



# PE File Browser

by

Software Verify

Copyright © 2017-2026 Software Verify Limited

# PE File Browser

## PE File contents inspector

---

*by Software Verify Limited*

*Welcome to the PE File Browser software tool.*

*PE File Browser is a software tool that allows you to inspect the contents of PE files (.exe and .dll files).*

*We hope you will find this document useful.*

# DbgHelp Browser Help

**Copyright © 2015-2026 Software Verify Limited**

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: December 2025 in United Kingdom.

# Table of Contents

Foreword	1
<b>Part I How to get PE File Browser</b>	<b>2</b>
<b>Part II What does PE File Browser do?</b>	<b>4</b>
<b>Part III Menu</b>	<b>6</b>
1 File .....	7
2 Inspect .....	7
3 Software Updates .....	7
4 Help .....	11
<b>Part IV The user interface</b>	<b>13</b>
1 The data display .....	14
File Header .....	14
Optional Header .....	16
Optional Header Directories .....	19
Additional Information .....	21
.Net Information .....	22
.Net Header .....	24
.Net MetaData .....	25
Imported Modules and Functions .....	26
Delay Loaded Modules .....	26
All Dependent Modules .....	27
Imported Functions .....	28
Exported Functions .....	28
Sections / Segments .....	29
Data Bounds .....	32
FPO Data .....	33
x64 Exception Handling .....	35
Version Information .....	36
Manifest .....	37
Digital Signatures .....	37
Thread Local Storage .....	38
Resources .....	39
Debug Information .....	39
PDB Symbols .....	40
COFF Symbols .....	42
CodeView Symbols .....	43
STABS Symbols .....	45
Misc Debug Data .....	46
2 View Memory Dialog .....	46
3 Search Memory Dialog .....	47
<b>Part V Command Line Interface</b>	<b>50</b>

<b>1</b>	<b>Alphabetic Reference .....</b>	<b>51</b>
<b>2</b>	<b>Usage Reference .....</b>	<b>52</b>
	<b>Index</b>	<b>0</b>



**Part**



## 1 How to get PE File Browser

PE File Browser is free for commercial use. PE File Browser can be downloaded for Software Verify's website at <https://www.softwareverify.com/product/pe-file-browser/>.

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

Windows Help	<a href="https://www.softwareverify.com/documentation/chm/peFileBrowser.chm">https://www.softwareverify.com/documentation/chm/peFileBrowser.chm</a>
PDF	<a href="https://www.softwareverify.com/documentation/pdfs/peFileBrowser.pdf">https://www.softwareverify.com/documentation/pdfs/peFileBrowser.pdf</a>
Online	<a href="https://www.softwareverify.com/documentation/html/peFileBrowser/index.html">https://www.softwareverify.com/documentation/html/peFileBrowser/index.html</a>

Whilst PE File Browser is free for commercial use, PE File Browser 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

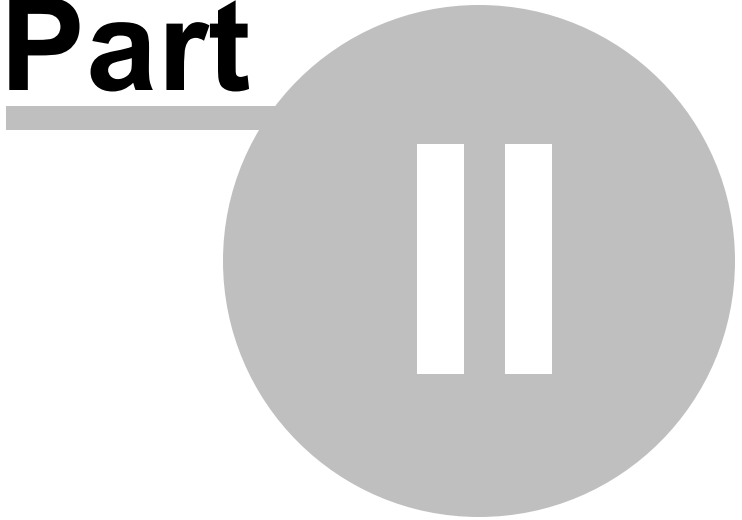
Contact Software Verify at:

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

email [sales@softwareverify.com](mailto:sales@softwareverify.com)  
web <https://www.softwareverify.com>  
blog <https://www.softwareverify.com/blog>  
twitter <http://twitter.com/softwareverify>

Visit our blog to read our articles on debugging techniques and tools.  
Follow us on twitter to keep track of the latest software tools and updates.

**Part**



## 2 What does PE File Browser do?

PE File Browser allows you to inspect the contents of a PE format file (.exe or .dll).

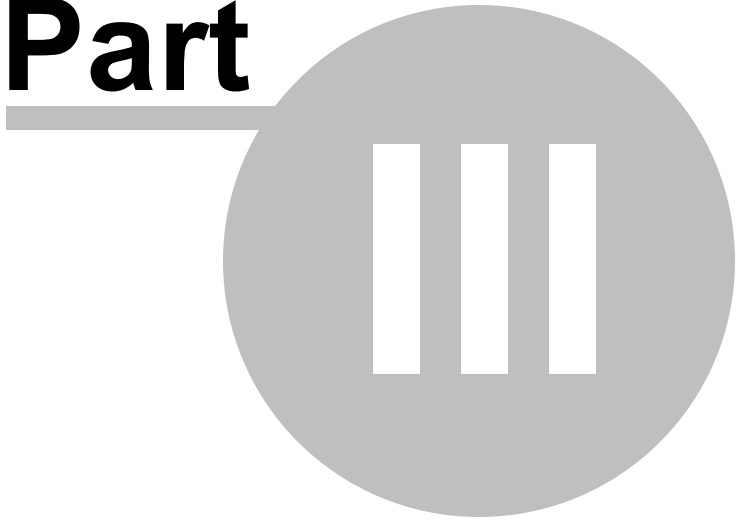
### 32 bit and 64 bit

PDB files created by 32 bit and 64 bit software are supported. On 64 bit Operating systems if a 64 bit PDB file is opened the 64 bit version PE File Browser is automatically started.

### History

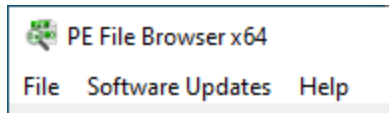
PE File Browser has been an internal tool at Software Verify for many years. We recently decided to make it a bit more user friendly and to make it available for public use.

**Part**



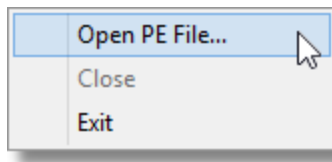
## 3 Menu

The main menu contains three menus, File, Software Updates and Help.



### 3.1 File

The File menu controls loading of PE files, clearing the display and exiting the program.



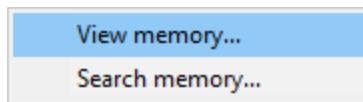
**File** menu > **Open PE File...** > loads a PE File (exe or dll) and displays information about the PE file.

**File** menu > **Close** > clear all results, unloads the PE file.

**File** menu > **Exit** > closes PE File Browser.

### 3.2 Inspect

The Inspect menu allows you to view arbitrary memory, or to search for memory.



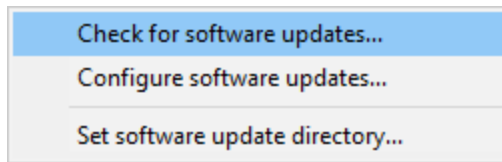
**Inspect** menu > **View memory...** > view memory at an arbitrary address. The View Memory Dialog is displayed.

**Inspect** menu > **Search memory...** > search for a text string or a sequence of bytes. The Search Memory Dialog is displayed.

### 3.3 Software Updates


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

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

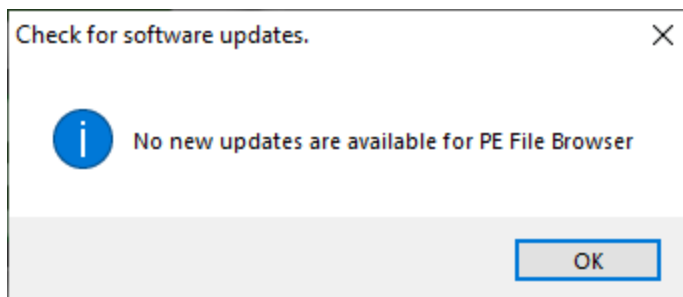


 **Software Updates** menu > **Check for software updates** > checks for updates and shows the software update dialog if any exist

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

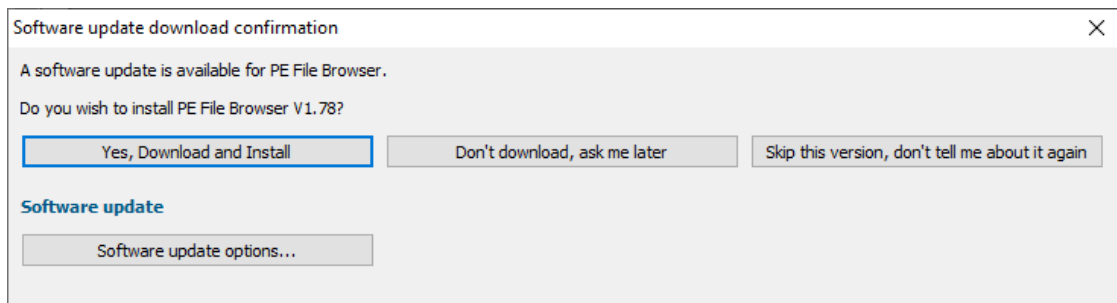
 Before updating the software, close the help manual, and end any active session by closing target programs.

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

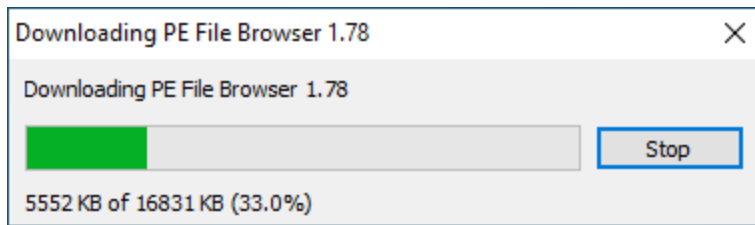


## Software Update dialog

If a software update is available for PE File Browser you'll see the software update dialog.



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



Once the update has downloaded, PE File Browser 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 PE File Browser
- **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 PE File Browser as follows:

- Ensure PE File Browser 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

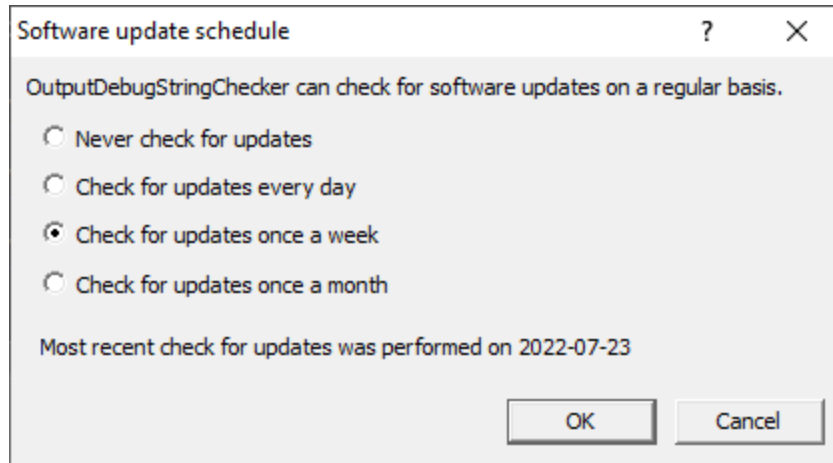
PE File Browser can automatically check to see if a new version of PE File Browser is available for downloading.

 **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.

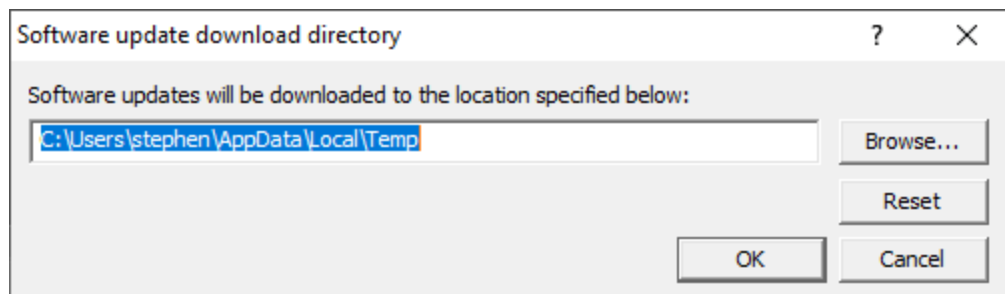


## Software update directory

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)

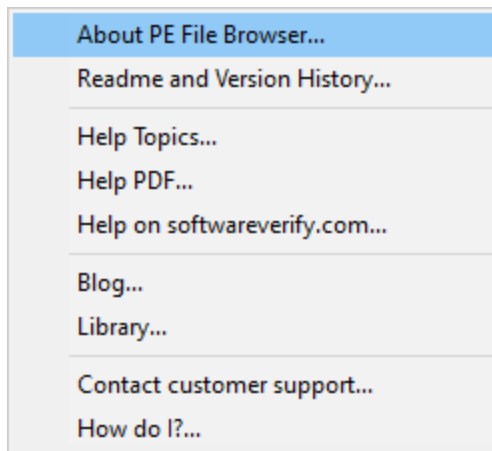
 When modifying the download directory, you should ensure the directory will continue to be valid. Updates may no longer occur if the download location is later invalidated.

- **Reset** > reverts the download location to the user's `TEMP` directory

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

## 3.4 Help

The Help menu controls displaying this help document and displaying information about PE File Browser.



**Help menu** > **About PE File Browser...** > displays information about PE File Browser.

**Help menu** > **Readme and Version History...** > displays the readme and version history.

**Help menu** > **Help Topics...** > displays this help file.

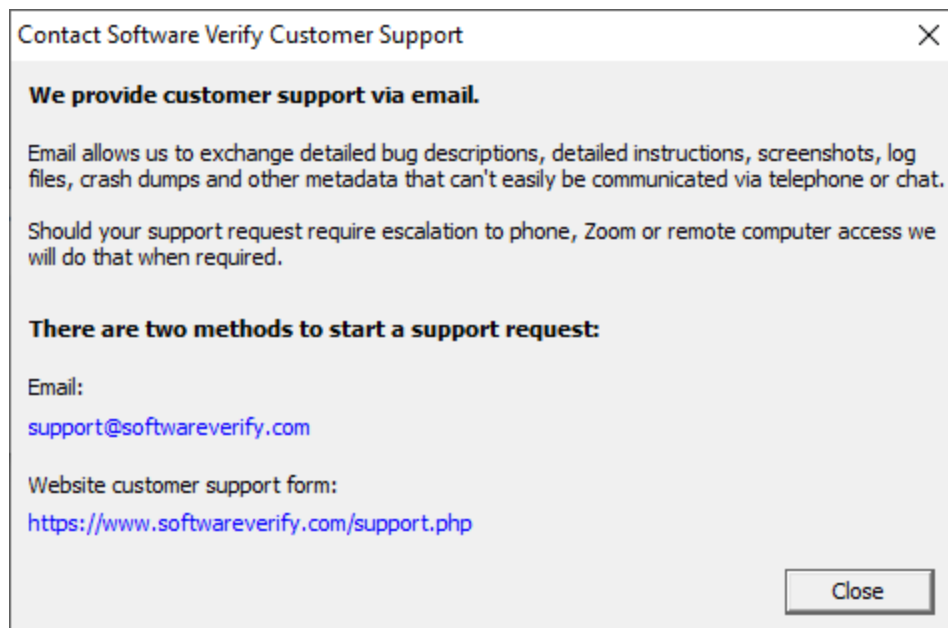
**Help menu** > **Help PDF...** > displays this help file in PDF format.

**Help menu** > **Help on softwareverify.com...** > display the Software Verify documentation web page where you can view online documentation or download compiled HTML Help and PDF help documents.

**Help menu** > **Blog...** > display the Software Verify blog.

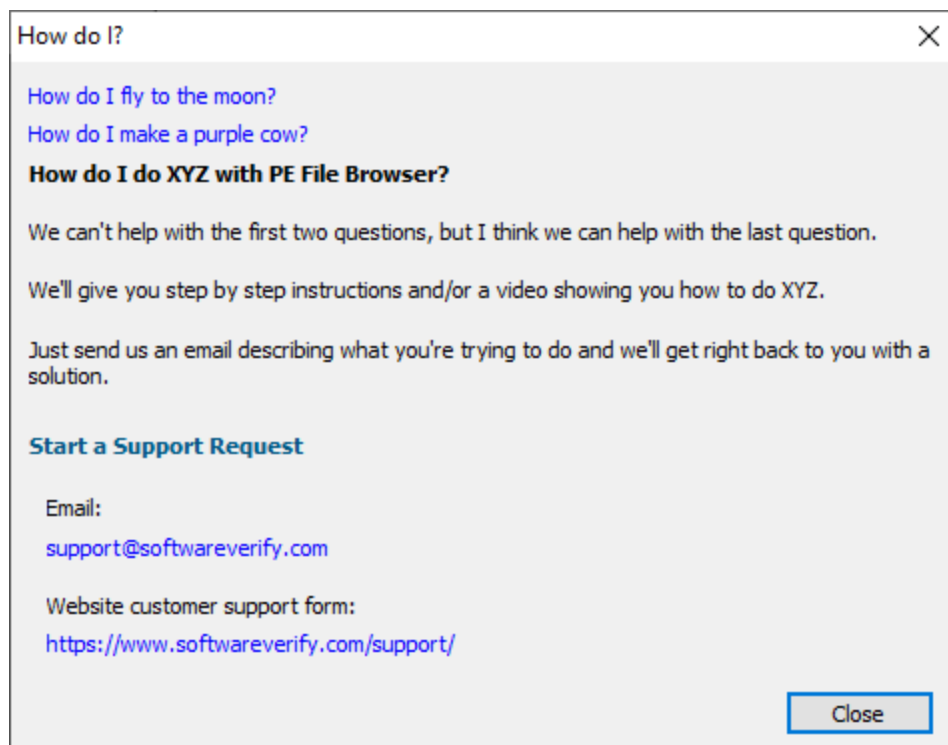
**Help menu** > **Library...** > display the Software Verify library - our best blog articles grouped by related topics.

**Help menu** > **Contact customer support...** > displays the options for contacting customer support.



Click a link to contact customer support.

**Help** menu > **How do I?...** > displays the options for asking us how to do a particular task.



**Part**

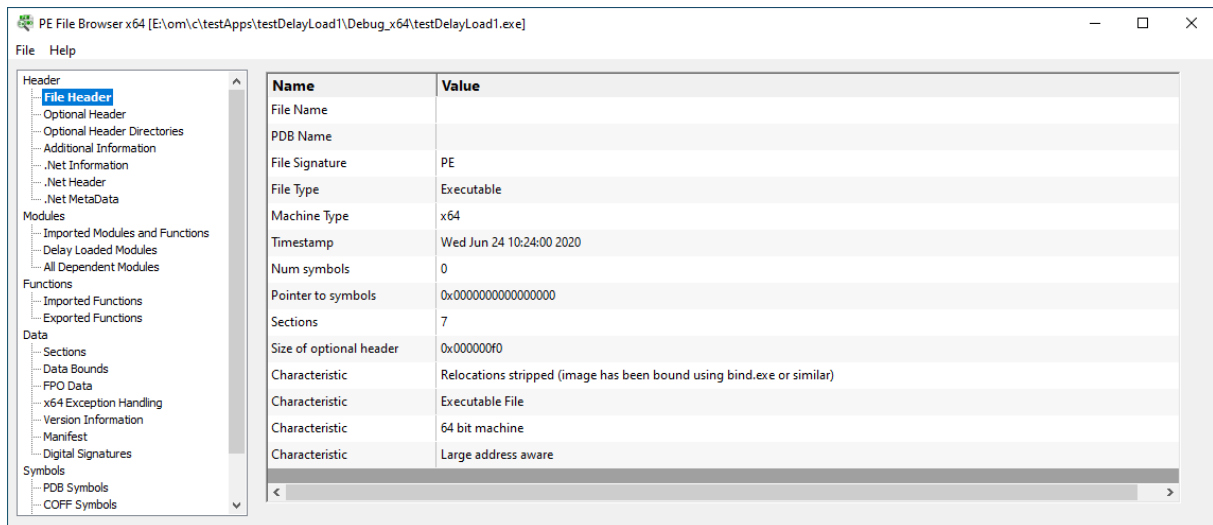


## 4 The user interface

Enter topic text here.

### 4.1 The data display

The PE File Browser user interface is shown below.



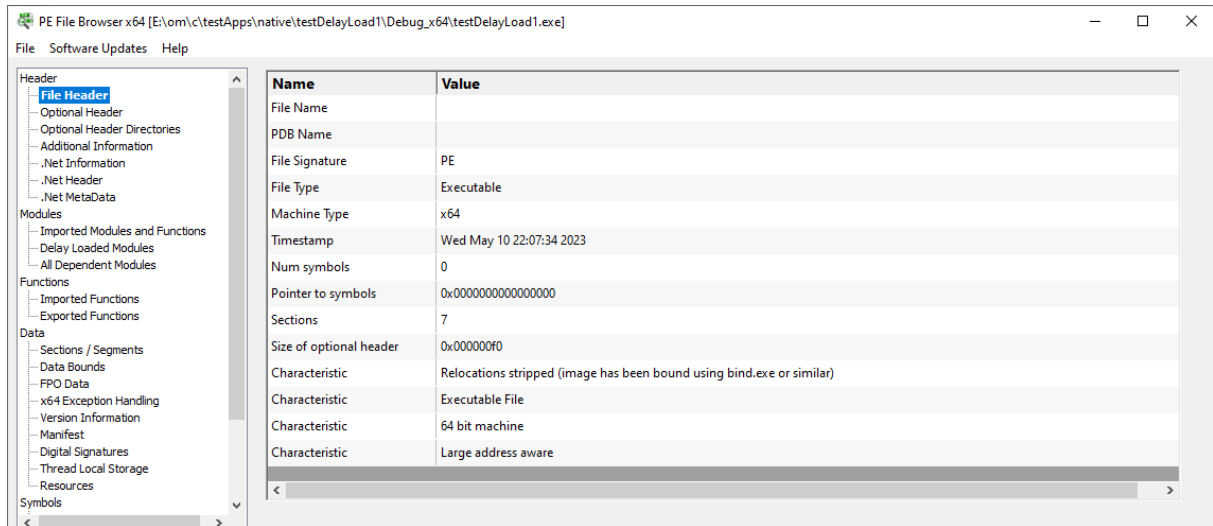
The general principle guiding what data we choose to display and how we group the data is as follows:

- Is the data logically grouped inside the PE file?  
This covers data that is in the structure of the PE file, such as File Headers, Optional Headers, Sections, etc.
- Is the data logically grouped from the perspective of the user of the software?  
This covers data such as debugging symbols.

There have been a few areas where we've blended data together to either compress the display (the File Header section has a few additional entries) or to synthesize extra information not expressly provided by the PE file format but which can be determined from the data in the PE file (the Additional Information section is an example of this).

#### 4.1.1 File Header

The File Header display shows the contents of the PE file File Header.



### File Name

The name of the PE file. This is often blank.

### PDB Name

The name of the corresponding PDB file. This is often blank.

### File Signature

This can be many values, but for a PE file, will be the value "PE".

### File Type

This can be one of three values:

- DLL. A loadable module.
- Executable. An executable program.
- System. A system file.

### Machine Type

This can be one of many values, although we typically only expect "i386" and "x64".

Valid values are:

- Unknown
- i386
- x64
- MIPS R3000
- MIPS R4000
- MIPS R10000
- Alpha
- Power PC
- Hitachi SH3
- Hitachi SH4
- ARM
- MIPS WinCE v2
- Hitachi SH3E
- Thumb
- IA64 (Merced/Itanium)
- MIPS 16
- MIPS FPU

- MIPS FPU 16
- ALPHA64

**Timestamp**

This is the time the PE file was created. This is often not set.

**Num Symbols**

The number of symbols embedded in the PE file.

**Pointer to Symbols**

Pointer to any symbols embedded in the PE file.

**Sections**

The number of different named sections in the PE file.

**Size of optional header**

The size of the optional header in the PE file.

**Characteristics**

A PE file can have many characteristics. The remainder of the File Header lists only the characteristics that have been set, one per line.

The following characteristics can be set. Many values can apply at the same time. Many of these characteristics are obsolete.

- Relocations stripped (image has been bound using bind.exe or similar)
- Executable File
- DLL
- System File
- UniProcessor systems only
- 32 bit machine
- 64 bit machine
- Line numbers stripped
- Local symbols stripped
- Debug info stripped (held in separate .DBG file)
- Aggressive working set trim
- Large address aware
- Machine is LSB precedes MSB
- Run from swap if image is on removable media
- Run from swap if image is on the network
- Machine is MSB precedes LSB

### 4.1.2 Optional Header

The Optional Header display shows the contents of the PE file Optional Header.

Name	Value	Description
Magic Number	0x000020b	64 bit
Linker Version	10.0	Visual Studio 2010
Size of code	0x00062200	392.50 KB
Size of initialised data	0x0001c400	113.00 KB
Size of uninitialised data	0x00000000	0 B
Entry point RVA	0x00002290	
Base of code	0x00001000	4.00 KB
Image base	0x000000140000000	Fixed load address
Section alignment	0x00001000	4.00 KB
File alignment	0x00000200	512 B
O.S. version	5.2	Windows XP
Image version	0.0	
Subsystem version	5.2	Windows XP
Win32 version	0	
Size of image	0x00085000	532.00 KB

### Magic Number

This is the magic number stored in the PE file to identify the file type.

Valid values are:

- 0x0107      ROM
- 0x010B      32 bit
- 0x020B      64 bit

### Linker Version

This is the version of the linker used to build the image.

The description field indicates which version of Visual Studio may have been used to build this file.

If Visual Studio 2012 is set to build using Visual Studio 2010 libraries this field will indicate Visual Studio 2010.

If a different compiler/linker is used (Embarcadero, Delphi, MingW) the description field will be incorrect.

### Size of code

Size of the executable code in the PE file.

### Size of initialised data

Size of the initialised data in the PE file.

### Size of uninitialised data

Size of the uninitialised data in the PE file.

### Entry point RVA

The relative virtual address of the entry point to the executable. This is an offset from the start of the PE file.

This is DIIMain for a DLL, or the start of the program for an EXE.

### Base of code

This is the relative virtual address of the start of the code in the PE file.

**Base of data**

This is the relative virtual address of the start of the data in the PE file.

**Section alignment**

Sections are aligned at boundaries that are a multiple of this value after the PE file is mapped into memory.

**File alignment**

Sections are aligned at boundaries that are a multiple of this value before the PE file is mapped into memory.

**O.S. version**

The minimum version of the Windows operating system that this software can execute.

**Image version**

A user definable version number.

**Subsystem version**

The windows subsystem version number.

**Win32 version**

The Win32 version number

**Size of image**

The size of the parts of the PE file that the DLL loader needs to be concerned with.

**Size of headers**

The size of the PE header and the section (object) table.

**Subsystem**

The Windows subsystem.

Valid values are:

- Unknown
- Native
- Windows GUI
- Windows Character UI
- OS2 Character UI
- Posix Character UI
- Windows 95/98 Driver
- Windows CE GUI
- EFI Application
- EFI Boot Service Driver
- EFI Runtime Driver
- EFI ROM
- XBox
- Windows Boot Application

**Characteristics**

These are characteristics that affect how the PE file is treated. This is a bitmask. Many values can apply at the same time.

Valid characteristics are:

- Call when DLL is first loaded into a process's address space
- Call when a thread terminates

- Call when a thread starts up
- Call when DLL exits
- Unknown(0x0010)
- Unknown(0x0020)
- Relocatable
- Fixed load address
- Force code integrity check
- Data execution prevention compatible
- No isolation
- No SEH
- Do not bind
- WDM driver
- Reserved(0x4000)
- Terminal server aware

**Stack reserve**

This is the amount of memory reserved for the first thread's stack.

**Stack commit**

This is the amount of memory committed for the first thread's stack.

**Heap reserve**

This is the amount of memory reserved for the process heap.

**Heap commit**

This is the amount of memory committed for the process heap.

**Loader flags**

The Loader flags value is obsolete. This value is normally set to 0. Known values appear to be related to debugging support.

1. Invoke a breakpoint instruction before starting the process
2. Invoke a debugger on the process after it's been loaded

### 4.1.3 Optional Header Directories

The Optional Header Directories display shows the contents of the PE file Optional Header Directories.

The screenshot shows the PE File Browser x64 interface. The left sidebar is expanded to 'Optional Header Directories'. The main window displays a table with the following data:

Name	RVA	Address	Size
Export	0x00000000		
Import	0x00081000	0x0000000140081000	0x00000050
Resource	0x00084000	0x0000000140084000	0x00000459
Exception	0x0007d000	0x000000014007d000	0x00002ad8
Security / Certificate table	0x00000000		
Base Relocation	0x00000000		
Debug	0x00064ce0	0x0000000140064CE0	0x0000001c
Architecture Specific	0x00000000		
Global Pointer	0x00000000		
Thread Local Storage	0x00000000		
Load Configuration	0x00000000		
Bound Import	0x00000000		
Import Address Table	0x000814e0	0x00000001400814E0	0x00000490
Delay Load Import Table	0x00083000	0x0000000140083000	0x00000060
CLR / COM Runtime Table	0x00000000		
Reserved	0x00000000		

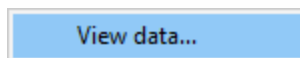
Each optional header directory is listed, with the RVA, the actual address and size. Address and Size are only displayed if the RVA is valid (non-zero).

The following header directories are displayed.

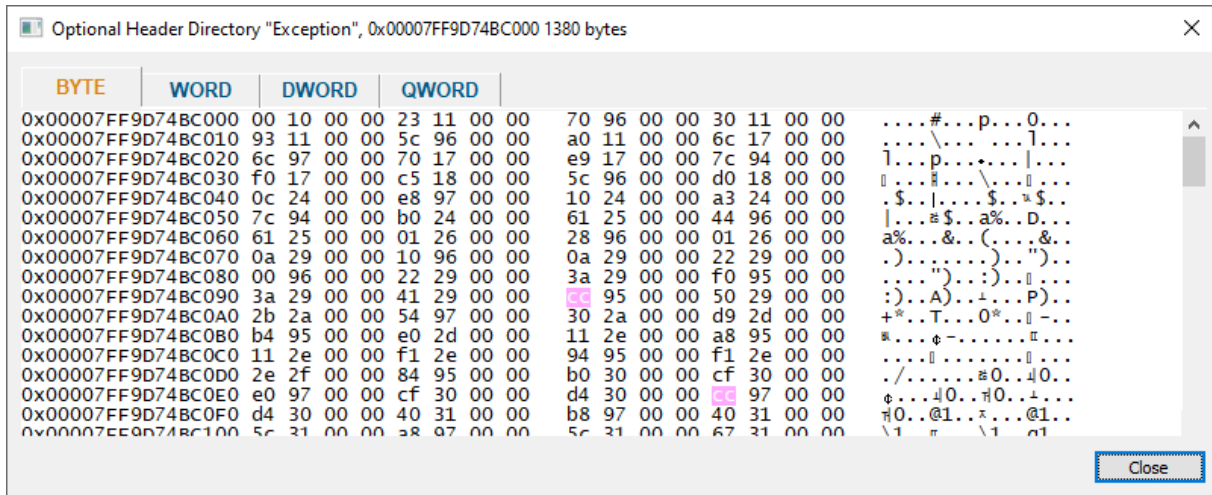
- Export
- Import
- Resource
- Exception
- Security
- Base Relocation
- Debug
- Architecture Specific
- Global Pointer
- Thread Local Storage
- Load Configuration
- Bound Import
- Import Address Table
- Delay Load Import Table
- COM Runtime Table
- Reserved

## Context Menu

A context menu provides a single option:

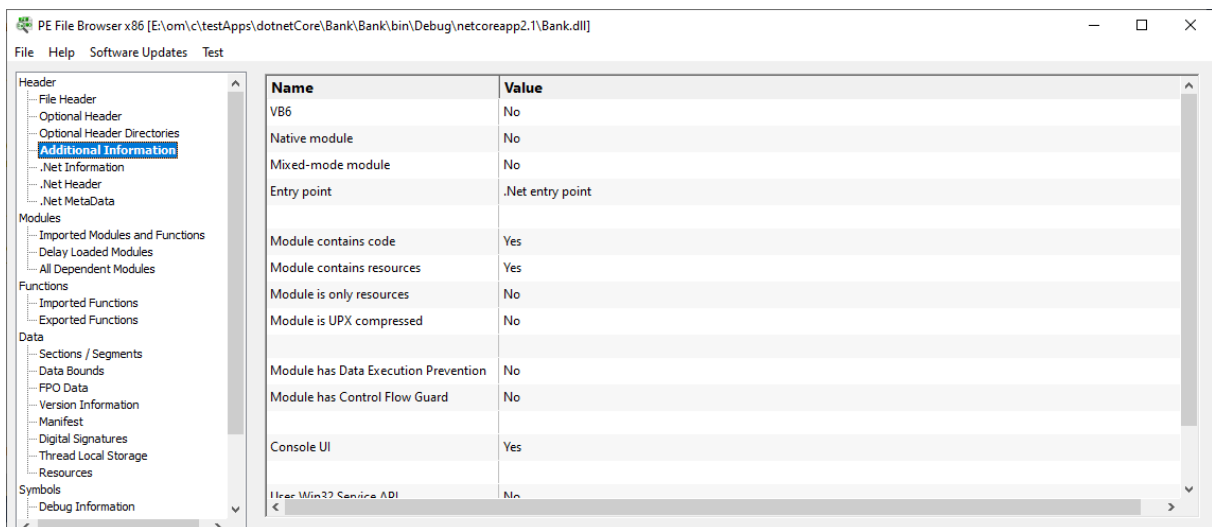


Clicking **View data...** opens a memory inspection dialog, allowing you to view the memory as BYTES, WORDS, DWORDs or QWORDS.



#### 4.1.4 Additional Information

The Additional Information displays some additional information about a PE that can be determined from a PE file, but which is not an obvious value in the PE file.



#### VB6

Is this module a Visual Basic 6 application?

#### Native module

Is this a native executable?

#### Mixed-mode module

Is this a mixed mode executable (contains native code and .Net code)?

#### Entry point

The entry point field describes the type of entry point for the DLL.

Valid values are:

- No entry point
- .Net entry point
- Native entry point
- Unknown entry point type

**Module contains code**

Does this module contain executable code?

**Module contains resources**

Does this module contain resources?

**Module is only resources**

Does this module only contain resources?

**Module is UPX compressed**

This module has been compressed using the UPX compression algorithm.

**Module has Data Execution Prevention**

This module implements data execution prevention security measures.

**Module has Control Flow Guard**

This module implements control flow guard security measures.

**Console UI**

Is this module a console application?

**Uses Win32 Service API**

This EXE/DLL uses functions in the Win32 Service API. This EXE/DLL may be part of a service.

**Uses Visual Leak Detector**

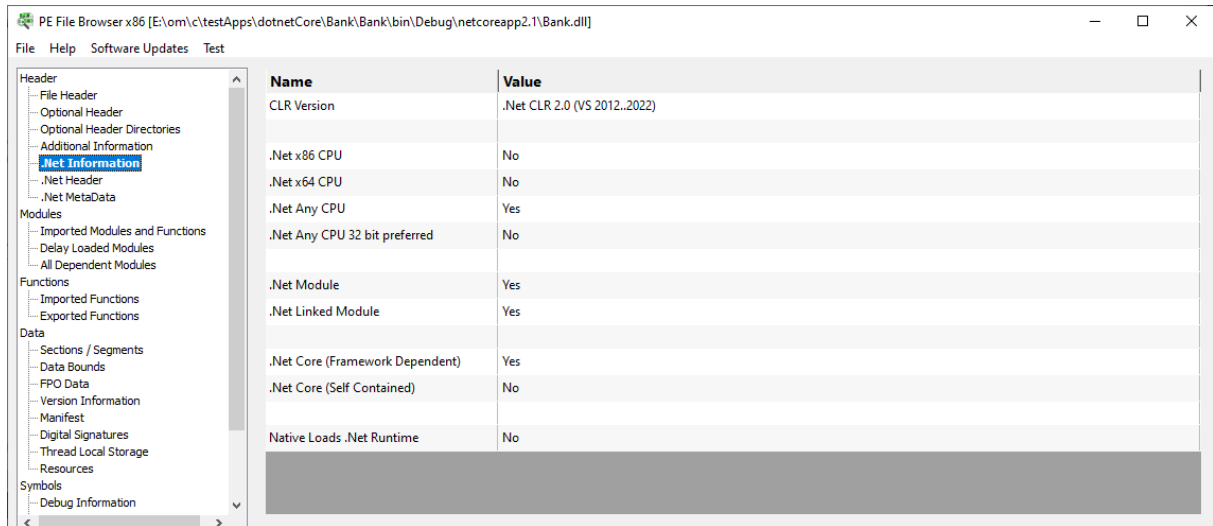
This EXE/DLL uses Visual Leak Detector.

**Uses Address Sanitizer**

This EXE/DLL uses Address Sanitizer.

#### 4.1.5 .Net Information

The .Net Information displays some .Net specific information about a PE that can be determined from a PE file, but which is not an obvious value in the PE file.



### CLR Version

If this module uses the .Net runtime, this entry displays which version of the CLR is required.

Valid values are:

- .Net CLR 1.0 (VS 2002)
- .Net CLR 1.1 (VS 2003)
- .Net CLR 2.0 (VS 2005..2010)
- .Net CLR 2.0 (VS 2012..2017)
- .Net CLR 2.0 or later

The last value will be displayed if a CLR version that is not recognised is encountered.

### .Net x86 CPU

Is this module a .Net module that is compiled for use on x86 processors?

### .Net x64 CPU

Is this module a .Net module that is compiled for use on x86 processors?

### .Net Any CPU

Is this module a .Net module that is compiled for use on any CPU?

### .Net Any CPU 32 bit preferred

Is this module a .Net module that is compiled for use on any CPU but prefers 32 bit processors?

### .Net Module

Is this module a pure .Net module (contains no native code)?

### .Net Linked Module

Is this module linked to the .Net runtime?

### .Net Core (Framework Dependent)

Is this module built for use with .Net Core as a framework dependent DLL?

### .Net Core (Self Contained)

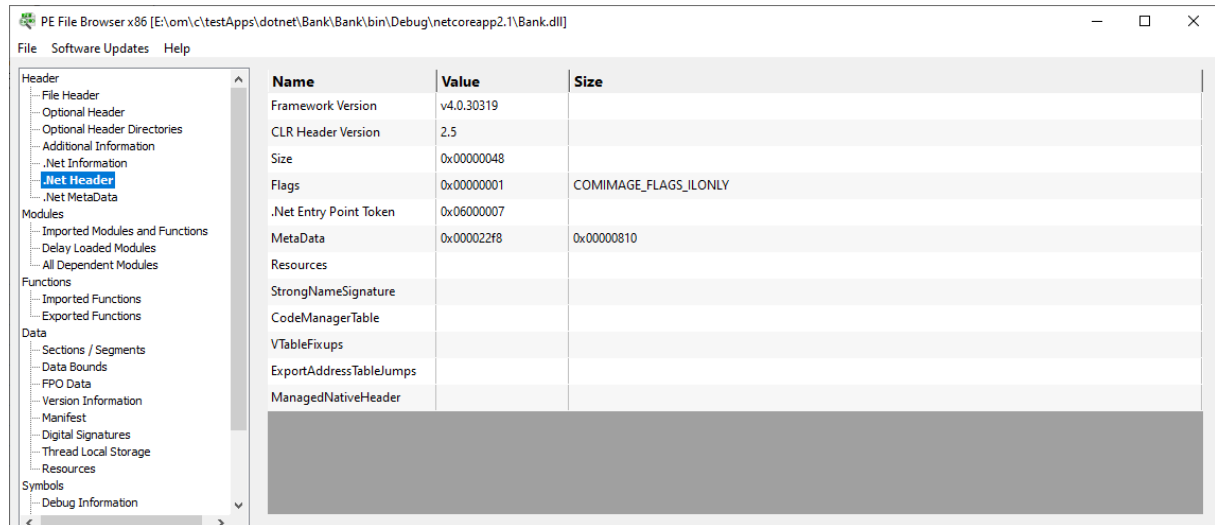
Is this module built for use with .Net Core as a self contained executable?

## Native Loads .Net Runtime

Is this a native module that loads the .Net runtime to executable .Net code?

### 4.1.6 .Net Header

The .Net Header displays some information the .Net Header (if any) that is present in the PE file.



#### Framework version

The .Net Framework version

#### MetaData Version

The .Net Metadata version.

#### MetaData Flags

The .Net Metadata flags.

#### Size

Size of the .Net Header

#### Flags

CLR flags. As specified by **.corflags** directive in ILASM or **/FLAGS** with ILASM compiler.

#### .Net Entry Point RVA

The relative virtual address that defines the native entry point for this .Net application. This is not displayed if a .Net Entry Point Token is defined.

#### .Net Entry Point Token

The .Net method token that identifies the startup function for this application. This is not displayed if a .Net Entry Point RVA is defined.

#### MetaData

Relative virtual address (RVA) of the .Net Metadata inside the PE file.

#### Resources

Relative virtual address (RVA) of the managed resources inside the PE file.

**StrongNameSignature**

Relative virtual address (RVA) of the strong name signature for this PE file.

**CodeManagerTable**

Relative virtual address (RVA) of the Code Manager table.

**VTableFixups**

Relative virtual address (RVA) of the virtual fixup table.

**ExportAddressTableJumps**

Relative virtual address (RVA) of jump thunks.

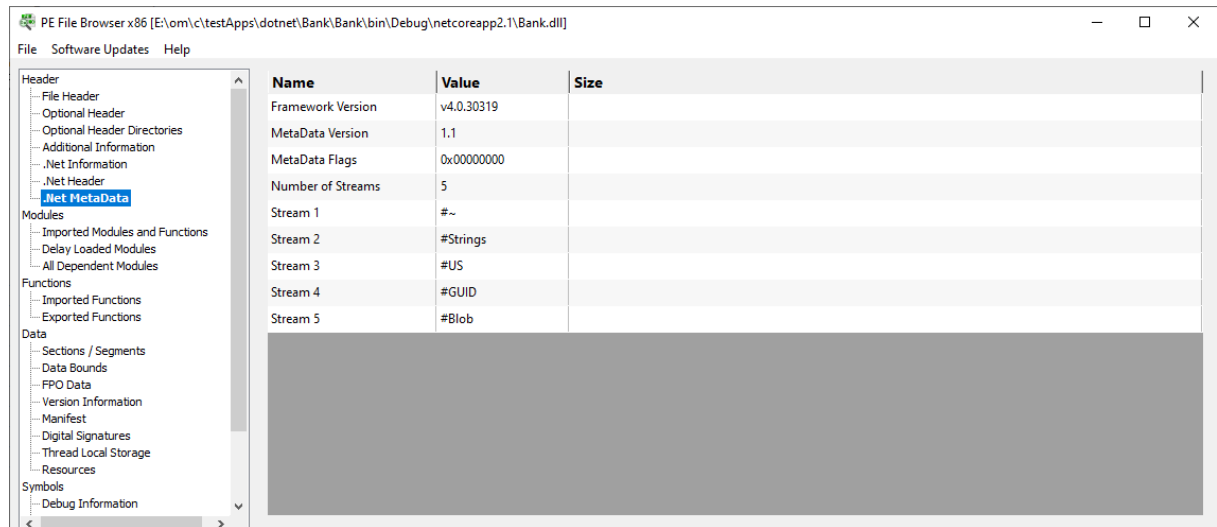
**ManagedNativeHeader**

Relative virtual address (RVA) of the managed native header. Reserved for precompiled images.

*For a more detailed explanation of the fields mentioned in this topic see Table 4-6, Chapter 4 of Expert .Net 2.0 Assembler by Serge Lidin.*

**4.1.7 .Net MetaData**

The .Net MetaData displays some information the .Net MetaData (if any) that is present in the PE file.

**Framework version**

The .Net Framework version

**MetaData Version**

The .Net Metadata version.

**MetaData Flags**

The .Net Metadata flags.

**Number of Streams**

How many streams are in the .Net MetaData.

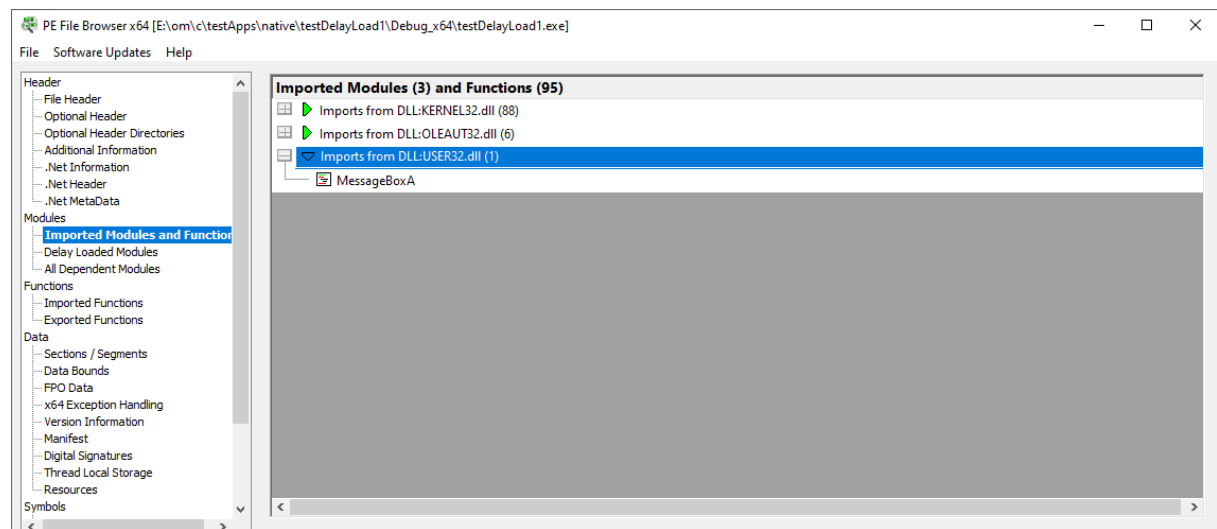
### Stream ??

For each metadata stream present the name of the metadata stream is shown.

There are typically no more than 5 streams. The expected names are #~, #Strings, #US, #GUID, #Blob.

## 4.1.8 Imported Modules and Functions

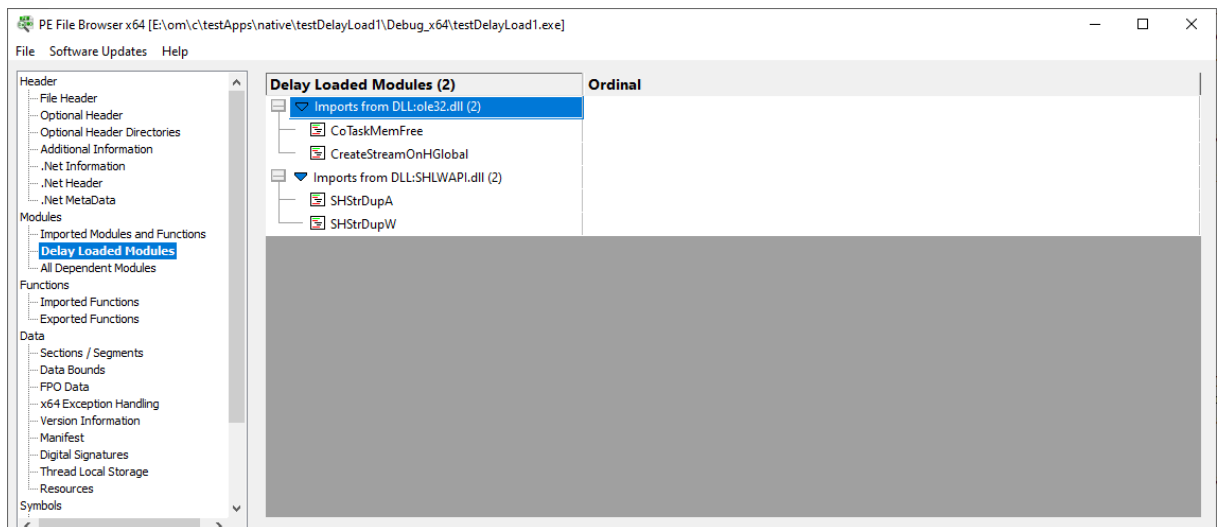
The Imported Modules and Functions display shows the modules that are linked to the PE file. The display also shows the functions imported from each module.



In the image above you can see that this PE file imports functions from 3 DLLs. The first DLL in the list, KERNEL32.DLL imports 88 functions to the PE file.

## 4.1.9 Delay Loaded Modules

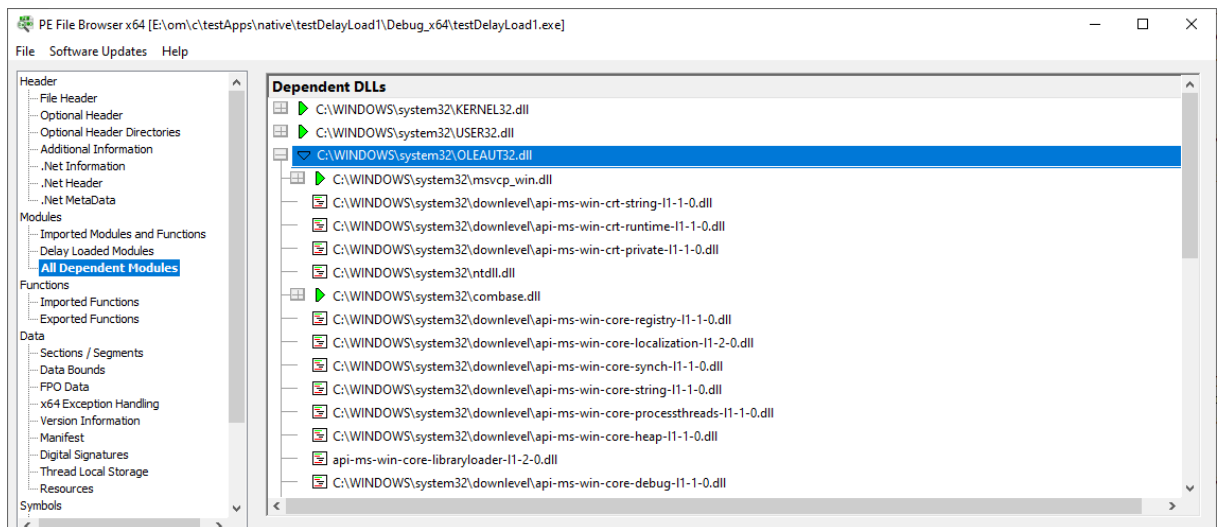
The Delay Loaded Modules display shows the modules that are linked to the PE file but which are delay loaded (they load when the a delay loaded function is called).



In the image above you can see that this PE file delay loads two DLLs, ole32.dll and shlwapi.dll. Each DLL in this example imports 2 functions.

#### 4.1.10 All Dependent Modules

The All Dependent Modules display shows the all the modules that the PE file depends on.

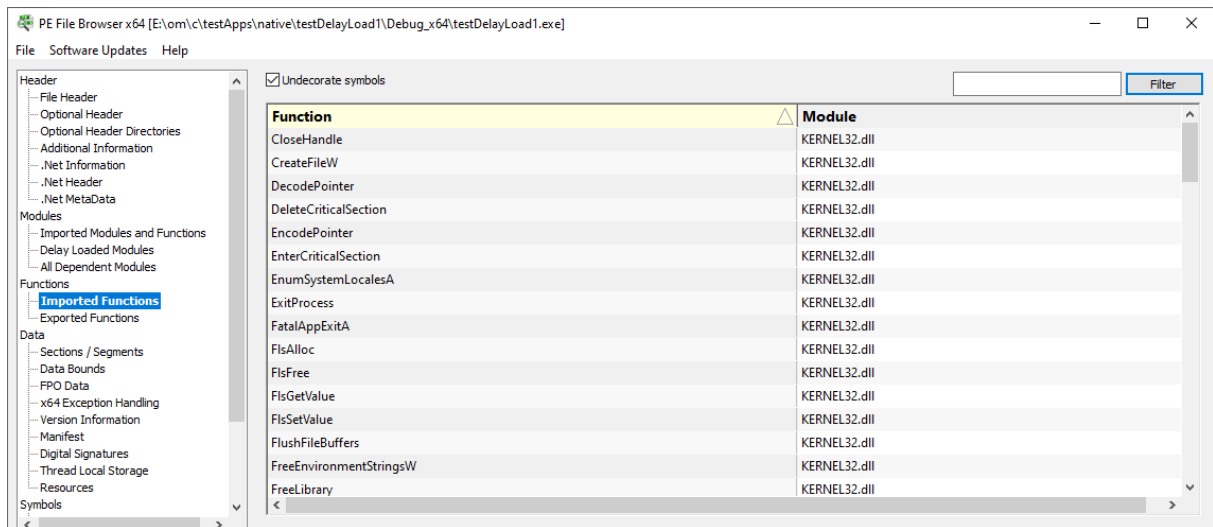


If you expand the entries you can see which DLLs those DLLs depend on.

Any DLLs that can't be found on the \$PATH or in the same folder as the main PE file will be displayed in red and marked as a "Missing DLL".

### 4.1.11 Imported Functions

The Imported Functions display shows the all the functions imported to the PE file.



#### Function

The name of the imported function.

#### Module

The name of the module the function is imported from.

#### Sorting

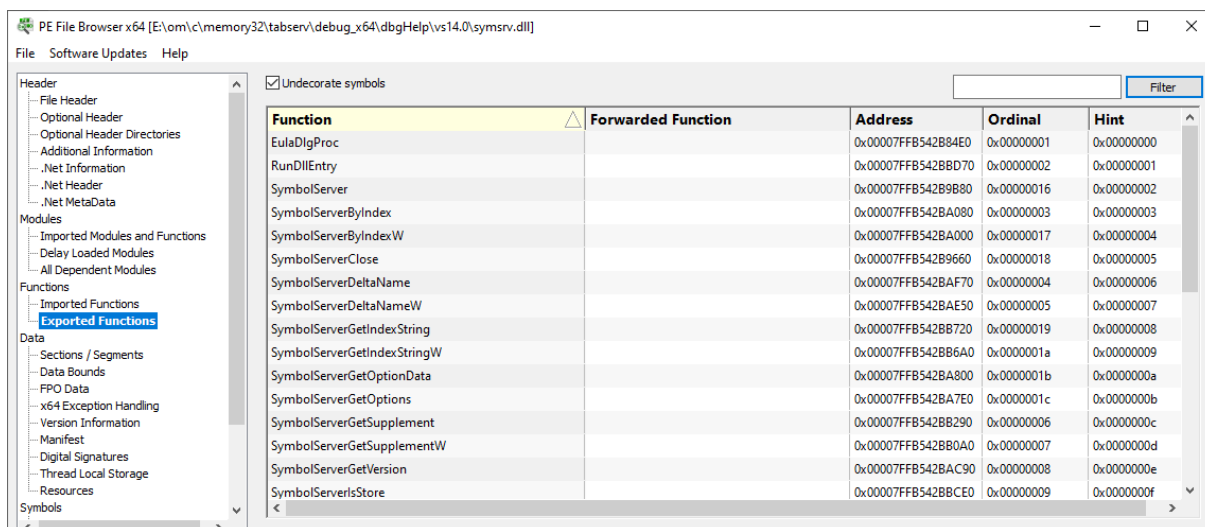
The data can be sorted by Function or Module. Click the column header to select sorting by Function or Module. Click the same header to reverse the sort direction.

#### Filter

If a filter string is typed into the edit field and the **Filter** button is clicked the display shows on functions that contain the filter string. The search is case sensitive.

### 4.1.12 Exported Functions

The Exported Functions display shows the all the functions exported from the PE file.



### Function

The name of the exported function.

### Forwarded Function

If function is being forwarded from another DLL, it is listed in this column.

### Address

The address of the exported function.

### Ordinal

The ordinal of the exported function.

### Hint

The ordinal hint of the exported function.

### Sorting

The data can be sorted by any column. Click the column header to select sorting by that column. Click the same header to reverse the sort direction.

### Filter

If a filter string is typed into the edit field and the **Filter** button is clicked the display shows on functions that contain the filter string. The search is case sensitive.

## 4.1.13 Sections / Segments

The Sections display shows the all the sections in the PE file.

Sections are also known as segments, often mentioned in MAP files and in the context of debugging information.

Symbol information is often provided as an offset and a segment. The symbol address is the module load address + the segment offset inside the dll, plus the symbol offset inside the segment.

Segment	Name	Relocations	Line Numbers	Length	Virtual	Raw Size	Character
1	.text	0	0	0x000141df	0x00001000	0x00014200	Code, Execu
2	.rdata	0	0	0x00010e64	0x00016000	0x00011000	Init data, Re
3	.data	0	0	0x00023364	0x00027000	0x00000600	Init data, Re
4	.pdata	0	0	0x00000fe4	0x0004b000	0x00001000	Init data, Re
5	.didat	0	0	0x000001c8	0x0004c000	0x00000200	Init data, Re
6	.mrdata	0	0	0x00002d30	0x0004d000	0x00002e00	Init data, Re
7	.rsrc	0	0	0x00001998	0x00050000	0x00001a00	Init data, Re
8	.reloc	0	0	0x000009e0	0x00052000	0x00000a00	Init data, Dis

### Segment

The segment identifier.

### Name

The section name.

### Relocations

The number of relocations in this section.

### Line Numbers

The number of line numbers in this section.

### Physical

The physical address of this section.

### Virtual

The virtual address of this section.

### Size

The size of this section.

### Characteristics

The characteristics of this section. This is a bitmask. Many values can apply at the same time. Some of these values are obsolete.

- Code
- Init data
- Uninit data
- Comments/info
- Not part of image
- COMDAT
- Reset speculative exception
- GP relative data
- FAR data
- Purgeable data
- 16 bit data

- Locked data
- Preloadable data
- Extended relocations
- Discardable
- Not cacheable
- Not pageable
- Shareable
- Executable
- Readable
- Writeable
- Alignment:1
- Alignment:2
- Alignment:4
- Alignment:8
- Alignment:16
- Alignment:32
- Alignment:64
- Alignment:128
- Alignment:256
- Alignment:512
- Alignment:1024
- Alignment:2048
- Alignment:4096
- Alignment:8192

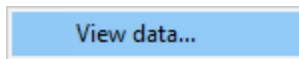
The alignment values are exclusive. Only one alignment value can be applied.

### Sorting

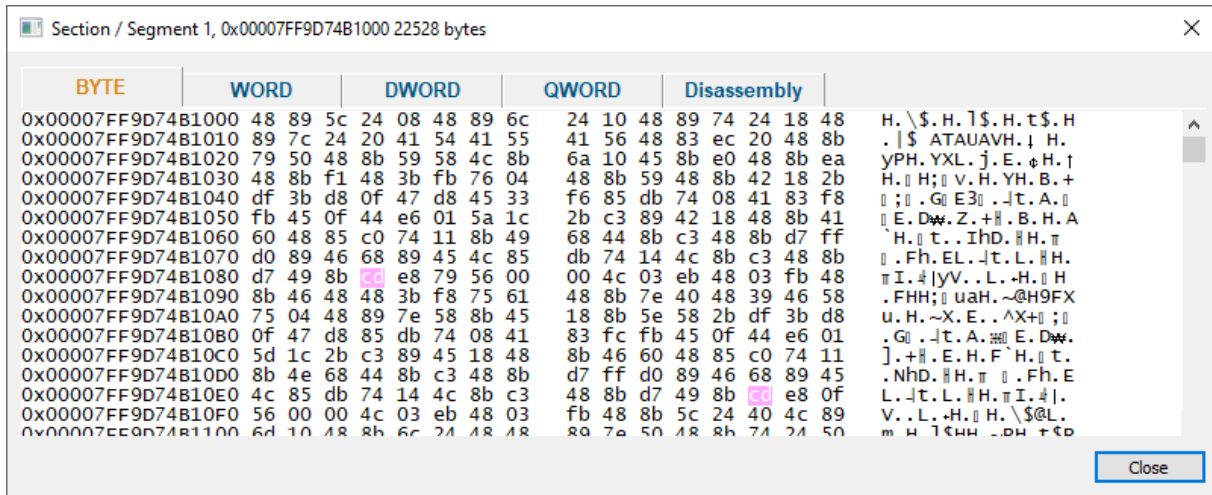
The data can be sorted by any column. Click the column header to select sorting by that column. Click the same header to reverse the sort direction.

### Context Menu

A context menu provides a single option:

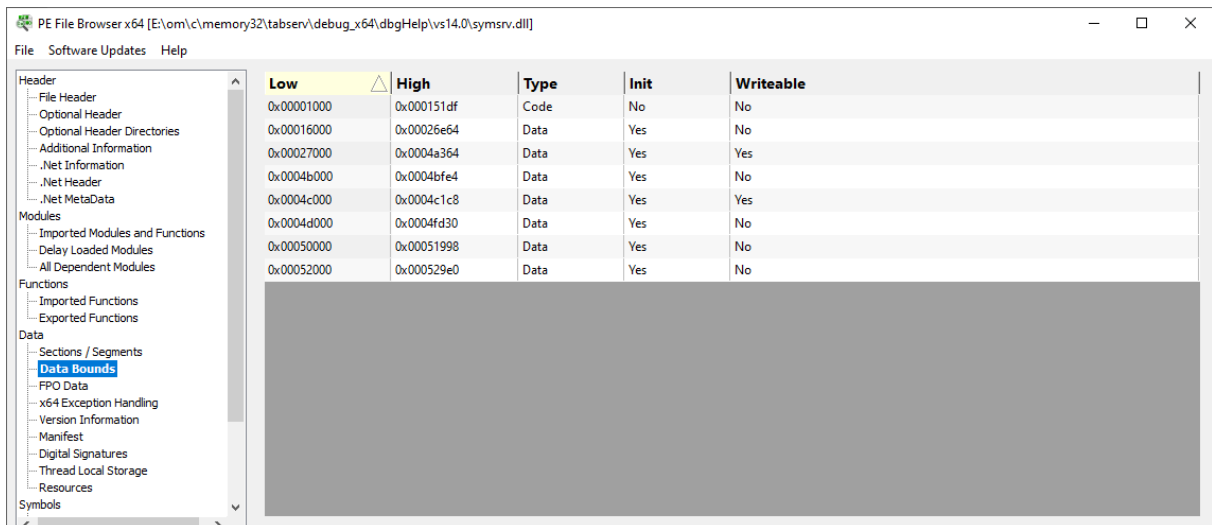


Clicking **View data...** opens a memory inspection dialog, allowing you to view the memory as BYTES, WORDs, DWORDs or QWORDs. For executable code a disassembly view is provided.



#### 4.1.14 Data Bounds

The Data Bounds display shows the all the different data and code area in the PE file.



#### Low

The low RVA (offset from the start of the PE file) of the data bound.

#### High

The high RVA (offset from the start of the PE file) of the data bound.

#### Type

This indicates if this area of memory is code or data.

#### Init

Is this area of memory initialised?

### Writeable

Is this area of memory writeable?

### Sorting

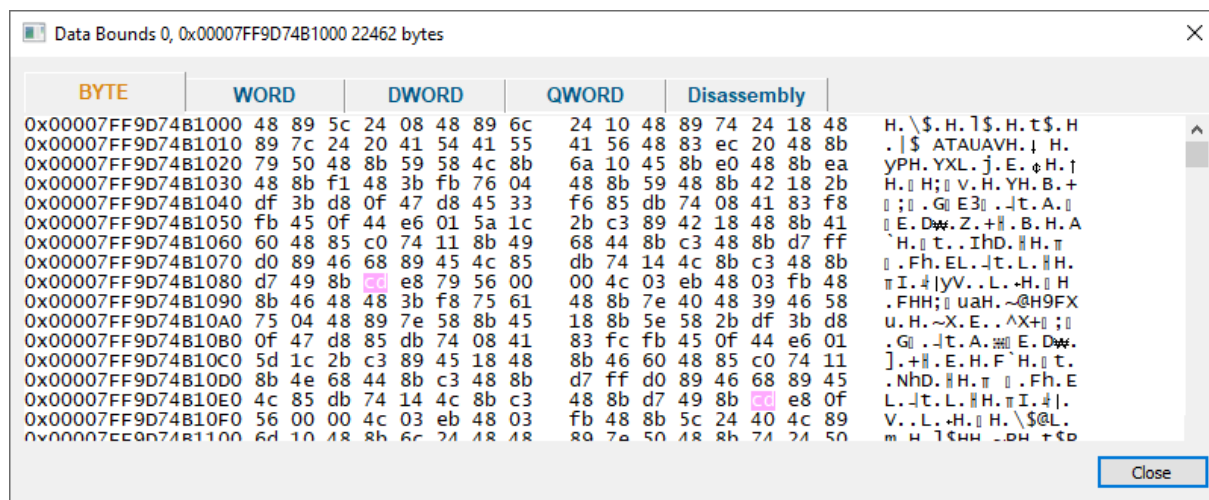
The data can be sorted by any column. Click the column header to select sorting by that column. Click the same header to reverse the sort direction.

## Context Menu

A context menu provides a single option:

View data...

Clicking **View data...** opens a memory inspection dialog, allowing you to view the memory as BYTES, WORDs, DWORDs or QWORDs. For executable code a disassembly view is provided.



### 4.1.15 FPO Data

The FPO display shows the all the Frame Pointer Omission data in the PE file.

Frame Pointer Omission data is largely obsolete these days. Since Windows Vista, no Windows binaries have been shipped that use Frame Pointer Omission. However the option to use Frame Pointer Omission remains in the compiler.

Type	RVA	Size	Locals	Params	Prolog	Registers	SEH	Uses EBP
FPO	0x0002c719	13	0	0	0	0	No	No
FPO	0x00028cbd	35	0	4	0	0	No	No
FPO	0x00028ce0	5	0	0	0	0	No	No
FPO	0x00028ce5	1	0	0	0	0	No	No
FPO	0x00089afc	25	0	0	0	0	No	No
FPO	0x0002914b	104	0	0	0	2	No	No
FPO	0x000291b3	63	0	8	0	0	No	No
FPO	0x000291f2	7	0	0	0	0	No	No
FPO	0x000291f9	23	0	4	0	0	No	No
FPO	0x00029210	3	0	4	0	0	No	No
FPO	0x00029210	3	0	4	0	0	No	No
FPO	0x00029210	3	0	4	0	0	No	No
FPO	0x00029210	3	0	4	0	0	No	No
FPO	0x00029210	3	0	4	0	0	No	No
FPO	0x00029210	3	0	4	0	0	No	No
FPO	0x00029210	3	0	4	0	0	No	No
FPO	0x00029210	3	0	4	0	0	No	No

**Type**

The type of frame.

Valid values are:

- FPO
- Trap
- TSS
- Non FPO

**RVA**

The relative virtual offset of the function that has Frame Pointer Omission data

**Size**

This is the size of the function that has Frame Pointer Omission data.

**Locals**

The number of DWORD local variables this function has.

**Params**

The number of DWORD parameters this function has.

**Prolog**

The number of bytes in the function prolog.

**Registers**

How many registers are saved.

**SEH**

Does the function use Structured Exception Handling?

**Uses EBP**

Does the function use the EBP register?

**Sorting**

The data can be sorted by any column. Click the column header to select sorting by that column. Click the same header to reverse the sort direction.

### 4.1.16 x64 Exception Handling

The x64 Exception Handling display shows the all the exception handling data in the x64 PE file. These are `RUNTIME_FUNCTION` records.

x86 PE files handle exceptions using a different technique, as such there is no corresponding data in an x86 PE file. This display is not available when working with a 32 bit (x86) PE file.

#	Function	Filename	Line	Begin	End	Unwind	Version	Flag
0	CFont::~scalar deleting de...			0x00001030	0x0000106a	0x0009ec44	1	0x00
1	CBrush::~scalar deleting d...			0x00001080	0x000010ba	0x0009ec44	1	0x00
2	CGdiObject::~scalar deleti...			0x000010c0	0x000010f0	0x0009ec44	1	0x00
3	CGdiObject::~CGdiObject	c:\program files (x86)\microsoft vi...	77	0x000010f0	0x0000113b	0x0009efac	1	0x03
4	CBitmap::~scalar deleting ...			0x00001140	0x0000117a	0x0009ec44	1	0x00
5	CGridCell::CreateObject	e:\om\c\3rd_src\virtualtreegridcon...	41	0x00001410	0x0000148c	0x0009ec44	1	0x00
6	CGridDefaultCell::Create...	e:\om\c\3rd_src\virtualtreegridcon...	42	0x00001490	0x000014c4	0x000a1f60	1	0x02
7	CGridCell::CGridCell	e:\om\c\3rd_src\virtualtreegridcon...	48	0x000014f0	0x0000154e	0x0009ec44	1	0x00
8	CGridCell::~scalar deletin...			0x00001550	0x000015ac	0x0009ec44	1	0x00
9	CGridCell::~CGridCell	e:\om\c\3rd_src\virtualtreegridcon...	55	0x000015b0	0x000015ef	0x000a23b8	1	0x00
10	CGridCell::Reset	e:\om\c\3rd_src\virtualtreegridcon...	71	0x000015f0	0x00001662	0x0009ec44	1	0x00
11	CGridCell::SetFont	e:\om\c\3rd_src\virtualtreegridcon...	91	0x00001670	0x00001719	0x0009c24c	1	0x00
12	CGridCell::GetFont	e:\om\c\3rd_src\virtualtreegridcon...	121	0x00001720	0x00001751	0x0009c414	1	0x00
13	CGridCell::GetFont	e:\om\c\3rd_src\virtualtreegridcon...	135	0x00001760	0x000017a9	0x000a23b8	1	0x00
14	CGridCell::GetFormat	e:\om\c\3rd_src\virtualtreegridcon...	175	0x000017e0	0x00001810	0x0009c414	1	0x00
15	CGridCell::GetMargin	e:\om\c\3rd_src\virtualtreegridcon...	188	0x00001810	0x00001840	0x0009c414	1	0x00
16	CGridCell::Edit	e:\om\c\3rd_src\virtualtreegridcon...	204	0x00001840	0x00001972	0x000a1f20	1	0x02

Each line of the grid displays the following information. If you expand the entry you can see the codes used for that exception handler.

#### #

The index of the `RUNTIME_FUNCTION` in the PE file.

#### Function

The name of the function this `RUNTIME_FUNCTION` relates to  
*This value is only populated if debug information can be read for this PE file.*

#### Filename

The filename of the function this `RUNTIME_FUNCTION` relates to  
*This value is only populated if debug information can be read for this PE file.*

#### Line

The line number of the function this `RUNTIME_FUNCTION` relates to  
*This value is only populated if debug information can be read for this PE file.*

#### Begin

The relative virtual address of the start of the area affected by the exception handler.

#### End

The relative virtual address of the end of the area affected by the exception handler.

#### Unwind

The relative virtual address of the unwind data for the exception handler.

### Version

The version format of the exception handling data

### Flags

Flags that indicate how what this exception handler is.

Valid values are:

- Exception Handler
- Termination Handler
- Chained unwind info

### Prolog

Size of the function prologue.

### Frame Reg

The name of frame register.

### Frame Offset

The offset for the frame.

### Codes

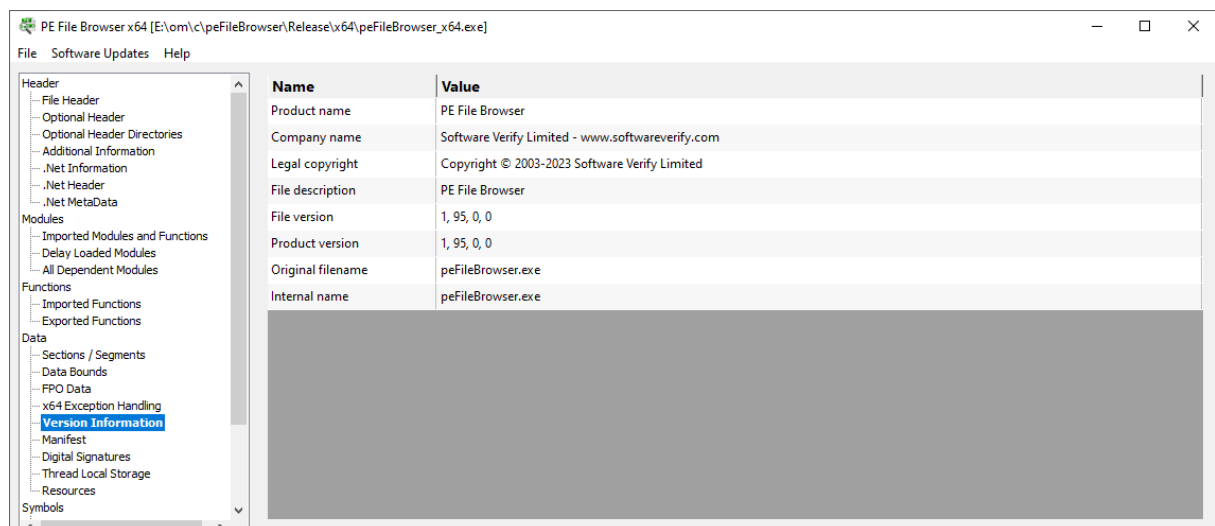
The number of codes used by this exception handler.

### Except RVA

The relative virtual address of the exception handler.

## 4.1.17 Version Information

The Version Information display shows the all the version information strings in the PE file.

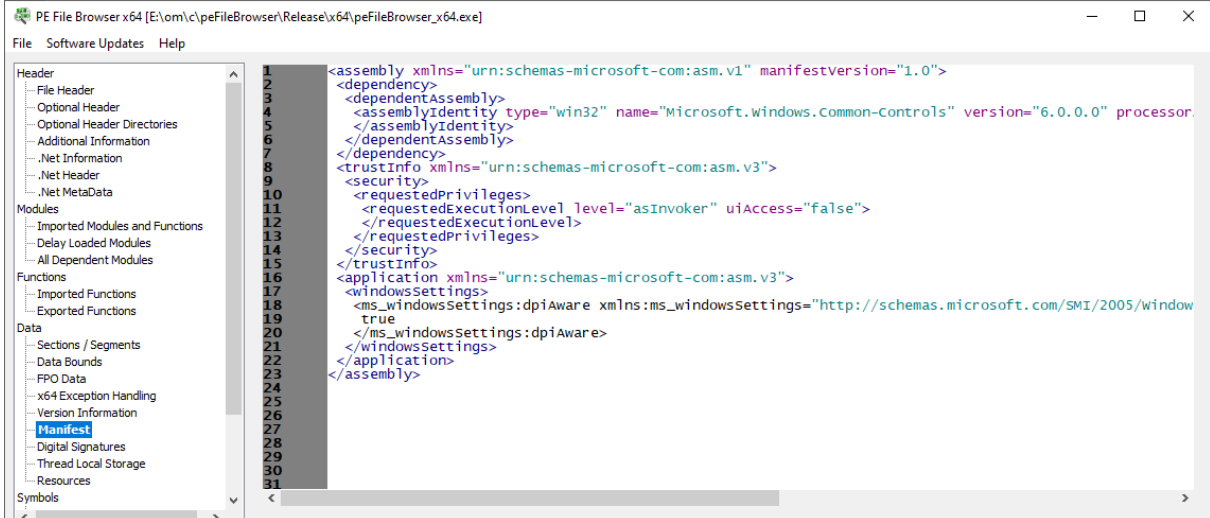


The following version strings are queried to see if they are present in the PE file. Only values that are present are displayed.

- Product name
- Company name
- Legal copyright
- Legal Trademarks
- File description
- File version
- Product version
- Original filename
- Internal name
- Private build
- Special build
- Comments

### 4.1.18 Manifest

The Manifest display shows the manifest that is used by the PE file.



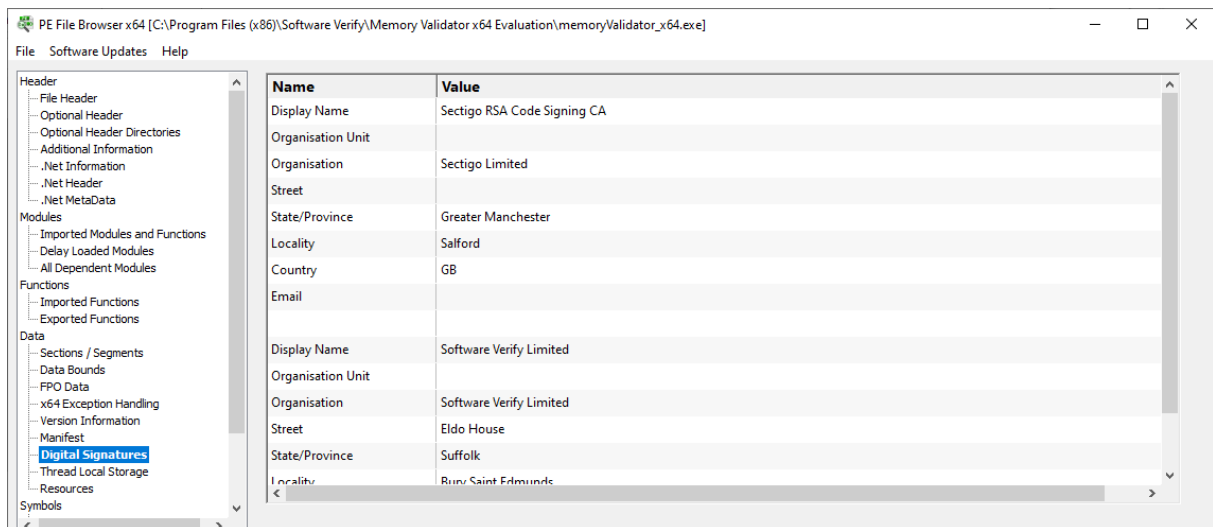
The screenshot shows the PE File Browser application window. The left sidebar is expanded to 'Manifest'. The main pane displays the following XML code:

```
1 <assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestversion="1.0">
2 <dependency>
3 <dependentAssembly>
4 <assemblyIdentity type="win32" name="Microsoft.Windows.Common-Controls" version="6.0.0.0" processor
5 </assemblyIdentity>
6 </dependentAssembly>
7 </dependency>
8 <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
9 <security>
10 <requestedPrivileges>
11 <requestedExecutionLevel level="asInvoker" uiAccess="false">
12 </requestedExecutionLevel>
13 </requestedPrivileges>
14 </security>
15 </trustInfo>
16 <application xmlns="urn:schemas-microsoft-com:asm.v3">
17 <windowsSettings>
18 <ms_windowsSettings:dpiAware xmlns:ms_windowsSettings="http://schemas.microsoft.com/SMI/2005/window
19 true
20 </ms_windowsSettings:dpiAware>
21 </windowsSettings>
22 </application>
23 </assembly>
24
25
26
27
28
29
30
31
```

The manifest will be reformatted to make it more readable if required. Embedded manifests have no formatting and are thus quite hard to read.

### 4.1.19 Digital Signatures

The Digital Signatures display shows the digital signature information in the PE file.

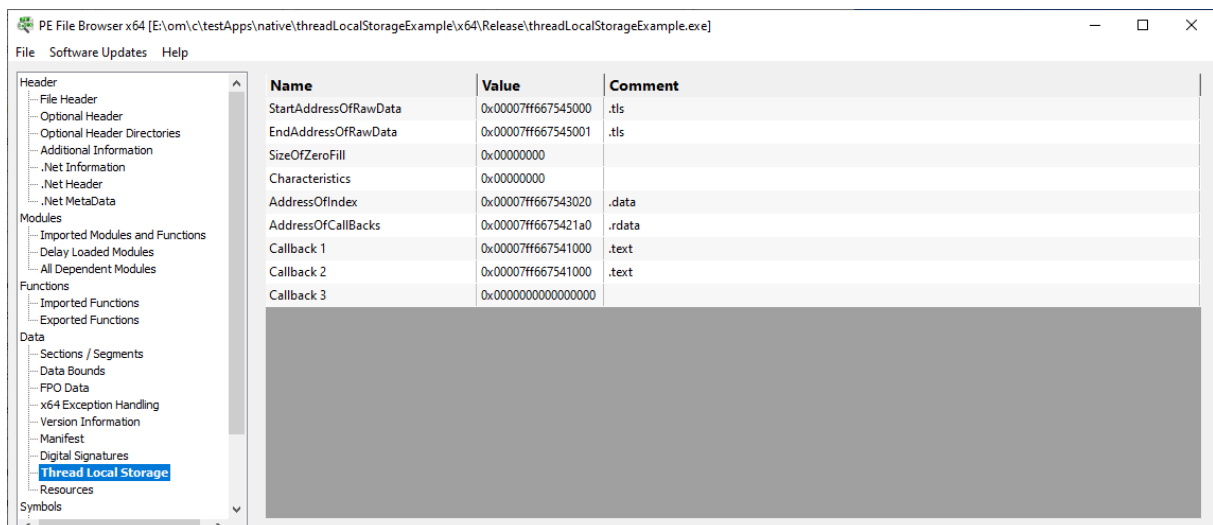


The digital signature is queried for the following information. Any queries that provide a result are displayed here.

- Display Name
- Organisation Unit
- Organisation
- Street
- State/Province
- Locality
- Country
- Email

#### 4.1.20 Thread Local Storage

The Thread Local Storage display shows the thread local storage in the PE file.

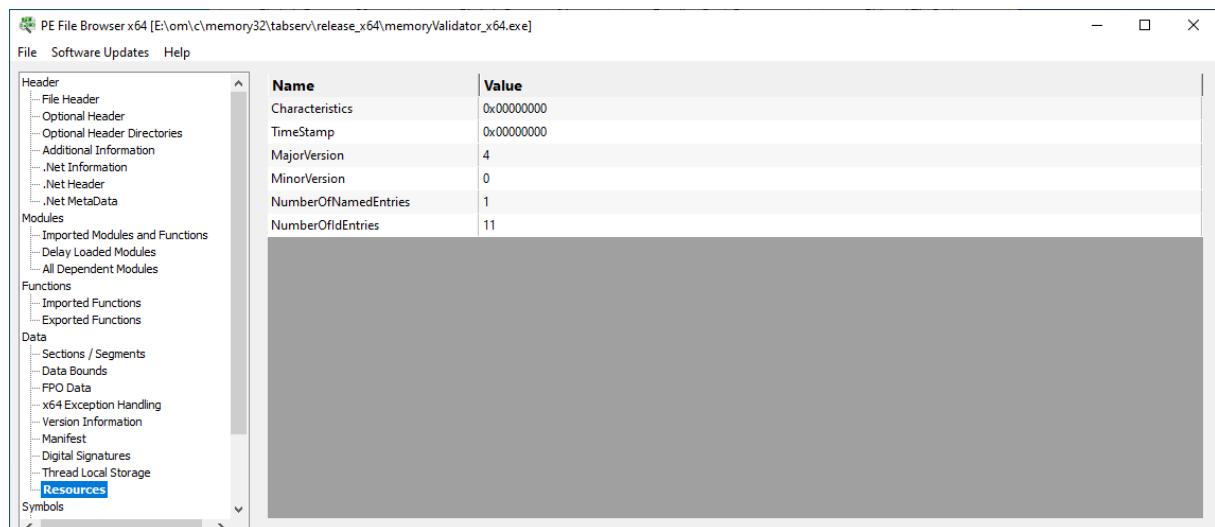


The thread local storage is queried for the following information. Any queries that provide a result are displayed here.

- StartAddressOfRawData
- EndAddressOfRawData
- SizeOfZeroFill
- Characteristics
- AddressOfIndex
- AddressOfCallBacks
- Callback 1
- Callback 2
- Callback 3

### 4.1.21 Resources

The Resources display shows the resources section in the PE file.

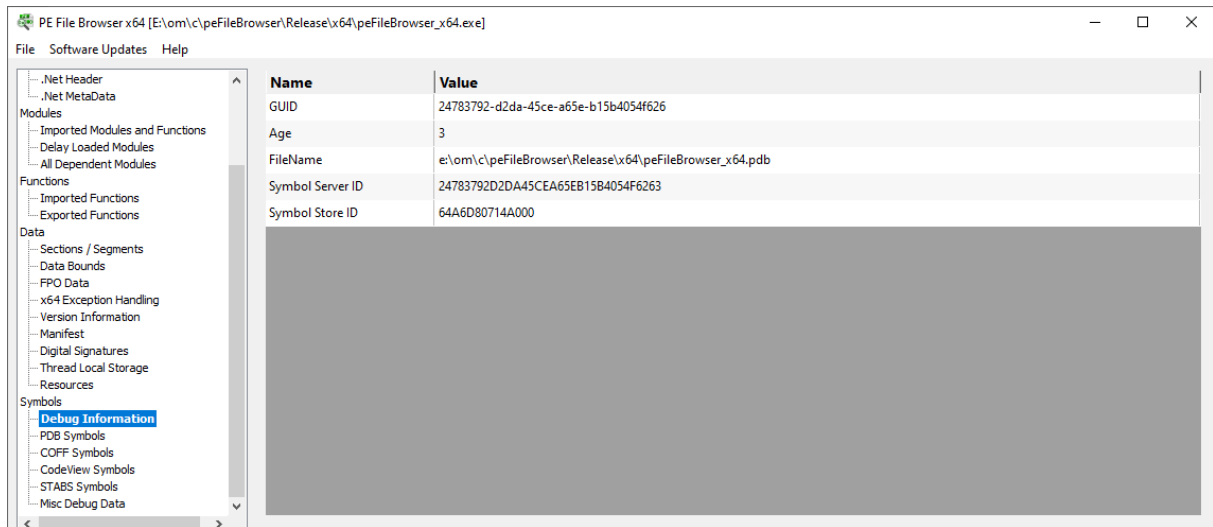


The resources header is queried for the following information. Any queries that provide a result are displayed here.

- Characteristics
- TimeStamp
- MajorVersion
- MinorVersion
- NumberOfNamedEntries
- NumberOfIdEntries

### 4.1.22 Debug Information

The Debug Information display shows the CV record for the module plus the symbol server id and local symbol store id.

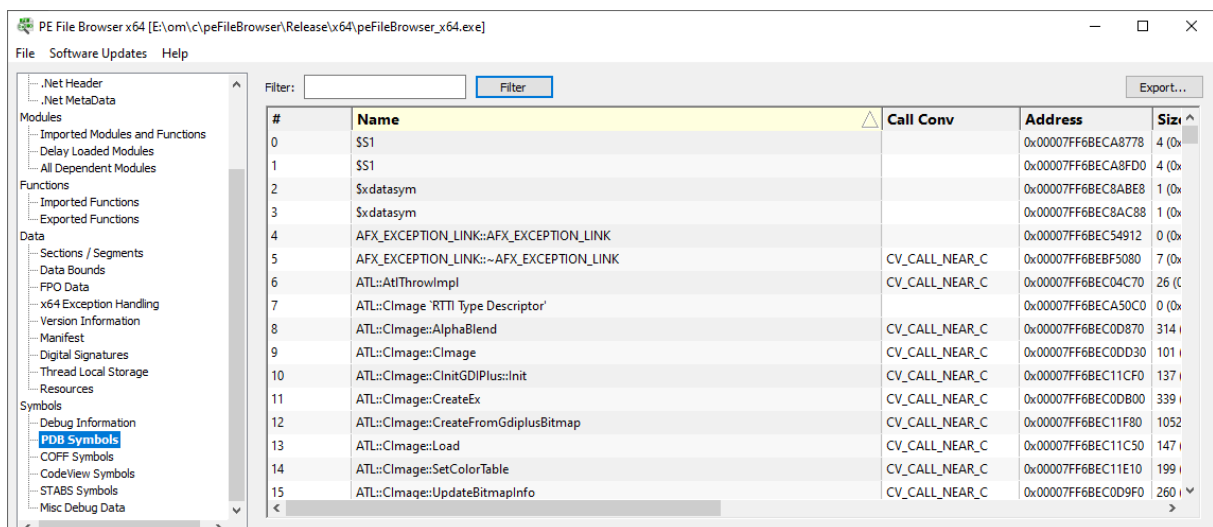


If the module contains a CV record the data in that record is shown here.

GUID	The globally unique identifier that identifies the module.
Age	The age of the module - this is the number of times the module has been linked after a clean or full rebuild.
Filename	The name of the PDB file at the time the module was built.
Symbol server id	The id that identifies a PDB file.
Symbol store id	The id that identifies a module.

### 4.1.23 PDB Symbols

The PDB Symbols display shows the all the PDB format symbols for the PE file.



### Columns

**#**

The ordinal of the symbol in the symbol data.

**Name**

The name of the symbol.

**Call Conv**

The calling convention for the symbol.

**Address**

The symbol address.

**Size**

The symbol size.

**Type**

The type of debug data.

Valid values are:

- SymTagNull
- SymTagExe
- SymTagCompiland
- SymTagCompilandDetails
- SymTagCompilandEnv
- SymTagFunction
- SymTagBlock
- SymTagData
- SymTagAnnotation
- SymTagLabel
- SymTagPublicSymbol
- SymTagUDT
- SymTagEnum
- SymTagFunctionType
- SymTagPointerType
- SymTagArrayType
- SymTagBaseType
- SymTagTypedef
- SymTagBaseClass
- SymTagFriend
- SymTagFunctionArgType
- SymTagFuncDebugStart
- SymTagFuncDebugEnd
- SymTagUsingNamespace
- SymTagVTableShape
- SymTagVTable
- SymTagCustom
- SymTagThunk
- SymTagCustomType
- SymTagManagedType
- SymTagDimension

**Flags**

Any flags that related to this debug information

**Line Number**

The line number for the symbol, if any.

**Filename**

The filename for the symbol, if any.

**Sorting**

The data can be sorted by any column. Click the column header to select sorting by that column. Click the same header to reverse the sort direction.

**Filtering**

To restrict the display to just symbols that match the filter, or symbols with a filename that matches the filter, enter your filter specification into the filter field then click **Filter**.

The filter can include the \* wildcard to match any sequence of characters.

Example filters:

```
Get*
*test*
*.cpp
```

To reset the filter, delete all text from the filter field then click **Filter**.

**Exporting Data**

To export the contents of the grid to a comma separated text file, click the **Export...** button then enter the name of the file you would like the data saved in.

**4.1.24 COFF Symbols**

The COFF Symbols display shows the all the COFF format symbols for the PE file.

The screenshot shows the PE File Browser application window. The left sidebar displays a tree view with 'COFF Symbols' selected. The main area contains a table of symbols with the following columns: Name, Address, Section, Type, Storage, Size, Lines, and File. The table lists various symbols including .bb, .bf, and .bss, with their respective addresses, sections, types, storage classes, sizes, and associated files.

Name	Address	Section	Type	Storage	Size	Lines	File
.bb	0x00002fa	Section 1	void	Block	0 (0x0000)		main.c
.bb	0x0000336	Section 1	void	Block	0 (0x0000)		functi
.bf	0x00002d0	Section 1	void	Function	0 (0x0000)		main.c
.bf	0x0000330	Section 1	void	Function	0 (0x0000)		functi
.bss	0x0000008	Section 4	int	Static	0 (0x0000)		new_o
.bss	0x0000008	Section 4	int	Static	0 (0x0000)		eh_pe
.bss	0x000040a8	Section 4	int	Static	0 (0x0000)		crstuf
.bss	0x000040a8	Section 4	int	Static	0 (0x0000)		pseud
.bss	0x000040a8	Section 4	int	Static	0 (0x0000)		crstuf
.bss	0x000040a8	Section 4	int	Static	0 (0x0000)		fake
.bss	0x000040a8	Section 4	int	Static	0 (0x0000)		fake
.bss	0x000040a8	Section 4	int	Static	0 (0x0000)		crstuf
.bss	0x000040a8	Section 4	int	Static	0 (0x0000)		pseud
.bss	0x00000000	Section 4	int	Static	0 (0x0000)		crt1.c
.bss	0x000040a8	Section 4	int	Static	0 (0x0000)		pseud
.bss	0x000040a8	Section 4	int	Static	0 (0x0000)		fake

**Columns**

**Name**

The name of the symbol.

**Address**

The relative virtual address of the symbol.

**Section**

The symbol section number.

**Type**

The datatype for the symbol.

**Storage**

The storage class for the symbol.

**Size**

The size of the symbol.

**Lines**

Number of lines for this function (blank if not available)

**Filename**

The symbol filename.

**Sorting**

The data can be sorted by any column. Click the column header to select sorting by that column. Click the same header to reverse the sort direction.

**Filtering**

To restrict the display to just symbols that match the filter, or symbols with a filename that matches the filter, enter your filter specification into the filter field then click **Filter**.

The filter can include the \* wildcard to match any sequence of characters.

Example filters:

Get\*

\*test\*

\*.cpp

To reset the filter, delete all text from the filter field then click **Filter**.

**Exporting Data**

To export the contents of the grid to a comma separated text file, click the **Export...** button then enter the name of the file you would like the data saved in.

**4.1.25 CodeView Symbols**

The CodeView Symbols display shows the all the CodeView format symbols for the PE file.

Name	Type	Address	Offset	Line Number	Filename
	Line Offset	0x00000000	0x000077a3	279	file_io.c
	Data	0xfd5a05b9	0x00000000	0	math_x87.obj
	Data	0x65c1ef1f	0x00000000	0	math_x87.obj
	Data	0x65c1ef1f	0x00000000	0	math_x87.obj
	Data	0x65c1ef1f	0x00000000	0	locale.obj
	Data	0x65c1ef1f	0x00000000	0	strtold.obj
	Data	0x65c1ef1f	0x00000000	0	math_x87.obj
	Data	0x7ce36145	0x00000000	0	wcstold.obj
	Data	0x7ce36145	0x00000000	0	strtold.obj
	Data	0x628bbaf4	0x00000000	0	strtold.obj
	Data	0x628bbaf4	0x00000000	0	ThreadLocalData.obj
	Data	0x99573ab6	0x00000000	0	locale.obj
	Data	0xbf50bbd0	0x00000000	0	locale.obj
	Data	0xc8e726b7	0x00000000	0	locale.obj
	Data	0x9a5be5b7	0x00000000	0	locale.obj
	Data	0x9b9a45b7	0x00000000	0	locale.obj

## Columns

### Name

The name of the symbol.

### Type

The type of the symbol.

### Address

The relative virtual address of the symbol.

### Offset

The symbol offset.

### Line Number

The line number, if any, of the symbol.

### Filename

The filename of the symbol.

### Sorting

The data can be sorted by any column. Click the column header to select sorting by that column. Click the same header to reverse the sort direction.

## Filtering

To restrict the display to just symbols that match the filter, or symbols with a filename that matches the filter, enter your filter specification into the filter field then click **Filter**.

The filter can include the \* wildcard to match any sequence of characters.

Example filters:

Get\*

\*test\*

\*.cpp

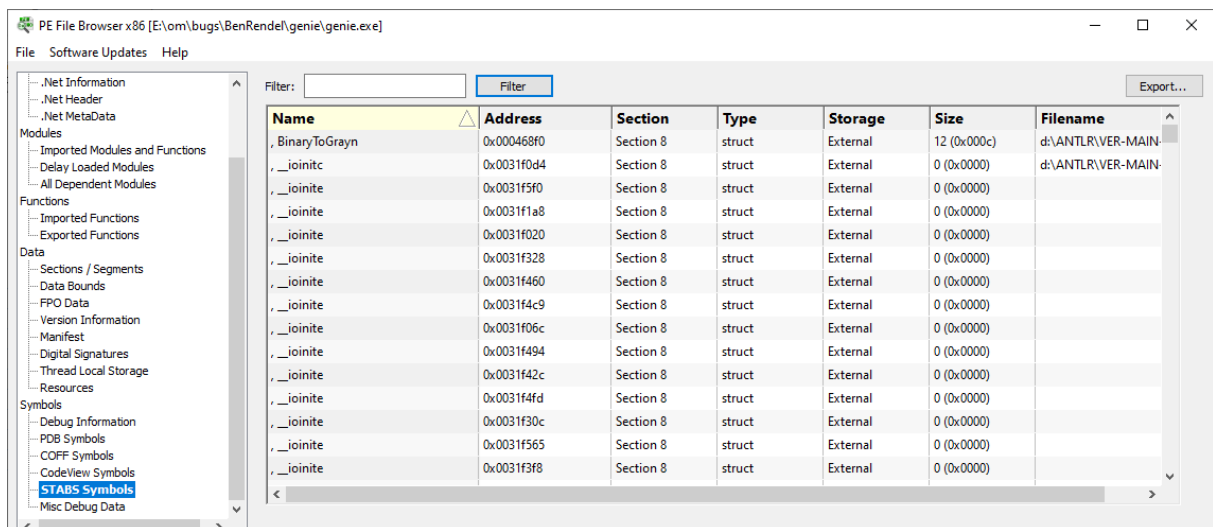
To reset the filter, delete all text from the filter field then click **Filter**.

## Exporting Data

To export the contents of the grid to a comma separated text file, click the **Export...** button then enter the name of the file you would like the data saved in.

### 4.1.26 STABS Symbols

The STABS Symbols display shows the all the STABS format symbols for the PE file.



## Columns

### Name

The name of the symbol.

### Address

The relative virtual address of the symbol.

### Section

The symbol section number.

### Type

The datatype for the symbol.

### Storage

The storage class for the symbol.

### Size

The size of the symbol.

### Filename

The symbol filename.

### Sorting

The data can be sorted by any column. Click the column header to select sorting by that column. Click the same header to reverse the sort direction.

### Filtering

To restrict the display to just symbols that match the filter, or symbols with a filename that matches the filter, enter your filter specification into the filter field then click **Filter**.

The filter can include the \* wildcard to match any sequence of characters.

Example filters:

```
Get*  
*test*  
*.cpp
```

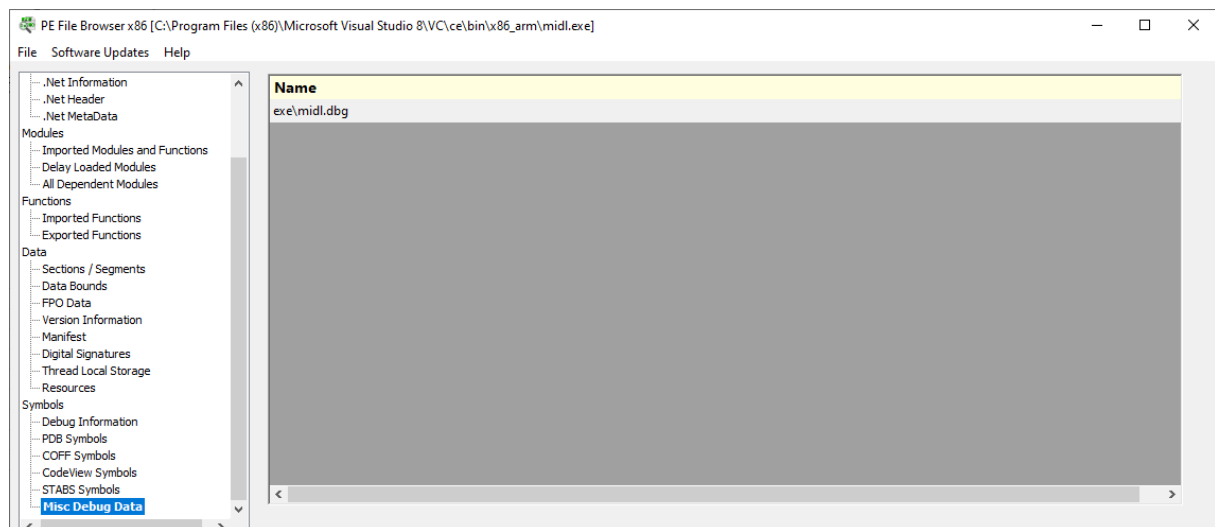
To reset the filter, delete all text from the filter field then click **Filter**.

### Exporting Data

To export the contents of the grid to a comma separated text file, click the **Export...** button then enter the name of the file you would like the data saved in.

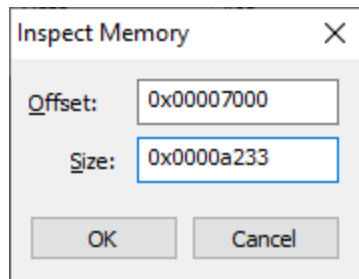
## 4.1.27 Misc Debug Data

The Misc Debug Data display shows the all the Miscellaneous debug symbols for the PE file.



## 4.2 View Memory Dialog

The Inspect Memory dialog is shown below.



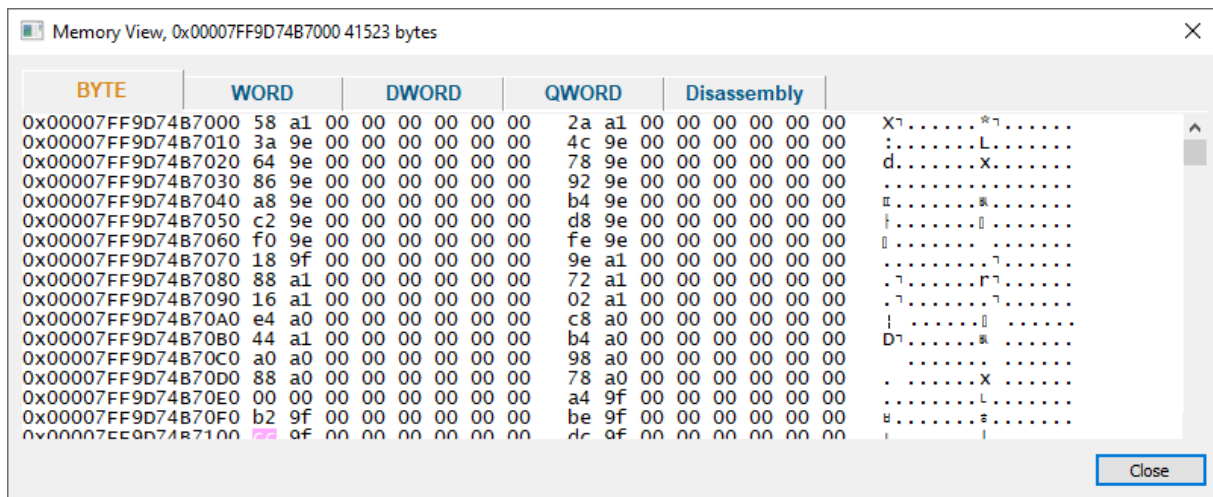
**Offset** > an offset inside the DLL to start viewing memory

**Size** > how many bytes to view

Specify an offset inside the DLL and the number of bytes to view.

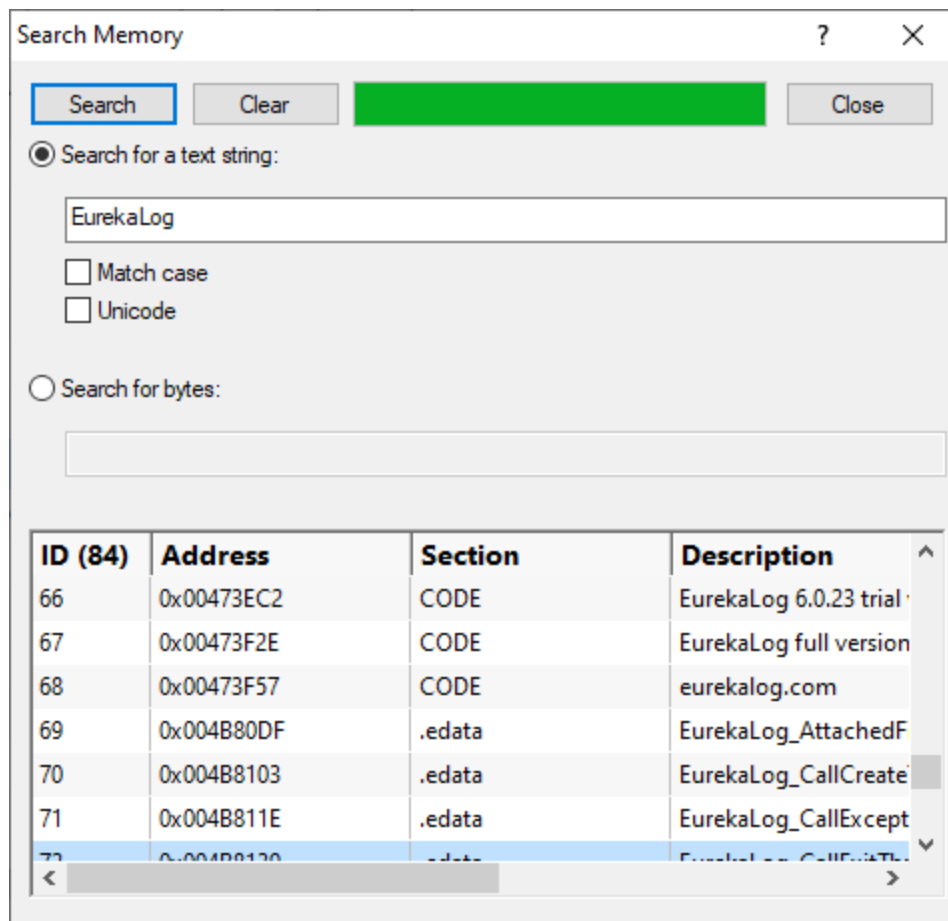
If the memory is readable, the memory will be displayed.

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



## 4.3 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

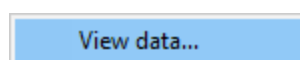
**Unicode** > select the check box if the string match should be Unicode. If the check box is not selected the string match is ANSI

**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

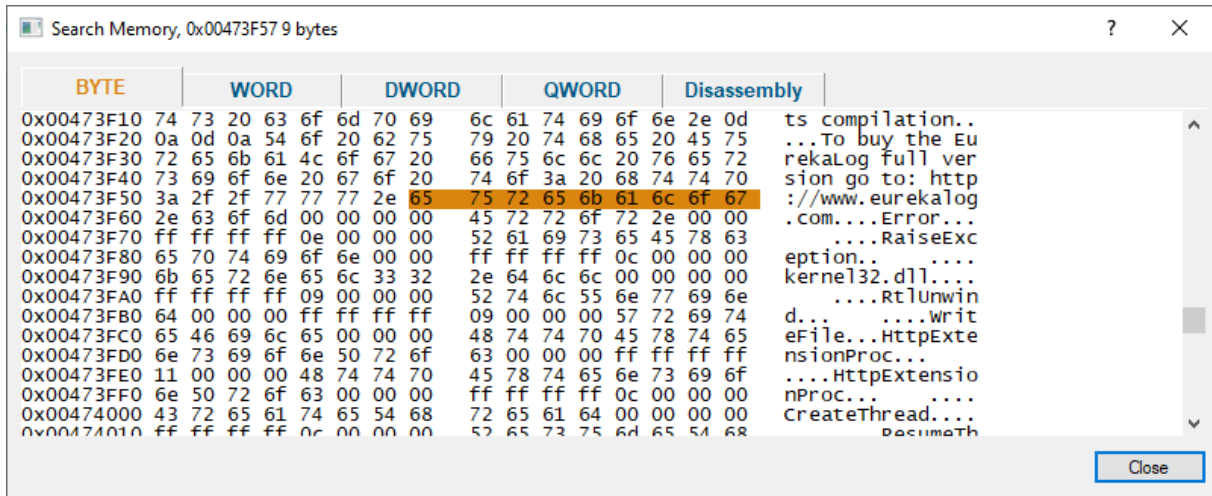
**Search** > perform the search. The progress of the search is shown on the progress bar, any matching search results are shown in the list.

**Clear** > clear the search results

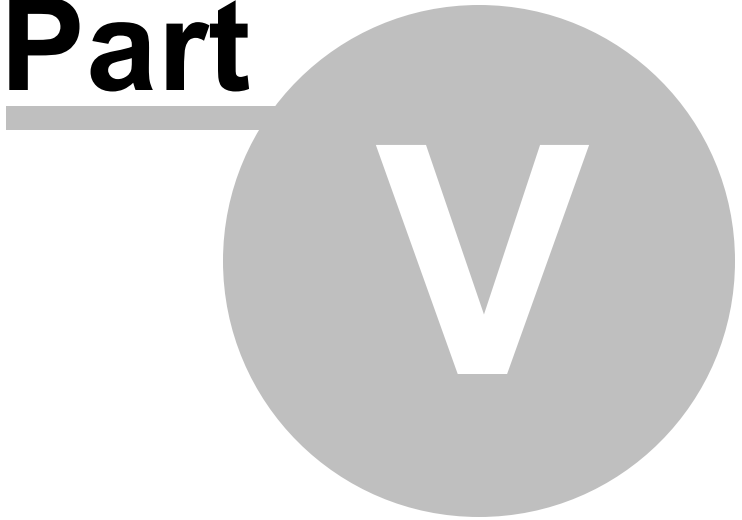
A context menu on the search results provides a single option:



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



**Part**



## 5 Command Line Interface

PE File Browser can be used from the command line as well as with the GUI.

The command line options allow you to launch PE File Browser from another an application so that you can display PE file internals for a specific DLL.

### 5.1 Alphabetic Reference

#### **/dll**

Specifies the PE File that will be displayed

`/dll path-to-executable`

Example: `/dll e:\om\c\test\release\test.dll`

#### **/exit**

Causes PE File Browser to exit after exporting modules

#### **/exportModules**

Specifies the filename to export the list of dependent modules to

`/exportModules path-to-export-filename`

Example: `/exportModules e:\om\export\dependentModules.txt`

#### **/exportModulesAndFunctions**

Specifies the filename to export the list of dependent modules and functions to

`/exportModulesAndFunctions path-to-export-filename`

Example: `/exportModulesAndFunctions e:\om\export\dependentModules.txt`

#### **/exportModulesTopLevel**

Specifies the filename to export the list of dependent modules to. Only the immediate dependent modules are exported. No descendant dependencies are exported.

`/exportModulesTopLevel path-to-export-filename`

Example: `/exportModulesTopLevel e:\om\export\dependentModules.txt`

## 5.2 Usage Reference

### Displaying a DLL

**/dll**

Specifies the PE File that will be displayed

`/dll path-to-executable`

Example: `/dll e:\om\c\test\release\test.dll`

### Export the modules that are dependent on the DLL

Specifies the filename to export the list of dependent modules to

`/exportModulesTopLevel path-to-export-filename`

Example: `/exportModulesTopLevel e:\om\export\dependentModules.txt`

### Export the modules (and their dependent modules) that are dependent on the DLL

Specifies the filename to export the list of dependent modules to

`/exportModules path-to-export-filename`

Example: `/exportModules e:\om\export\dependentModules.txt`

### Export the modules (and their dependent modules) and functions that are dependent on the DLL

Specifies the filename to export the list of dependent modules and functions to

`/exportModulesAndFunctions path-to-export-filename`

Example: `/exportModulesAndFunctions e:\om\export\dependentModules.txt`

### Close PE File Browser after exporting

`/exit`



