DbgHelp Browser

by

Software Verify

Software Verify
Troubleshoot your software

DbgHelp Browser

Visual Studio PDB contents inspector

by Software Verify Limited

Welcome to the DbgHelp Browser software tool. DbgHelp Browser is a software tool that allows you to inspect the contents of PDB files.

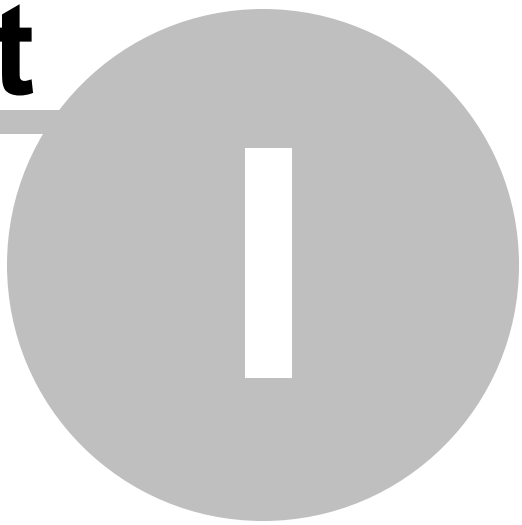
We hope you will find this document useful.

DbgHelp Browser Help

Printed: June 2024 in United Kingdom.

Table of Contents

Part



1 How to get DbgHelpBrowser

DbgHelpBrowser is free for commercial use. DbgHelpBrowser can be downloaded from Software Verify's website at https://www.softwareverify.com/product/dbghelp-browser/

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

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

Whilst DbgHelpBrowser is free for commercial use, DbgHelpBrowser 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.

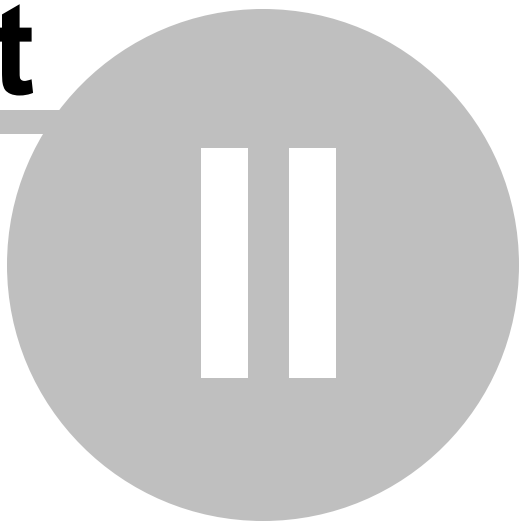
Contact Software Verify at:

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

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 DbgHelpBrowser do?

DbgHelpBrowser allows you to inspect the contents of a PDB (Program Database) file.

You can sort the data, filter the data by name or by type of data.

You can also query the data by address which can be useful for identifying what function is at a given address if all you have is a crash address and nothing else.

Query by address is supported four ways:
- Query by absolute address.
- Query by address offset from a DLL load address.
- Query by address offset from a symbol.
- Query using XML data from the Windows Event Log.

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 DbgHelp Browser is automatically started.

Native, .Net, .Net Core

PDB files created for native executables, for .Net executables, and for .Net Core executables (using the Portable PDB format) are all supported.

.Net Core

To provide method names and parameter information for .Net Core executables the computer DbgHelpBrowser is running on needs to have .Net Core installed.

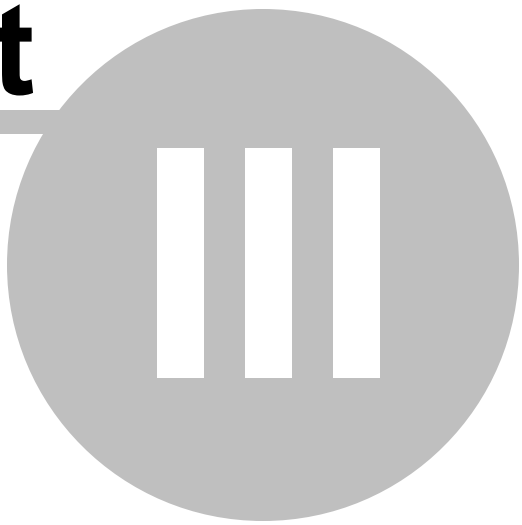
Without .Net Core installed the .Net Core metadata can't be read.

Method Token, filename and line number data will be available, but method names won't be available if .Net Core is not installed.

History

DbgHelpBrowser 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 What is a module?

A module is a block of executable code and data. For example, a DLL or EXE.

Some software vendors name their DLLs with different file extensions, for example .BPL, .ARX.

When you call LoadLibrary to load a module, you are returned a HMODULE, which is an opaque handle to a module. The HMODULE is most often the same as the module load address, but not always.
The lower few bits of the HMODULE can get OR'd with some flags to create a HMODULE value that is not the same as the module load address.

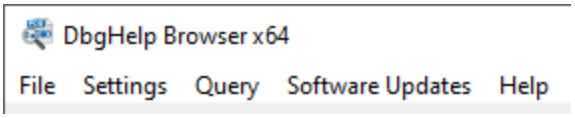
You can get the load address of a module from it's HMODULE by masking out the lower 16 bits of the HMODULE value then casting to a DWORD_PTR.

In this documentation when you read EXE or DLL or module, we are effectively referring to the same thing. It's easier to read and write "DLLs" rather than "DLLs or EXE".

Part IV

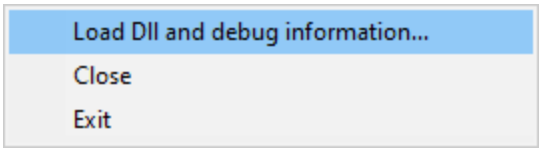
4 Menu

The main menu contains five menus, File, Settings, Query, Software Updates and Help.



4.1 File

The File menu controls loading of DLLs and debug information, clearing the display and exiting the program.



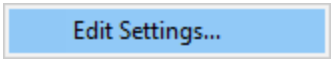
File menu > **Load Dll and debug information...** > loads a DLL and the debug information and displays it.

File menu > **Close** > clear all results, unloads the DLL and debug information.

File menu > **Exit** > closes DbgHelpBrowser.

4.2 Settings

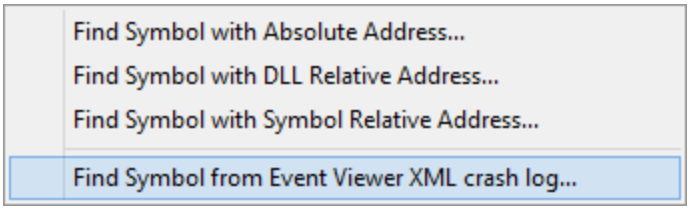
The Edit menu controls editing settings.



Settings menu > **Edit Settings...** > displays the settings dialog.

4.3 Query

The Query menu controls searching for symbols.



Query menu > **Find Symbol with Absolute Address...** > use this option to turn an absolute address in a process into a symbol, filename and line number.

See Decoding an absolute crash address for more details.

Query menu > **Find Symbol with DLL Relative Address...** > use this option to turn a relative address inside a DLL into a symbol, filename and line number.

See Decoding a relative crash address for more details.

Query menu > **Find Symbol with Symbol Relative Address...** > use this option to turn an address that is relative to a symbol inside a DLL into a symbol, filename and line number.

See Decoding a symbol relative crash address for more details.

Query menu > **Find Symbol from Event Viewer XML crash log...** > use this option to turn an XML crash log from the Microsoft Event Viewer to a symbol inside a DLL into a symbol, filename and line number.

See Decoding an Event Viewer XML crash log for more details.

.Net, .Net Core

The query options are not available for .Net and .Net executables as there is no direct translation from a crash address/offset to a .Net symbol.

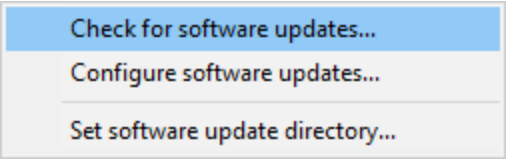
Without having access to the compiled .Net method address and the compiled address to ILASM instruction offset data it is impossible to translate crash addresses/offsets to .Net functions.

The compiled address to ILAMS instruction offset data is only available in the context of a running .Net application attached to a .Net debugger or a .Net profiler.

4.4 Software Updates

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

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

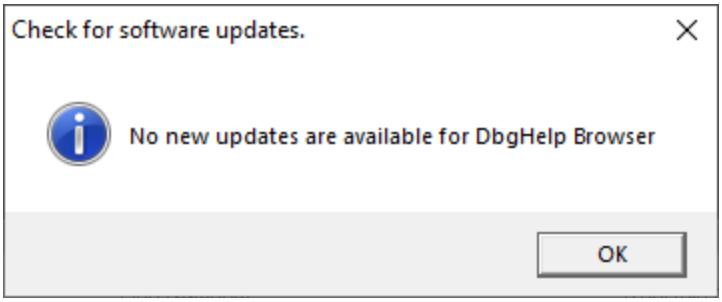


Software Updates menu > **Check for software updates** > checks for updates and shows the software update dialog if any exist

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

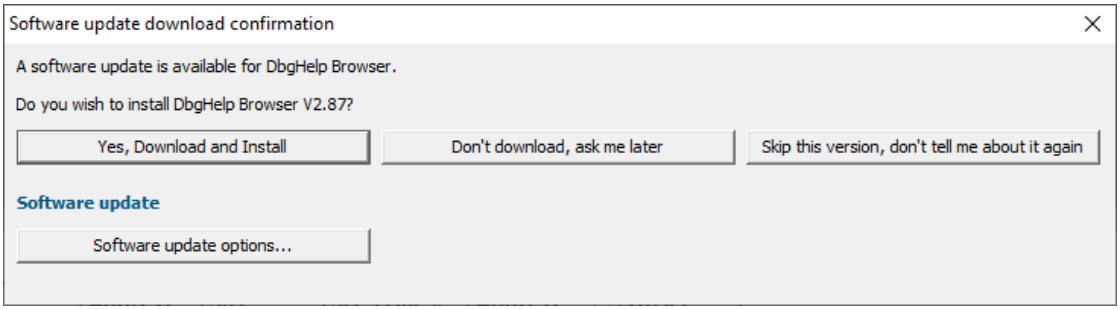
Before updating the software, close the help manual, and end any active session by closing target programs.

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

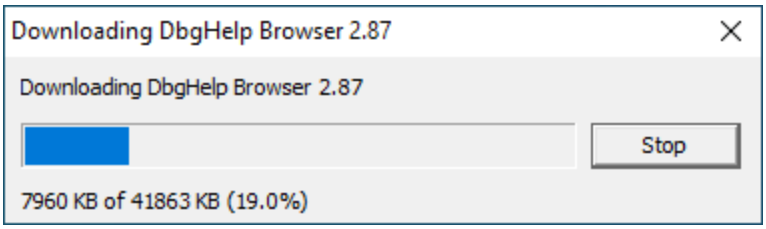


Software Update dialog

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



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



  Once the update has downloaded, DbgHelp 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 DbgHelp 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 DbgHelp Browser as follows:

- Ensure DbgHelp 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

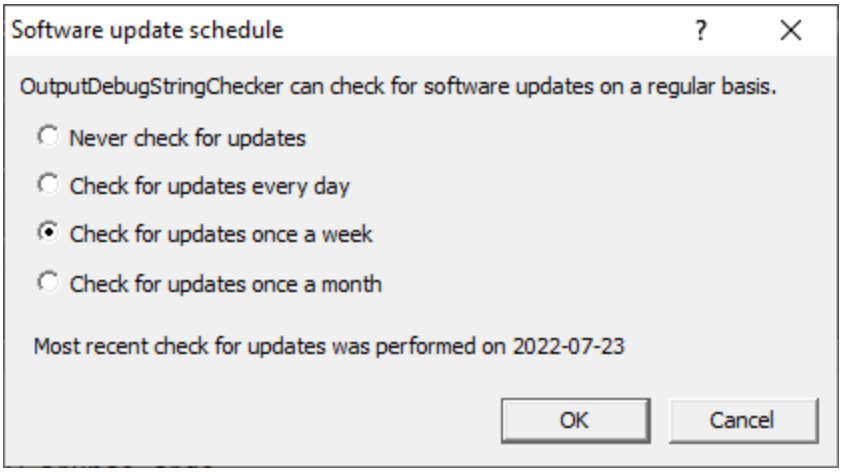
DbgHelp Browser can automatically check to see if a new version of DbgHelp Browser is available for downloading.

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.

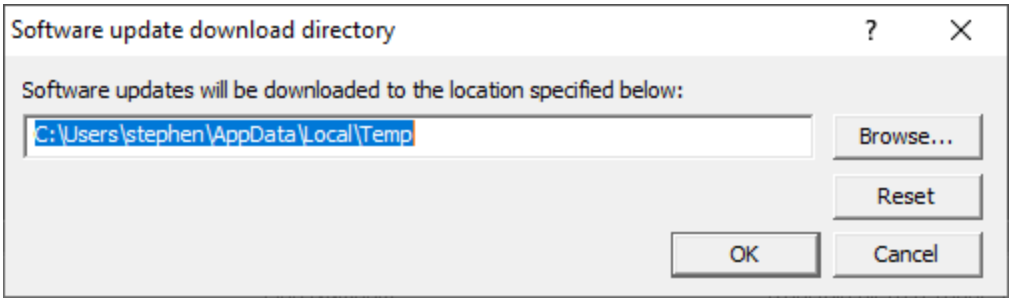


Software update directory

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)

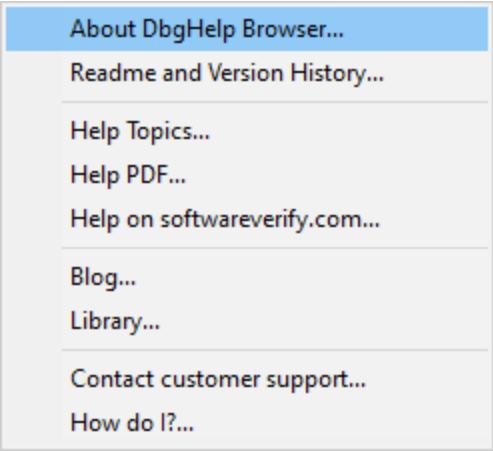
When modifying the download directory, you should ensure the directory will continue to be valid. Updates may no longer occur if the download location is later invalidated.

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

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

4.5 Help

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



Help menu > **About DbgHelp Browser...** > displays information about DbgHelp Browser.

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

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

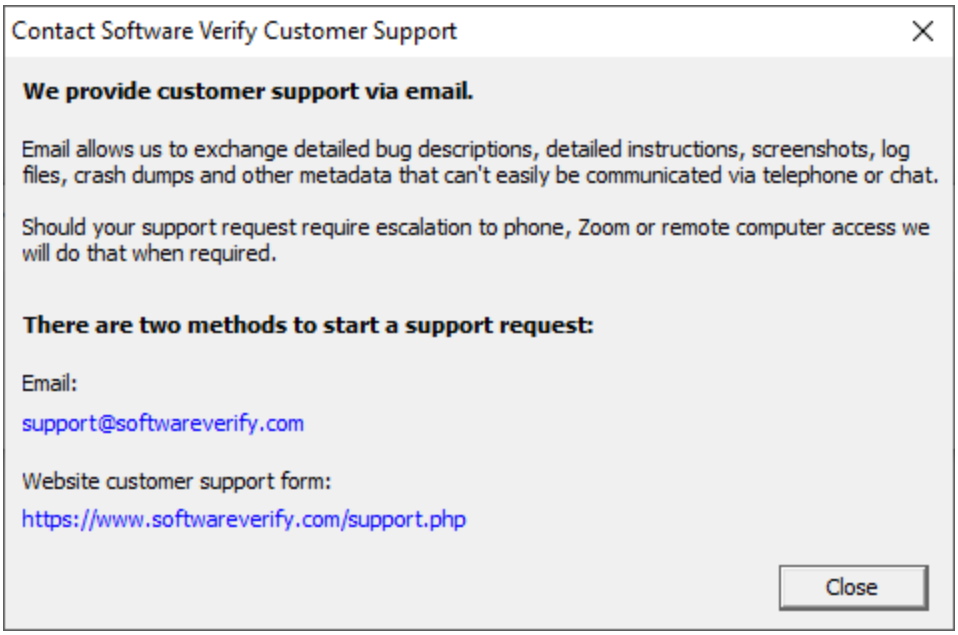
Help menu > **Help PDF...** > displays this help file in PDF format.

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.

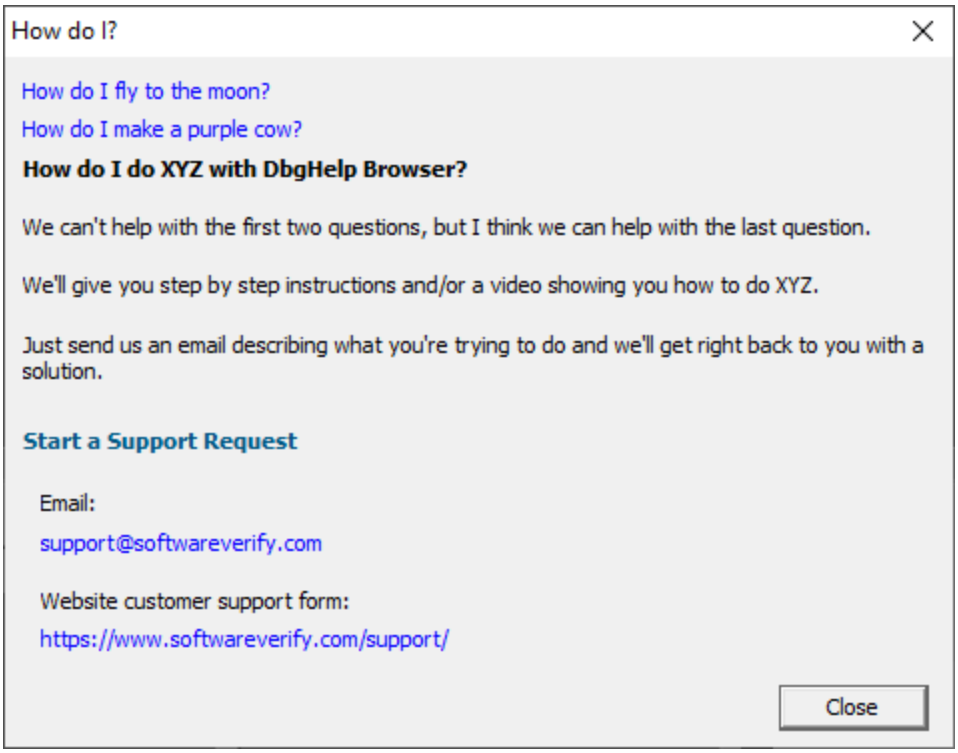
Help menu > **Library...** > display the Software Verify library - our best blog articles grouped by related topics.

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



Click a link to contact customer support.

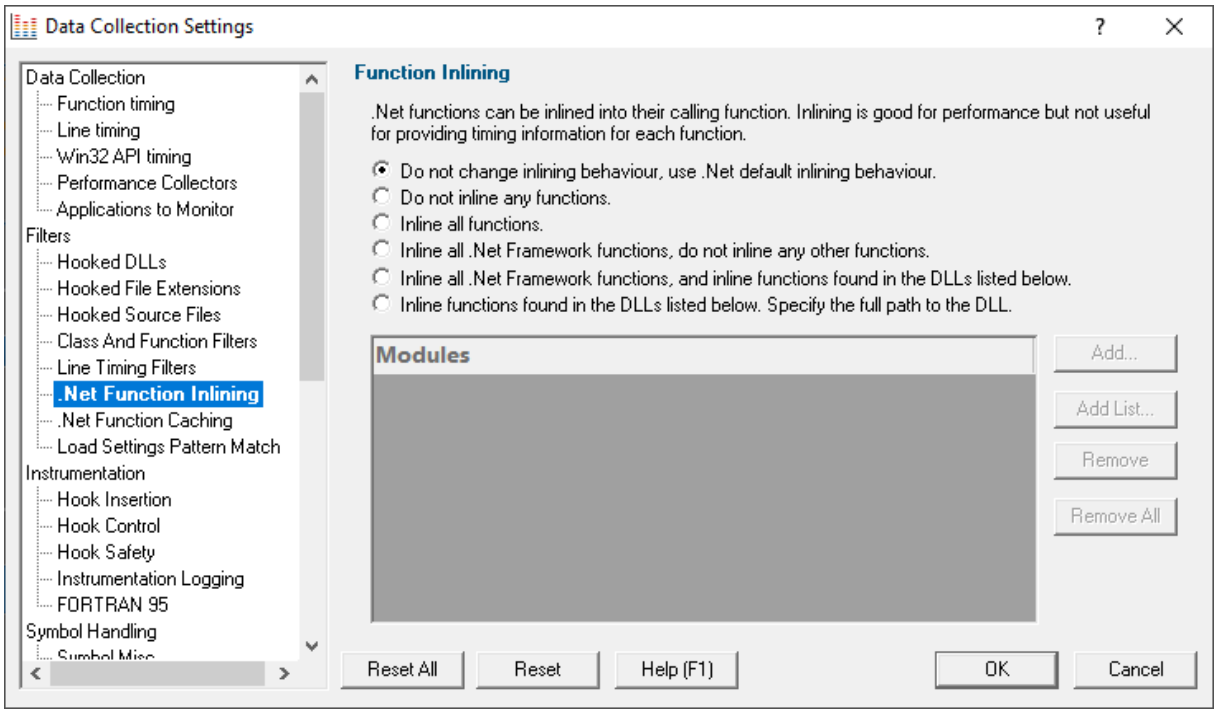
Help menu > **How do I?...** > displays the options for asking us how to do a particular task.



Part V

5 The user interface

The DbgHelpBrowser user interface is shown below.



The user interface consists of a main grid showing all main datatypes and functions in the debug help.

Below is a display for function parameters, function local variables, line numbers and a source code display for viewing the source code of any function or variable that is selected.

Selecting any item in the grid populates the lower grids and source code display as appropriate.

Querying any value will select the nearest item in the main grid and populate the other displays as appropriate.

Some basic filtering functionality is also provided.

PDB Information

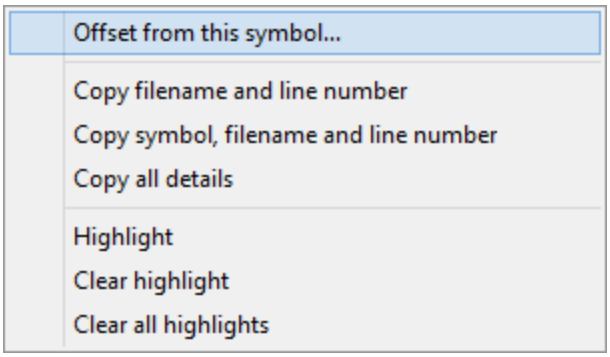
#	Name (7275)	Call Conv	Address	Size	Type	Flags	Filename
928	CFileFiltersSettingsDialog::OnButtonRemoveAll	CV_CALL_THISCALL	0x08A11FA0	67	SymTagFunction		e:\om\c\threadlockchecker\filefilt
929	CFileFiltersSettingsDialog::OnCancel	CV_CALL_THISCALL	0x08A11940	5	SymTagFunction		e:\om\c\threadlockchecker\filefilt
930	CFileFiltersSettingsDialog::OnDblGridClick	CV_CALL_THISCALL	0x08A12030	21	SymTagFunction		e:\om\c\threadlockchecker\filefilt
931	CFileFiltersSettingsDialog::OnHelpInfo	CV_CALL_THISCALL	0x08A11A50	29	SymTagFunction		e:\om\c\threadlockchecker\filefilt
932	CFileFiltersSettingsDialog::OnInitDialog	CV_CALL_THISCALL	0x08A120A0	406	SymTagFunction		e:\om\c\threadlockchecker\filefilt
933	CFileFiltersSettingsDialog::OnLGridClick	CV_CALL_THISCALL	0x08A11FF0	21	SymTagFunction		e:\om\c\threadlockchecker\filefilt
934	CFileFiltersSettingsDialog::OnOK	CV_CALL_THISCALL	0x08A11950	5	SymTagFunction		e:\om\c\threadlockchecker\filefilt
935	CFileFiltersSettingsDialog::OnRGridClick	CV_CALL_THISCALL	0x08A12010	21	SymTagFunction		e:\om\c\threadlockchecker\filefilt
936	CFileFiltersSettingsDialog::OnSelChanged	CV_CALL_THISCALL	0x08A12050	21	SymTagFunction		e:\om\c\threadlockchecker\filefilt
937	CFileFiltersSettingsDialog::OnVKeyToItem	CV_CALL_THISCALL	0x08A12070	30	SymTagFunction		e:\om\c\threadlockchecker\filefilt
938	CFileFiltersSettingsDialog::`RTTI Base Class Array'		0x08A4B088	0	SymTagPublicSymbol		
939	CFileFiltersSettingsDialog::`RTTI Base Class Descriptor at (0,-1,0,64)'		0x08A4B0A8	0	SymTagPublicSymbol		
940	CFileFiltersSettingsDialog::`RTTI Class Hierarchy Descriptor'		0x08A4B078	0	SymTagPublicSymbol		

The PDB information shows you the symbol name, calling convention, symbol address, symbol size, symbol type, and associated debugging flags and the filename and line number for the symbol.

You can sort the data by clicking no the column header and clicking again to reverse the direction of the sort.

If you select any item in the grid the lower grids and source code display are populated with data as appropriate.

If you right click any item a context is displayed which will allow you to perform a symbol relative query.



Parameters

#	Name (3)	Tag	Address	Scope	Siz
1	btUInt	SymTagBaseType	0x00000008	4	
2	CListBox	SymTagPointerType	0x0000000C	4	
3	btUInt	SymTagBaseType	0x00000010	4	

The parameters section lists all parameters for the selected symbol. For each parameter, the name, tag type, address, scope, size and flags are displayed.

Local Variables

#	Name (1)	Tag	Address	Scope	Siz
1	CFileFiltersSettingsDialog	SymTagPointerType	0x00000000	4	

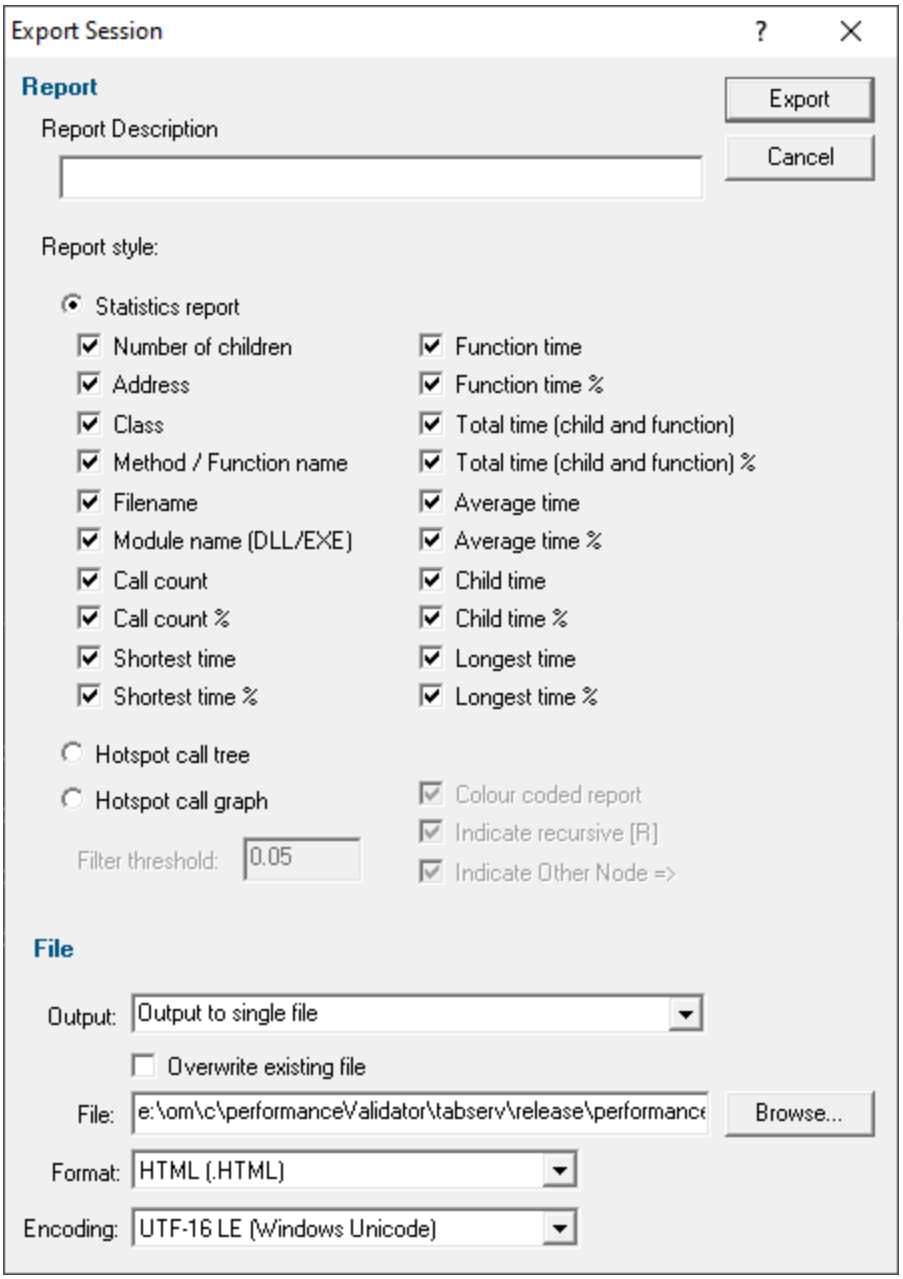
The parameters section lists all local variables for the selected symbol. For each parameter, the name, tag type, address, scope, size and flags are displayed.

Line Numbers

Line #...	Address	Offset
379	0x08A12070	0
380	0x08A12073	3
382	0x08A12079	9
383	0x08A1207E	14
389	0x08A12083	19
388	0x08A12087	23

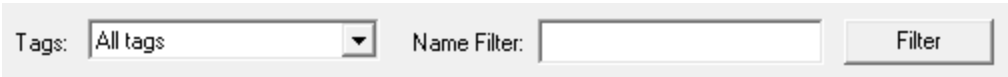
The line numbers section lists each line number, the address of that line and the offset of that line from the start of the owning function. Note that offsets can be negative as well as positive depending on how the compiler did it's work.

Source Code



The source code section displays the source code, highlights the selected line and displays information relating to filename, line number, function and address.

Filters

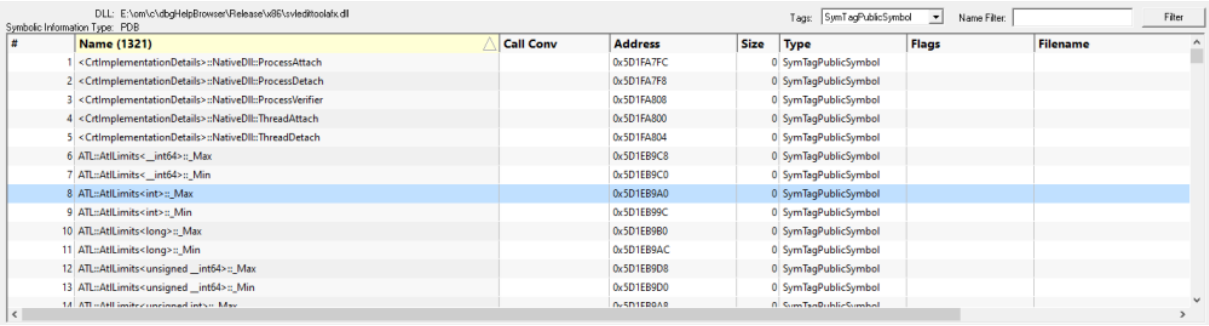


The filters section allows you to filter data in two ways:

Tags

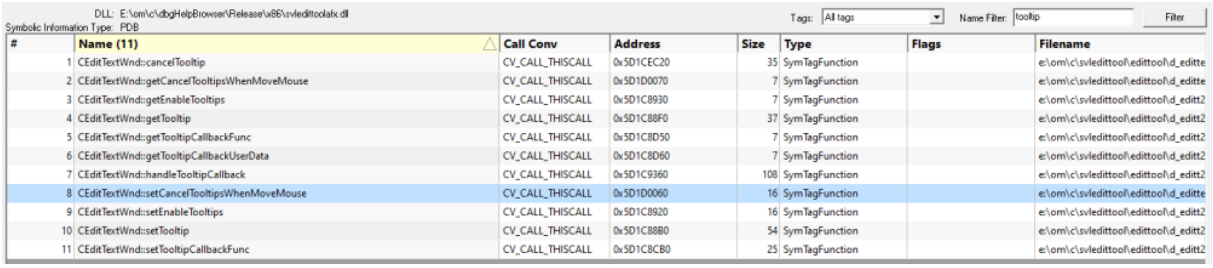
Filtering by tag allows you to reduce the amount of data to just the symbols that have the tag you choose. Available tags are:

All tags
Executable code
Data
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



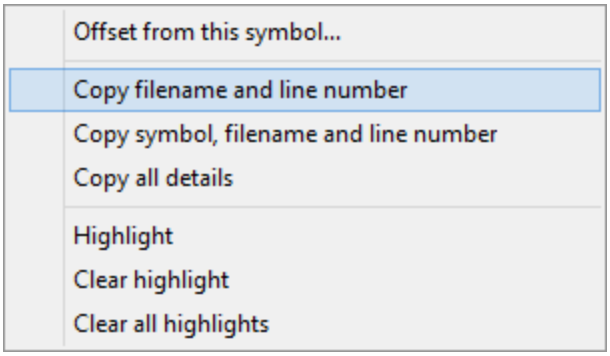
Symbol Name

Filtering by symbol name allows you to easily find a particular symbol. This is very useful when wanting to decode a crash address that has been provided as relative to a symbol (symbol + offset).


DLL: E:\om\c\dbgHelpBrowser\Release\x86\svledittoolafx.dll
Symbolic Information Type: PDB
Tags: All tags Name Filter: tooltip Filter

#	Name (11)	Call Conv	Address	Size	Type	Flags	Filename
1	CEditTextWnd::cancelTooltip	CV_CALL_THISCALL	0x5D1CEC20	35	SymTagFunction		e:\om\c\svledittool\edittool\d_editte
2	CEditTextWnd::getCancelTooltipsWhenMoveMouse	CV_CALL_THISCALL	0x5D1D0070	7	SymTagFunction		e:\om\c\svledittool\edittool\d_editte
3	CEditTextWnd::getEnableTooltips	CV_CALL_THISCALL	0x5D1C8930	7	SymTagFunction		e:\om\c\svledittool\edittool\d_editt2
4	CEditTextWnd::getTooltip	CV_CALL_THISCALL	0x5D1C88F0	37	SymTagFunction		e:\om\c\svledittool\edittool\d_editt2
5	CEditTextWnd::getTooltipCallbackFunc	CV_CALL_THISCALL	0x5D1C8D50	7	SymTagFunction		e:\om\c\svledittool\edittool\d_editt2
6	CEditTextWnd::getTooltipCallbackUserData	CV_CALL_THISCALL	0x5D1C8D60	7	SymTagFunction		e:\om\c\svledittool\edittool\d_editt2
7	CEditTextWnd::handleTooltipCallback	CV_CALL_THISCALL	0x5D1C9360	108	SymTagFunction		e:\om\c\svledittool\edittool\d_editt2
8	CEditTextWnd::setCancelTooltipsWhenMoveMouse	CV_CALL_THISCALL	0x5D1D0060	16	SymTagFunction		e:\om\c\svledittool\edittool\d_editte
9	CEditTextWnd::setEnableTooltips	CV_CALL_THISCALL	0x5D1C8920	16	SymTagFunction		e:\om\c\svledittool\edittool\d_editt2
10	CEditTextWnd::setTooltip	CV_CALL_THISCALL	0x5D1C88B0	54	SymTagFunction		e:\om\c\svledittool\edittool\d_editt2
11	CEditTextWnd::setTooltipCallbackFunc	CV_CALL_THISCALL	0x5D1C8CB0	25	SymTagFunction		e:\om\c\svledittool\edittool\d_editt2

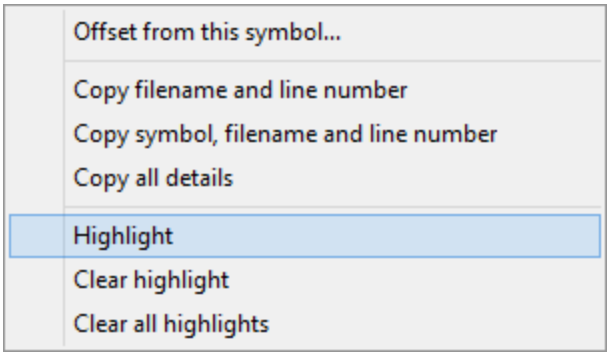
Clipboard



Options on the context menu to allow you to copy the following information to the clipboard:

- Filename and line number. `e:\om\c\svledittool\edittool\undotext.cpp 64`
- Symbol, filename and line number. `UndoText::doUndo e:\om\c\svledittool\edittool\undotext.cpp 64`
- All symbol details. `30 UndoText::doUndo CV_CALL_THISCALL 0x6C168230 540 SymTagFunction e:\om\c\svledittool\edittool\undotext.cpp 64`

Highlighting



Options on the context menu allow you to highlight multiple symbols, and to remove highlights.

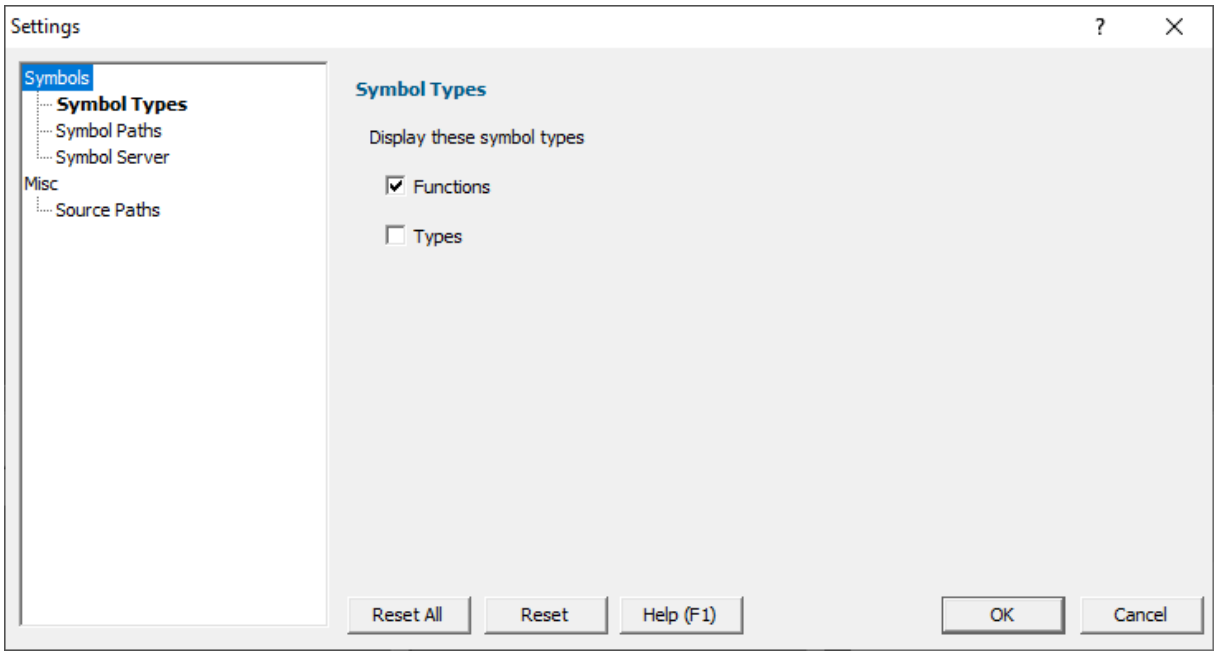
Highlighting can useful when you want to easily mark a symbol for future reference. Here's an example image showing some symbols that have been highlighted.

#	Name (7275)	Call Conv	Address	Size	Type	Flags	Filename
929	CFileFiltersSettingsDialog::OnCancel	CV_CALL_THISCALL	0x08A11940	5	SymTagFunction		e:\am\c\threadlockchecker\filefilt
930	CFileFiltersSettingsDialog::OnDblGridClick	CV_CALL_THISCALL	0x08A12030	21	SymTagFunction		e:\am\c\threadlockchecker\filefilt
931	CFileFiltersSettingsDialog::OnHelpInfo	CV_CALL_THISCALL	0x08A11A50	29	SymTagFunction		e:\am\c\threadlockchecker\filefilt
932	CFileFiltersSettingsDialog::OnInitDialog	CV_CALL_THISCALL	0x08A120A0	406	SymTagFunction		e:\am\c\threadlockchecker\filefilt
933	CFileFiltersSettingsDialog::OnLGridClick	CV_CALL_THISCALL	0x08A11FF0	21	SymTagFunction		e:\am\c\threadlockchecker\filefilt
934	CFileFiltersSettingsDialog::OnOK	CV_CALL_THISCALL	0x08A11950	5	SymTagFunction		e:\am\c\threadlockchecker\filefilt
935	CFileFiltersSettingsDialog::OnRGridClick	CV_CALL_THISCALL	0x08A12010	21	SymTagFunction		e:\am\c\threadlockchecker\filefilt
936	CFileFiltersSettingsDialog::OnSelChanged	CV_CALL_THISCALL	0x08A12050	21	SymTagFunction		e:\am\c\threadlockchecker\filefilt
937	CFileFiltersSettingsDialog::OnVKeyToItem	CV_CALL_THISCALL	0x08A12070	30	SymTagFunction		e:\am\c\threadlockchecker\filefilt
938	CFileFiltersSettingsDialog::`RTTI Base Class Array'		0x08A4B068	0	SymTagPublicSymbol		
939	CFileFiltersSettingsDialog::`RTTI Base Class Descriptor at (0,-1,0,64)'		0x08A4B0A8	0	SymTagPublicSymbol		
940	CFileFiltersSettingsDialog::`RTTI Class Hierarchy Descriptor'		0x08A4B07B	0	SymTagPublicSymbol		
941	CFileFiltersSettingsDialog::`RTTI Complete Object Locator'		0x08A4B064	0	SymTagPublicSymbol		
942	CFileFiltersSettingsDialog::`vector deleting destructor'	CV_CALL_THISCALL	0x08A11A90	106	SymTagFunction		

Part VI

6 Settings Dialog

The settings dialog allows you to configure how DbgHelpBrowser searches for symbols and source files.



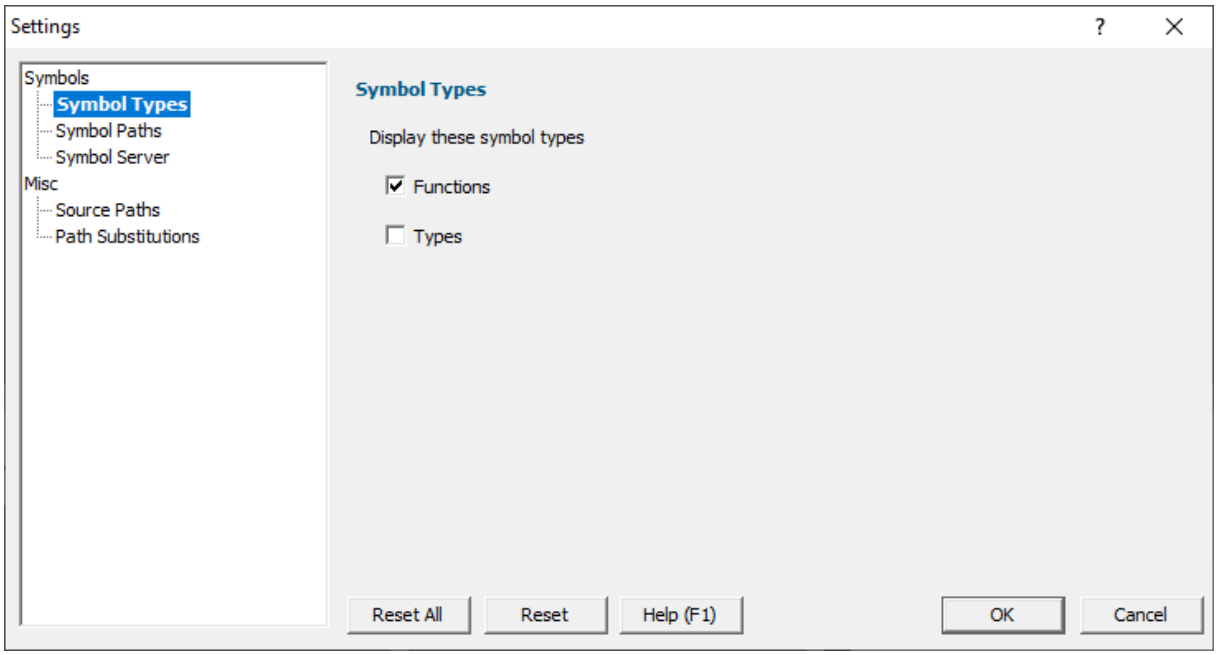
Reset

You can reset the settings to their default state at any time by clicking **Reset**.

6.1 Symbols

6.1.1 Symbol Types

The settings dialog allows you to configure how DbgHelpBrowser searches for symbols.



Symbol types to display

Select the types of symbols to be displayed in the main grid.

If you're interested in classes, methods and functions, choose **Functions**.

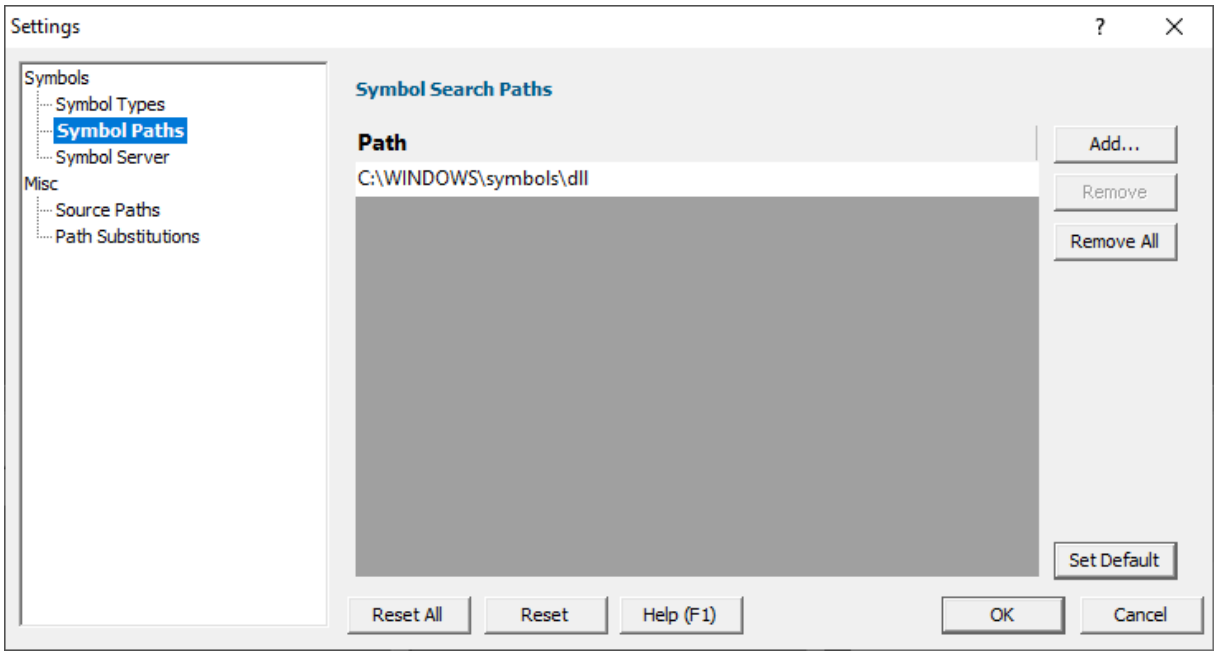
If you're interested in data, choose **Types**.

Reset

You can reset the settings to their default state at any time by clicking **Reset**.

6.1.2 Symbol Paths

The settings dialog allows you to configure how DbgHelpBrowser searches for symbols.



Symbol Search Paths

The symbol search paths section allows to specify locations on your computer that hold debug information for the DLLs you may wish to use with DbgHelpBrowser.

The **Add...** button allows you to choose a directory and add it to the list of directories that will be searched for PDB information.

The **Remove** button will remove any selected directory, the **Remove All** button, will remove all directories.

The **Set Default** button restores the default search path of c:\windows\symbols\dll.

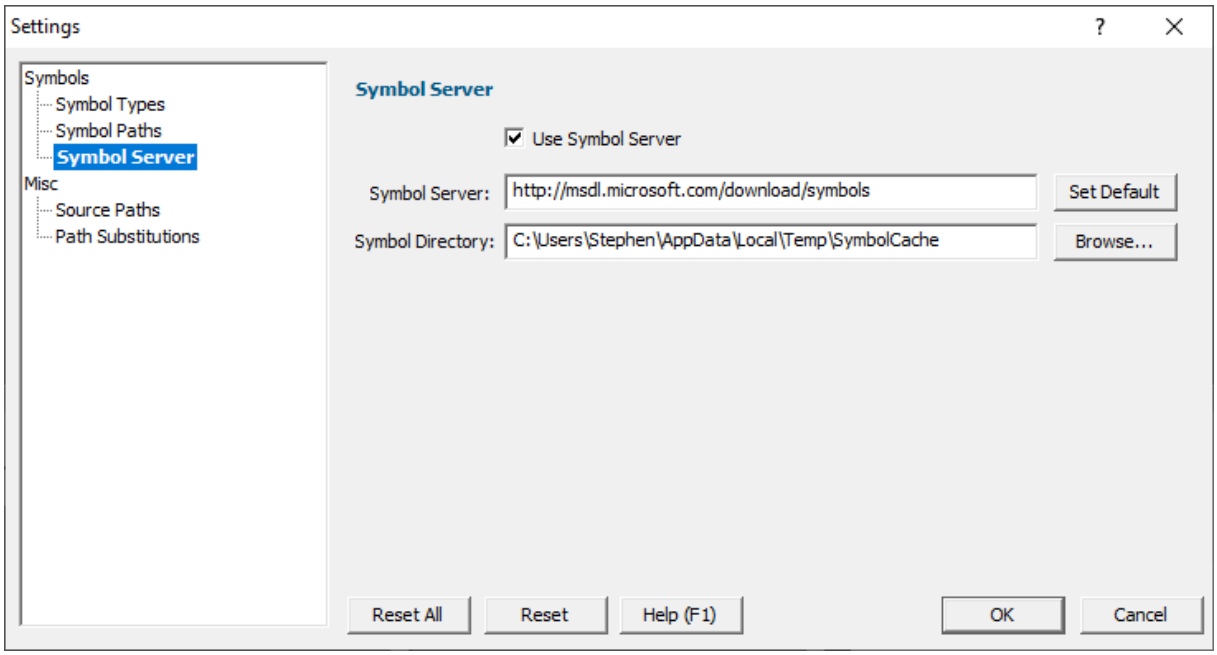
In the image above we can see the user has created their own directory `c:\MicrosoftSymbols` to store all the symbols from the Microsoft symbol server.

Reset

You can reset the settings to their default state at any time by clicking **Reset**.

6.1.3 Symbol Server

The settings dialog allows you to configure how DbgHelpBrowser searches for symbols.



Symbol Server

You may be using a symbol server to provide symbols for your DLLs. If that is the case you need to use the Symbol Server settings.

The defaults are configured for Microsoft's symbol server, but you can point them at any symbol server you want. If you wish to reset these at any time, the **Set Default** button will do that for you.

If you want to use the symbol server, you must enable it by selecting **Use Symbol Server** check box.

Type the full path to the symbol server in the **Symbol Server** field, and type the full path to where you wish to store a local copy of the symbols in the **Symbol Directory** field, or use the **Browse...** button to use a directory to identify the directory.

Reset

You can reset the settings to their default state at any time by clicking **Reset**.

6.2 Misc

6.2.1 Source Paths

The Source Paths settings allow you to specify where DbgHelp Browser looks for source code files.

The source code paths are used when a filename is incomplete - a filename without a path, a filename with a partial path, or a filename that isn't valid on this machine.

Manually adding path type directories

The Path list shows all the paths that will be searched for source code files.

You can modify the list of files for each path type in the following ways:

- **Add** › appends a row to the directory list › enter the directory path

Edit a directory path by double clicking the entry. The usual controls apply for removing list items:

- **Remove** › removes selected items from the list
- **Remove All** › clears the list
- **Set Default** › adds all valid directories found in the PATH environment variable

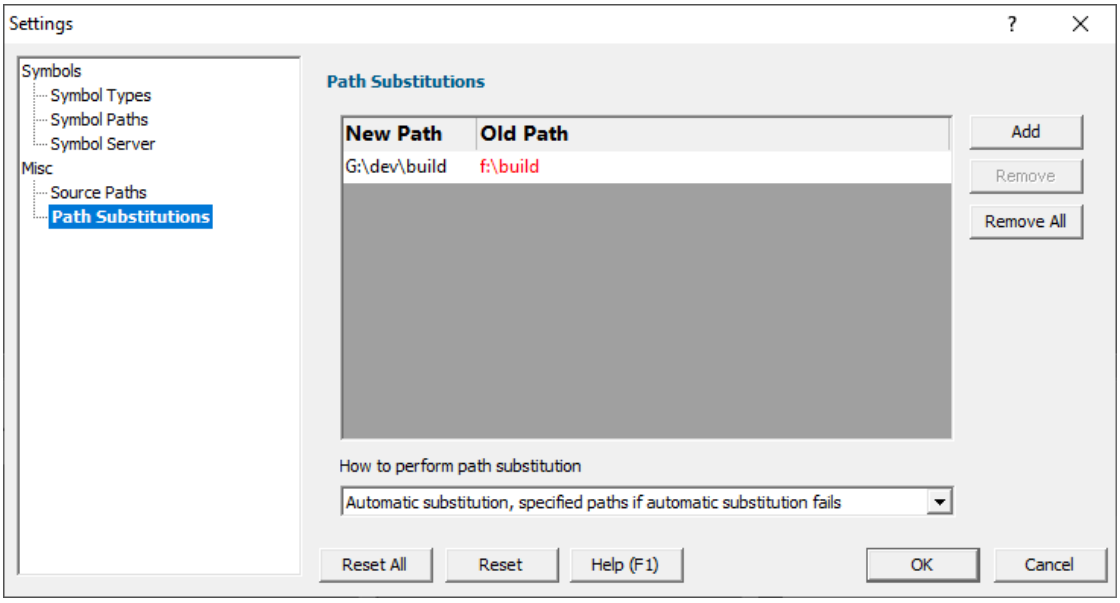
Alternatively, press Del to delete selected items, and Ctrl + A to select all items in the list first.

Reset - Resets **all** global settings, not just those on the current page. This includes removing any symbol servers added.

6.2.2 Path Substitutions

The **Path Substitutions** tab allows you to specify file path substitutions to handle copying builds from build machines to development or test machines .

The default settings are shown below:



Path Substitutions

Some software development schemes have multiple rolling builds of their software, often enabled by using substituted disk drive naming schemes.

When you download the build to your development machine for development and testing, debugging information may reference disk drives that don't exist on your machine, for example, drive X: while your machine only has C:, D:, and E: drives.

Or you may just be copying a build from a drive on a development machine to a subdirectory on a drive on your test machine.

These options let you remap the substitution so that the DbgHelp Browser looks in the correct place for the source code.

- **Add** › adds a row to the **File Paths Substitutions** table › enter the new path that will replace the old path in the **New Path** column › click in the Old **Path** column › enter the path that is being replaced

  For example, you might enter c:\users\stephen\documents for the new path and `f:\dev\build` for the old path.

You can double click to edit drives and paths in the table, or remove items:

- **Remove** › removes selected substitutions from the list
- **Remove All** › removes all substitutions from the list

Alternatively, press Del to delete selected items, and Ctrl + A to select all items in the list first.

Example: Changed disk drive

Project originally located at	m:\dev\build\testApp
Project copied to	e:\dev\build\testApp
New Path	e:\
Old Path	m:\

Example: Project copied to a new location

Project originally located at	f:\dev\build\testApp
Project copied to	C:\Users\Stephen\Documents\testApp
New Path	C:\Users\Stephen\Documents
Old Path	f:\dev\build

The slashes do not have to match, a forward slash will match a backslash when comparing path fragments. This is deliberate - to improve ease of use with libraries built by different compilers (LLVM and compilers that use it use forward slashes, whereas Visual Studio etc use backslashes).

Path Substitution Method

Path substitution can be turned off, use only manually specified paths, perform automatic path substitution based on best guesses based on information in the executable, or a combination.

Use the combo box to choose the appropriate path substitution method. The default is automatic path substitution and if that fails to try path substitution using the manually specified paths.

- **No path substitution** › path substitution does not happen
- **Only substitute specified paths** › path substitution uses the manually specified paths
- **Automatic substitution only** › path substitution is performed automatically using information in the executable
- **Automatic substitution, specified paths if substitution fails** › an attempt at automatic path substitution is made, if this fails path substitution is performed using the manually specified paths

The default is **Automatic substitution, specified paths if substitution fails**.

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

Reset - Resets the settings on the current page.

Part VII

7 How to use DbgHelpBrowser

Load PDB information

To load Debug information you need to have the PDB file containing debug information and the DLL that the PDB file relates to. You need to ensure the PDB files corresponds to the very same build as the DLL.

Use the **File > Load Dll and debug information...** option to load the appropriate DLL and it's debug information.

The grid displays various attributes of each debugging item. You can sort the grid by clicking the appropriate column header. Click the same header to reverse the sort order.

Select a symbol to see information about the parameters, locals, line numbers and source code.

Filtering

If you wish to only view one type of debugging data, select that datatype using the **Tags** combo.

You can also filter by name by typing the name into the **Name Filter** box and clicking the **Filter** button to perform the filtering.

Viewing function data

As each item in the list is selected the Parameters and Locals grid are populated, the Line Numbers are updated and the source code display updates to show the source code for the function. All lines in the function that contain executable code (as indicated by the debugging information) are coloured grey. The current line for the function is coloured bright green.

Querying data

You can query data by using the two Query fields below the main grid.

Relative query

Type the relative address (also know as address offset) into the Query by Offset field, then click Query. The symbol information is displayed.

The field accepts decimal or hexadecimcal values. Hex values must be prefixed with 0x.

Absolute query

Type the absolute address into the Query by Address field, type the absolute DLL load address into the Alternate Load Address field, then click Query. The symbol information is displayed.

The fields accept decimal or hexadecimcal values. Hex values must be prefixed with 0x.

7.1 Decoding an absolute crash address

Scenario:

A customer has supplied you with a crash report containing a callstack with addresses. The callstack also indicates which module relates to which address.
The customer has also supplied you with a list of module load addresses.

Example Data:

```
Exception code: C0000005 ACCESS_VIOLATION
Fault address:  0x005f5eec (base 0x00400000) C:\Program Files (x86)\Software Verification\
Exception Parameters:
   0: 0x00000000 [Read Error]
   1: 0x035f0034 [Address]

Registers:
   EAX:035F0034
   EBX:00000000
   ECX:FFFDD000
   EDX:00002370
   ESI:006F7D58
   EDI:035F0034
   CS:EIP:0023:005F5EEC
   SS:ESP:002B:0018FE14  EBP:0018FE3C
   DS:002B  ES:002B  FS:0053  GS:002B
   Flags:00010202

StackTrace

C:\Program Files (x86)\Software Verification\C++ Memory Validator\memoryValidator.exe : 0x
C:\Program Files (x86)\Software Verification\C++ Memory Validator\memoryValidator.exe : 0x
C:\Windows\syswow64\msvcrt.dll : 0x75D70000 : 0x75D7C3E4
C:\Windows\syswow64\msvcrt.dll : 0x75D70000 : 0x75D836B6
C:\Program Files (x86)\Software Verification\C++ Memory Validator\memoryValidator.exe : 0x
C:\Windows\syswow64\kernel32.dll : 0x754D0000 : 0x754E3365
C:\Windows\SysWOW64\ntdll.dll : 0x77920000 : 0x77959F6D
C:\Windows\SysWOW64\ntdll.dll : 0x77920000 : 0x77959F40
C:\Windows\SysWOW64\ntdll.dll : 0x77920000 : 0x77959F40
```

This is data from a real crash a few years ago, from C++ Memory Validator 5.80.

Question:
How do you decode these absolute addresses?

Answer:
In the above data we can see a callstack containing entries for ntdll.dll, msvcrt.dll, and memoryValidator.exe.

All the modules are Microsoft DLLs except for the EXE, which is part of C++ Memory Validator, one of our tools.

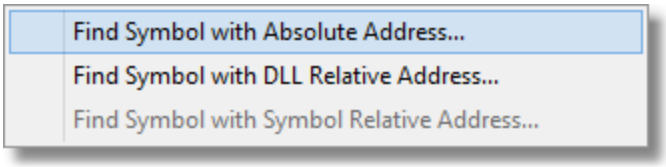
To decode these values, we load memoryValidator.exe into DbgHelpBrowser.exe, then for each symbol we take the following actions.

For our purposes here, we're going to show how to convert one symbol. We're going to use the first symbol from memoryValidator.exe in the example data above.

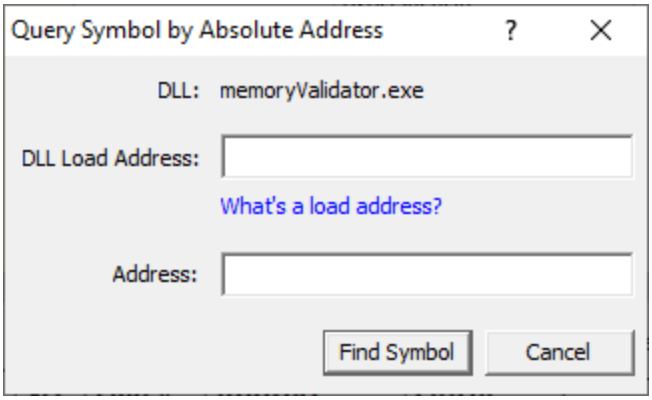
```
0x005f5eec (base 0x00400000)
```

The address is 0x005f5eec. The DLL loaded at 0x00400000. You'll notice the load address for all MemoryValidator.exe entries is 0x00400000.

From the Query menu choose **Find Symbol with Absolute Address...**.



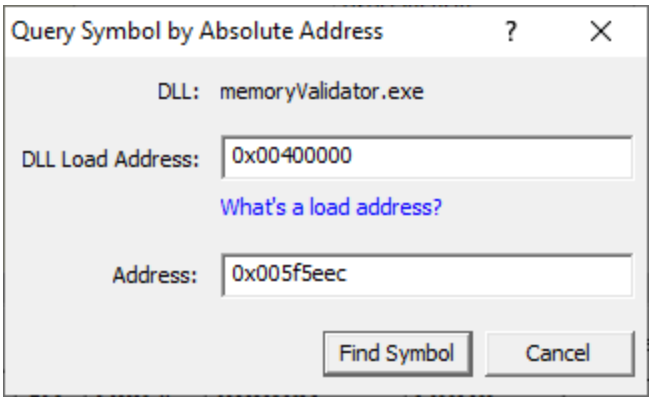
The Query Symbol by Absolute Address dialog is displayed.



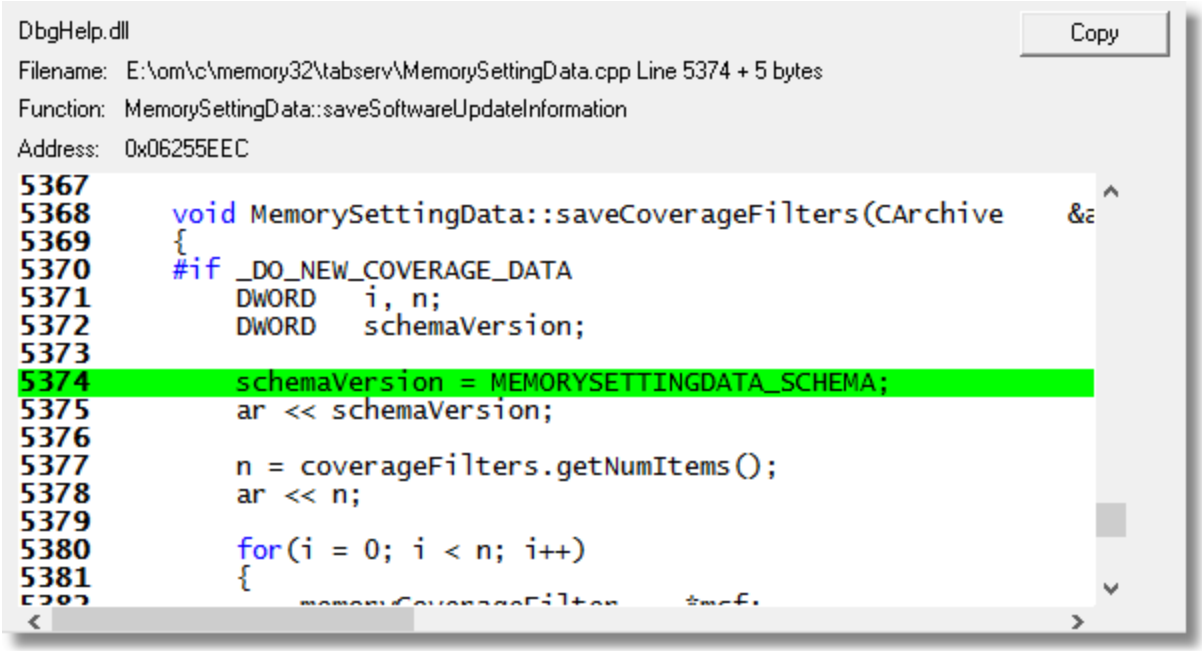
Type the DLL load address into the DLL Load Address field. Prefix any hexadecimal addresses with 0x.

Type the symbol address into the Address field. Prefix any hexadecimal addresses with 0x.

Click the **Find Symbol** button.



The appropriate location in the code is found and displayed.



Results:
Repeating the process for the data shown above resulted in this information.

```
0x005f5eec (base 0x00400000) C:\Program Files (x86)\Software Verification\C++ Memory Valid
C:\Program Files (x86)\Software Verification\C++ Memory Validator\memoryValidator.exe : 0x
C:\Program Files (x86)\Software Verification\C++ Memory Validator\memoryValidator.exe : 0x
C:\Windows\syswow64\msvcrt.dll : 0x75D70000 : 0x75D7C3E4
C:\Windows\syswow64\msvcrt.dll : 0x75D70000 : 0x75D836B6
C:\Program Files (x86)\Software Verification\C++ Memory Validator\memoryValidator.exe : 0x
C:\Windows\syswow64\kernel32.dll : 0x754D0000 : 0x754E3365
C:\Windows\SysWOW64\ntdll.dll : 0x77920000 : 0x77959F6D
C:\Windows\SysWOW64\ntdll.dll : 0x77920000 : 0x77959F40
C:\Windows\SysWOW64\ntdll.dll : 0x77920000 : 0x77959F40
```

Help! I have a crash address but I don't know what the load address is? What do I do?

You need to read about load addresses.

.Net, .Net Core

The query options are not available for .Net and .Net executables as there is no direct translation from a crash address/offset to a .Net symbol.

Without having access to the compiled .Net method address and the compiled address to ILASM instruction offset data it is impossible to translate crash addresses/offsets to .Net functions.

The compiled address to ILAMS instruction offset data is only available in the context of a running .Net application attached to a .Net debugger or a .Net profiler.

7.2 Decoding a relative crash address

Scenario:

A customer has supplied you with a crash report containing a callstack with relative offsets from DLLs. The callstack also indicates which module relates to which address.

Example Data:

```
Exception code: C0000005 ACCESS_VIOLATION
Fault offset:   0x00036FA3 C:\WINDOWS\system32\MSVCRT.dll
Exception Parameters:
   0: 0x00000000 [Read Error]
   1: 0x5f8f2000 [Address]

Registers:
   EAX:B3BEB6D4
   EBX:5F8CB6C8
   ECX:150BE5B5
   EDX:00000000
   ESI:5F8F2000
   EDI:01B98DEC
   CS:EIP:001B:77C46FA3
   SS:ESP:0023:0012F158  EBP:0012F160
   DS:0023  ES:0023  FS:003B  GS:0000
   Flags:00010212


StackTrace

C:\WINDOWS\system32\MFC42u.DLL : 0x0000270a
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x000db989
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x000db1f8
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x00121a83
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x00121b7e
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x00174ec5
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x00175094
C:\WINDOWS\system32\MFC42u.DLL : 0x00013724
C:\WINDOWS\system32\MFC42u.DLL : 0x00014245
C:\WINDOWS\system32\MFC42u.DLL : 0x00001b31
C:\WINDOWS\system32\MFC42u.DLL : 0x0008cba7
```

This is data from a real crash many years ago.

Question:
There are no DLL load addresses and the addresses aren't addresses, but offsets from the start of a DLL. How do you decode these relative offsets?

Answer:
In the above data we can see a callstack containing entries for mfc42u.dll, and memoryValidator.exe.

All the modules are Microsoft DLLs except for the EXE, which is part of C++ Memory Validator, one of our tools.

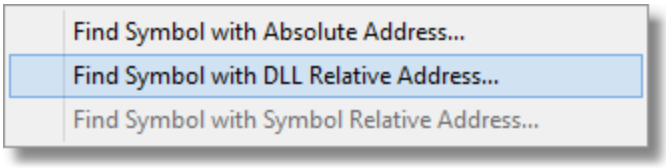
To decode these values, we load memoryValidator.exe into DbgHelpBrowser.exe, then for each symbol we take the following actions.

For our purposes here, we're going to show how to convert one symbol. We're going to use the first symbol from memoryValidator.exe in the example data above.

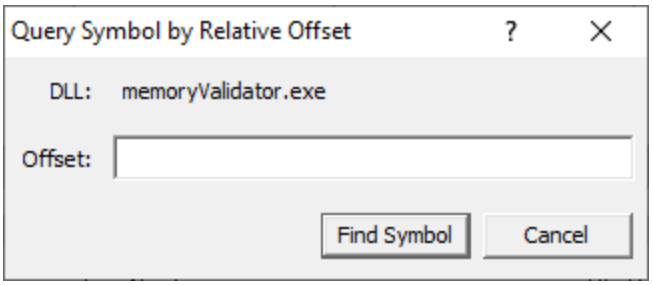
```
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x000db989
```

The relative address (or offset) is 0x000db989. We don't know the DLL load address.

From the Query menu choose **Find Symbol with DLL Relative Address...**.

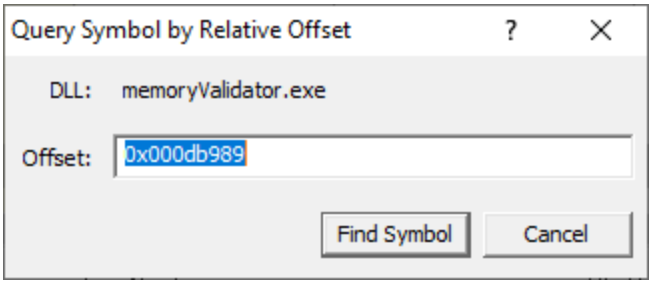


The Query Symbol by Absolute Address dialog is displayed.

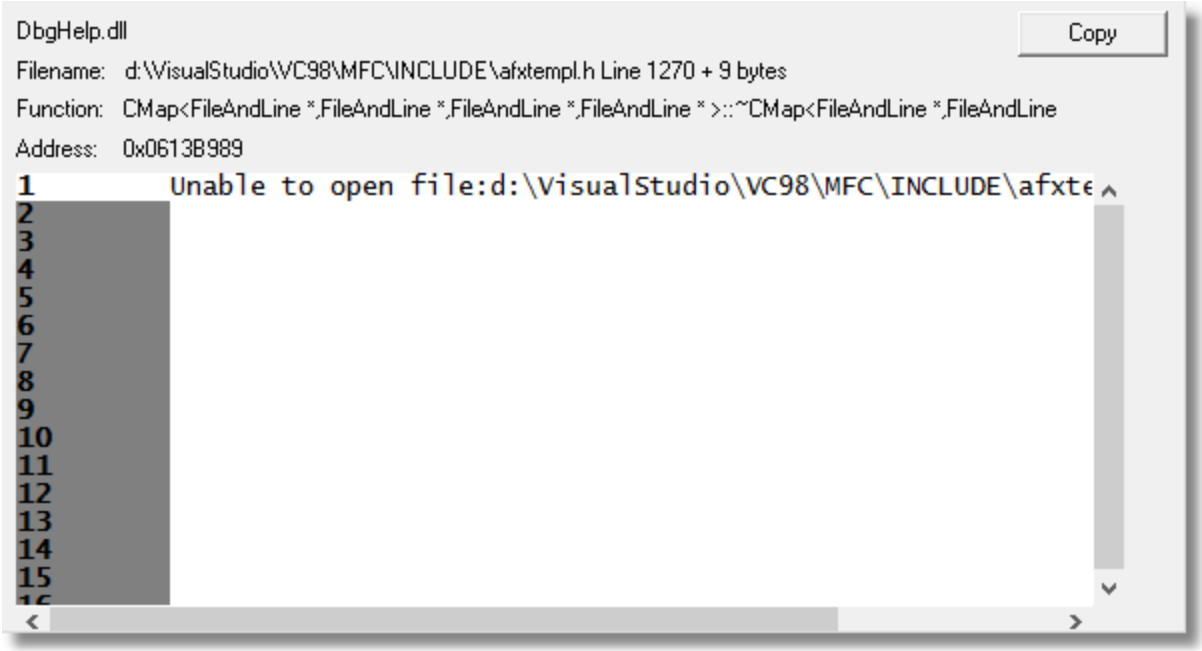


Type the relative address into the Offset field. Prefix any hexadecimal addresses with 0x.

Click the **Find Symbol** button.



The appropriate location in the code is found and displayed. In this example DbgHelpBrowser could not locate the source code (as the file location is not valid on this machine)



Results:
Repeating the process for the data shown above resulted in this information.

```
Exception code: C0000005 ACCESS_VIOLATION
Fault offset:   0x00036FA3 C:\WINDOWS\system32\MSVCRT.dll
Exception Parameters:
   0: 0x00000000 [Read Error]
   1: 0x5f8f2000 [Address]

Registers:
   EAX:B3BEB6D4
   EBX:5F8CB6C8
   ECX:150BE5B5
   EDX:00000000
   ESI:5F8F2000
   EDI:01B98DEC
   CS:EIP:001B:77C46FA3
   SS:ESP:0023:0012F158  EBP:0012F160
   DS:0023  ES:0023  FS:003B  GS:0000
   Flags:00010212


StackTrace

C:\WINDOWS\system32\MFC42u.DLL : 0x0000270a
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x000db989
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x000db1f8
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x00121a83
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x00121b7e
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x00174ec5
C:\Program Files\Software Verification\Memory Validator\memoryValidator.exe : 0x00175094
C:\WINDOWS\system32\MFC42u.DLL : 0x00013724
C:\WINDOWS\system32\MFC42u.DLL : 0x00014245
C:\WINDOWS\system32\MFC42u.DLL : 0x00001b31
C:\WINDOWS\system32\MFC42u.DLL : 0x0008cba7
```

.Net, .Net Core

The query options are not available for .Net and .Net executables as there is no direct translation from a crash address/offset to a .Net symbol.

Without having access to the compiled .Net method address and the compiled address to ILASM instruction offset data it is impossible to translate crash addresses/offsets to .Net functions.

The compiled address to ILAMS instruction offset data is only available in the context of a running .Net application attached to a .Net debugger or a .Net profiler.

7.3 Decoding a symbol relative crash address

Scenario:

A customer has supplied you with a crash report containing a callstack with symbol relative offsets from DLLs. The callstack also indicates which module relates to which address.

Example Data:

```
ntoskrnl.exe!KeSynchronizeExecution+0x2246
ntoskrnl.exe!KeWaitForMultipleObjects+0x135e
ntoskrnl.exe!KeWaitForMultipleObjects+0xdd9
ntoskrnl.exe!KeWaitForSingleObject+0x373
ntoskrnl.exe!KeStallWhileFrozen+0x1977
ntoskrnl.exe!_misaligned_access+0x13f9
ntoskrnl.exe!KeWaitForMultipleObjects+0x152f
ntoskrnl.exe!KeWaitForMultipleObjects+0xdd9
ntoskrnl.exe!KeWaitForSingleObject+0x373
ntoskrnl.exe!NtWaitForSingleObject+0xb2
ntoskrnl.exe!setjmpex+0x34a3
ntdll.dll!ZwWaitForSingleObject+0xa
KERNELBASE.dll!WaitForSingleObjectEx+0x98
svlcoveragevalidatorstub_x64.dll!sendCommandLineAndStartTimeToGUI+0x2868
svlcoveragevalidatorstub_x64.dll!setValidatorFeedbackHookingComplete+0x1fa6
svlcoveragevalidatorstub_x64.dll!svl_sendMessageRawToUserInterface+0x21837
svlcoveragevalidatorstub_x64.dll!svl_sendMessageRawToUserInterface+0x218cb
KERNEL32.DLL!BaseThreadInitThunk+0x22
ntdll.dll!RtlUserThreadStart+0x34
```

This is real data from a bug at Software Verify Ltd. This is one thread from many in a dump relating to a deadlock bug we were investigating.

Question:
How do you decode these symbol relative offsets?

Answer:
In the above data we can see a callstack containing entries for ntoskrnl.exe, ntdll.dll, kernelbase.dll, kernel32.dll and svlcoveragevalidatorstub_x64.dll.

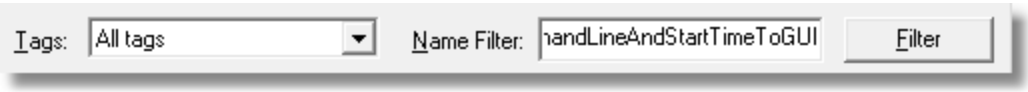
All the modules are Microsoft DLLs except for one DLL, which is part of C++ Coverage Validator, one of our tools.

To decode these values, we load svlCoverageValidatorStub_x64.dll into DbgHelpBrowser.exe (64 bit), then for each symbol we take the following actions.

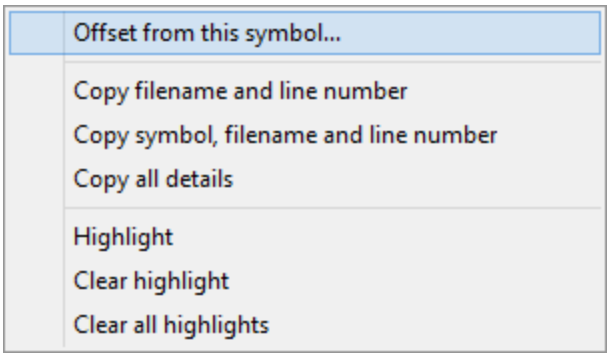
For our purposes here, we're going to show how to convert one symbol. We're going to use the first symbol from svlCoverageValidatorStub_x64.dll in the example data above.

```
svlcoveragevalidatorstub_x64.dll!sendCommandLineAndStartTimeToGUI+0x2868
```

Type the symbol name into the **Name Filter** field, then click **Filter**. This makes it easy to find the symbol we want.

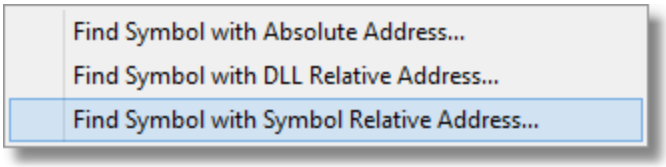


Once we have found the symbol, right click on the symbol to display the context menu and choose **Offset from this symbol...**.

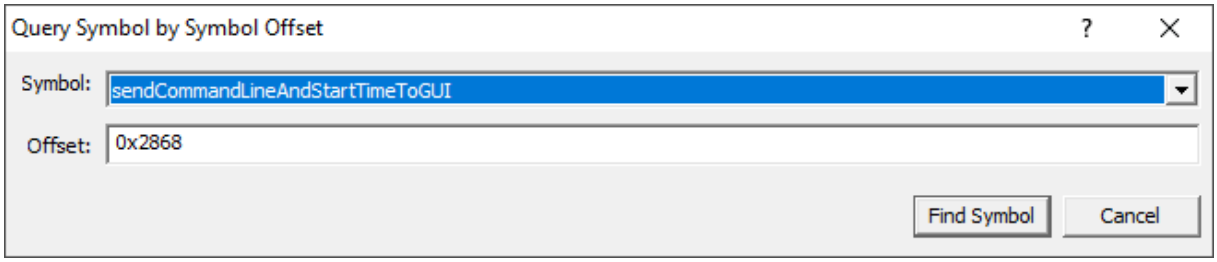


An alternate method is to click on the symbol to select it, then from the Query menu choose **Find Symbol with Symbol Relative Address...**.

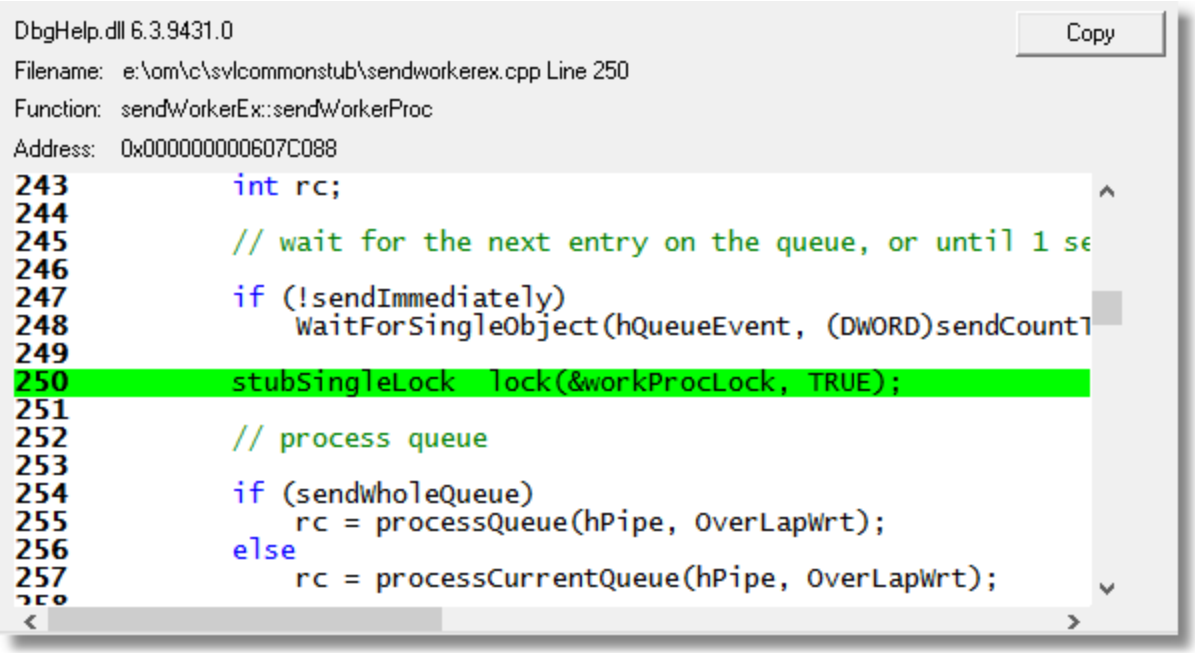
Or, from the Query menu choose **Find Symbol with Symbol Relative Address...** then choose the symbol you want from the combo box.



Type the offset into the dialog (hex values must be prefixed with 0x) and click OK.



The appropriate location in the code is found and displayed.



Results:
Repeating the process for the data shown above resulted in this information.

```
svlcoveragevalidatorstub_x64.dll!sendCommandLineAndStartTimeToGUI+0x2868          sendWorke
svlcoveragevalidatorstub_x64.dll!setValidatorFeedbackHookingComplete+0x1fa6       stubSendC
svlcoveragevalidatorstub_x64.dll!svl_sendMessageRawToUserInterface+0x21837        memcpy
svlcoveragevalidatorstub_x64.dll!svl_sendMessageRawToUserInterface+0x218cb        wcscpy
```

.Net, .Net Core

The query options are not available for .Net and .Net executables as there is no direct translation from a crash address/offset to a .Net symbol.

Without having access to the compiled .Net method address and the compiled address to ILASM instruction offset data it is impossible to translate crash addresses/offsets to .Net functions.

The compiled address to ILASM instruction offset data is only available in the context of a running .Net application attached to a .Net debugger or a .Net profiler.

7.4 Decoding an Event Viewer XML crash log

Scenario:

A customer has supplied you with data from Windows Event Viewer about a crash. The log contains XML and you don't know which values are relevant.

The event log data will have a provider name of "Windows Error Reporting" or "Application Error".

The XML data is found on the "Details" tab with the XML View radio box selected.

Example Data:


```xml
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Windows Error Reporting" />
    <EventID Qualifiers="0">1001</EventID>
    <Level>4</Level>
    <Task>0</Task>
    <Keywords>0x80000000000000</Keywords>
    <TimeCreated SystemTime="2020-02-10T17:39:08.000000000Z" />
    <EventRecordID>260219</EventRecordID>
    <Channel>Application</Channel>
    <Computer>hydra</Computer>
    <Security />
  </System>
  <EventData>
    <Data>2023787729086567941</Data>
    <Data>1</Data>
    <Data>APPCRASH</Data>
    <Data>Not available</Data>
    <Data>0</Data>
    <Data>testDeliberateCrash.exe</Data>
    <Data>1.0.0.1</Data>
    <Data>5e419525</Data>
    <Data>testDeliberateCrash.exe</Data>
    <Data>1.0.0.1</Data>
    <Data>5e419525</Data>
    <Data>c0000005</Data>
    <Data>000017b2</Data>
    <Data />
    <Data />
    <Data>C:\Users\stephen\AppData\Local\Temp\WERA14E.tmp.WERInternalMetadata.xml</Data>
    <Data>C:\Users\stephen\AppData\Local\Microsoft\Windows\WER\ReportArchive\AppCrash_testDelibera
    <Data />
    <Data>0</Data>
    <Data>3cc45263-4c2c-11ea-83d3-001e4fdb3956</Data>
    <Data>0</Data>
    <Data>54756af49aec84f97c15f03794ffd605</Data>
  </EventData>
</Event>
```

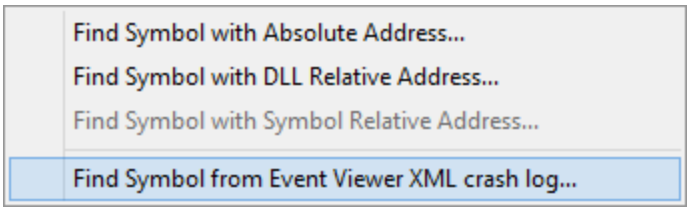

This is data from a test program that is designed to crash.

Question:
There the event log indicates a DLL, but no load address, two different addresses, an exception code and an offset from the start of the DLL. How do you decode this relative offset?

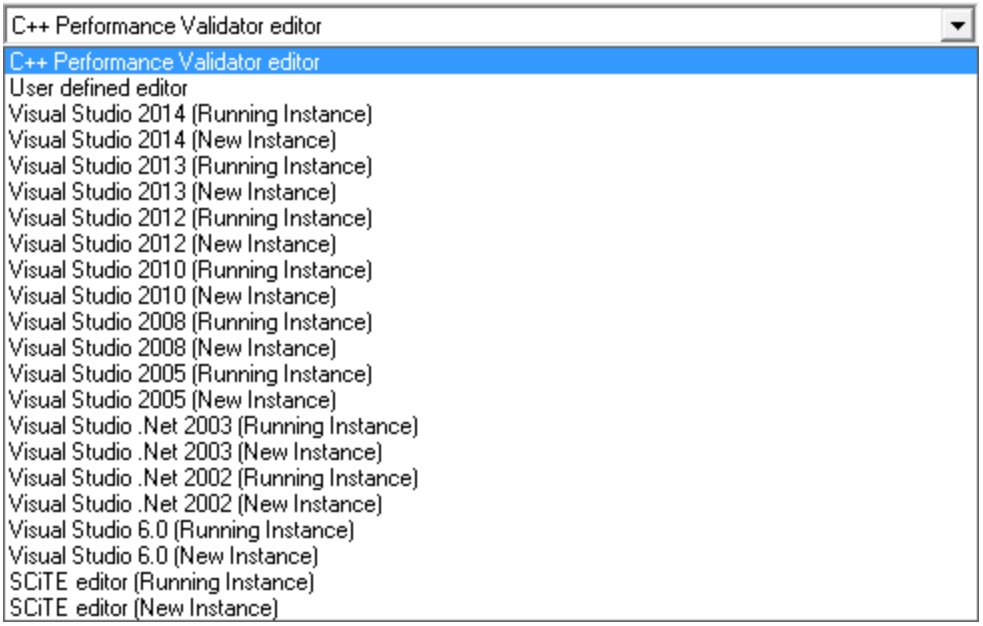
Answer:
DbgHelpBrowser has an option specifically for this occasion.

The XML data indicates the crash happened in **testDeliberateCrash.exe**. Load this into DbgHelpBrowser being sure to load the correct build version and that the PDB file can be found so that symbols get loaded.

From the Query menu choose **Find Symbol from Event Viewer XML crash log...**.

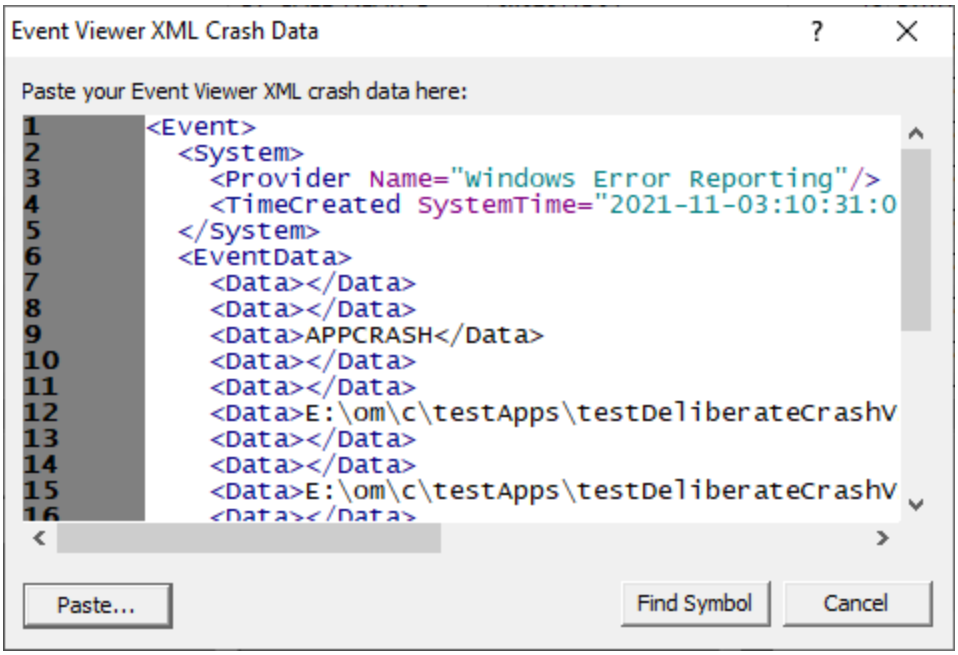


The Query Symbol by Absolute Address dialog is displayed.

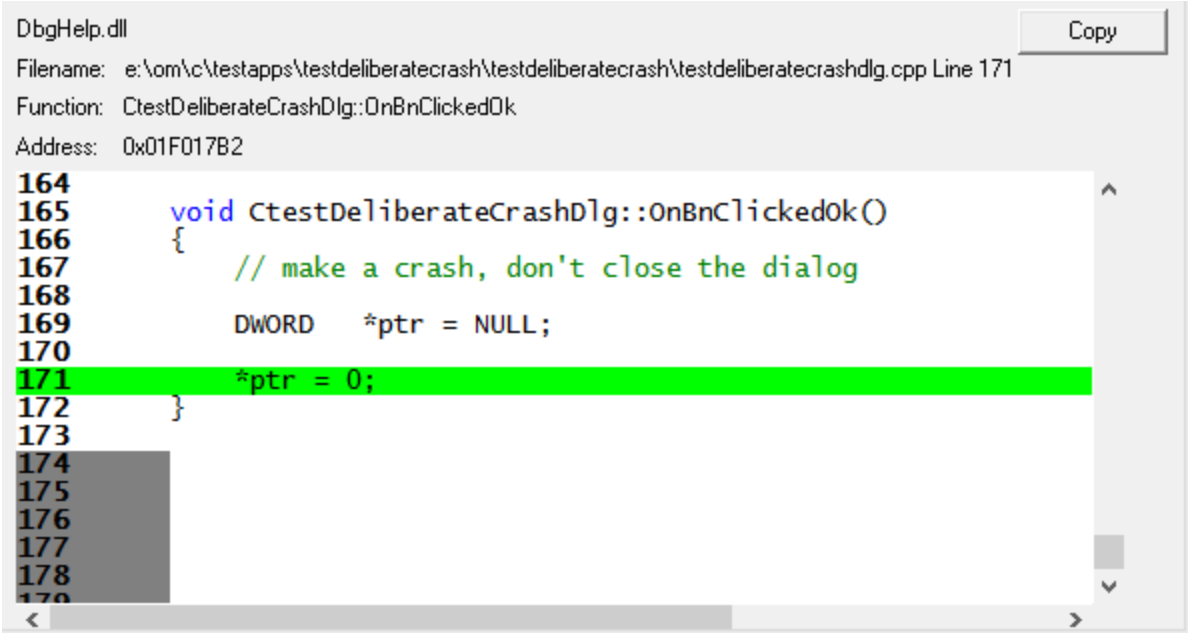


Paste the XML data from the Event Viewer into the text field.

Click the **Find Symbol** button.



The appropriate location in the code is found and displayed.



.Net, .Net Core

The query options are not available for .Net and .Net executables as there is no direct translation from a crash address/offset to a .Net symbol.

Without having access to the compiled .Net method address and the compiled address to ILASM instruction offset data it is impossible to translate crash addresses/offsets to .Net functions.

The compiled address to ILAMS instruction offset data is only available in the context of a running .Net application attached to a .Net debugger or a .Net profiler.

7.5 What is a load address?

A load address is the address at which a DLL loads.

All versions of Microsoft Windows load modules (.dll, .exe) into address space that is reserved using a call to VirtualAlloc().

The allocation of VirtualAlloc() can be queried by calling Win32 API GetSystemInfo() and examining the value returned in **dwAllocationGranularity**. For all versions of Microsoft Windows this has been 64KB.

Why is the load address important?

The load address is important because without it we can't calculate the offset inside the DLL so that we can obtain a symbol.

That's why a crash address with no DLL Load Address isn't very useful - we don't know which DLL the crash is in, nor do we know where the DLL was loaded.

But I don't have a load address. What can I do?

Depending upon how your module (DLL/EXE) was built we may be able to guess the correct load address.

If the OS you are using is Windows XP or earlier, we can guess the address.

First a brief chat about Address Space Layout Randomisation...

If the OS you are using is Windows Vista or later, we may be able to guess the load address. The reason this is not precises is because something known as Address Space Layout Randomisation (ASLR) was introduced with Microsoft Vista to improve security against many malicious computer attacks. Any program built with ASLR enabled when run on Vista (or later) will have the load address for all modules (including the .exe) randomised, making guessing the load address a waste of time.

ASLR is enabled by the /DYNAMICBASE in the linker settings of Visual Studio.

If you are using Visual Studio 2005 or earlier this setting is not available, your program is not affected by ASLR.

If you are using Visual Studio 2008 or later you will need to check to see if this option is present. If it is not present, your program is not affected by ASLR.

If you are not using Visual Studio to build your program then you may not be affected by this option, consult your compiler/linker documentation.

If your program is not affected by ASLR...

We can try to guess the load address of your DLL/EXE. We can do this regardless of which compiler/linker you used to build your program. All the programs I mention here are free to download at the time of writing this help file.

VM Validator
https://www.softwareverify.com/cpp-virtual-memory.php

This works for 32 bit and 64 bit programs.

Method 1

- Start your program using VM Validator or attach to your running program with VM Validator.
- On the Summary tab, inspect the DLLs sub tab in the lower half of the display.
- Find the DLL name in the DLL column.
- The load address is the value in the Address column.

DLLs | Page Faults

DLL (133)	Fault Count	Address	Size	Commit	Reserve	CPU
E:\om\c\dbgHelpBrowser\Release\x86\dbgHelpBrowser.exe	0	0x00400000	1176.00 KB	1176.00 KB	0.00 KB	x86
E:\om\c\testApps\testDeliberateCrash\Release\testDeliberateCrash.exe	0	0x00640000	100.00 KB	100.00 KB	0.00 KB	x86
E:\om\c\dbgHelpBrowser\Release\x86\svlPeInfo.dll	0	0x006C0000	172.00 KB	172.00 KB	0.00 KB	x86

Method 2

- Start your program using VM Validator or attach to your running program with VM Validator.
- Go to the Paragraphs tab.
- Find any purple entry, check the DLL name in the Description field.
- The load address is the value in the Address column.

Summary | Virtual | Pages | Paragraphs

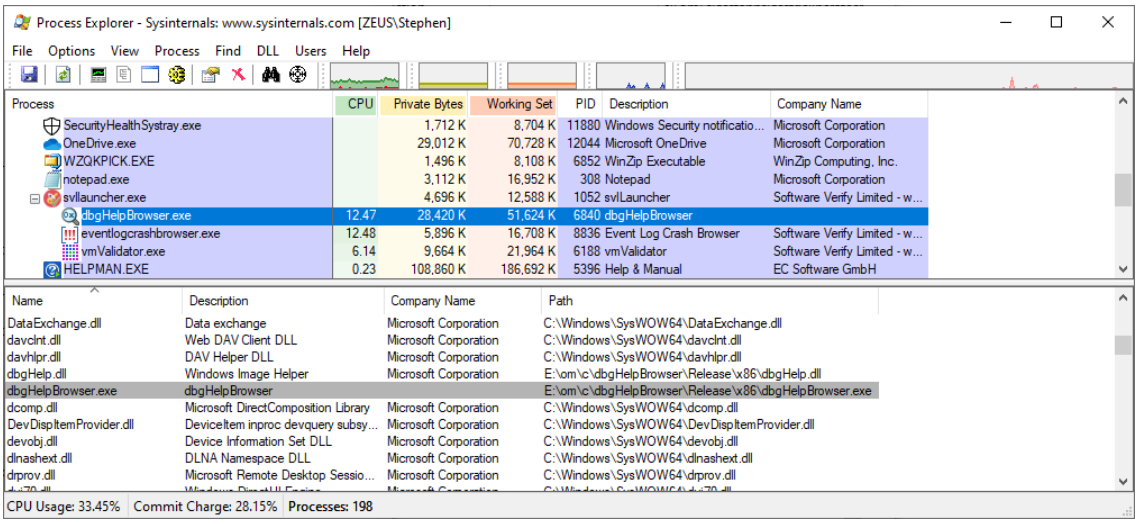
Address	Size	Type	Protect	Working Set	Shared	Swap	Description
0x002C0000	64 KB	Private	Read, Write	Read/write.			Commited, Reserved
0x002D0000	152 KB	Private					Reserved, Commited, Reserved
0x00300000	1,024 KB	Private					Reserved
0x00400000	1,176 KB	Image	Read Only	Read-only. Executable and read-only.	Shared: 98		e:\om\c\dbghelpbrowser\release\x86\dbghelpbrowser.exe
0x00530000	796 KB	Mapped	Read Only	Read-only.	Shared: 49		Commited, Free

In the examples above, for dbgHelpBrowser.exe, the load address is 0x00400000.

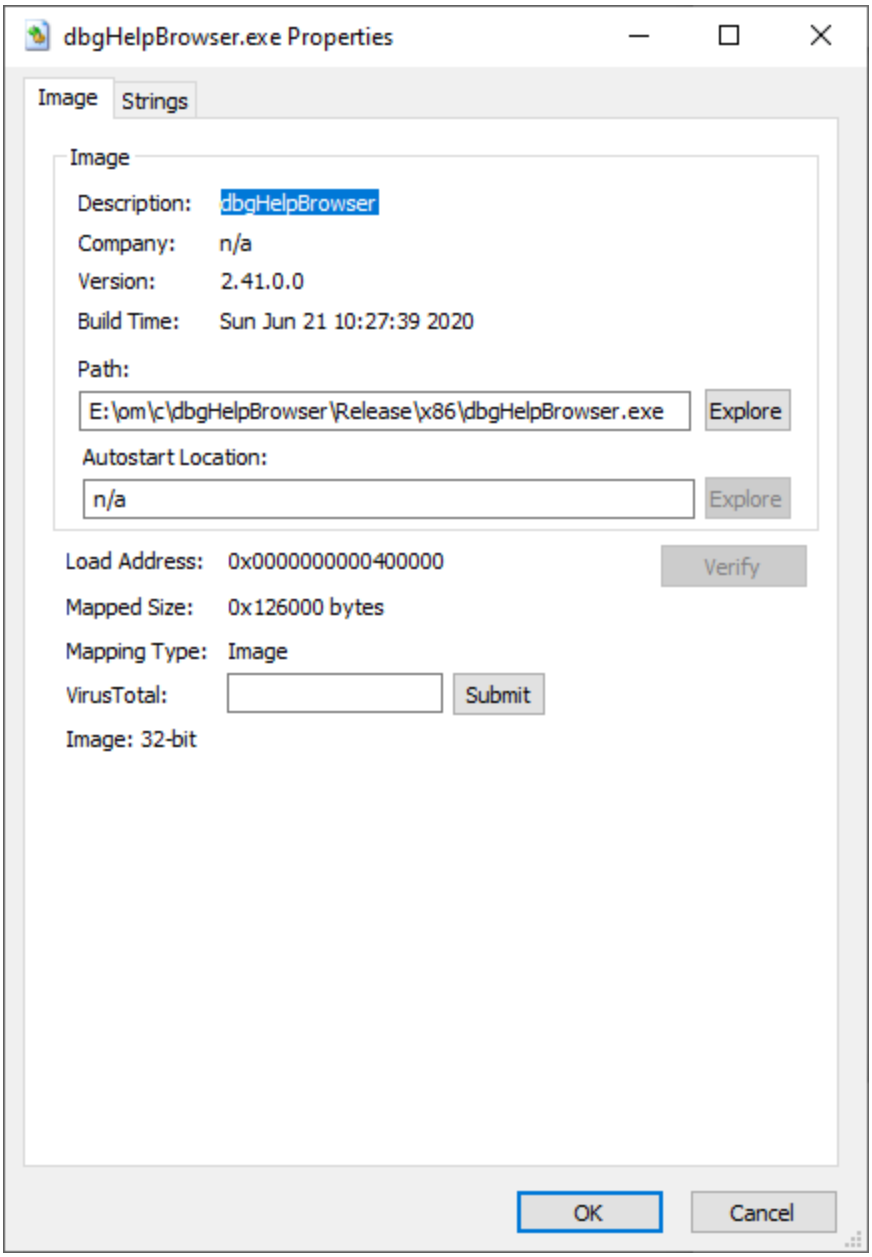
Process Explorer
https://technet.microsoft.com/en-us/sysinternals/processexplorer.aspx

This works for 32 bit and 64 bit programs.

- Start your program
- Start Process Explorer. *If your program is a service or runs as administrator you'll need to start Process Explorer as administrator.*
- In Process Explorer, enable View -> Show Lower Pane. Then for View -> Lower Pane Window, choose DLLs.
- Select your program in the top window.
- Find your DLL in the bottom window. Right click. Choose Properties from the Context menu.



- In the Properties dialog, read the load address.

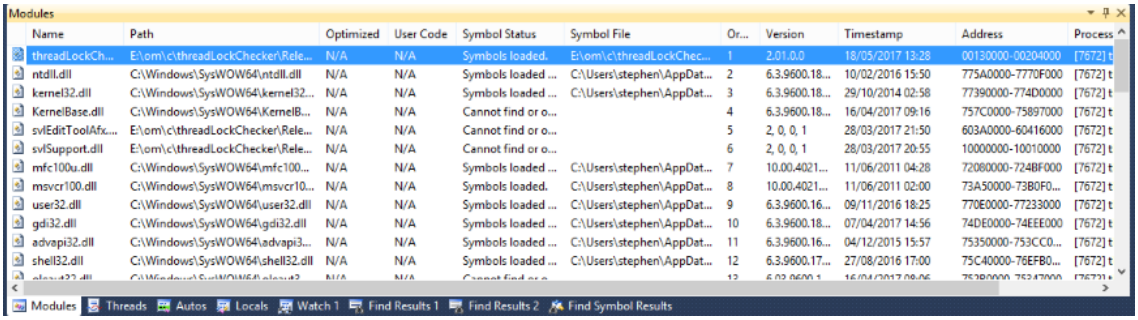


In the example above, for dbgHelpBrowser.exe, the load address is 0x00400000.

Visual Studio (any version)
https://www.visualstudio.com/

- Start Visual Studio.
- From the Project menu, choose File -> Open -> Solution. Choose your executable.
- From the Debug menu, choose Start Debugging.
- From the Debug menu, choose Windows -> Modules.

- In the Modules window, find your DLL, then read the Address column.

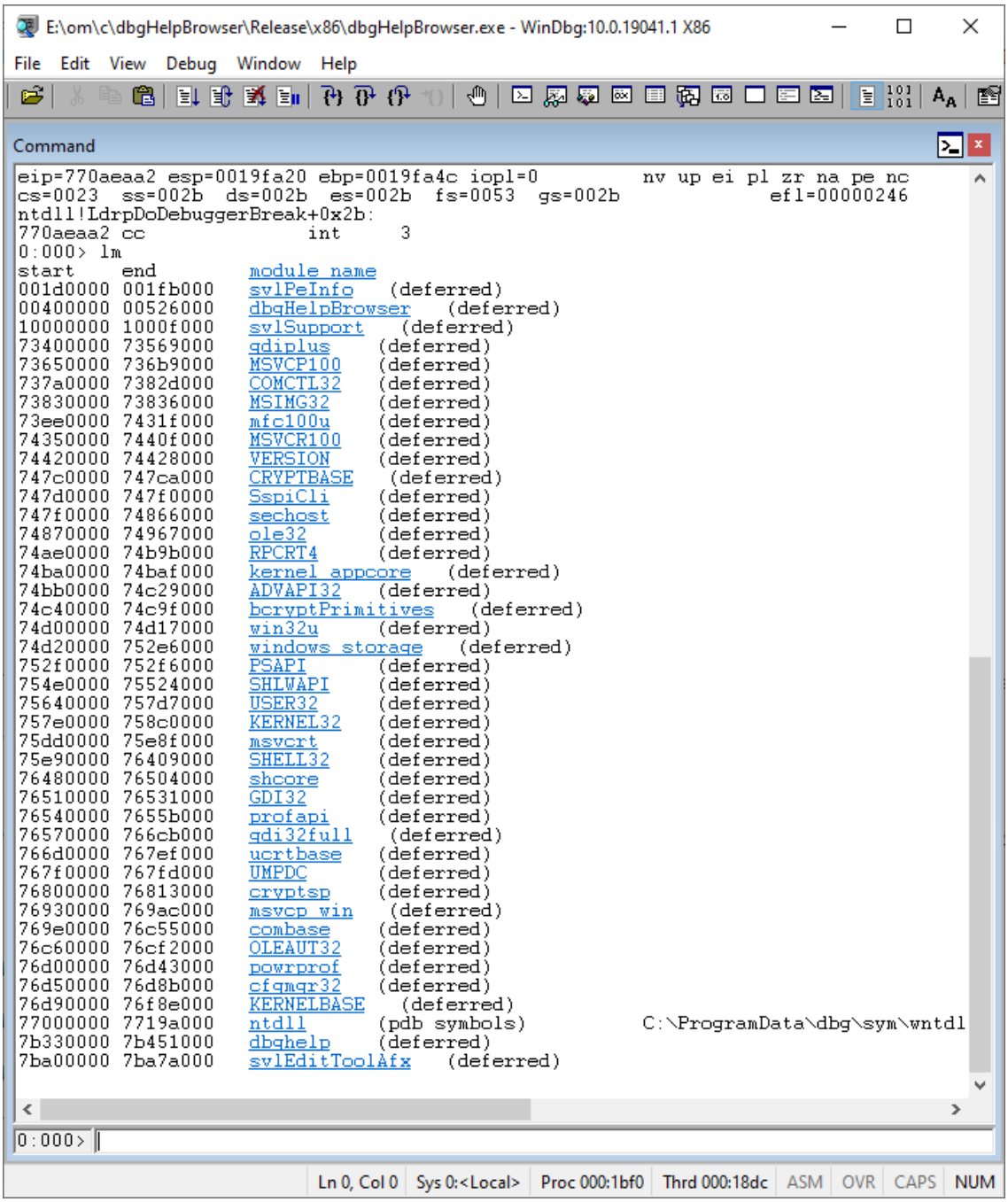


Name	Path	Optimized	User Code	Symbol Status	Symbol File	Or...	Version	Timestamp	Address	Process
threadLockCh...	E:\om\c\threadLockChecker\Rele...	N/A	N/A	Symbols loaded.	E:\om\c\threadLockChec...	1	2.01.0.0	18/05/2017 13:28	00130000-00204000	[7672] t
ntdll.dll	C:\Windows\SysWOW64\ntdll.dll	N/A	N/A	Symbols loaded ...	C:\Users\stephen\AppDat...	2	6.3.9600.18...	10/02/2016 15:50	775A0000-7770F000	[7672] t
kernel32.dll	C:\Windows\SysWOW64\kernel32...	N/A	N/A	Symbols loaded ...	C:\Users\stephen\AppDat...	3	6.3.9600.18...	29/10/2014 02:58	77390000-774D0000	[7672] t
KernelBase.dll	C:\Windows\SysWOW64\KernelB...	N/A	N/A	Cannot find or o...		4	6.3.9600.18...	16/04/2017 09:16	757C0000-75897000	[7672] t
svlEditToolAfx....	E:\om\c\threadLockChecker\Rele...	N/A	N/A	Cannot find or o...		5	2, 0, 0, 1	28/03/2017 21:50	603A0000-60416000	[7672] t
svlSupport.dll	E:\om\c\threadLockChecker\Rele...	N/A	N/A	Cannot find or o...		6	2, 0, 0, 1	28/03/2017 20:55	10000000-10010000	[7672] t
mfc100u.dll	C:\Windows\SysWOW64\mfc100...	N/A	N/A	Symbols loaded ...	C:\Users\stephen\AppDat...	7	10.00.4021...	11/06/2011 04:28	72080000-724BF000	[7672] t
msvcr100.dll	C:\Windows\SysWOW64\msvcr10...	N/A	N/A	Symbols loaded.	C:\Users\stephen\AppDat...	8	10.00.4021...	11/06/2011 02:00	73A50000-73B0F0...	[7672] t
user32.dll	C:\Windows\SysWOW64\user32.dll	N/A	N/A	Symbols loaded ...	C:\Users\stephen\AppDat...	9	6.3.9600.16...	09/11/2016 18:25	770E0000-77233000	[7672] t
gdi32.dll	C:\Windows\SysWOW64\gdi32.dll	N/A	N/A	Symbols loaded ...	C:\Users\stephen\AppDat...	10	6.3.9600.18...	07/04/2017 14:56	74DE0000-74EEE000	[7672] t
advapi32.dll	C:\Windows\SysWOW64\advapi3...	N/A	N/A	Symbols loaded ...	C:\Users\stephen\AppDat...	11	6.3.9600.16...	04/12/2015 15:57	75350000-753CC0...	[7672] t
shell32.dll	C:\Windows\SysWOW64\shell32.dll	N/A	N/A	Symbols loaded ...	C:\Users\stephen\AppDat...	12	6.3.9600.17...	27/08/2016 17:00	75C40000-76EFB0...	[7672] t

In the example above, for threadLockChecker.exe, the load address is 0x00130000.

WinDbg
https://msdn.microsoft.com/en-gb/library/windows/hardware/ff551063(v=vs.85).aspx

- Start WinDbg
- From the File menu, choose Open Executable. Choose your executable.
- Type lm, then press return.
- All modules are listed. Find your module. The start address is the load address.



In the example above, for threadLockChecker.exe, the load address is 0x00130000.

Final Comments

OK, you should now know how to find the load address of a DLL or an EXE (or any module type). Remember that a load address obtained this way is only valid for symbol decoding if the executable doesn't have ASLR applied to it.

If your crash reporting code only grabs crash addresses and not DLL load addresses, you need to update your code so that you grab DLL load addresses at the time of the crash. That way you know for sure what the load addresses were and you won't have to guess the load addresses in future.

Part VIII

8 Command Line Interface

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

The command line options allow you to view PDB debug information that is embedded in an executable file, and optionally highlight a symbol at a specified offset.

/fileName

Specifies the module to load. This is typically a .exe or a .dll.

/fileName path-to-executable

Example: /fileName e:\om\c\test\release\test.exe

/offset

Specifies an offset inside the executable. DbgHelp Browser will highlight the symbol that occupies this location.

Typically this offset will be calculated from a crash location.

For example:

If a DLL is loaded at 0x00400000 and a crash happens at 0x00420192, the offset is calculated by subtracting the DLL load address from the crash address.

That is: 0x00420192 - 0x00400000, which gives 0x00020192.

The offset is 0x00020192.

The offset must be specified in hexadecimal with a leading 0x.

/offset value

Example: /offset 0x00020192

Example Command Line

32 bit applications

```
dbgHelpBrowser.exe /fileName e:\test\release\test.exe /offset 0x00020192
```

64 bit applications

```
dbgHelpBrowser_x64.exe /fileName e:\test\release\test.exe /offset 0x00020192
```