above. Whilst your program is running you can display data in VM Validator and using the information from VM Validator you may gain insight into a bug you are looking at in the debugger.

2.1 Menu Reference

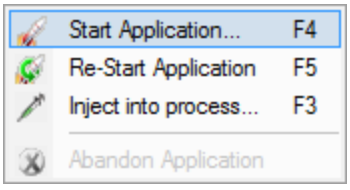
This section lists the various menus in VM Validator and provides links to the appropriate section of the help manual.

File



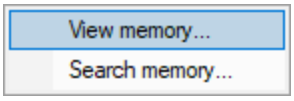
- Load Session
- Save Session
- Load Minidump
- Save Minidump
- Exit

Launch



- Start Application
- Re-Start Application
- Inject
- Abandon Application

Inspect



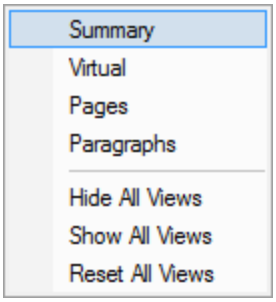
- View Memory Dialog
- Search Memory Dialog

Settings

- Settings

Data Views

The Data Views provides easy control of which tabs are displayed in the main view.



- Summary
- Virtual
- Pages
- Paragraph

Selecting any of the items shows the relevant tab (if it's not visible already), and makes it the current selected tab.

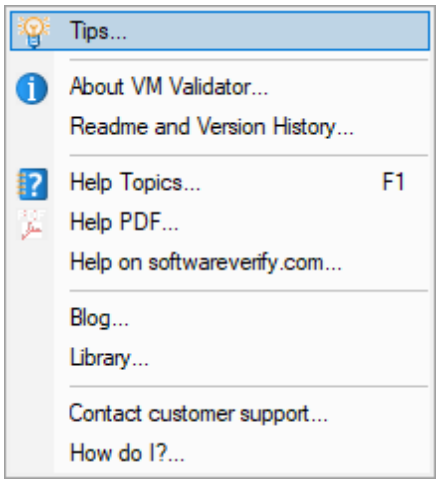
- **Hide All Views** › hides all tabs except the one that's currently visible
- **Show All Views** › shows all the listed tabs, and in that order
- **Reset All Views** › shows the original settings tabs.

Software Updates

The Software Updates menu controls how often software updates are downloaded.

This is discussed in detail in Software Updates.

Help



- Tips
- About VM Validator

  Version and licence information for VM Validator.
- Readme and Version History

  Displays the VM Validator readme and version history.
- Help Topics

  Help in compiled HTML format.
- Help PDF

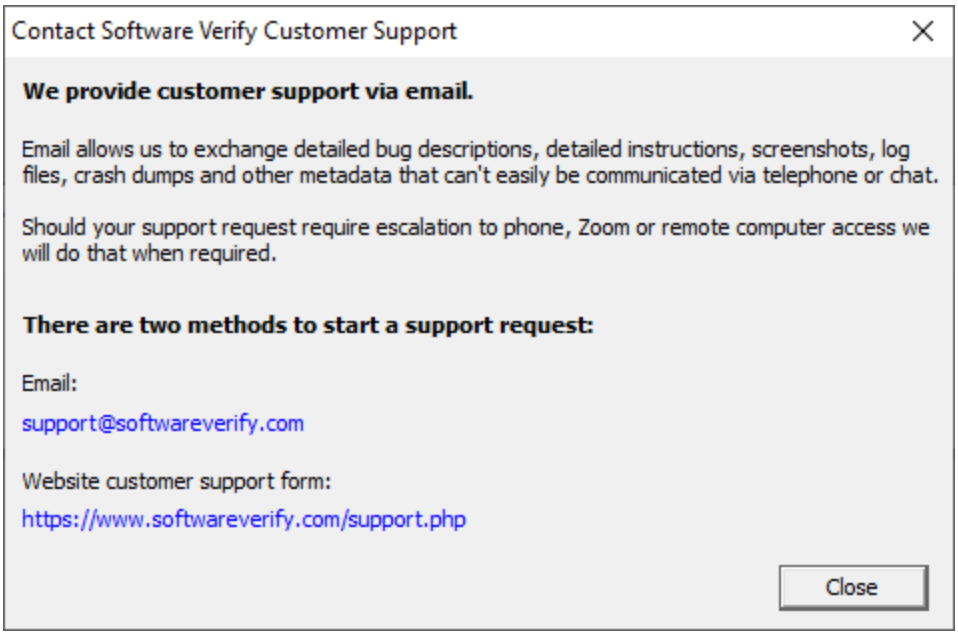
  Help in PDF format.

- Help on softwareverify.com

  Displays the documentation page on the software verify website.
- Blog

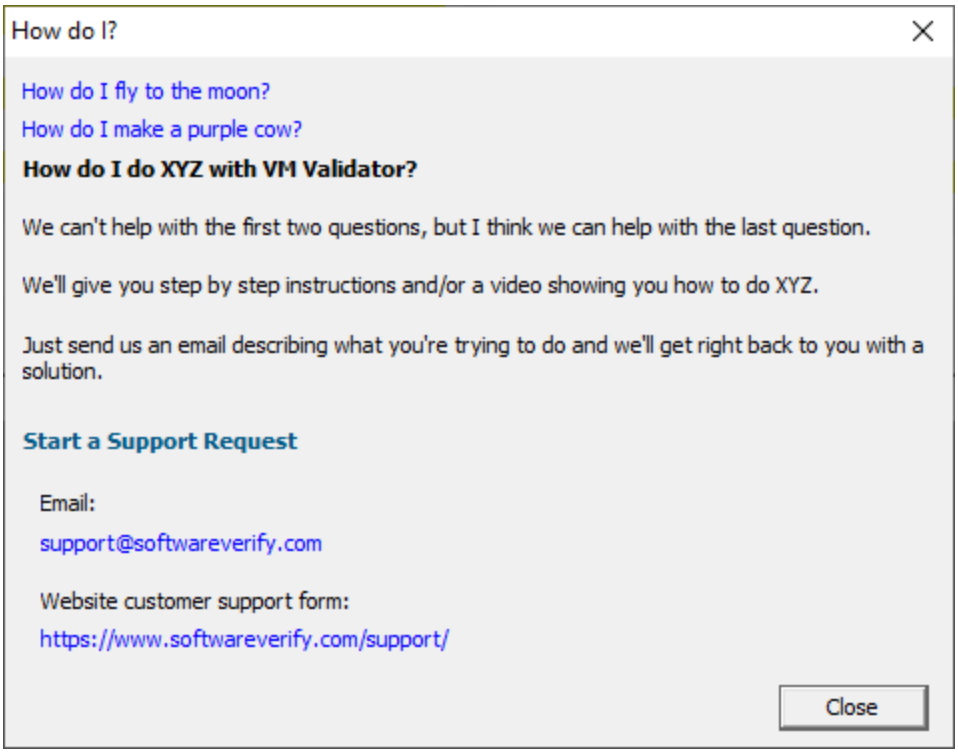
  Displays the Software Verify blog.
- Library

  Displays the Software Verify library - our best articles grouped by related topics.
- Contact customer support



Click a link to contact customer support

- How Do I?



2.2 Toolbar Reference

This section describes the toolbar in VM Validator and provides links to the appropriate section of the help manual. The icons are described in sequence from left to right in the toolbar.

- Settings
- Inject into application
- Launch application
- Re-Launch application
- Stop application

2.3 The main display

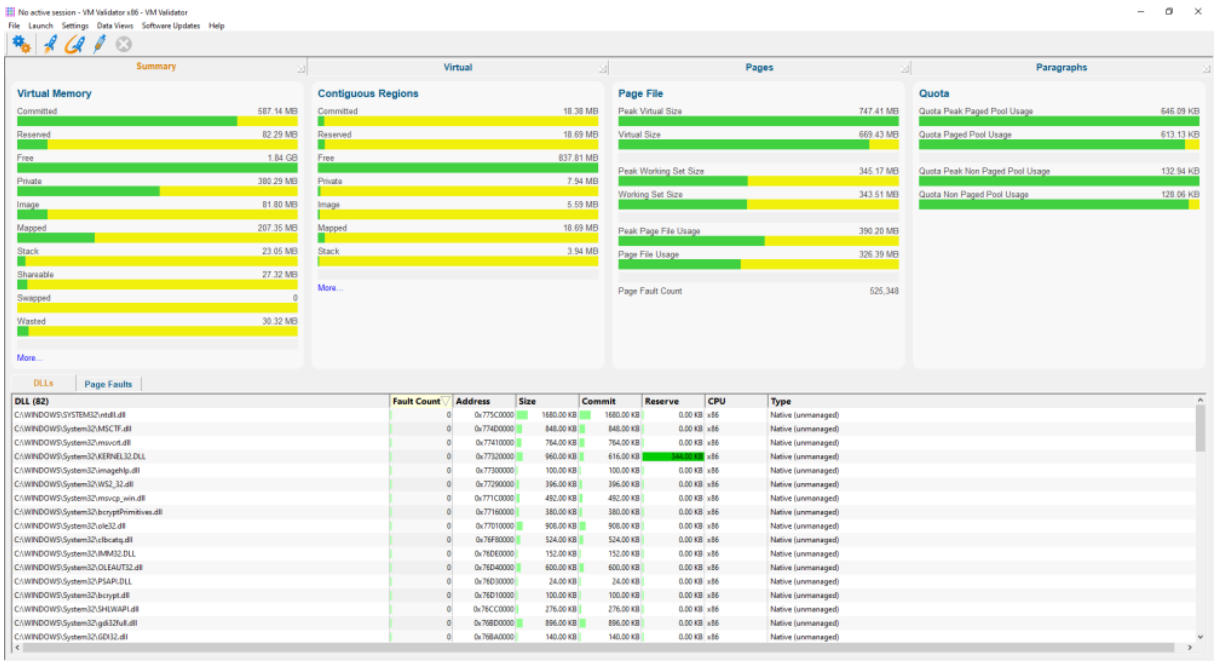
The main display of VM Validator consists of four tabbed windows. Each tabbed window allows the data collected by VM Validator to be viewed, inspected and queried in different and complimentary ways. Typically when using VM Validator a user will use one window to analyse a problem that is present in the target program. The four tabbed windows are:

- Summary
- Virtual
- Pages
- Paragraphs

2.3.1 Summary

The Summary view allows you to get an overview of your application's memory usage.

Four main panels highlight key statistics and a lower panel with two tabs provides information about DLLs and page faults.



Overview

Virtual Memory - statistics about the virtual memory usage of the application.

- **Committed**. Committed memory.
- **Reserved**. Reserved memory.
- **Free**. Free memory.
- **Private**. Memory private to this application.
- **Image**. Executable files (EXE, DLL, etc).
- **Mapped**. Memory mapped files.
- **Stack**. Memory used for thread stacks.
- **Shareable**. Memory shared with other applications.
- **Swapped**. Memory swapped in/out of memory.
- **Wasted**. Memory allocated by VirtualAlloc() but not available to the application to use. What is wasted memory?

Clicking on one of these bars will take you to the Pages tab to display data about the type of memory that was clicked on.

Contiguous Regions - data about the largest contiguous region for each type of virtual memory. These values will let you easily determine if there is enough memory to satisfy a particular allocation request.

- **Committed**. Committed memory.
- **Reserved**. Reserved memory.
- **Free**. Free memory.
- **Private**. Memory private to this application.
- **Image**. Executable files (EXE, DLL, etc).
- **Mapped**. Memory mapped files.
- **Stack**. Memory used for thread stacks.

- **Shareable**. Memory shared with other applications.

Clicking on one of these bars will take you to the Pages tab to display data about the type of contiguous memory that was clicked on.

Page File - data about the working set and the page file.

- Peak Virtual Size
- Virtual Size
- Peak Working Set Size
- Working Set Size
- Peak Page File Usage
- Page File Usage

Quota - data about the paged and non-paged pool used by the operating system and device drivers.

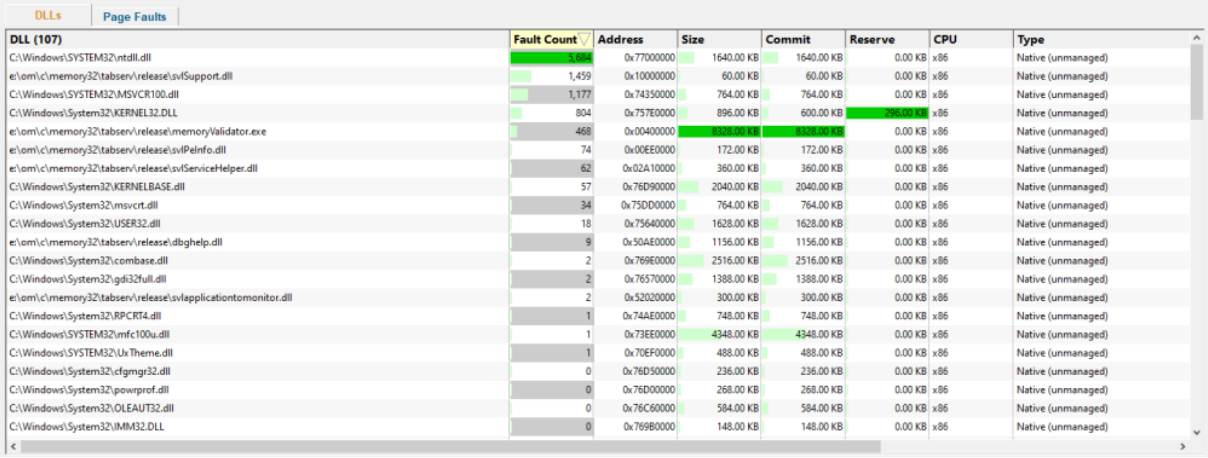
- Quota Peak Page Pool Usage
- Quota Page Pool Usage
- Quota Peak Non Paged Pool Usage
- Quota Non Paged Pool Usage

For more information about paged and non-paged pool see
`https://blogs.technet.microsoft.com/markrussinovich/2009/03/10/pushing-the-limits-of-windows-paged-and-nonpaged-pool/`

DLLs

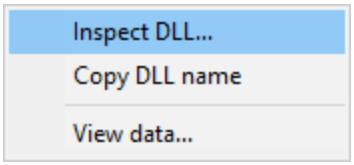
A list of all DLLs in the application with statistics on page faults, load address, dll size, amount of committed memory, amount of reserved memory, the CPU it was built for (x86, x64, AnyCPU) and the type of DLL (native, managed, mixed-mode).

Note that AnyCPU, managed and mixed-mode only apply if a .Net application is being examined.



DLL (107)	Fault Count	Address	Size	Commit	Reserve	CPU	Type
C:\Windows\SYSTEM32\ntdll.dll	5,684	0x77000000	1640.00 KB	1640.00 KB	0.00 KB	x86	Native (unmanaged)
e:\om\c\memory32\tabserv\release\svlSupport.dll	1,459	0x10000000	60.00 KB	60.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\SYSTEM32\MSVCR100.dll	1,177	0x74350000	764.00 KB	764.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\KERNEL32.DLL	804	0x757E0000	896.00 KB	600.00 KB	296.00 KB	x86	Native (unmanaged)
e:\om\c\memory32\tabserv\release\memoryValidator.exe	468	0x00400000	8328.00 KB	8328.00 KB	0.00 KB	x86	Native (unmanaged)
e:\om\c\memory32\tabserv\release\svlPeInfo.dll	74	0x00EE0000	172.00 KB	172.00 KB	0.00 KB	x86	Native (unmanaged)
e:\om\c\memory32\tabserv\release\svlServiceHelper.dll	62	0x02A10000	360.00 KB	360.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\KERNELBASE.dll	57	0x76D90000	2040.00 KB	2040.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\msvcrt.dll	34	0x75DD0000	764.00 KB	764.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\USER32.dll	18	0x75640000	1628.00 KB	1628.00 KB	0.00 KB	x86	Native (unmanaged)
e:\om\c\memory32\tabserv\release\dbghelp.dll	9	0x50AE0000	1156.00 KB	1156.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\combase.dll	2	0x769E0000	2516.00 KB	2516.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\gdi32full.dll	2	0x76570000	1388.00 KB	1388.00 KB	0.00 KB	x86	Native (unmanaged)
e:\om\c\memory32\tabserv\release\svlapplicationtomonitor.dll	2	0x52020000	300.00 KB	300.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\RPCRT4.dll	1	0x74AE0000	748.00 KB	748.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\SYSTEM32\mfc100u.dll	1	0x73EE0000	4348.00 KB	4348.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\SYSTEM32\UxTheme.dll	1	0x70EF0000	488.00 KB	488.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\cfgmgr32.dll	0	0x76D50000	236.00 KB	236.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\powrprof.dll	0	0x76D00000	268.00 KB	268.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\OLEAUT32.dll	0	0x76C60000	584.00 KB	584.00 KB	0.00 KB	x86	Native (unmanaged)
C:\Windows\System32\IMM32.DLL	0	0x769B0000	148.00 KB	148.00 KB	0.00 KB	x86	Native (unmanaged)

A context menu is available for the DLLs grid, to allow DLL inspection using PE File Browser, and to copy the DLL name to clipboard.

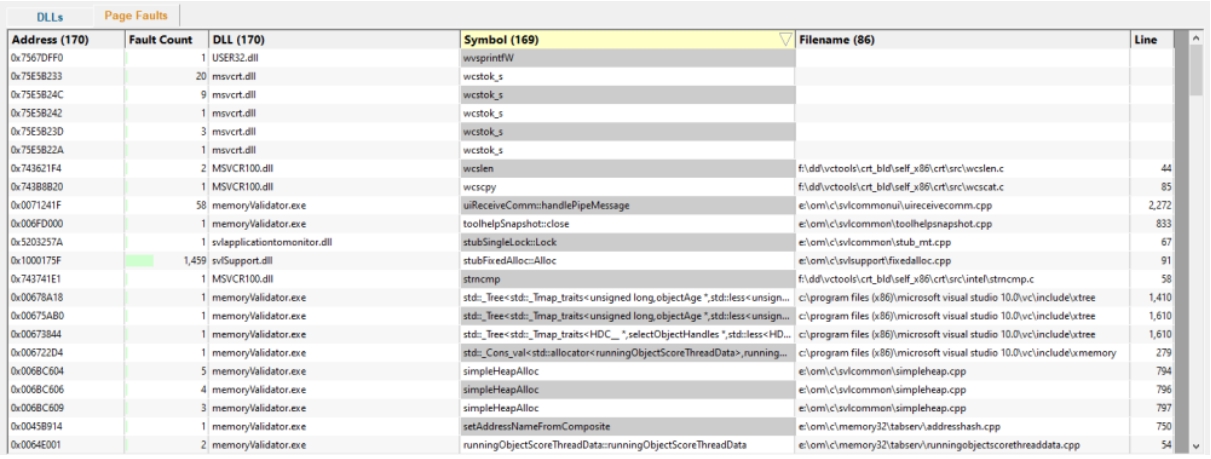


Page Faults

A list of all page faults that have been monitored, which DLL they were in (if any) and what symbol, filename and line number are related to that address.

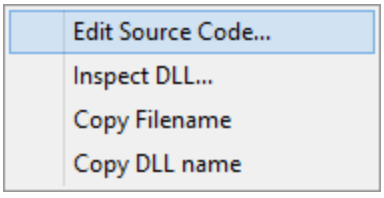
If you have problems getting symbols you may want to change the settings used to obtain symbols.

This display doesn't include all page fault information. If you want detailed page fault information, download Page Fault Monitor from `https://www.softwareverify.com/cpp-page-fault-monitor.php`


DLLs | Page Faults

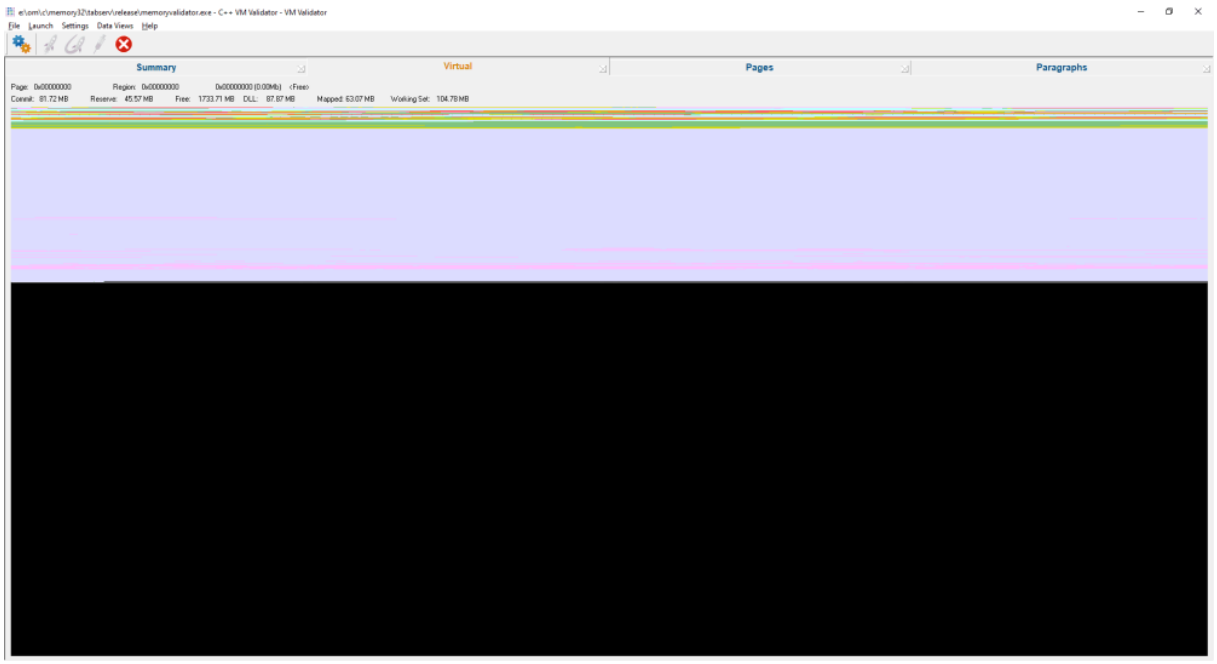
Address (170)	Fault Count	DLL (170)	Symbol (169)	Filename (86)	Line
0x7567DFF0	1	USER32.dll	wvsprintfW		
0x75E5B233	20	msvcrt.dll	wcstok_s		
0x75E5B24C	9	msvcrt.dll	wcstok_s		
0x75E5B242	1	msvcrt.dll	wcstok_s		
0x75E5B23D	3	msvcrt.dll	wcstok_s		
0x75E5B22A	1	msvcrt.dll	wcstok_s		
0x743621F4	2	MSVCR100.dll	wcslen	f:\dd\vctools\crt_bld\self_x86\crt\src\wcslen.c	44
0x743B8B20	1	MSVCR100.dll	wcscpy	f:\dd\vctools\crt_bld\self_x86\crt\src\wcscat.c	85
0x0071241F	58	memoryValidator.exe	uiReceiveComm::handlePipeMessage	e:\om\c\svlcommonui\uireceivecomm.cpp	2,272
0x006FD000	1	memoryValidator.exe	toolhelpSnapshot::close	e:\om\c\svlcommon\toolhelpsnapshot.cpp	833
0x5203257A	1	svlapplicationtomonitor.dll	stubSingleLock::Lock	e:\om\c\svlcommon\stub_mt.cpp	67
0x1000175F	1,459	svlSupport.dll	stubFixedAlloc::Alloc	e:\om\c\svlsupport\fixedalloc.cpp	91
0x743741E1	1	MSVCR100.dll	strncmp	f:\dd\vctools\crt_bld\self_x86\crt\src\intel\strncmp.c	58
0x00678A18	1	memoryValidator.exe	std::_Tree<std::_Tmap_traits<unsigned long,objectAge *,std::less<unsign...	c:\program files (x86)\microsoft visual studio 10.0\vc\include\xtree	1,410
0x00675AB0	1	memoryValidator.exe	std::_Tree<std::_Tmap_traits<unsigned long,objectAge *,std::less<unsign...	c:\program files (x86)\microsoft visual studio 10.0\vc\include\xtree	1,610
0x00673844	1	memoryValidator.exe	std::_Tree<std::_Tmap_traits<HDC__ *,selectObjectHandles *,std::less<HD...	c:\program files (x86)\microsoft visual studio 10.0\vc\include\xtree	1,610
0x006722D4	1	memoryValidator.exe	std::_Cons_val<std::allocator<runningObjectScoreThreadData>,running...	c:\program files (x86)\microsoft visual studio 10.0\vc\include\xmemory	279
0x006BC604	5	memoryValidator.exe	simpleHeapAlloc	e:\om\c\svlcommon\simpleheap.cpp	794
0x006BC606	4	memoryValidator.exe	simpleHeapAlloc	e:\om\c\svlcommon\simpleheap.cpp	796
0x006BC609	3	memoryValidator.exe	simpleHeapAlloc	e:\om\c\svlcommon\simpleheap.cpp	797
0x0045B914	1	memoryValidator.exe	setAddressNameFromComposite	e:\om\c\memory32\tabserv\addresshash.cpp	750
0x0064E001	2	memoryValidator.exe	runningObjectScoreThreadData::runningObjectScoreThreadData	e:\om\c\memory32\tabserv\runningobjectscorethreaddata.cpp	54

A context menu is available for the page faults grid, providing source code editing, DLL inspection, and copy the DLL name or filename to clipboard.



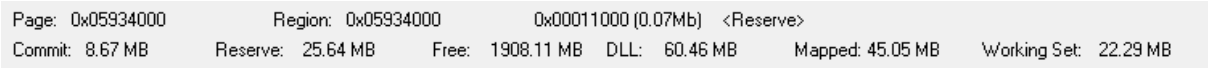
2.3.2 Virtual

The **Virtual** tab displays a visual representation of all the memory in the target program. This is known as the virtual view. A visual display of the allocation state of each memory page in the program.



The virtual view displays information about a process's virtual memory usage whilst the process is running. If there is no target program being monitored, the virtual view does not display any data. The display will be black.

Virtual Memory Information



When the mouse cursor is moved on the display area, some information on the page address, start and end of the region of memory, and the use of the memory are displayed.
If the mouse is over a region of memory that is a DLL the name of the DLL is displayed.

The **Page Address** field displays the start address of the page of virtual memory that the mouse cursor is pointing at.

The **Region Address** field displays the start address and the size of the region of virtual memory that the mouse cursor is pointing at.

2.3.3 Pages

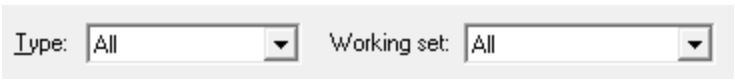
The **Pages** tab displays a tabular representation of all the memory in the target program. This is known as the pages view.

The pages view displays information about a process's virtual memory usage whilst the process is running. If there is no target program being monitored, the pages view does not display any data.

The data displayed is data per virtual memory page (as defined by the operating system). For 32 bit x86 Windows Operating Systems a page is 4Kb.

Filtering

The displayed data can be filtered by Type and by Working Set.

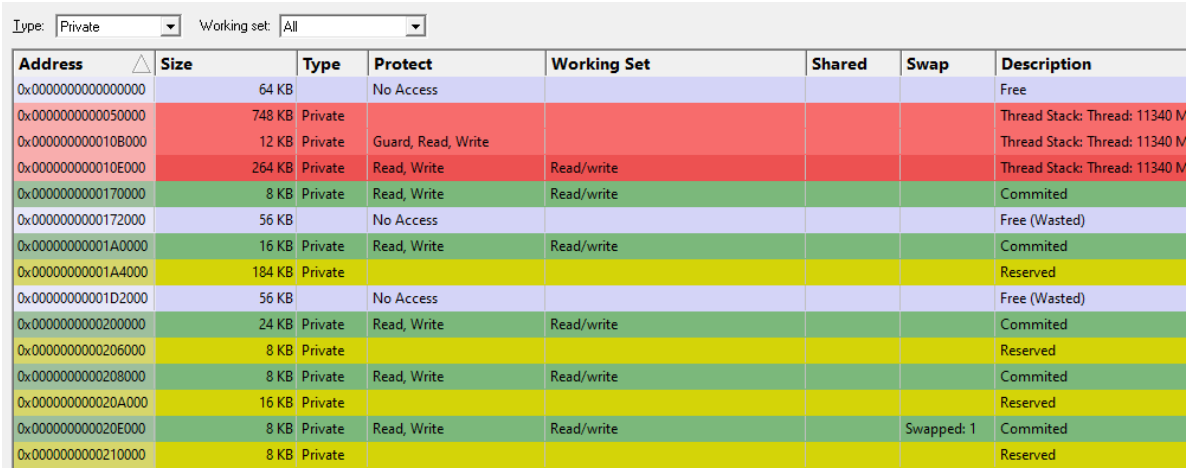


Type

Filtering by type filters memory for the following types of memory:

- **All.** All types of memory.
- **Committed**. Committed memory.
- **Reserved**. Reserved memory.
- **Free**. Free memory.
- **Private**. Memory private to this application.
- **Image**. Executable files (EXE, DLL, etc).
- **Mapped**. Memory mapped files.

- **Stack**. Memory used for thread stacks.
- **Shareable**. Memory shared with other applications.
- **Swapped**. Memory swapped in/out of memory.
- **Wasted**. Memory allocated by VirtualAlloc() but not available to the application to use. What is wasted memory?


Type: Private Working set: All

Address	Size	Type	Protect	Working Set	Shared	Swap	Description
0x0000000000000000	64 KB		No Access				Free
0x0000000000050000	748 KB	Private					Thread Stack: Thread: 11340 M
0x000000000010B000	12 KB	Private	Guard, Read, Write				Thread Stack: Thread: 11340 M
0x000000000010E000	264 KB	Private	Read, Write	Read/write			Thread Stack: Thread: 11340 M
0x0000000000170000	8 KB	Private	Read, Write	Read/write			Commited
0x0000000000172000	56 KB		No Access				Free (Wasted)
0x00000000001A0000	16 KB	Private	Read, Write	Read/write			Commited
0x00000000001A4000	184 KB	Private					Reserved
0x00000000001D2000	56 KB		No Access				Free (Wasted)
0x0000000000200000	24 KB	Private	Read, Write	Read/write			Commited
0x0000000000206000	8 KB	Private					Reserved
0x0000000000208000	8 KB	Private	Read, Write	Read/write			Commited
0x000000000020A000	16 KB	Private					Reserved
0x000000000020E000	8 KB	Private	Read, Write	Read/write		Swapped: 1	Commited
0x0000000000210000	8 KB	Private					Reserved

The above image shows the pages display filtered by Private memory, showing memory that is in the working set and memory that is not in the working set.

Working Set

Filtering by working set filters memory for the following working set criteria:

- **All.** Displays all memory.
- **Working Set**. Only displays memory that is in the working set.
- **Not Working Set**. Only displays memory that is not in the working set.


Type: All Working set: Working Set

Address	Size	Type	Protect	Working Set	Shared	Swap	Description
0x0000000000010000	4 KB	Mapped	Read Only	Read-only	Shared: 7		Memory Mapped File
0x0000000000020000	4 KB	Mapped	Read Only	Read-only	Shared: 7		Memory Mapped File
0x0000000000030000	116 KB	Mapped	Read Only	Read-only	Shared: 7		Memory Mapped File
0x000000000010E000	264 KB	Private	Read, Write	Read/write			Thread Stack: Thread: 11340 M
0x0000000000150000	16 KB	Mapped	Read Only	Read-only	Shared: 7		Memory Mapped File
0x0000000000160000	4 KB	Mapped	Read Only	Read-only	Shared: 7		Memory Mapped File
0x0000000000170000	8 KB	Private	Read, Write	Read/write			Commited
0x0000000000180000	4 KB	Mapped	Read Only	Read-only	Shared: 7		Memory Mapped File
0x0000000000190000	64 KB	Mapped	Read, Write	Read/write	Shared: 7		Memory Mapped File
0x00000000001A0000	16 KB	Private	Read, Write	Read/write			Commited
0x00000000001E0000	4 KB	Mapped	Read Only	Read-only	Shared: 7		Memory Mapped File
0x00000000001F0000	4 KB	Mapped	Read Only	Read-only	Shared: 7		Memory Mapped File
0x0000000000200000	24 KB	Private	Read, Write	Read/write			Commited
0x0000000000208000	8 KB	Private	Read, Write	Read/write			Commited

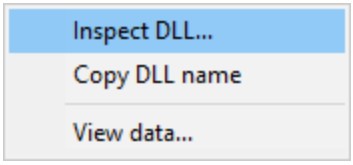
The above image shows the pages display showing memory that is in the working set.

Sorting

Each column can be sorted by clicking the column header to move sorting to that column. Clicking the current column header again reverses the sort direction.

Context menu

A context menu is available to allow DLL inspection using PE File Browser, and to copy the DLL name to clipboard.



Viewing swap behaviour

To view virtual memory being swapped select the **Swap** column so that the sort arrow points downwards.

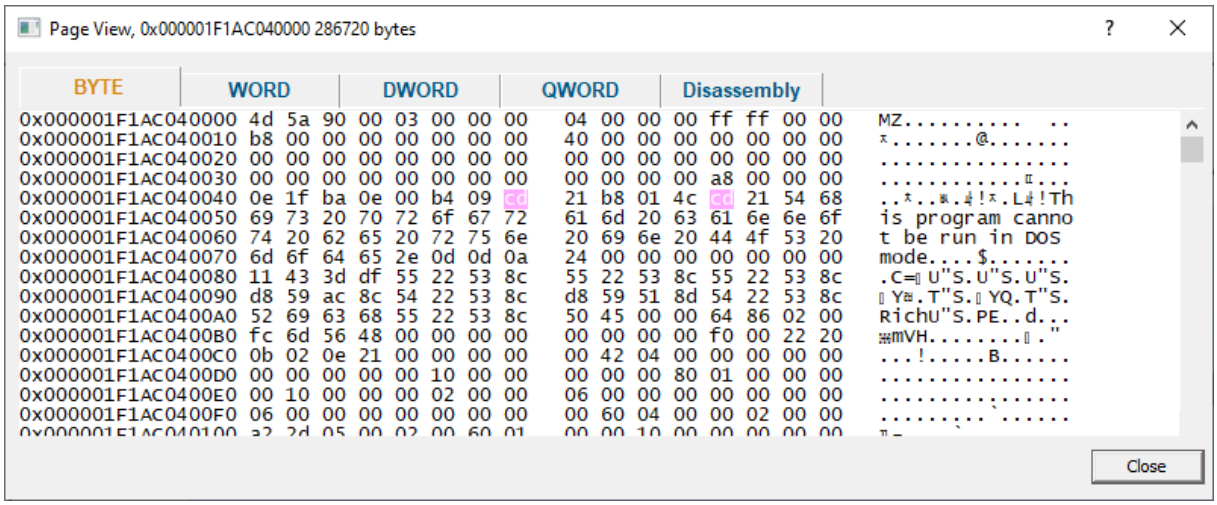
The display will show pages that have been swapped with the number of times they have been swapped shown in the Swap column.

When a page gets swapped it will be promoted at the top of the swap column for 1 second then be displayed back in it's regular sorted position.

When combined with the **highlight swapped memory** option on the Display Settings dialog you can very easily see pages being swapped.

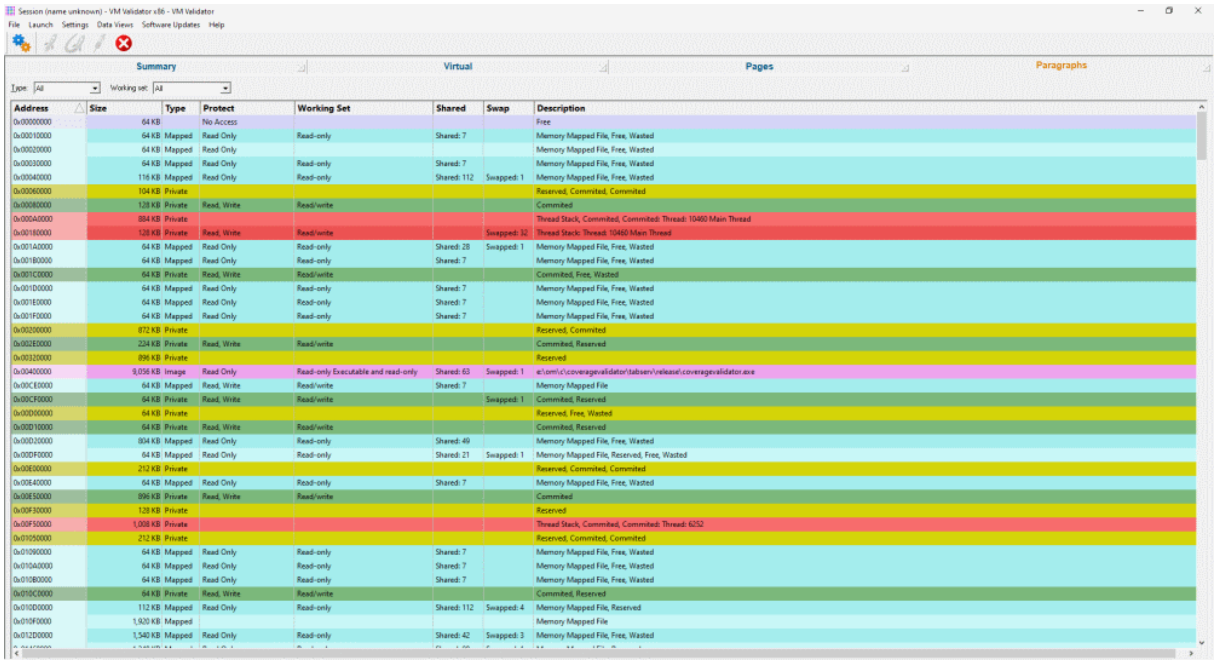
View data

Clicking **View data...** opens a memory inspection dialog, allowing you to view the memory as BYTEs, WORDs, DWORDs or QWORDs.



2.3.4 Paragraphs

The **Paragraphs** tab displays a tabular representation of all the memory in the target program. This is known as the paragraphs view.



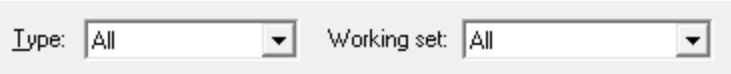
The paragraphs view displays information about a process's virtual memory usage whilst the process is running. If there is no target program being monitored, the paragraphs view does not display any data.

For memory paragraphs where more than one memory status (or swap count etc) is present for the pages in the memory paragraph the display will list the aggregate values (if two pages have been swapped a total of 33 times (18 and 15 times each) the display will show a swap count of 33).

The data displayed is data per virtual memory paragraph (as defined by the operating system dwAllocationGranularity returned from Win32 API call **GetSystemInfo()**).
A memory paragraph is 64Kb.

Filtering

The displayed data can be filtered by Type and by Working Set.



Type

Filtering by type filters memory for the following types of memory:

- **All.** All types of memory.
- **Committed**. Committed memory.
- **Reserved**. Reserved memory.
- **Free**. Free memory.
- **Private**. Memory private to this application.
- **Image**. Executable files (EXE, DLL, etc).
- **Mapped**. Memory mapped files.
- **Stack**. Memory used for thread stacks.
- **Shareable**. Memory shared with other applications.
- **Swapped**. Memory swapped in/out of memory.
- **Wasted**. Memory allocated by VirtualAlloc() but not available to the application to use. What is wasted memory?

Working Set

Filtering by working set filters memory for the following working set criteria:

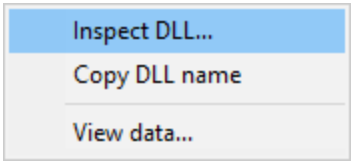
- **All.** Displays all memory.
- **Working Set**. Only displays memory that is in the working set.
- **Not Working Set**. Only displays memory that is not in the working set.

Sorting

Each column can be sorted by clicking the column header to move sorting to that column. Clicking the current column header again reverses the sort direction.

Context menu

A context menu is available to allow DLL inspection using PE File Browser, and to copy the DLL name to clipboard.



Viewing swap behaviour

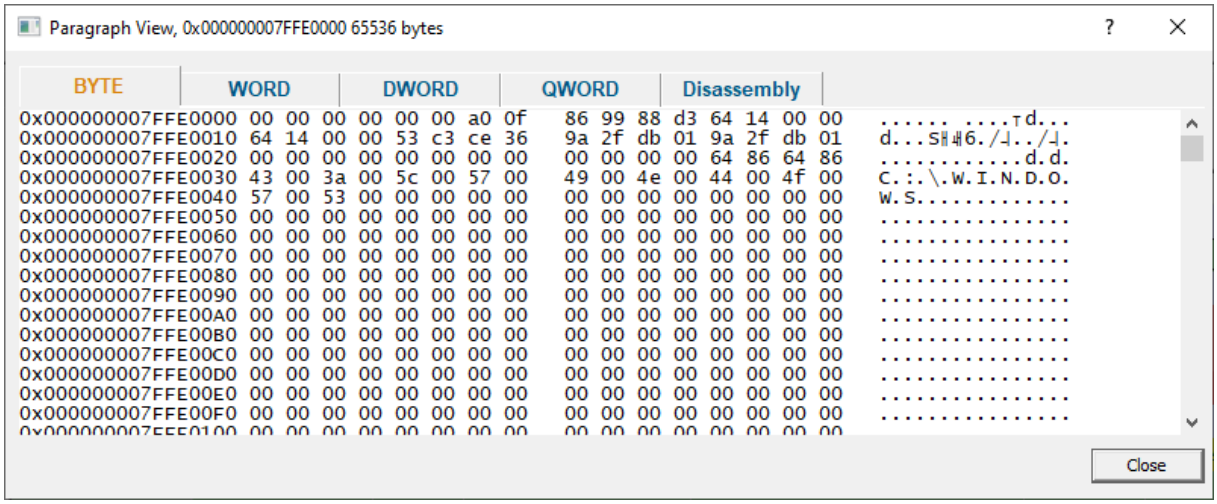
To view virtual memory being swapped select the **Swap** column so that the sort arrow points downwards.

The display will show pages that have been swapped with the number of times they have been swapped shown in the Swap column.

When a page gets swapped it will be promoted at the top of the swap column for 1 second then be displayed back in it's regular sorted position.
When combined with the **highlight swapped memory** option on the Display Settings dialog you can very easily see pages being swapped.

View data

Clicking **View data...** opens a memory inspection dialog, allowing you to view the memory as BYTEs, WORDs, DWORDs or QWORDs.



2.4 Load Session

Load Session allows you to load a previously saved virtual memory session that was saved using Save Session.

To load a session, choose **File** menu › **Load Session...**

A Microsoft file browser dialog is shown. Choose the session you wish to load.

When the session is loaded the displays will show the virtual memory information, dll names, thread names, page fault information that were in the session when it was saved.

File extensions

32 bit session files have extension vm.
64 bit session files have extension vm_x64.

2.5 Save Session

Save Session allows you to save the current virtual memory information so that you can inspect it in the future or send it to a colleague for inspection.

To save a session, choose **File** menu › **Save Session...**

A Microsoft file browser dialog is shown. Choose where you'd like to save the session.

When the session is saved all the virtual memory information, dll names, thread names, page fault information that are in the session when are saved to the file.

You can keep this data for future reference, provide to a colleague for examination or send to a vendor to help debugging a product.

File extensions

32 bit session files have extension vm.
64 bit session files have extension vm_x64.

2.6 Load Minidump

Load Minidump allows you to load a previously saved minidump that was saved using Save Minidump, or a minidump from any other source.

To load a minidump, choose **File** menu › **Load Minidump...**

A Microsoft file browser dialog is shown. Choose the minidump you wish to load.

When the minidump is loaded the displays will show the virtual memory information, dll names, thread names, that were in the minidump when it was created.

Page Faults

Minidumps don't contain any page fault related information - the page fault display will be empty when a minidump is loaded.

2.7 Save Minidump

Save Minidump allows you to save a minidump based on the process being viewed by VM Validator.

To save a minidump, choose **File** menu › **Save Minidump...**

A Microsoft file browser dialog is shown. Choose where you'd like to save the minidump.

You can keep this minidump for future reference, provide to a colleague for examination (with Minidump Browser or a debugger) or send to a vendor to help debugging a product.

2.8 Settings

VM Validator settings allow you to control the colours of displayed data and units used to represent memory usage, and how symbols are fetched.

Settings are grouped into two sections, colours and symbols.

Opening the settings dialog

To view the settings dialog, choose **File** menu › **Settings...**

Or use the option on the Session Toolbar:

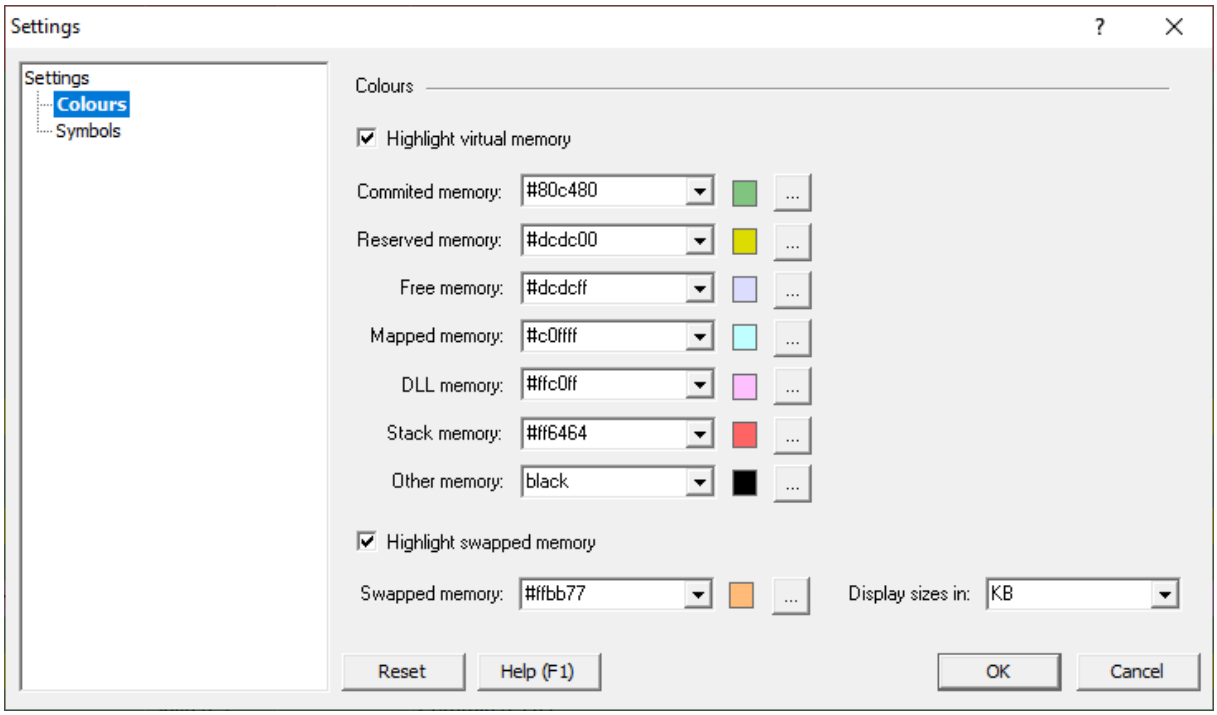
Keyboard shortcuts:

F6

Using the settings dialog

The dialog has a scrolled list on the left hand side, grouping the topics. When a topic is clicked, its related controls are displayed on the right hand side.

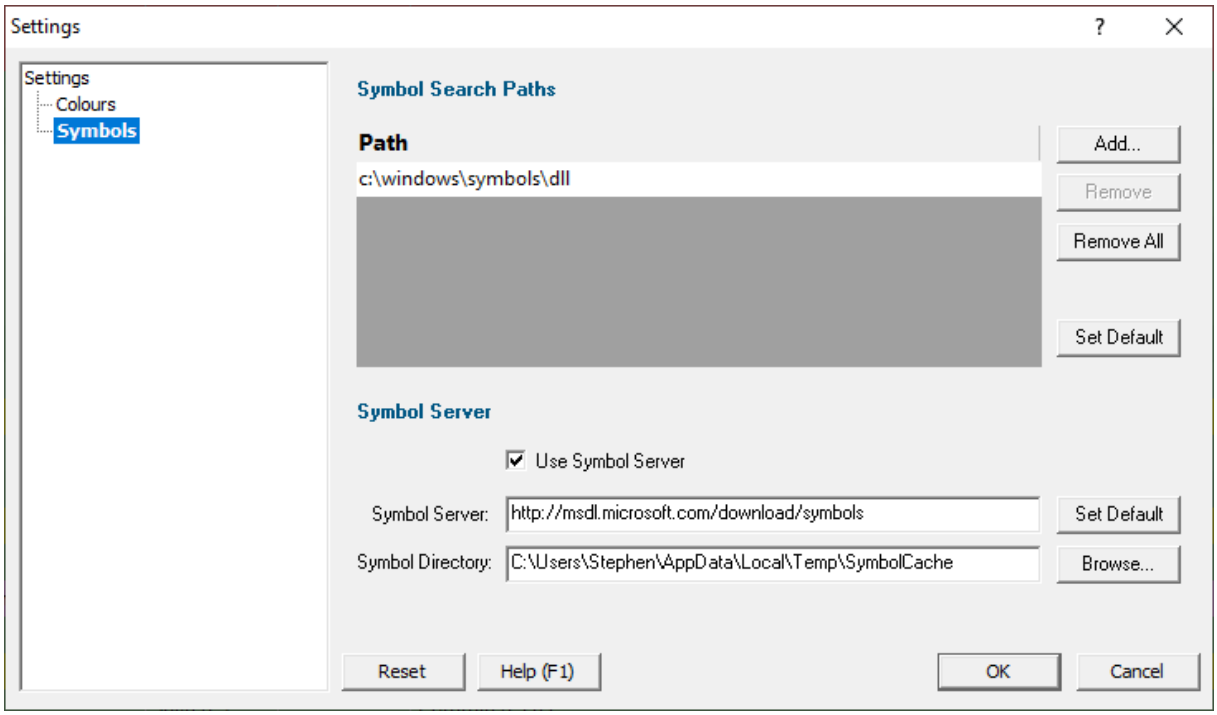
The default display of the dialog is shown below with the first topic selected.



2.8.1 Symbols

The **Symbols** tab lets you specify the paths used to find symbols for the application you are examining.

The defaults are setup to pull symbols from Microsoft's symbol servers.



Symbol Search Paths

We use debug information found in PDB files to turn addresses into human readable symbols, filenames and lines.

By default there is one path in the symbol search paths: **c:\windows\symbols\dll**.

If you wish to add more paths to symbols you can add or remove them using the **Add...**, **Remove** and **Remove All** buttons.

You can restore the default symbol paths using the **Set Default** button.

Symbol Server

If the symbol server is enabled (**Use Symbol Server**) Page Fault Monitor will download symbols using a symbol server.

Symbols are downloaded to a **Symbol Directory**. This directory must be valid for the symbol server to work. You can type the directory name or click **Browse...** to use the Microsoft folder browser to select the directory.

Microsoft DLLs

For Microsoft DLLs that are found in the Windows System32 directory we download symbols from Microsoft's symbol server:

```
http://msdl.microsoft.com/download/symbols
```

None Microsoft DLLs

For none Microsoft DLLs we download symbols from the symbol server specified on the settings dialog in the **Symbol Server** field.

You can set the symbol server to it's default value using the **Set Default** button.

Please note that if you have symbol servers enabled there may be a delay in providing symbol information the first symbols for a specific DLL are downloaded from the symbol server.

Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

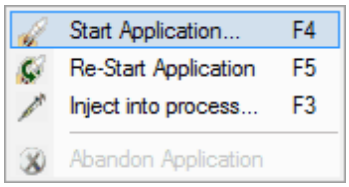
2.9 Starting your target program

VM Validator provides a choice of 2 methods to start a target program and have VM Validator collect data from the target program about the program's execution.

- You can start a program in a specified directory and with as many command line arguments as you want.
- You can monitor an already running program.

2.9.1 Launching the program

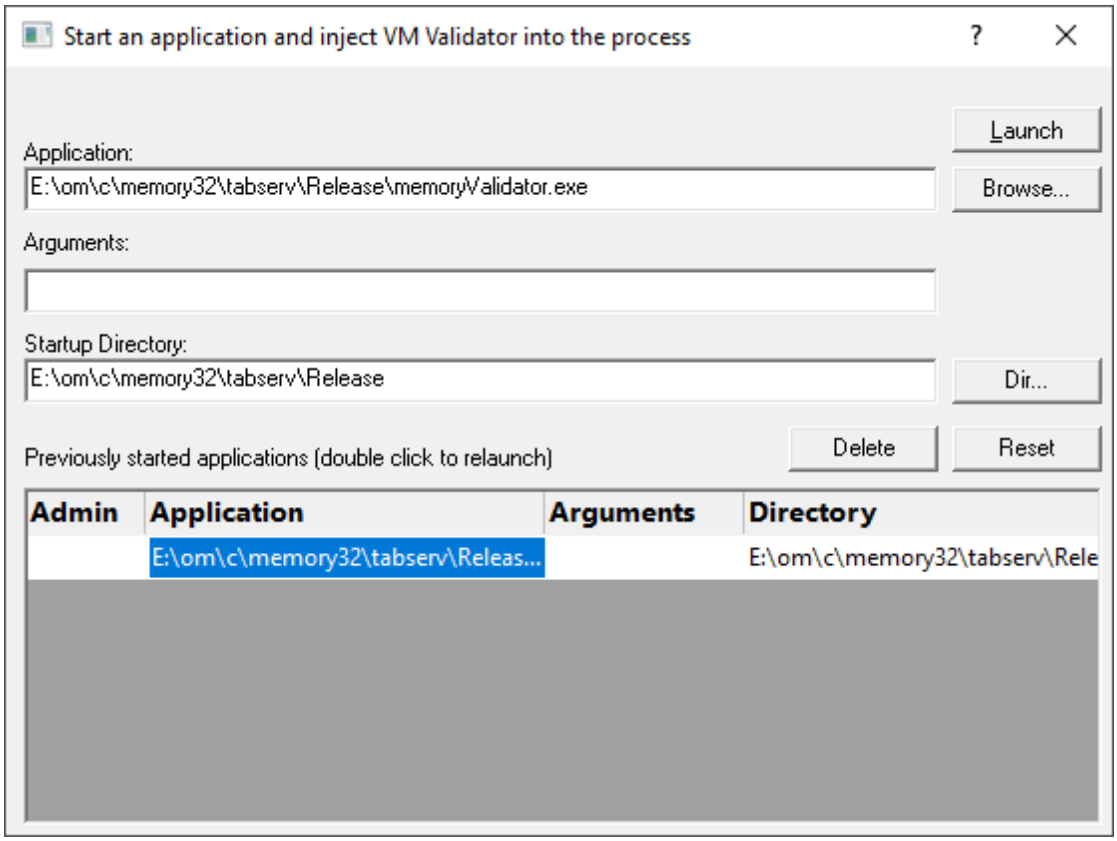
To start your program choose **Start Application...** on the **Launch** menu,



or click on the launch icon on the session toolbar.

Keyboard shortcuts:

The launch program dialog will be displayed.



The picture shows some already launched programs shown in the scrollable list. You can relaunch these programs by double clicking their entry in the list. Depending on what you want to do with your program you have a choice of launching the program.

Detailed program start (1)

1) Type in the program name in the **Application** field, or click the **Browse...** button to use a file browser to choose the program to launch.
2) Type in the program arguments in the **Arguments** field.
3) Type in the program directory in the **Startup Directory** field, or click the **Dir...** button to use a directory browser to choose the directory.
4) Click on the **Go!** button to launch the program.

Detailed program start (2)

1) Click on the program shown in the scrolled list that you want to start. The details for the program will appear in the **Application**, **Arguments** and **Startup Directory** fields.
2) Modify any of the **Application**, **Arguments** and **Startup Directory** fields as appropriate.
3) Click on the **Go!** button to launch the program.

Repeat program start

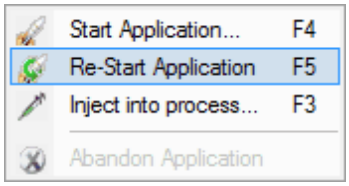
Double click on the program shown in the scrolled list. The program will be launched automatically.

VM Validator will start the program and inject the stub into the program. A progress dialog will be displayed whilst the stub is being injected into the program. The progress dialog lets you know what task it is performing during the injection sequence. When the stub is correctly installed in the target program the stub will establish communications with VM Validator.

If at any point you would like the fields reset, click on the **Reset** button.

2.9.2 Re-Launching the program

To re-start your program choose **Re-Start Application...** on the **Launch** menu,



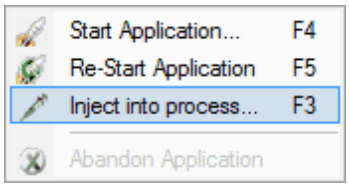
or click on the relaunch icon on the session toolbar.

Keyboard shortcuts:

The application that was most recently launched is started.

2.9.3 Injecting into a running program

To inject the stub into an already running program choose **Inject...** on the **Launch** menu,

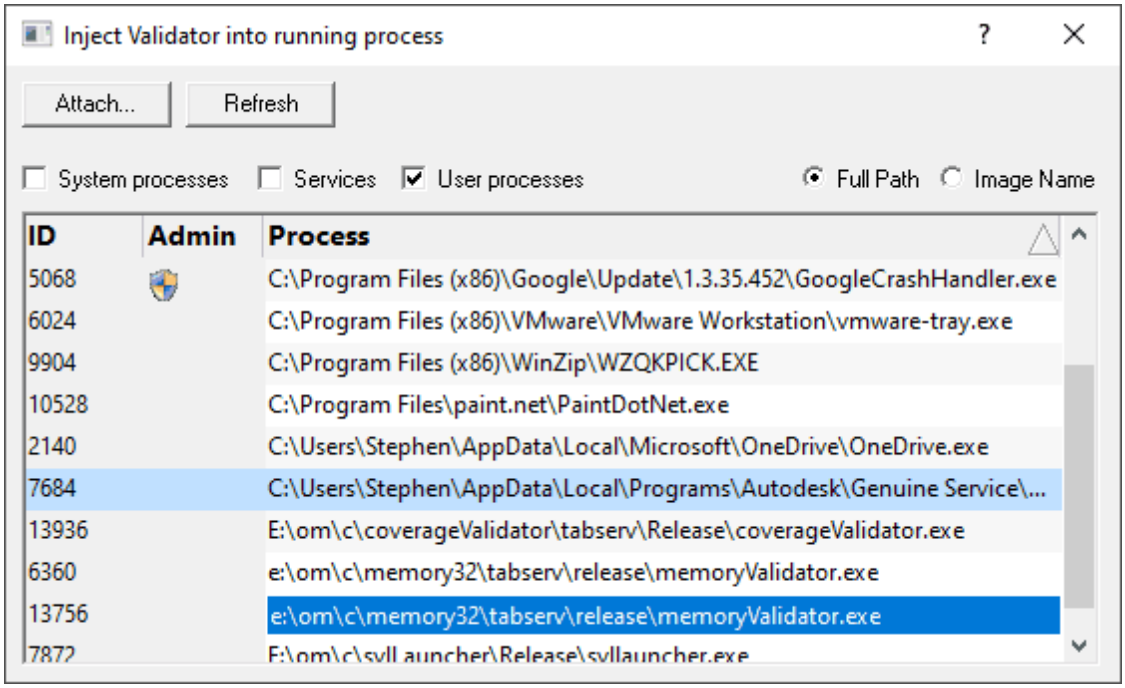


or click on the inject icon on the session toolbar.

Keyboard shortcuts:

F3

The inject program dialog will be displayed.



Injecting into running process

1) Select the process to attach to.
2) Click on the **Attach...** button.

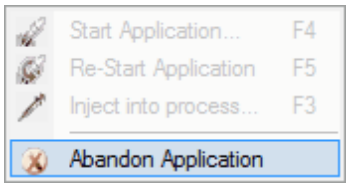
VM Validator will start the program and inject the stub into the program. A progress dialog will be displayed whilst the stub is being injected into the program. The progress dialog lets you know what task it is performing during the injection sequence. When the stub is correctly installed in the target program the stub will establish communications with VM Validator.

You can choose if you want to view system and/or user processes. If you need to refresh the list of active processes, click the **Refresh** button.

2.10 Stopping your target program

Sometimes you will start a run of your target program and realise that you made a mistake, forgot to do something or need a different option enabled and need to restart the test from the beginning. You can stop the target program using Task Manager, using the debugger, or using VM Validator.

VM Validator provides you with a simple way of stopping the target program. Choose **Abandon Application** on the **Launch** Menu,



or click on the large red X on the session toolbar.

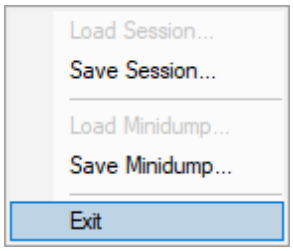
The target program is terminated using TerminateProcess(). When VM Validator has ended the target program, there will be no data to display.

Using VM Validator to stop the target program is usually quicker than using Task Manager or the debugger because the session is thrown away for you, rather than you having to terminate the current session.

Please note that if you wish your program to end in a controlled manner you should close your program yourself and should no use the **Abandon Application** option in VM Validator.

2.11 Closing VM Validator

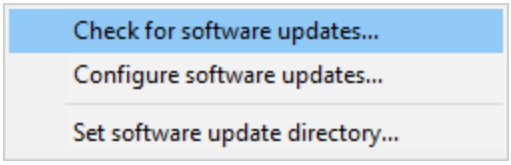
To finish working with VM Validator, choose **Exit** on the **File** menu.



2.12 Software Updates

The Software Updates menu controls how often software updates are downloaded.

If you've been notified of a new software release to VM Validator or just want to see if there's a new version, this feature makes it easy to update.

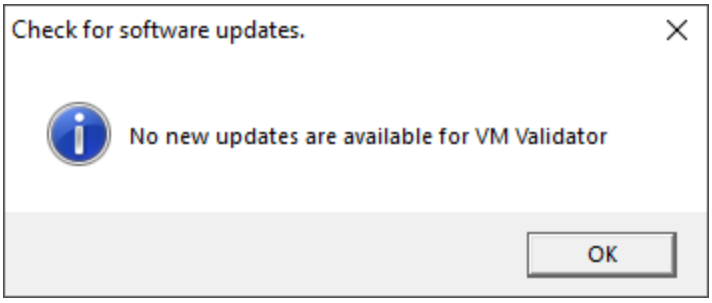


Software Updates menu › **Check for software updates** › checks for updates and shows the software update dialog if any exist

An internet connection is needed to be able to make contact with our servers.

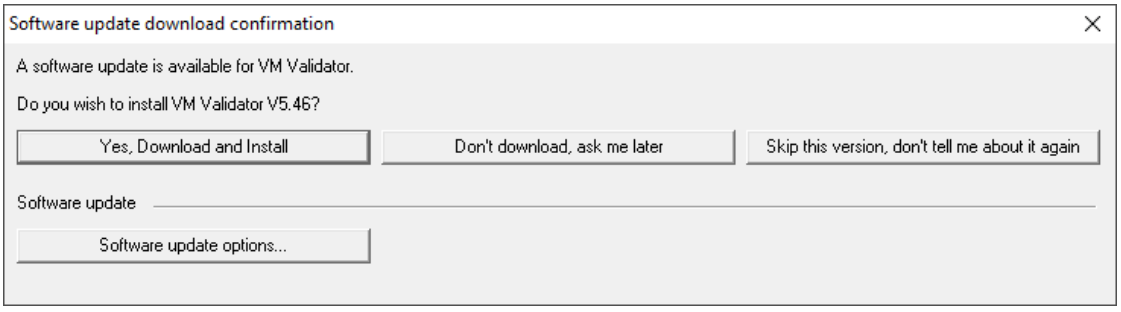
Before updating the software, close the help manual, and end any active session by closing target programs.

If no updates are available, you'll just see this message:

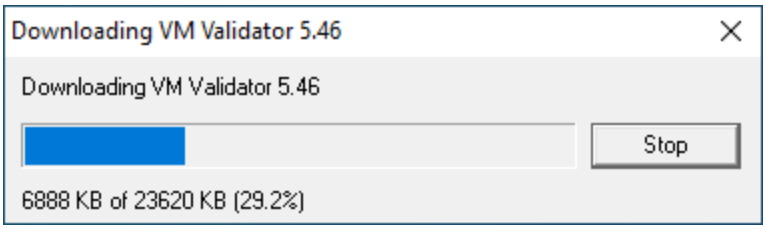


Software Update dialog

If a software update is available for VM Validator you'll see the software update dialog.



- **Download and install** › downloads the update, showing progress



Once the update has downloaded, VM Validator will close, run the installer, and restart.

You can stop the download at any time, if necessary.

- **Don't download...** › Doesn't download, but you'll be prompted for it again next time you start VM Validator
- **Skip this version...** › Doesn't download the update and doesn't bother you again until there's an even newer update
- **Software update options...** › edit the software update schedule

Problems downloading or installing?

If for whatever reason, automatic download and installation fails to complete:

- Download the latest installer manually from the software verify website.

Make some checks for possible scenarios where files may be locked by VM Validator as follows:

- Ensure VM Validator and its help manual is also closed
- Ensure any error dialogs from the previous installation are closed

You should now be ready to run the new version.

Software update schedule

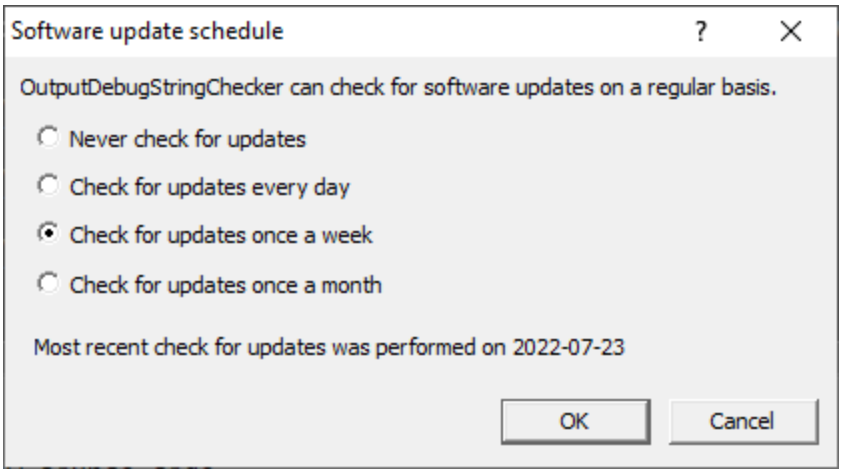
VM Validator can automatically check to see if a new version of VM Validator is available for downloading.

Software Updates menu › **Configure software updates** › shows the software update schedule dialog

The update options are:

- never check for updates
- check daily (the default)
- check weekly
- check monthly

The most recent check for updates is shown at the bottom.

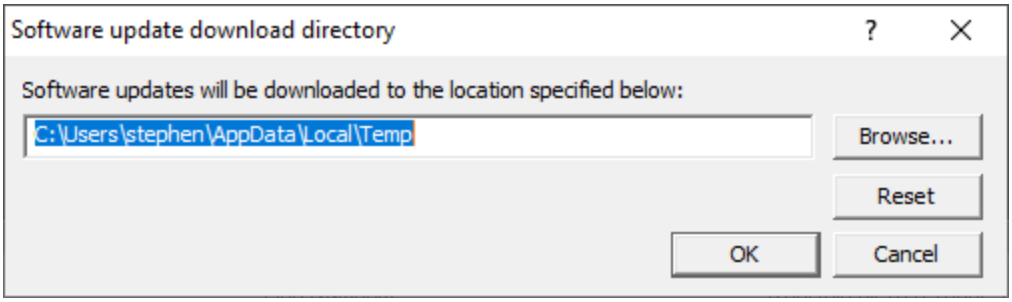


Software update directory

It's important to be able to specify where software updates are downloaded to because of potential security risks that may arise from allowing the `TMP` directory to be executable. For example, to counteract security threats it's possible that account ownership permissions or antivirus software blocks program execution directly from the `TMP` directory.

The `TMP` directory is the default location but if for whatever reason you're not comfortable with that, you can specify your preferred download directory. This allows you to set permissions for `TMP` to deny execute privileges if you wish.

Software Updates menu › **Set software update directory** › shows the Software update download directory dialog



An invalid directory will show the path in red and will not be accepted until a valid folder is entered.

Example reasons for invalid directories include:

- the directory doesn't exist
- the directory doesn't have write privilege (update can't be downloaded)
- the directory doesn't have execute privilege (downloaded update can't be run)

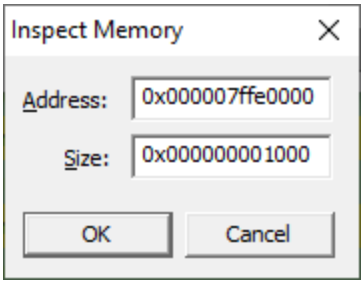
When modifying the download directory, you should ensure the directory will continue to be valid. Updates may no longer occur if the download location is later invalidated.

- **Reset** › reverts the download location to the user's `TMP` directory

  The default location is `c:\users\[username]\AppData\Local\Temp`

2.13 View Memory Dialog

The Inspect Memory dialog is shown below.



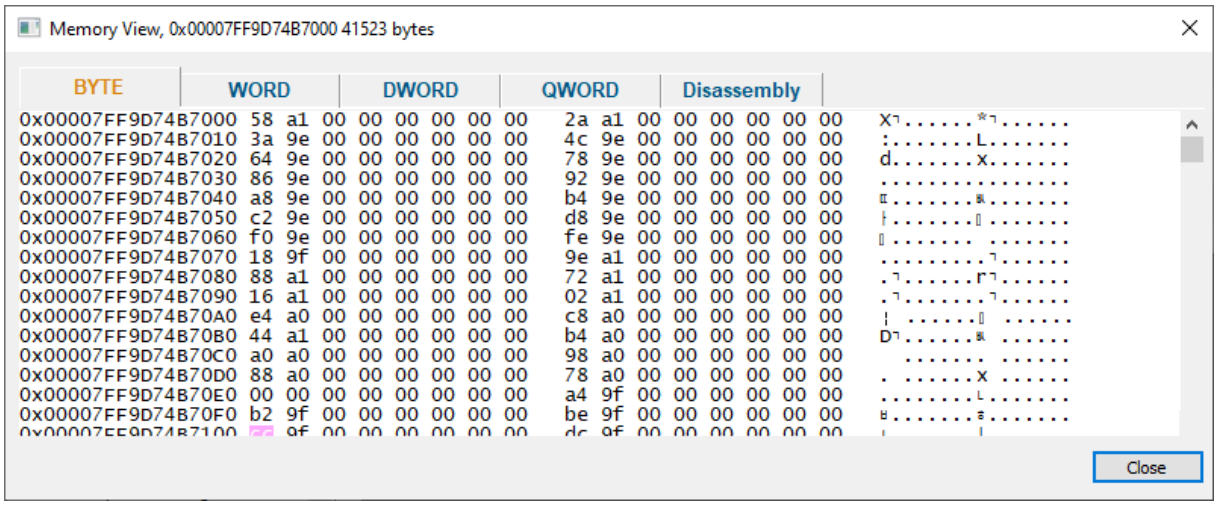
Address › an address inside the application to start viewing memory

Size › how many bytes to view

Specify an address inside the application and the number of bytes to view.

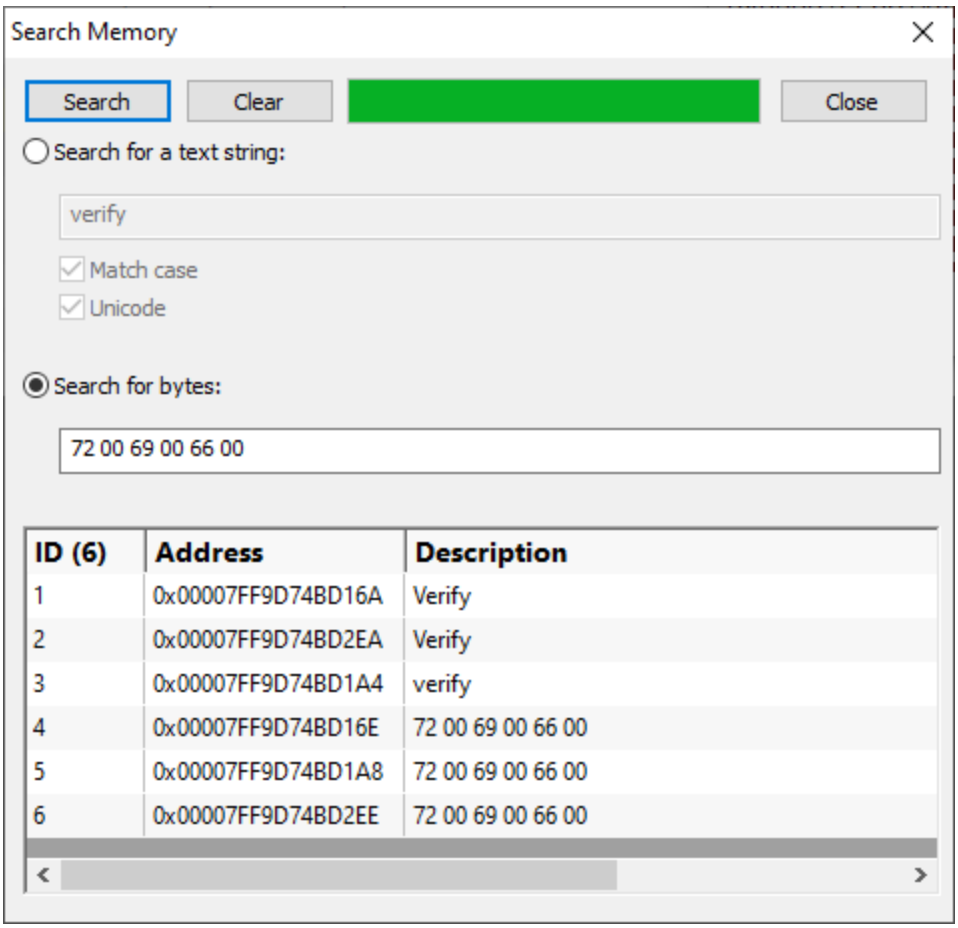
If the memory is readable, the memory will be displayed.

If the memory is executable, a disassembly view is also availble.



2.14 Search Memory Dialog

The Search Memory dialog is shown below.



You can search for text strings or you can search for byte sequences.

Search for a text string › type the string you wish to search for into the text box

Match case › select the check box if the string match should be case sensitive

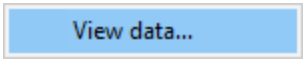
Unicode › select the check box if the string match should be Unicode. If the check box is not selected the string match is ANSI

Search for bytes › type the bytes you wish to search for into the text box. A byte should be specified as a two digit hex value. Separate bytes with spaces

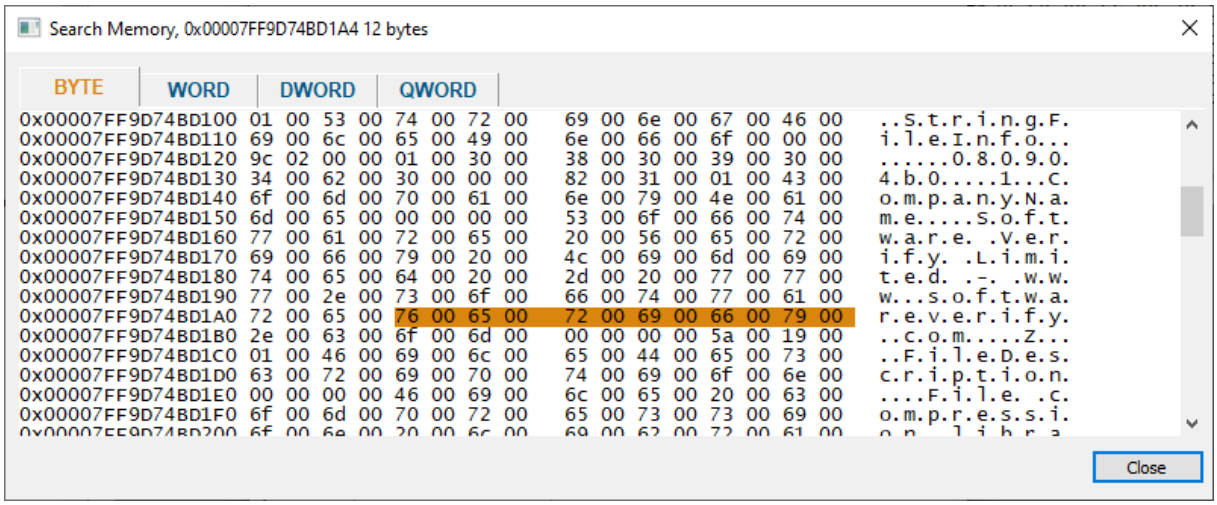
Search › perform the search. The progress of the search is shown on the progress bar, any matching search results are shown in the list.

Clear › clear the search results

A context menu on the search results provides a single option:



Clicking **View data...** opens a memory inspection dialog, allowing you to view the search results memory as BYTEs, WORDs, DWORDs or QWORDs.



2.15 Help

VM Validator provides a help system to help you get the most from VM Validator.

2.15.1 Tip of the day

The tip of the day dialog provides helpful tips on how to get the most from VM Validator.

2.15.2 About VM Validator

Information about this version of VM Validator, copyright notices and user information.

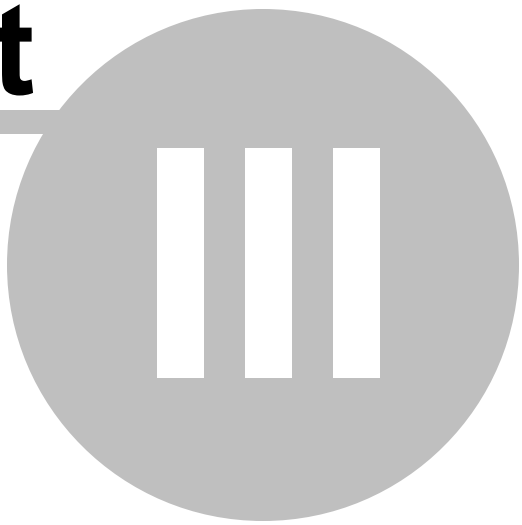
2.15.3 Help Topics

Display the online help for VM Validator.

PDF Help

PDF versions of the help file are available from https://www.softwareverify.com/helpPDFs.html.

Part



3 Command Line Interface

VM Validator can be used from the command line as well as with the GUI.

The command line options allow you to launch an application that VM Validator will monitor, or to monitor an already running process.

Use in your own debugging work

One usage scenario which we have at Software Verify is to launch VM Validator to monitor a worker process we've just launched.

The **/process processId** option is very useful for this.

3.1 Alphabetic Reference

/allArgs

Specifies the arguments passed to the process being started. All remaining arguments on the command line are consumed by this command.

/args arguments to pass to process.

Example: /allArgs weebles wobble but they don't fall down

/arg

Specifies an argument to be passed to the process being started

/arg argument to pass to process. Enclose in quotes if any whitespace.

Examples:

/arg webbles
/arg "webbles wobble"
/arg webbles /arg wobble /arg but /arg "they don't fall down"

/dir

Specifies the startup directory for the process being started

/dir directory. Enclose in quotes if any whitespace.

Examples:

/dir e:\om\c\
/dir "e:\om\c path with spaces\"

/process

Specifies the process that will be monitored

/process processId

Example: /process 1344

/minidump

Specifies the minidump that will be loaded

/minidump path-to-minidump. Enclose in quotes if any whitespace.

Examples:

/minidump e:\om\c\test\release\test.dmp
/minidump "e:\om\c\test path with spaces\release\test.dmp"

/program

Specifies the executable that will be started

/program path-to-executable. Enclose in quotes if any whitespace.

Examples:

/program e:\om\c\test\release\test.exe
/program "e:\om\c\test path with spaces\release\test.exe"

/session

Specifies the saved session that will be loaded. The session file was previously saved using Save Session.

/session path-to-session-file. Enclose in quotes if any whitespace.

Examples:

/session e:\om\c\test\release\test.vm
/session "e:\om\c\test path with spaces\release\test.vm"

File extensions

32 bit session files have extension vm.
64 bit session files have extension vm_x64.

3.2 Usage Reference

Launch Application

Launch an application that will be monitored by VM Validator.

/allArgs

Specifies the arguments passed to the process being started. All remaining arguments on the command line are consumed by this command.

/args arguments to pass to process.

Example: /allArgs weebles wobble but they don't fall down

/arg

Specifies an argument to be passed to the process being started

/arg argument to pass to process. Enclose in quotes if any whitespace.

Examples:
/arg webbles
/arg "webbles wobble"
/arg webbles /arg wobble /arg but /arg "they don't fall down"

/dir
Specifies the startup directory for the process being started

/dir directory. Enclose in quotes if any whitespace.

Examples:
/dir e:\om\c\
/dir "e:\om\c path with spaces\"

/program
Specifies the executable that will be started

/program path-to-executable. Enclose in quotes if any whitespace.

Examples:
/program e:\om\c\test\release\test.exe
/program "e:\om\c\test path with spaces\release\test.exe"

Monitor Application

Monitor an application that is already running.

/process
Specifies the process that will be monitored

/process processId

Example: /process 1344

Load a saved session

Load a session saved by VM Validator.

/session
Specifies the saved session that will be loaded. The session file was previously saved using Save Session.

/session path-to-session-file. Enclose in quotes if any whitespace.

Examples:
/session e:\om\c\test\release\test.vm
/session "e:\om\c\test path with spaces\release\test.vm"

File extensions

32 bit session files have extension vm.

64 bit session files have extension vm_x64.

Load a minidump

Load a minidump saved by VM Validator, Exception Tracer, Task Manager, Visual Studio.

/minidump
Specifies the minidump that will be loaded

/minidump path-to-minidump. Enclose in quotes if any whitespace.

Examples:
/minidump e:\om\c\test\release\test.dmp
/minidump "e:\om\c\test path with spaces\release\test.dmp"

3.3 Command Line Examples

This section provides some example command lines. For each example we'll break it down, argument by argument so that you can see how the command line works.

In all these examples the executable to run is **vmValidator.exe**. You will need to provide the full path to vmValidator.exe, or add the path to vmValidator install directory to your $PATH.

32 bit application monitoring: **vmValidator.exe**
64 bit application monitoring: **vmValidator_x64.exe**

Example 1

vmValidator.exe /exe e:\om\c\test\release\test.exe /dir e:\om\c\test\release /args "-add 3 4"

/program e:\om\c\test\release\test.exe

Start application e:\om\c\test\release\test.exe

/dir e:\om\c\test\release

Start the application in e:\om\c\test\release

/args "-add 3 4"

Pass the arguments -add 3 4 to the application being launched

Example 2

vmValidator.exe /process 1344

/process 1344

Attach to process 1344 and monitor that process.

Example 3

Launching VM Validator from your own code to monitor a process you've just started. The arguments are the same as for Example 2.

```
int attachVMValidatorToRunningProcess(DWORD         processId,          // process id to monitor
                                      const TCHAR   *dir)               // startup dir, can be NUI
{
   CString      vmValidator;
   int          bRet;

   vmValidator = getVMValidatorPath(dir);
   if (GetFileAttributes(vmValidator) != INVALID_FILE_ATTRIBUTES)
   {
      // only try to launch if exception tracer is a valid filename

      TCHAR               commandLine[1000];
      STARTUPINFO         stStartInfo;
      PROCESS_INFORMATION stProcessInfo;

      memset(&stStartInfo, 0, sizeof(STARTUPINFO));
      memset(&stProcessInfo, 0, sizeof(PROCESS_INFORMATION));

      stStartInfo.cb = sizeof(STARTUPINFO);

      _stprintf_s(commandLine, sizeof(commandLine) / sizeof(commandLine[0]), _T("%s /process %u"),

      bRet = CreateProcess(NULL,
                           commandLine,
                           NULL,
                           NULL,
                           FALSE,
                           0,
                           NULL,
                           dir,
                           &stStartInfo,
                           &stProcessInfo);
      if (bRet)
      {
         WaitForInputIdle(stProcessInfo.hProcess, 10 * 1000);   // 10 seconds

         CloseHandle(stProcessInfo.hProcess);
         CloseHandle(stProcessInfo.hThread);
      }
   }
   else
      bRet = FALSE;

   return bRet;
}
```

Example 4

vmValidator.exe /session e:\crashes\2022_10_05\osc1.vvm

/session e:\crashes\2022_10_05\osc1.vvm

Load session file e:\crashes\2022_10_05\osc1.vm

Example 5

vmValidator.exe /minidump e:\crashes\2022_10_05\crash34.dmp

/minidump e:\crashes\2022_10_05\crash34.dmp

Load minidump e:\crashes\2022_10_05\crash34.dmp

Part

4 Frequently Asked Questions

This section lists the commonly asked questions about VM Validator.

4.1 I have an idea for a feature, can it be added to VM Validator?

We have tried to add as many features to VM Validator that we thought would be useful to the potential users of VM Validator. In fact every feature in VM Validator has been used to solve problems and bugs on consulting work carried out for our customers and on internal projects at Software Verify Limited. We know the features we have put in the product are useful.

However it is possible we have overlooked a feature that you may find very useful. We will be pleased to consider all ideas for new features to VM Validator. Please contact us at the address provided on the contact page.

A Quake III mode, however much fun, is not useful in solving bugs. Sorry guys!

4.2 What file extensions does VM Validator use for itself?

VM Validator stores all of the configuration data it needs in the registry.

VM Validator does not use an specific file extensions.

4.3 What is Wasted Memory?

Wasted memory is a term we use to identify memory that is wasted when memory allocations are made with VirtualAlloc() that are smaller than the Windows operating system allocation granularity (which is typically 64KB).

This is best explained with an example;

```
ptr = VirtualAlloc(NULL, 24 * 1024, MEM_COMMIT, PAGE_READWRITE);
```

For example if you use VirtualAlloc() to commit 24KB of memory, VirtualAlloc() will set aside 64KB of memory, and then commit 24KB of that 64KB memory. There will be two chunks of memory as a result:

1	24KB	Committed	PAGE_READWRITE
2	40KB	Free	PAGE_NO_ACCESS

The first block is pointed to by the pointer returned from VirtualAlloc().

The second block isn't pointed to by anything. You can calculate where it is if you know about the allocation and the size of the allocation.

But in most cases, for this type of allocation, the second block is there implicitly - the caller probably doesn't realise they've used 64KB of memory to make that 24KB allocation.

How to prevent wasted memory?

The best way to prevent wasted memory is to use a different allocator (HeapAlloc, malloc, new, etc) to allocate the memory you need, and if you need to change the memory protection on that memory then call VirtualProtect() to change the memory protection on that memory (bear in mind this works on 4KB blocks at a minimum, aligned on 4KB page boundaries - for x86, x64 machines).

Alternatively you would write a simple memory allocator that uses VirtualAlloc() and VirtualFree() as it's backing store. You allocator would allocate when necessary using VirtualAlloc() and then subdivide those blocks to pass back suitable locations inside the larger blocks, thus avoiding the memory wastage.

Index

- C -

Commit 15
CRT 15

- D -

Data Collection 26, 28
Detailed program start 26
Detailed program start (2) 26
DLL 15

- E -

Exit 30

- F -

Free 15
Frequently Asked Questions 45

- H -

Heap 15

- I -

inject 28
Injecting into running process 28

- K -

KB 15

- M -

Mapped 15
MB 15

- P -

page 15
Page Information 15
Pages 15

- Q -

Quick program start 26

- R -

Refresh 15
Region Information 15
Repeat program start 26
Reserve 15

- S -

Stack 15
Start 15
start a target program 26
stop 15, 30
Stub 15

- U -

Update Interval 15

- V -

Virtual 15
virtual memory 15