Thread Validator

by

Software Verify

SoftwareVerify
Troubleshoot your software

Thread Validator

Thread deadlock and error detection for Win32 applications

by Software Verify Limited

Welcome to the Thread Validator software tool. Thread Validator is a software tool designed to detect threading errors and thread deadlocks in software applications.

Thread Validator provides metrics about each lock in your application, monitors historical data about lock entry, lock exit and thread state and has the ability to detect thread deadlocks and display the cause of the deadlock.

We hope you will find this document useful.

Thread Validator Help

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: May 2025 in United Kingdom.

Table of Contents

Part I

1 Overview

Hi, welcome to the Thread Validator help manual.

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

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

Tutorials for Thread Validator are available at https://www.softwareverify.com/tutorial/thread-validator-tutorial/.

Before reading this manual, it's worth taking a quick look at the notation used.

Read background information

The overview section covers things like:

- the capabilities of Thread Validator
- how it works
- what's supported
- how to purchase

If you've already purchased, thank you!

Learn about getting started

You *can* skip the background information, but do make sure you're aware of how to prepare your target program in the getting started section.

Dive right in

The quick start section shows how to launch your application.

To find your way around the rest of the features and settings then read about the user interface, or browse the examples.

If you're already feeling confident you can learn about some of the advanced features such as comparing sessions.

1.1 Notation used in this help

› Instruction › steps
☰ Menu action › steps

Throughout the help you'll find instruction steps like this:

- **Filter...** › shows the session comparison private filters dialog

or

☰ **Settings** Menu › **Edit Settings...** › **Data Collection** in the list › **Trace Hooks**

This is a shorthand notation for performing consecutive steps in the user interface.

The first example indicates that the action of clicking the **Filter...** will result in showing the dialog described.

The second example directs you to open the Settings menu (from the menu bar in this case), and then choose the Settings item, and in the dialog that appears, open the Data Collection option via the list and select the Trace Hooks child entry.

🖱 Right mouse button menu

Where you see this mouse menu the instruction is to use the right mouse button menu (a.k.a. popup menu or context menu) and select the menu option that follows this symbol.

For example: use 🖱 **Edit Source Code...**

👆 Interactive images

Shown next to a picture, the hand symbol 👆 indicates the image is interactive and can be clicked on in order to jump directly to the help section most relevant to the part of the image under the cursor.

↗ External Links

You may see this symbol ↗ after some links. Those links lead to an external website (shown in your default browser), as opposed to jumping to another section in the help. Naturally, if you have no internet access, these links will be unavailable.

For example: Software Verifiy Limited↗

Notes

📝 Warning notes

Notes pertaining to the current topic are indicated by the 📝 symbol. Notes may include exceptions to a rule, items to watch out for, or other asides to the main topic.

Notes that act as warnings will use the similar 📝 symbol, for example where there's a danger of crashing your application. Don't panic though - there aren't many of these!

➡ See also

Where there are other pages in the help that have more detail on the topic at hand, or if there is additional reading that is not already linked within the content, you will find these sections linked after the ➡ symbol.

1.2 Introducing Thread Validator

What is Thread Validator?

Thread Validator is an **automatic deadlock detector** for Windows.

It works with versions of Windows from NT® 4.0 and above, running on the Intel i386 (and compatible) family of processors.

For the purposes of this documentation, we'll call Thread Validator just 'Thread Validator'.

What does Thread Validator do?

Thread Validator can:

- detect deadlocks
- detect potential deadlocks
- find bad lock strategies

  When deadlocks, potential deadlocks and bad lock strategies are detected, you can examine the order in which each lock was acquired, complete with a stack trace for each lock.

- perform deadlock postmortem analysis

  If your program deadlocks you can attach to it with Thread Validator and ask for a complete stack trace for each thread in the application.

  Typically if you try to do this with a debugger, each deadlocked thread just shows one or two stack entries inside NTDLL.DLL and no stack entries for your program, so you don't know which part of *your* program deadlocked.

Thread Validator uses proprietary stack-walking routines to walk the entire stack, so that a complete stack trace can be displayed for deadlocked threads.

The results are displayed as a variety of comprehensive but easily explorable tabulated and hierarchical formats.

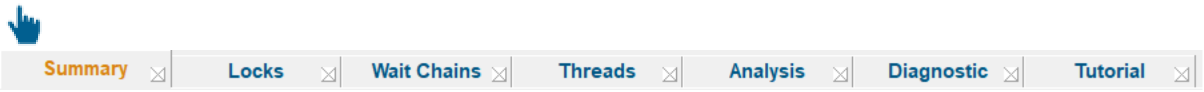
Source code editing is provided with colour coded lines so that you can see at a glance which functions consume more time than others.

The performance overhead is very low and there is no need to recompile or relink the target program.

Thread Validator can also be used as part of a regression testing strategy by Quality Assurance teams.

The main sections of Thread Validator

The user interface is split into tabbed report sections (plus Tutorials), each presenting or analysing locks in the target program at different levels of granularity.



Here's a summary of those sections, each of which is covered in full in The User Interface section.

Summary A high level overview of all threading activity, showing data on threads, locks, contentions, recursions, waits, errors and thread coverage.

Locks Master tab for three different tabs, each providing a different view of all the locks, waits and threads in the application.

Wait Chains Displays information about the lock wait chains

Threads This display provides a graphical view of thread activity over time.

Analysis Master tab for four different tabs providing query, thread coverage, object creation data, and object handles.

Diagnostic Logs diagnostic information collected by the stub, including functions that could not be hooked.

1.3 Why Thread Validator?

Adapts to everyone's workflow

Thread Validator allows you to find otherwise hard to isolate errors using an intuitive colour-coded user interface.

Data can be collected using a variety of different query methods. When faced with a large amount of data you can search or filter the data.

If you want to see the source code that caused a deadlock, just double click on the display entry to view source code in the adjacent window.

Alternatively, to edit the code, double click on a code fragment shown and the appropriate source code file will be loaded into Thread Validator's colour-coded editor, or into Microsoft® Visual Studio®, or you can choose your preferred awesome editor.

You can save sessions, reload them at a later date and still interact with the analysis data.

You can also export session data to HTML or XML which can be used to create reports targeted to the appropriate audience: the management team; quality assurance team; or to create detailed stack trace reports for the software engineers.

Designed with principles

Thread Validator and the other products in our suite of tools have been created with the following principles in mind:

MINIMUM IMPACT
INDEPENDENT
RELIABLE
FLEXIBLE
DO NO HARM

- **must not adversely effect the program's behaviour**

  Any hooks placed into the target program's code must not affect the registers or the condition code flags of that program. The program must behave in the same way when being inspected by Thread Validator as without.

- **must be reliable and avoid causing the target program to crash**

  Since we can't know exactly which DLLs and other components are present on every computer that Thread Validator is installed on, every hook can be enabled or disabled, and/or installed or not installed.

  Thus if a new DLL is released in the future that causes problems with certain functions, you can disable the hooks for those functions, and continue using Thread Validator until a fix for the new DLL's behaviour is available.

- **must have as little impact on the target program's performance as possible**

  Thread Validator has very little effect on the target applications performance, but you can also enable and disable as many or as few threading hooks as you wish.

For example:

If you're only interested in monitoring performance of a particular area of code, you can pick only that directory to be hooked.

If you're only interested in a selection of in-house DLLs, choose only those modules to be hooked.

- **must have a user interface independent of the target program**

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

This means:

If the target program crashes, the user interface will not crash - you will still have data to work with.

If the target program is stopped in the debugger, Thread Validator's user interface will continue to work.

In the unlikely event that the Thread Validator's own user interface crashes, your target program will not crash.

- **must be flexible**

We know our users like choices! Where there are multiple ways of presenting the data, the user is given a choice over how that display works, so that not all users have to work with the same settings.

1.4 What do you need to run Thread Validator?

Compilers

The following makes of compiler are supported:

- Microsoft® Visual Studio®
- Borland C++
- Borland Delphi
- Intel
- Metrowerks
- MinGW
- QtCreator
- Fortran (various)

➡ Supported compilers for more details regarding versions and caveats.

User Privileges

Thread Validator uses the `CreateRemoteThread()` Win32 function. You must have access privileges that allow you to create threads in other programs.

Typically **Administrator** and **Power User** user types have the appropriate privileges. Ordinary user accounts can be easily modified to have the required privileges.

➡ Learn more about user privileges in the section on User Permissions.

Registry Access Privileges

Thread Validator requires read and write access to:

- `HKEY_CURRENT_USER\Software\SoftwareVerification\ThreadValidator`
- `HKEY_USERS\.DEFAULT\Software\SoftwareVerification\ThreadValidator`.

  This is used when working with services

If read and write access is *not* allowed:

- Thread Validator will use default settings (thus any user selections will not apply)
- Error messages will be displayed when Thread Validator tries to access the registry key

  These error messages can be suppressed if they are not desired. For example, if you're not working with services, then there's no requirement to access the second registry key, and all error messages relating to it can be ignored.

➡ The question relating to creating Power User accounts for Windows XP.

Operating System

Any 'modern' windows machine is suitable to run Thread Validator.

At a minimum, Thread Validator requires Windows NT® 4.0 or better.

Thread Validator will *not* run on the following platforms, because the `CreateRemoteThread()` Win32 function and named pipes are not available:

- Windows 95®
- Windows 98®
- Windows Me®

Any newer operating systems do not *need* any additional service packs but we generally recommend being up to date where possible anyway.

For older systems, we recommend using the minimum service pack levels below:

- Windows NT 4.0: Service Pack 6
- Windows 2000: Service Pack 2

1.5 Buying Thread Validator and Support

The best way to purchase Thread Validator is online from Software Verify Limited - just click the ***Purchasing*** link at the top of the website.

Purchase options

There are options for single or multiple licenses, per-user or floating licenses, and although you can of course purchase it as a single product, you can save significantly by buying Thread Validator as part of a suite of products. All the details are online.

Pre-purchase questions?

If you have any pre-purchase questions not answered in this help manual, or niggling little doubts about something, we can be contacted as below.

email: sales@softwareverify.com *(recommended)*

web: https://www.softwareverify.com

or by old fashioned post:

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

After sales support

If you need support after purchase, check our frequently asked questions and then drop us a line below with as much detail as possible about your problem.

email: support@softwareverify.com

1.6 How does Thread Validator work?

The Stub and the UI - more than the sum of its parts

Thread Validator has two main parts - the *stub* and the *user interface*.

The **stub** is typically injected into the target program and communicates with the Thread Validator user interface.

The stub is injected into the target program using the `CreateProcess()` or `CreateRemoteThread()` Win32 function. Communication between the stub and the user interface is via named pipes. There is no human readable data sent between the two parts of the program. Both the stub and the user interface are multi-threaded. If required the stub can be linked into the program so that it doesn't need to be injected into the program.

The stub walks the entire program image detecting the start of each source code line using PDB and/or MAP files.

The stub rewrites the DLL import address table to make functions call into the stub's hooks. Each function is checked to see if it can safely be hooked without corrupting the code for another line or function, or changing the function of the program. The line is hooked if possible, otherwise the user interface is informed of the function hook failure.

The stub monitors thread and lock related data and stores the information in data structures to be displayed by the user interface.

1.7 Supported Compilers

Thread Validator will work with any portable executable (PE) file format and supports C++, Delphi, Fortran 95 and Visual Basic.

C/C++ runtime functions are provided by individual compiler vendors, not all of whom still maintain, support the products listed.

The following C / C++ compilers are supported by Thread Validator.

Microsoft http://www.microsoft.com

Thread Validator requires your application to be built using Microsoft® Visual Studio® 6.0 service pack 3 or later.
In practice, you may find that applications built with Developer Studio 4.2 and later can be used with Thread Validator.

- **Microsoft Developer Studio 4.0**
- **Microsoft Developer Studio 5.0**

- **Microsoft Developer Studio 6.0**
- **Microsoft Visual Basic 6.0**
- **Microsoft Visual Studio 6.0 -** service pack 3 or later
- **Microsoft Visual Studio 7.0 / .net 2002**
- **Microsoft Visual Studio 7.1 / .net 2003**
- **Microsoft Visual Studio 8.0 / .net 2005**
- **Microsoft Visual Studio 9.0 / .net 2008**
- **Microsoft Visual Studio 10.0 / .net 2010**
- **Microsoft Visual Studio 11.0 / .net 2012**
- **Microsoft Visual Studio 12.0 / .net 2013**
- **Microsoft Visual Studio 14.0 / .net 2015**
- **Microsoft Visual Studio 15.0 / .net 2017**
- **Microsoft Visual Studio 16.0 / .net 2019**
- **etc...**

Microsoft Developer Studio and Microsoft Visual Studio products support both C++ and Visual Basic.

➡ Visual Studio and Visual Basic 6 in the *Getting Started* section.

Intel http://www.intel.com

- **Intel performance compiler -** The Intel compiler uses the Microsoft runtime already installed on your computer rather than supply its own

- **Intel Fortran**

Intel use Microsoft's PDB proprietary symbol information format. If your compiler uses PDB symbol information Thread Validator will be able to use it.

Metrowerks

- **Metrowerks CodeWarrior for Windows Version 8.0**
- **Metrowerks CodeWarrior for Windows Version 9.0**

You will need to ensure the debug information is stored as *CodeView* information and not a custom Metrowerks debug format. Metrowerks symbolic information is embedded in the .exe/.dll as CodeView information. Please consult the documentation for CodeWarrior to include debug information (including filenames and line numbers) in the CodeView information.

Embarcadero https://www.embarcadero.com/

This includes compilers formerly produced by Borland.

- **C++ Builder 5.0 to C++ Builder 11**
- **Delphi 6.0 to Delphi 11**
- **Rad Studio**

➡ C++ Builder and Delphi in the *Getting Started* section.

MinGW http://www.mingw.org

- **MinGW** (Minimalist GNU for Windows)

MinGW can create symbols in a variety of formats. If you configure MinGW to produce DWARF symbols, STABS symbols or COFF symbols Thread Validator can read them.

➡ MinGW compiler in the *Getting Started* section.

Qt (Digia, Nokia, Trolltech) http://qt.io

- **QtCreator**

➡ Ensure that debug information is created in DWARF, STABS or COFF formats.

Salford Software now maintained by SilverFrost

- **Salford Software Fortran 95**

Salford Software Fortran 95 uses COFF symbol information. If your compiler uses COFF symbol information Thread Validator will be able to use that information.

Note: While Fortran 95 provides multi-threading capabilities, FORTRAN 90 Standard does not.

Compaq

- **Compaq Visual Fortran 6.6**

The Compaq Visual Fortran product may be compatible with Thread Validator. If you are using Compaq Visual Fortran and wish to use Thread Validator please contact us.

Other...?

If the compiler you are using is not listed here, please contact us for advice. We add compilers as we receive requests for them, and many of the above were added at the request of customers.

1.8 User Permissions

This section details the privileges a user requires to successfully run Thread Validator.

Typically, *Administrator* and *Power User* user types will already have the appropriate privileges.

Why do user privileges matter?

Debugging tools such as Thread Validator are intrusive tools - they require specific privileges not normally granted to typical applications.

Thread Validator requires specific privileges to write to the default user profile in the registry.

This is so that when Thread Validator is working with services (or any application run on an account which is not the current user's account) it can read the registry and the correct configuration data.

If the account upon which a service or application is running is *not* the user's account, the fallback position is the DEFAULT account in `HKEY_USERS\.DEFAULT`.

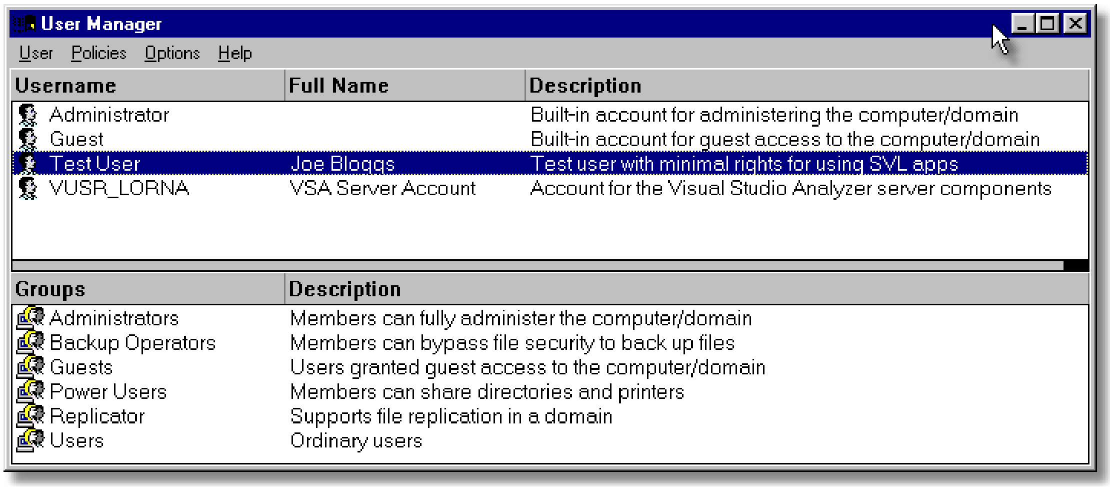
You can enable and disable various warnings using the User Permissions Warnings dialog.

User privileges

Thread Validator requires the following privilege to allow debugging of applications and services:

Debug Programs (SE_DEBUG_NAME)

Ordinary users will need to be granted these permissions using the Administrative *User Manager* tool. The example below shows the NT4 User Manager - the Windows 2000 User Manager and Windows XP User Manager will be different but similar in principle.

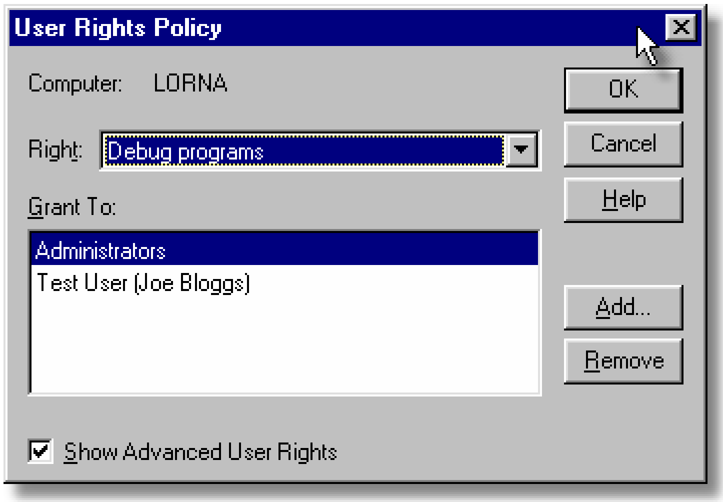


Username	Full Name	Description
Administrator		Built-in account for administering the computer/domain
Guest		Built-in account for guest access to the computer/domain
Test User	Joe Bloggs	Test user with minimal rights for using SVL apps
VUSR_LORNA	VSA Server Account	Account for the Visual Studio Analyzer server components

Groups	Description
Administrators	Members can fully administer the computer/domain
Backup Operators	Members can bypass file security to back up files
Guests	Users granted guest access to the computer/domain
Power Users	Members can share directories and printers
Replicator	Supports file replication in a domain
Users	Ordinary users

In the User Manager select the user - in this case "Test User".

Choose: **Policies** Menu › **User Rights** › check **Show Advanced User Rights** › select **Debug Programs** in the **Right** drop down list



Computer: LORNA

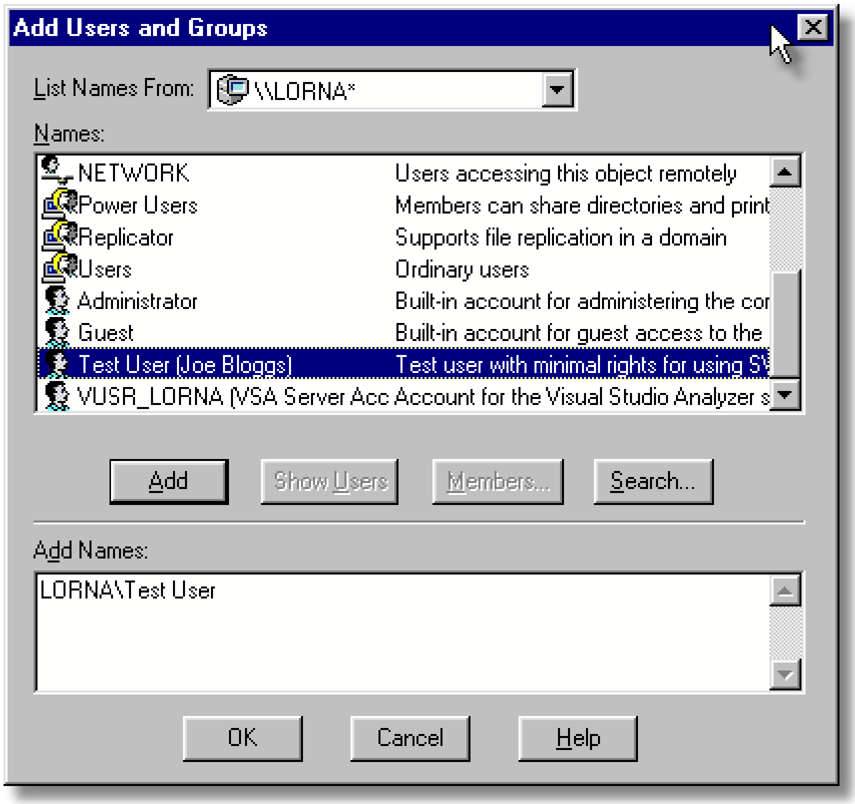
Right: Debug programs

Grant To:

- Administrators
- Test User (Joe Bloggs)

☑ Show Advanced User Rights

Click **Add.... > Show Users**



List Names From: \\LORNA*

Names:

NETWORK	Users accessing this object remotely
Power Users	Members can share directories and print
Replicator	Supports file replication in a domain
Users	Ordinary users
Administrator	Built-in account for administering the cor
Guest	Built-in account for guest access to the
Test User (Joe Bloggs)	Test user with minimal rights for using S\
VUSR_LORNA (VSA Server Acc Account for the Visual Studio Analyzer s	

Add Names:

LORNA\Test User

Select **[ComputerName]\Test User** in the top list. Click **Add > OK > OK >** Close the User Manager.

Registry access privileges

Thread Validator requires read and write access to:

- `HKEY_CURRENT_USER\Software\SoftwareVerification\ThreadValidator`

- `HKEY_USERS\.DEFAULT\Software\SoftwareVerification\ThreadValidator.`

  This is used when working with services

If read and write access is *not* allowed:

- Thread Validator will use default settings (thus any user selections will not apply)
- Error messages will be displayed when Thread Validator tries to access the registry key

  These error messages can be suppressed if they are not desired. For example, if you're not working with services, then there's no requirement to access the second registry key, and all error messages relating to it can be ignored.

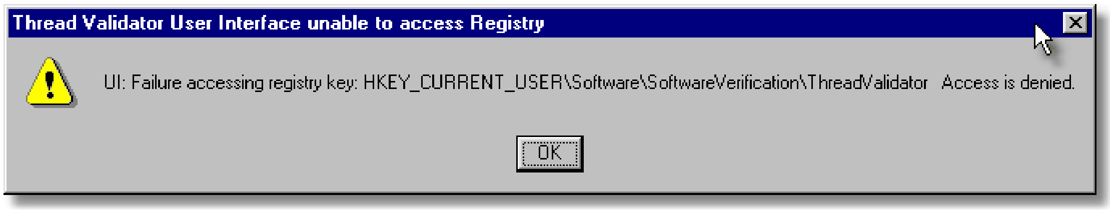
You can modify the registry access permissions using the **regedt32.exe** tool Security menu (or similar). Ask your administrator to modify your registry access permissions if you can't do this yourself.

➡ What's the difference between Regedit and Regedt32?

Error notifications

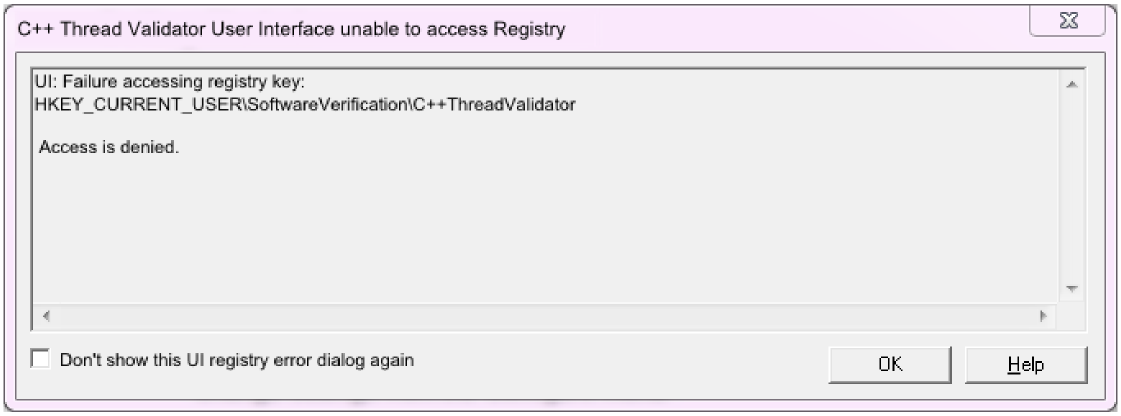
When Thread Validator fails to gain access for read or write to the registry a message box is displayed indicating if the error is for the user interface (UI) or Services. The message indicates the name of the registry key that failed and the failure reason.

This simple message box is displayed during early startup and late close-down of Thread Validator:


Thread Validator User Interface unable to access Registry

UI: Failure accessing registry key: HKEY_CURRENT_USER\Software\SoftwareVerification\ThreadValidator Access is denied.

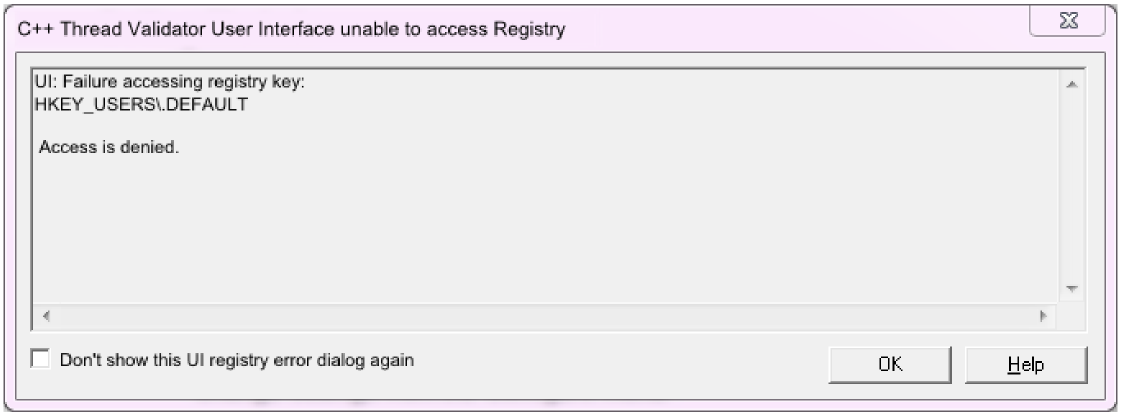
Message boxes like the following are displayed when Thread Validator is not starting up or closing down. The messages differ in the registry key.


C++ Thread Validator User Interface unable to access Registry

UI: Failure accessing registry key:
HKEY_CURRENT_USER\SoftwareVerification\C++ThreadValidator

Access is denied.

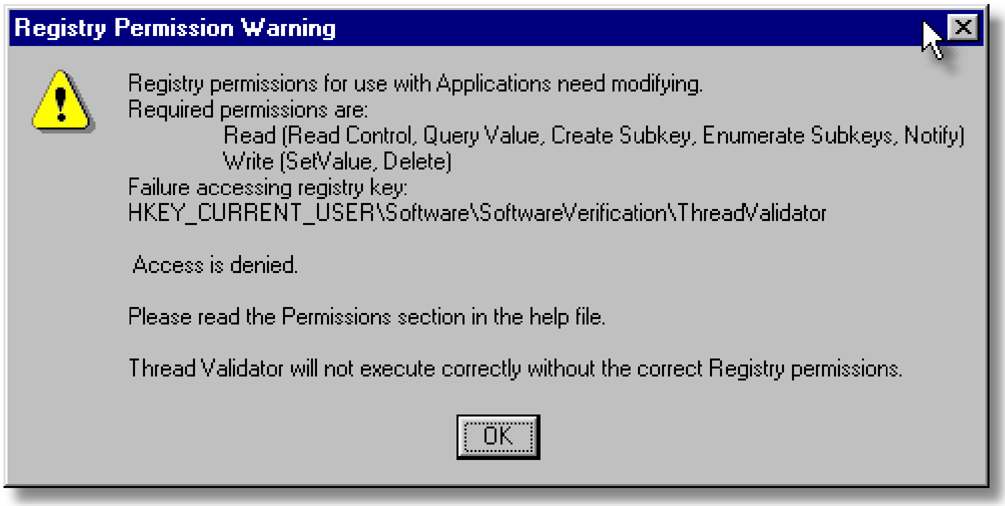
□ Don't show this UI registry error dialog again


C++ Thread Validator User Interface unable to access Registry

```
UI: Failure accessing registry key:
HKEY_USERS\.DEFAULT

 Access is denied.
```

Detailed registry access error messages

The following detailed registry access error message is also displayed when failing to gain access to the registry keys.


Registry Permission Warning

Registry permissions for use with Applications need modifying.
Required permissions are:
Read (Read Control, Query Value, Create Subkey, Enumerate Subkeys, Notify)
Write (SetValue, Delete)
Failure accessing registry key:
HKEY_CURRENT_USER\Software\SoftwareVerification\ThreadValidator

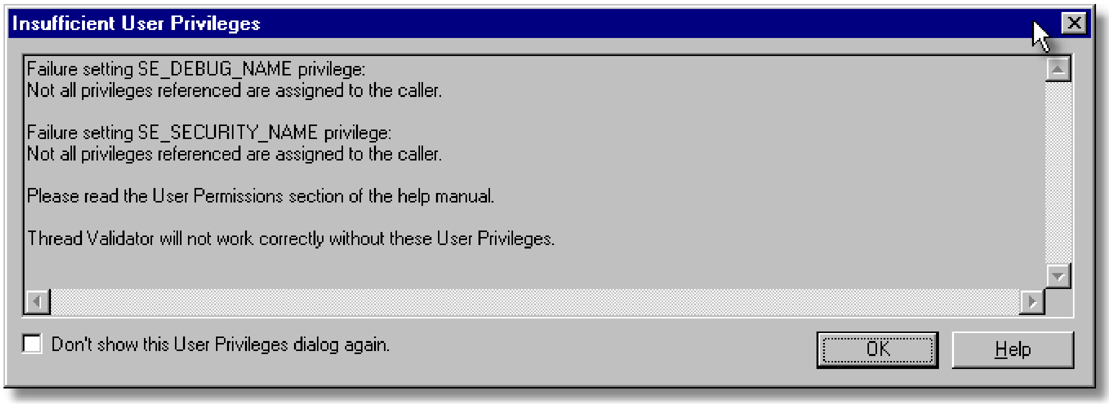
Access is denied.

Please read the Permissions section in the help file.

Thread Validator will not execute correctly without the correct Registry permissions.

Insufficient user privileges

The following dialog is displayed if a user has insufficient privileges to use the software correctly.



Insufficient User Privileges

Failure setting SE_DEBUG_NAME privilege:
Not all privileges referenced are assigned to the caller.

Failure setting SE_SECURITY_NAME privilege:
Not all privileges referenced are assigned to the caller.

Please read the User Permissions section of the help manual.

Thread Validator will not work correctly without these User Privileges.

☐ Don't show this User Privileges dialog again.

Without the **Debug Programs** privilege, Thread Validator will not work correctly with *Services*, and may not work correctly with *Applications*.

➡ How to create Power User accounts for Windows XP.

Part II

2 Getting Started

For those that wish to 'dive in', this section will make you aware of how to prepare your target program before giving a quick start introduction.

Otherwise skip right on to the next chapter - The User Interface.

Diving in?

You're probably wanting to 'dive in' and start analysing the threads in your application, looking for deadlocks right away.

However, if you choose to read this manual first, you'll find out more about Thread Validator and how to leverage it to its full advantage.

For new users of Thread Validator, a configuration wizard appears the first time you run the application. This ends with a brief overview video.

We also recommend watching the tutorials online - it's an easy way to explore the functionality available.

2.1 Enabling Debugging

To get the best from our tools you will need to enable debugging information for your compiler and your linker.

Detailed instructions are available for these IDEs / compilers:

- Visual Studio
- Visual Basic 6
- C++ Builder
- Delphi
- MingW, gcc, g++
- Dev C++
- Salford Software FORTRAN 95
- Metrowerks Code Warrior

Debug Information Formats

Thread Validator can understand debugging information in the following formats:

- Microsoft Program Database (PDB)
- Turbo Debugger Symbols (TDS)
- COFF
- DWARF
- STABS

The Intel Performance Compiler and Intel Fortran compilers produce symbols in Microsoft's PDB format.

2.2 Quick Start

If you are:

- an admin level user
- using Microsoft compilers
- on a modern OS
- already know that you create debug info in your debug and release product

...then you're more than likely good to go and dive in!

Otherwise, we recommend reading these topics from the Overview section before starting:

- What do you need to run Thread Validator?
- Supported Compilers
- User Permissions

Testing on the Example Program

You can test drive the capabilities of Thread Validator by launching the example program supplied with Thread Validator - **nativeExample.exe**.

The example program can be used in conjunction with the Thread Validator tutorials.

Ensure you have debugging information

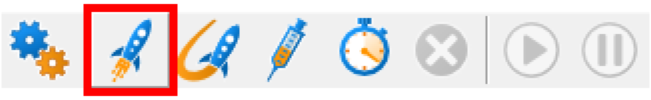
Your application needs to be compiled to produce debugging information and linked to make that debugging information available.

We provide some details for enabling debugging information with Visual Studio, C++ Builder, Delphi, MinGW, and other compilers.

If you have no PDB debugging information but you do have a Microsoft format MAP file available, it *must* contain line number information by using the /MAPINFO:LINES linker directive.

Launching

To start your program click on the launch icon on the Session toolbar.

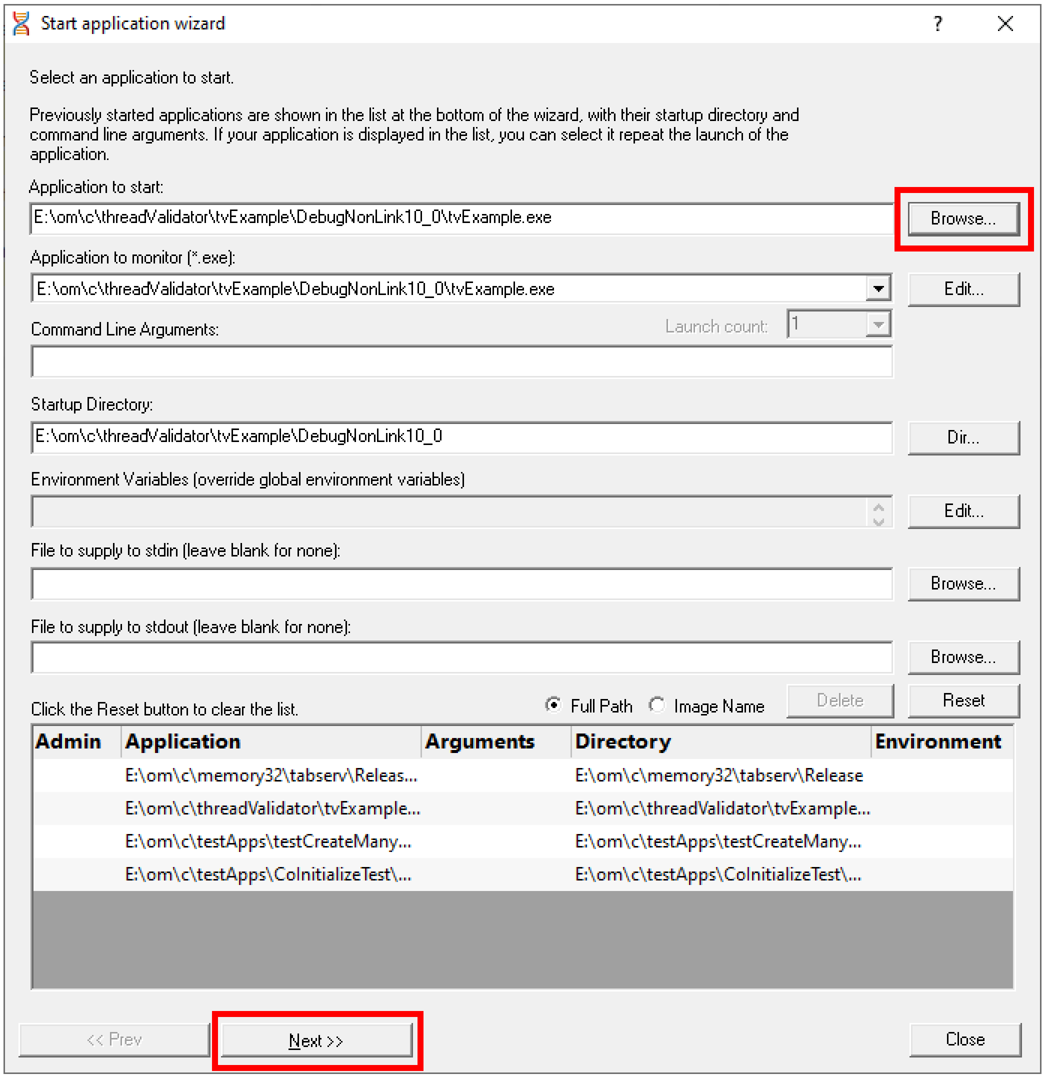


What you see next depends on the user interface mode (wizard or dialog style).

The Launch Wizard...

If you've just installed the software you will be shown the launch wizard:

Click **Browse...** to choose a program to launch › **Next** › **Next** › **Next** › **Start Application...**



Select an application to start.

Previously started applications are shown in the list at the bottom of the wizard, with their startup directory and command line arguments. If your application is displayed in the list, you can select it repeat the launch of the application.

Application to start: E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe

Application to monitor (*.exe): E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe

Command Line Arguments:

Launch count: 1

Startup Directory: E:\om\c\threadValidator\tvExample\DebugNonLink10_0

Environment Variables (override global environment variables)

File to supply to stdin (leave blank for none):

File to supply to stdout (leave blank for none):

Click the Reset button to clear the list.

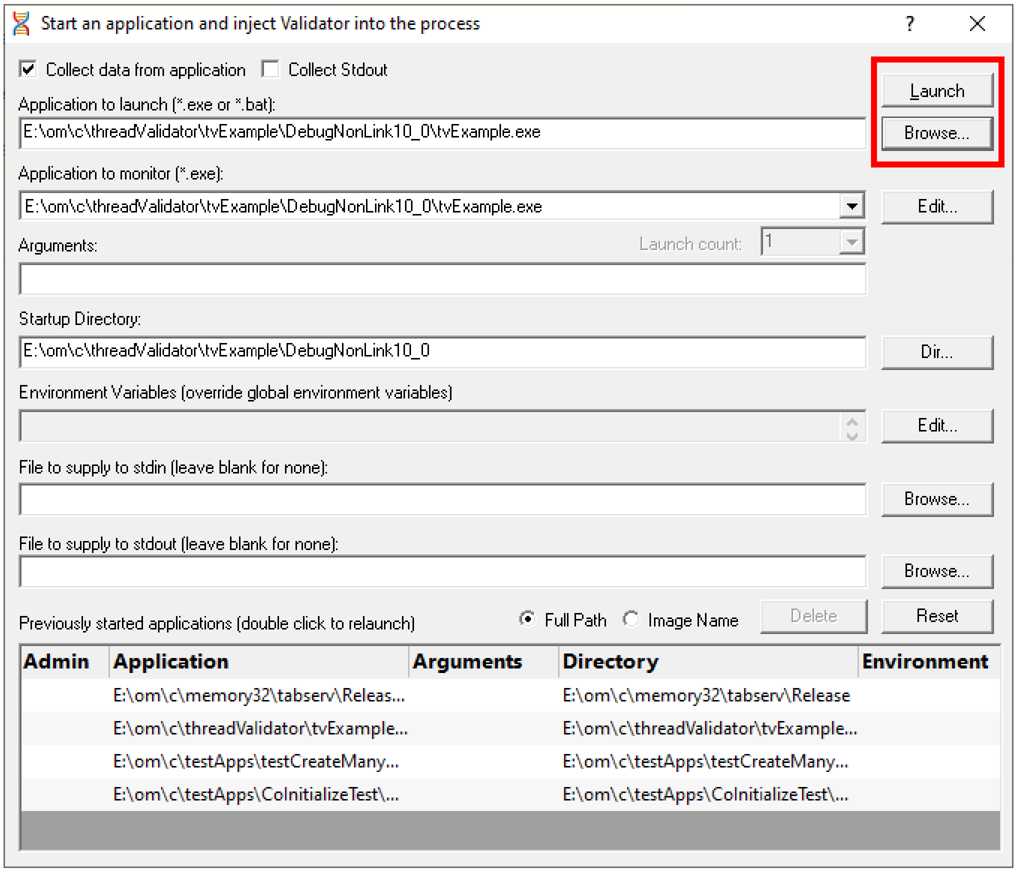
● Full Path ○ Image Name

Admin	Application	Arguments	Directory	Environment
	E:\om\c\memory32\tabserv\Releas...		E:\om\c\memory32\tabserv\Release	
	E:\om\c\threadValidator\tvExample...		E:\om\c\threadValidator\tvExample...	
	E:\om\c\testApps\testCreateMany...		E:\om\c\testApps\testCreateMany...	
	E:\om\c\testApps\CoInitializeTest\...		E:\om\c\testApps\CoInitializeTest\...	

...or the Launch Dialog

If you have switched to Dialog mode you will be shown the launch dialog:

Click **Browse...** to choose a program to launch › **Launch**



Start an application and inject Validator into the process

☑ Collect data from application ☐ Collect Stdout

Application to launch (*.exe or *.bat):
E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe

Application to monitor (*.exe):
E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe

Arguments:

Launch count: 1

Startup Directory:
E:\om\c\threadValidator\tvExample\DebugNonLink10_0

Environment Variables (override global environment variables)

File to supply to stdin (leave blank for none):

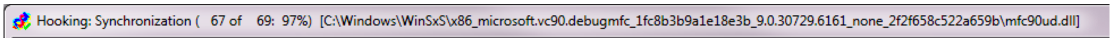
File to supply to stdout (leave blank for none):

Previously started applications (double click to relaunch) ● Full Path ○ Image Name

Admin	Application	Arguments	Directory	Environment
	E:\om\c\memory32\tabserv\Releas...		E:\om\c\memory32\tabserv\Release	
	E:\om\c\threadValidator\tvExample...		E:\om\c\threadValidator\tvExample...	
	E:\om\c\testApps\testCreateMany...		E:\om\c\testApps\testCreateMany...	
	E:\om\c\testApps\CoInitializeTest\...		E:\om\c\testApps\CoInitializeTest\...	

During launch

Thread Validator will start and inject the stub into the target program. Progress during this phase is displayed in the title header of the main window.



Hooking: Synchronization (67 of   69: 97%) [C:\Windows\WinSxS\x86_microsoft.vc90.debugmfc_1fc8b3b9a1e18e3b_9.0.30729.6161_none_2f2f658c522a659b\mfc90ud.dll]

Once correctly installed in the target program, the stub will establish communications with Thread Validator and data can be collected and viewed until the target program exits.

When a deadlock or a bad lock strategy is detected, the data relating to this on each tab is displayed in the relevant colours.

Most tabs will update at intervals (unless set otherwise) to show data collected so far.

After exit - examining the output

When the target program exits, Thread Validator stops collecting more data but keeps all the session data active for you to explore. The data collection icons on the session toolbar are disabled, and the launch icons are enabled again.

Most tabs are automatically updated to reflect the final data, and the other tabs also continue to let you explore the data until the session is closed.

Ending the session

Even though the target program has exited, the session is still active and can be examined or saved until the session is closed:

File menu › **Close Session**

You can have more than one session open at a time.

Part III

3 The User Interface

The part of Thread Validator that you get to see and interact with, is the user interface, but that's only one half of the story.

Behind the scenes, the stub installs and controls the data hooks in the target program and interacts with the user interface.

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

Typical workflow

Typical usage of Thread Validator is very simple:

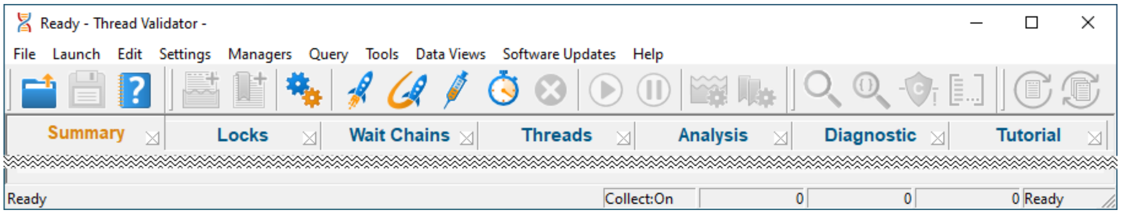
- Start your target program
- Examine the thread lock acquisition strategy of the program
- Close the program
- Analyse final data - saving or exporting data if needed

However, there is much more to Thread Validator than this simple workflow. For example, whilst your program is running, you can display data and gain insight into a specific bug you are looking at in the debugger, or you can monitor the program as a whole, looking out for potential deadlock issues and check coverage of thread related functions.

The user interface

The user interface consists of the menus, toolbars, status bar and the main display tabs.

Read on to find out about all those features, or click parts of the image below to jump directly to any of the menus, tabs or other sections of interest.

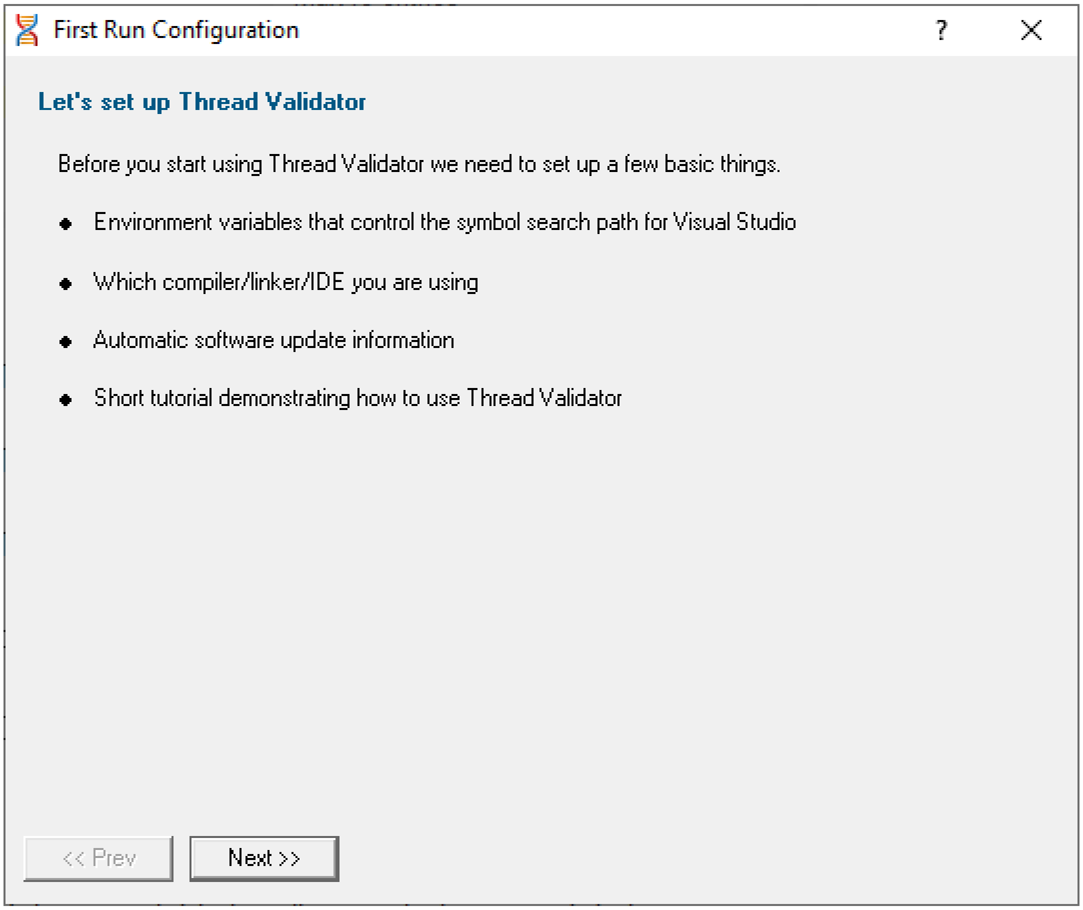


3.1 First run configuration

First run configuration

For new users of Thread Validator, a configuration wizard appears the first time you run the application.

The wizard collects a few details about environment, tools, update requirements (for non-evaluation users) and ends with a short video tutorial.



First Run Configuration

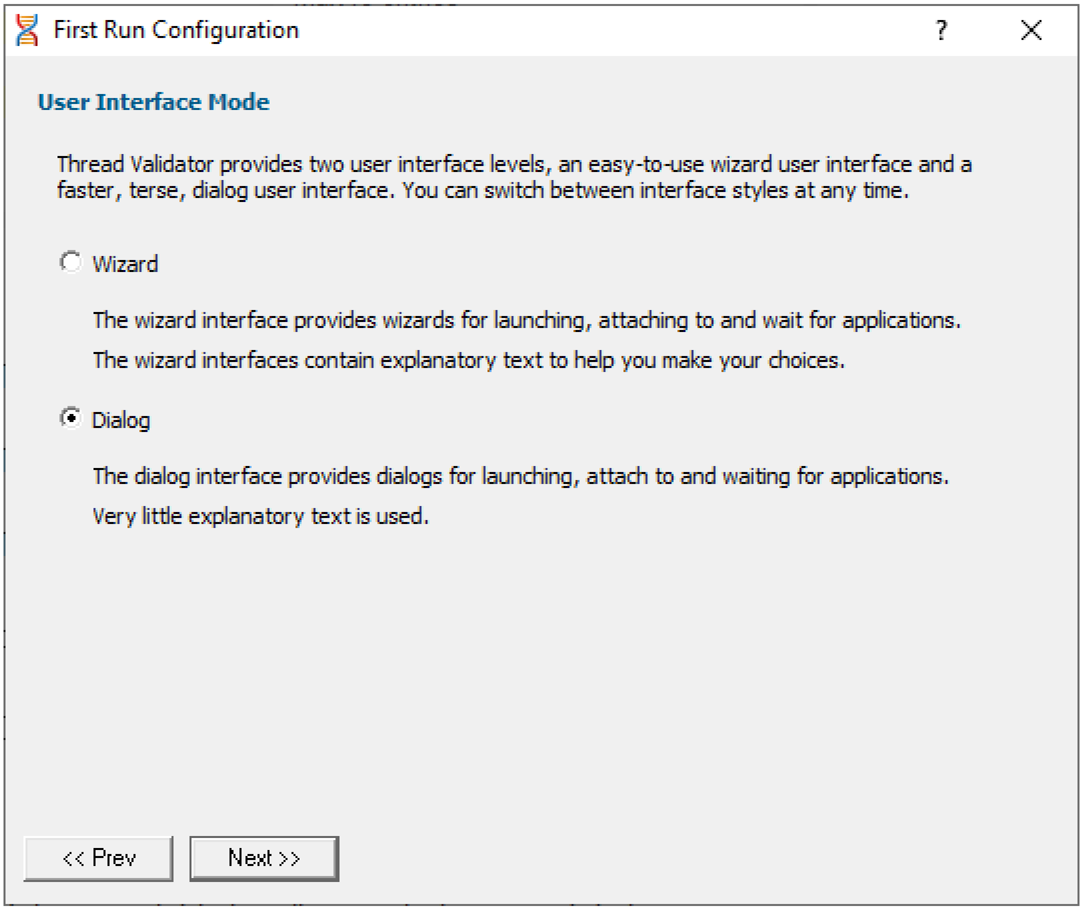
Let's set up Thread Validator

Before you start using Thread Validator we need to set up a few basic things.

- Environment variables that control the symbol search path for Visual Studio
- Which compiler/linker/IDE you are using
- Automatic software update information
- Short tutorial demonstrating how to use Thread Validator

User interface mode

After the introductory page, the wizard presents options for configuring the how the launch, inject and wait dialogs present information to you.



First Run Configuration

User Interface Mode

Thread Validator provides two user interface levels, an easy-to-use wizard user interface and a faster, terse, dialog user interface. You can switch between interface styles at any time.

○ Wizard

The wizard interface provides wizards for launching, attaching to and wait for applications.

The wizard interfaces contain explanatory text to help you make your choices.

● Dialog

The dialog interface provides dialogs for launching, attach to and waiting for applications.

Very little explanatory text is used.

- **Wizard mode** › guides you through the tasks in a linear fashion
- **Dialog mode** › all options are contained in a single dialog

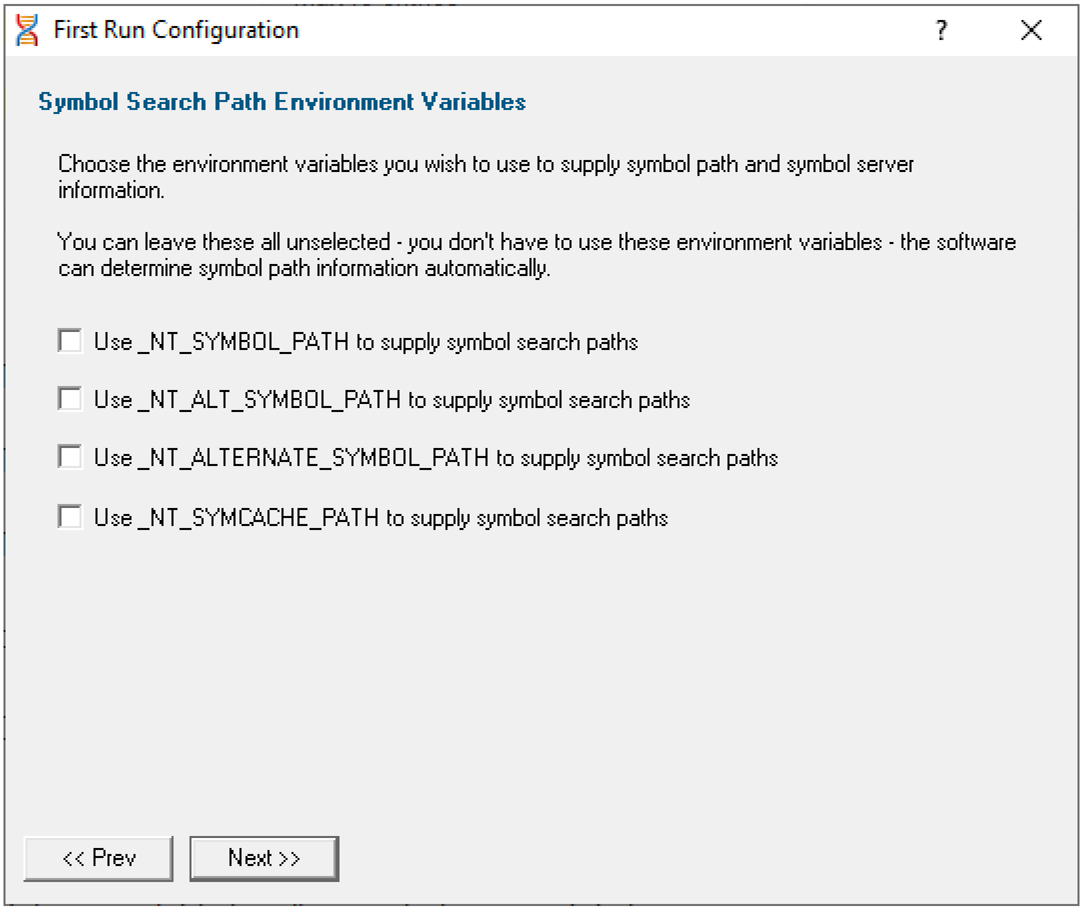
  Experienced users will find this mode quicker to use

These settings can be changed at any time via the User Interface Mode option on the Settings menu.

Symbol search path environment variables

After the introductory page, the wizard presents options for using environment variables for symbol search paths when finding PDB symbols.

You don't *have* to choose any of these options as Thread Validator will try to automatically determine symbol path information.



Symbol Search Path Environment Variables

Choose the environment variables you wish to use to supply symbol path and symbol server information.

You can leave these all unselected - you don't have to use these environment variables - the software can determine symbol path information automatically.

☐ Use _NT_SYMBOL_PATH to supply symbol search paths

☐ Use _NT_ALT_SYMBOL_PATH to supply symbol search paths

☐ Use _NT_ALTERNATE_SYMBOL_PATH to supply symbol search paths

☐ Use _NT_SYMCACHE_PATH to supply symbol search paths

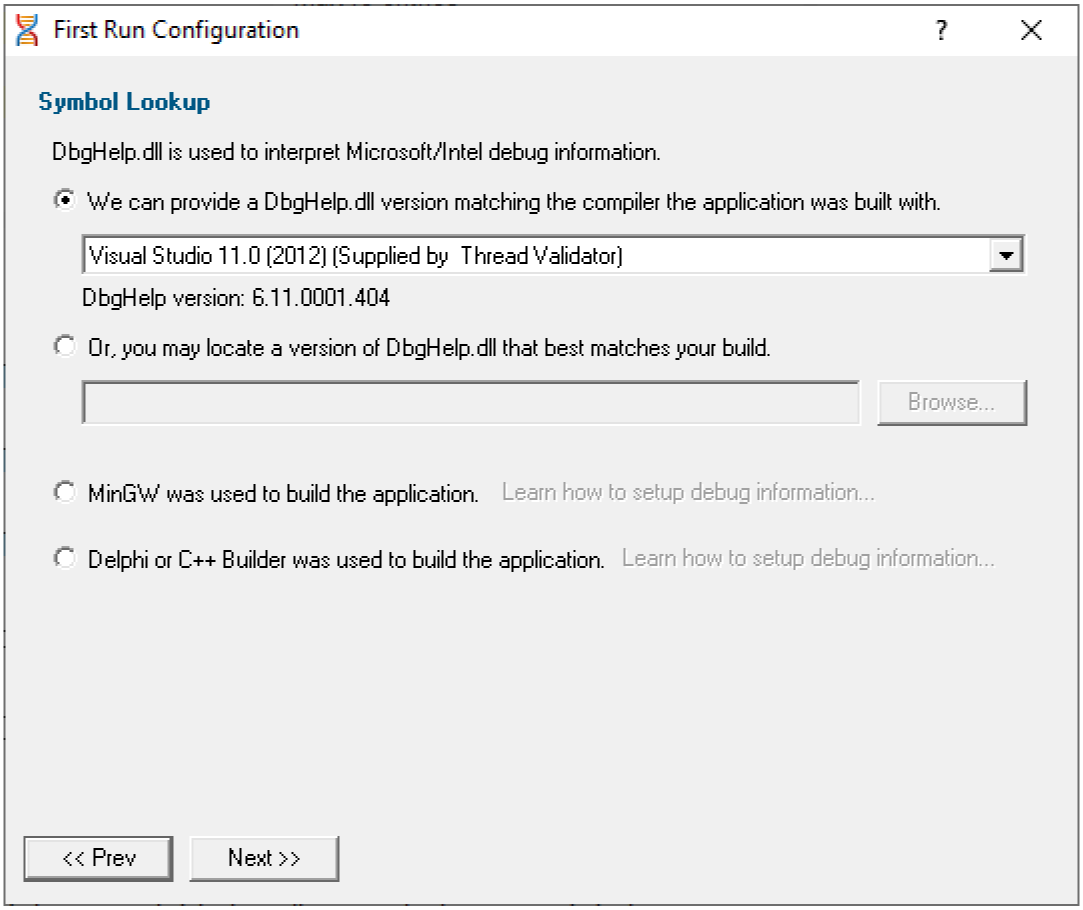
These environment settings can be changed at any time via the Configure Symbol Handling Environment Variables on the Symbol Server page of the Settings Dialog.

Symbol lookup

The next page of the wizard allows you to specify which IDE, Compiler or Linker you're using.

This is important as it affects how symbol lookup is performed. Visual Studio has various quirks in its history of symbol handling and we have to work around that.

The default settings are shown below, although the Visual Studio version may vary.



Symbol Lookup

DbgHelp.dll is used to interpret Microsoft/Intel debug information.

● We can provide a DbgHelp.dll version matching the compiler the application was built with.

Visual Studio 11.0 (2012) (Supplied by Thread Validator)

DbgHelp version: 6.11.0001.404

○ Or, you may locate a version of DbgHelp.dll that best matches your build.

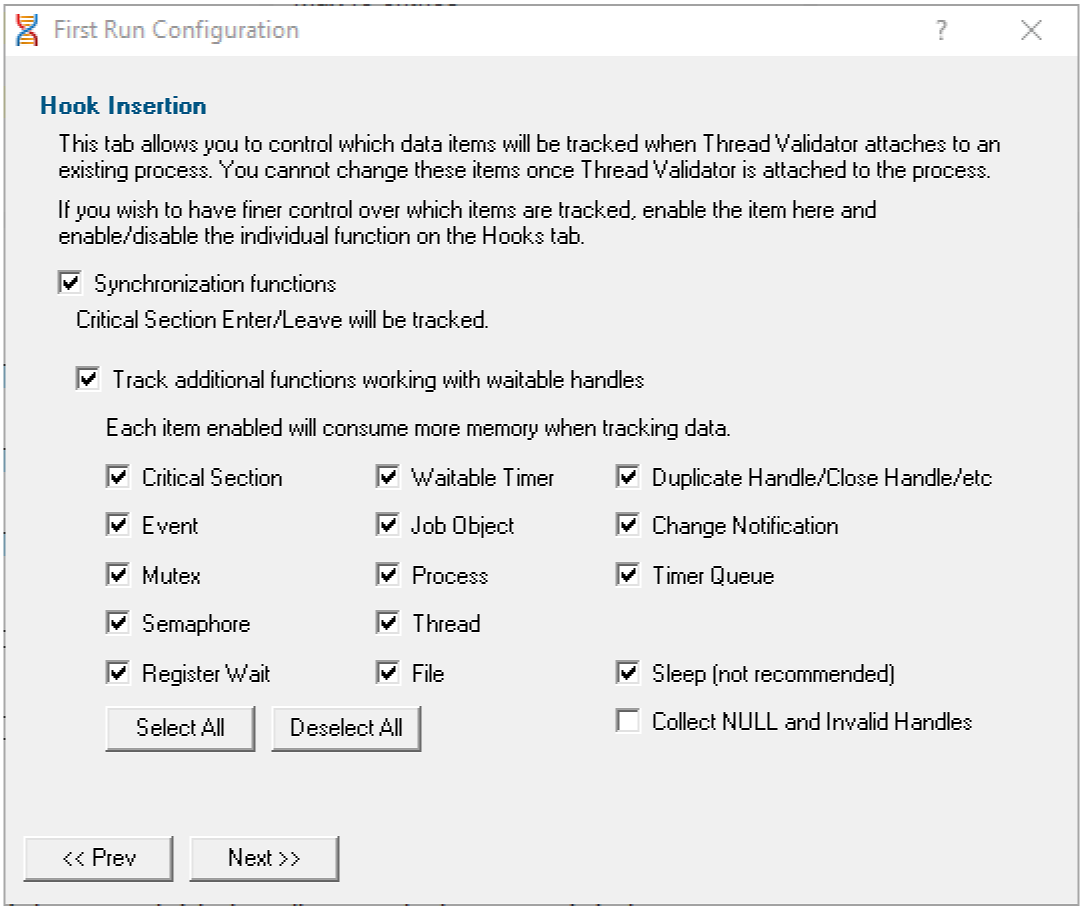
○ MinGW was used to build the application. Learn how to setup debug information...

○ Delphi or C++ Builder was used to build the application. Learn how to setup debug information...

These lookup settings can be changed at any time on the Symbol Lookup page of the Settings Dialog.

Hook Insertion

The next page of the wizard allows you to specify which synchronization functions will be monitored.



Hook Insertion

This tab allows you to control which data items will be tracked when Thread Validator attaches to an existing process. You cannot change these items once Thread Validator is attached to the process.

If you wish to have finer control over which items are tracked, enable the item here and enable/disable the individual function on the Hooks tab.

☑ Synchronization functions

Critical Section Enter/Leave will be tracked.

☑ Track additional functions working with waitable handles

Each item enabled will consume more memory when tracking data.

☑ Critical Section ☑ Waitable Timer ☑ Duplicate Handle/Close Handle/etc

☑ Event ☑ Job Object ☑ Change Notification

☑ Mutex ☑ Process ☑ Timer Queue

☑ Semaphore ☑ Thread

☑ Register Wait ☑ File ☑ Sleep (not recommended)

□ Collect NULL and Invalid Handles

Two check boxes control which groups of hooks are inserted into your application.

- **Synchronization functions ›** inserts hooks for tracking synchronization objects into the target program

  If enabled, the default behaviour is that critical section enter and leave events will be tracked. Additional functions can then be specified below.

- **Functions working with waitable handles ›** hooks (below) for tracking functions that create, manage and destroy waitable handles will be inserted into the target program

  A few additional functions (Sleep, and critical section related functions) are also monitored.

  Specify which groups of functions should have data collected from them using the individual check boxes. (See note about memory below).

  Any collected data can be viewed on the Active Objects and Analysis tabs.

Groups of items	Functions that...
• **Critical Section**	initialize, modify and destroy
• **Event**	create, open and destroy
• **Mutex**	create, open and destroy

Semaphore	create, open and destroy
Register Wait	register and unregister
Waitable Timer	create, open and destroy
Job Object	create, open and destroy
Process	create, open and destroy
Thread	create, open and destroy
File	create, open and destroy files (of any type)
Duplicate Handle	miscellaneous handle functions
Change Notification	create, open and destroy file change notifications
Timer Queue	create, open and destroy timer queues and timer queue timers

Other options:

Sleep	`Sleep` and `SleepEx` For many applications, enabling `Sleep` will generate a lot of data, as various threads sleep from time to time.
Collect NULL and Invalid Handles	information about failed handle creation When collecting waitable handles, you may also want to collect information about failed handle creation using this option.

- **Select/Deselect All** > switches all the above functions allocating waitable handles - i.e. not `Sleep` and invalid handles

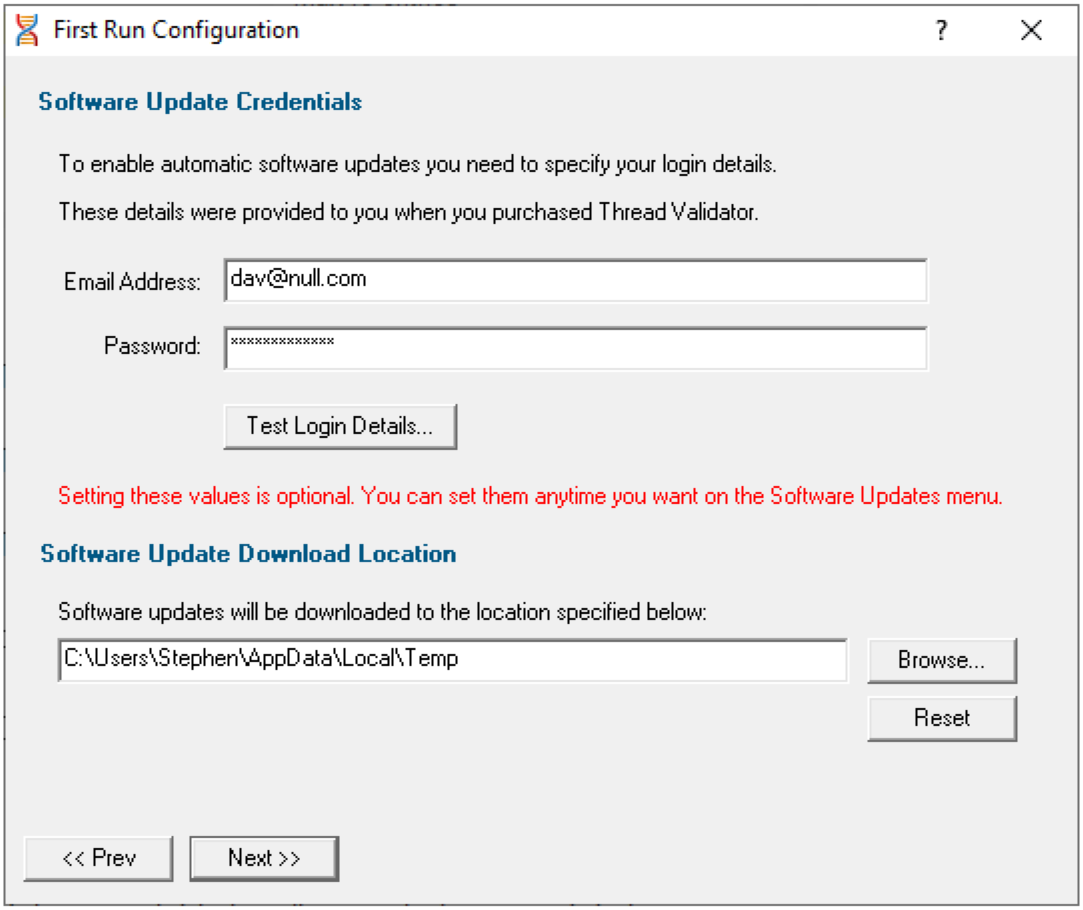
It's recommended that options in this section are only enabled if you need to monitor the data. If you subsequently find that memory consumption is very high, consider changing the settings here, or on the Historical Data tab.

These hook insertion settings can be changed at any time on the Hook Insertion page of the Settings Dialog.

Software update credentials

The software updates page of the wizard is only shown to non-evaluation users.

You can configure your software update credentials within the application and where updates are downloaded to.



First Run Configuration

Software Update Credentials

To enable automatic software updates you need to specify your login details.

These details were provided to you when you purchased Thread Validator.

Email Address: dav@null.com

Password: *************

Setting these values is optional. You can set them anytime you want on the Software Updates menu.

Software Update Download Location

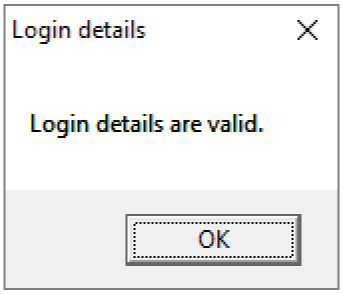
Software updates will be downloaded to the location specified below:

C:\Users\Stephen\AppData\Local\Temp

You can test the login details to ensure they are valid:

- **Test login details ›** click to check your entered details are valid (requires an internet connection)

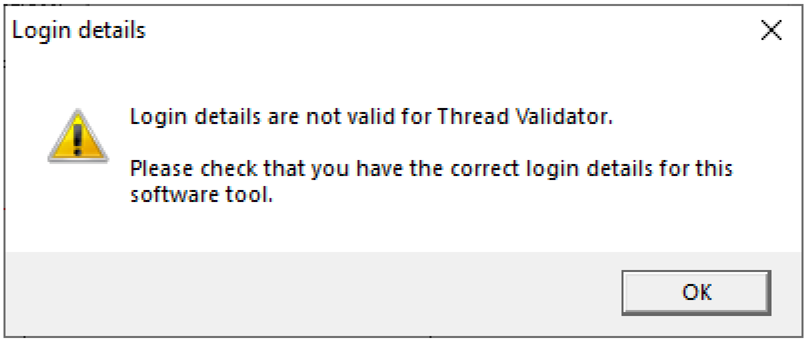
  Valid details will be confirmed:



Login details

Login details are valid.

Invalid details may mean you entered credentials for another application in the Validator suite, or they could have been entered incorrectly.


Login details

Login details are not valid for Thread Validator.

Please check that you have the correct login details for this software tool.

You should have received the correct credentials when you purchased Thread Validator.

If you experience problems, check with your system administrator or contact Software Verify.

These update credentials can be changed at any time from the Software Updates menu.

Software update download location

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.

We use the `TMP` directory as a default, but if 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.

An invalid directory, e.g. one that does not exist, will show text in red and will not be accepted until a valid folder is entered.

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

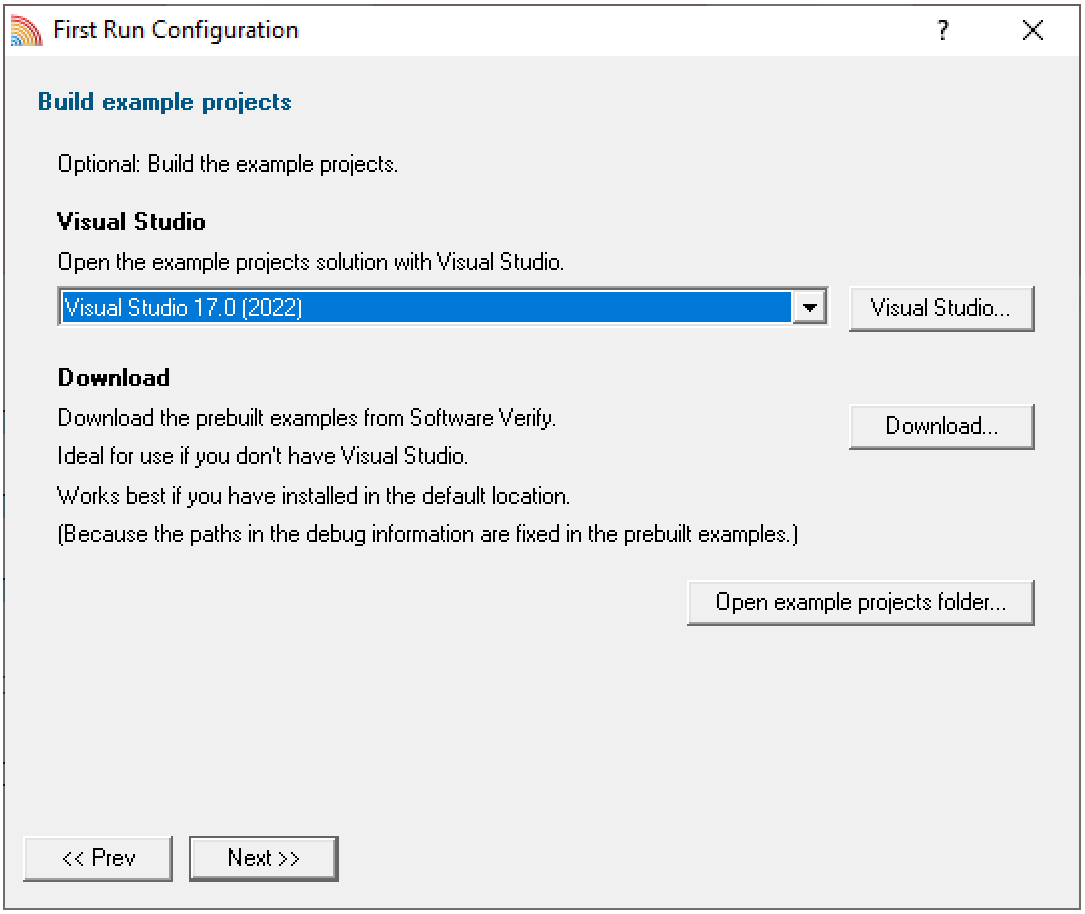
  The default location is `c:\users\[username]\AppData\Local\Temp`

The download location can be changed at any time from the Software Updates menu.

Build example projects

The next page of the wizard allows you to build the example projects that ship with Thread Validator.

The example projects demonstrate various application types containing bug you may wish to investigate with Thread Validator.



First Run Configuration

Build example projects

Optional: Build the example projects.

Visual Studio

Open the example projects solution with Visual Studio.

Visual Studio 17.0 (2022)

Download

Download the prebuilt examples from Software Verify.
Ideal for use if you don't have Visual Studio.
Works best if you have installed in the default location.
(Because the paths in the debug information are fixed in the prebuilt examples.)

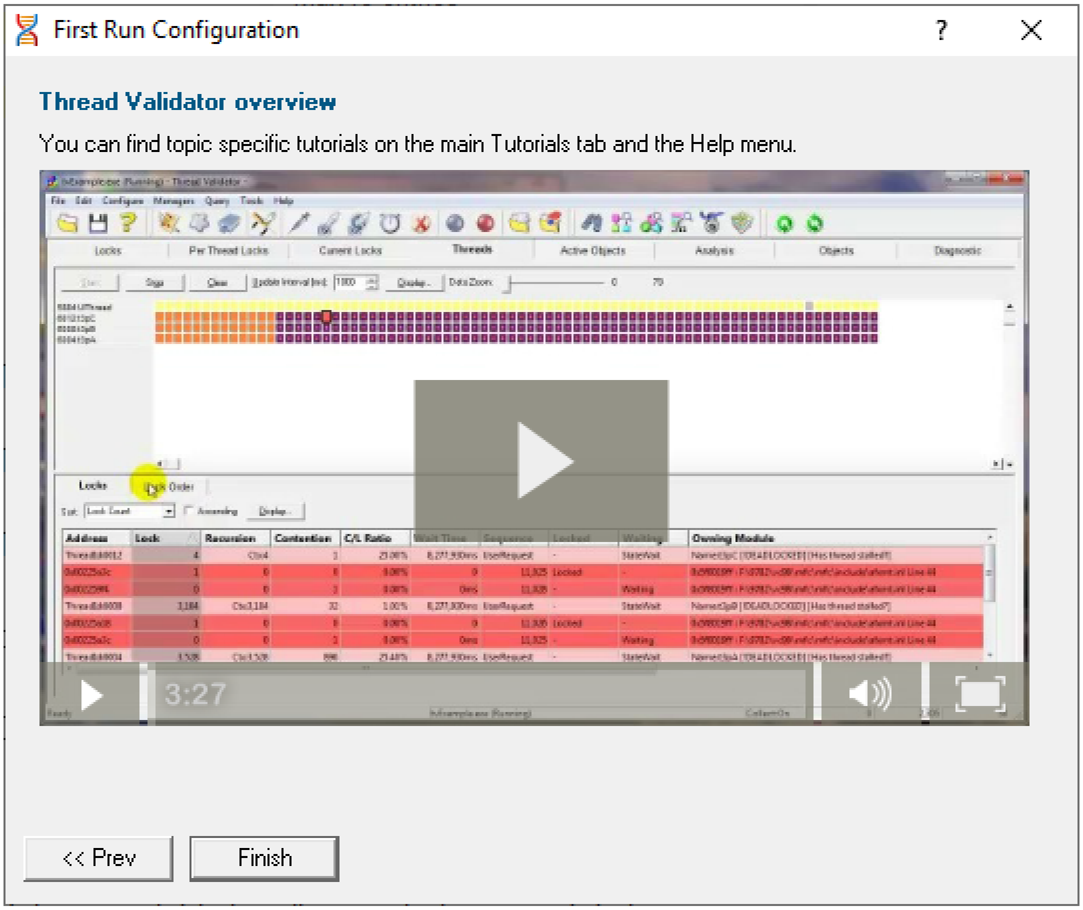
- **Visual Studio...** › opens the example projects solution with the version of Visual Studio selected
- **Download...** › downloads a prebuilt version of the example projects, unzips them and installs them in the examples folder in the Thread Validator installation directory

  If you choose this option and you have not installed Thread Validator in the default location (assuming a 64 bit OS) the source file paths in the debug information will be incorrect - you will need to use the File Locations settings to inform Thread Validator of the correct location(s).
- **Open example projects folder...** › opens the folder that contains all example projects

Video overview of application

The final page of the wizard presents a short video overview of Thread Validator.

The video has audio



First Run Configuration

Thread Validator overview

You can find topic specific tutorials on the main Tutorials tab and the Help menu.

More help is available via the Tutorials tab and the Help menu.

The video is also available on the product website. Visit https://www.softwareverify.com/products.php and find the product link for Thread Validator.

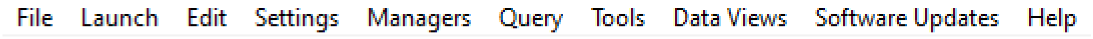
- **Finish** › closes the First Run Configuration dialog leaving the application ready to use

3.2 Menu Reference

The menus provide access to all the major features in Thread Validator. The most common ones are also directly accessible via the toolbars.

The next few pages provide a convenient collection of links to the detailed help pages on each menu item.

Alternatively, click in the picture below to jump to the page for any menu:



3.2.1 File menu

Files and sessions

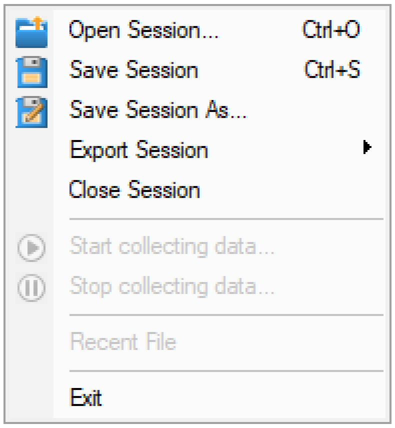
The **File** menu allows you to:

- open, close and save sessions
- manage the launching of an application
- control the collection of data
- exit the application

Most of these actions (those with an icon to the left) are also available via the standard or session toolbars.

Near the bottom of the menu, a list of recently used filenames allows you to easily reload a previously saved session.

Click on an item in the picture below to find out more:



The last two items are not linked to topics. **Exit** is self explanatory and above that is a list of recently opened files that will vary for each user.

3.2.2 Launch menu

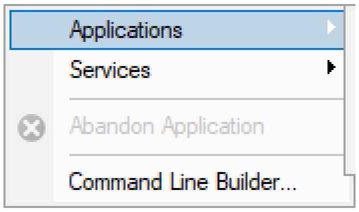
The **Launch** menu allows you to:

- start applications and restart applications
- inject into running applications
- wait for applications to start then attach to them
- monitor services and ISAPI extensions
- stop monitoring an application

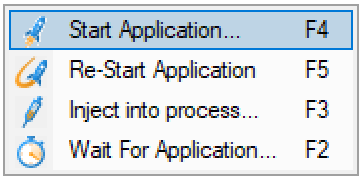
Most of these actions are also available via the standard or session toolbars.

These actions are grouped into submenus according to whether they involve applications or services.

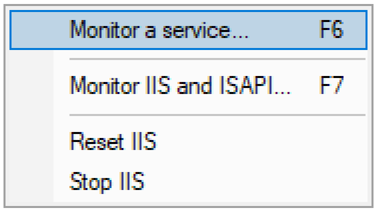
Click on an item in the pictures below to find out more:



Applications



Services



In addition to the function key short cuts shown above, you can redisplay the previously chosen launch dialog by using Ctrl + F4

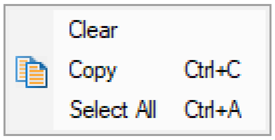
3.2.3 Edit menu

Using the clipboard and selections

The **Edit** menu options can be used to:

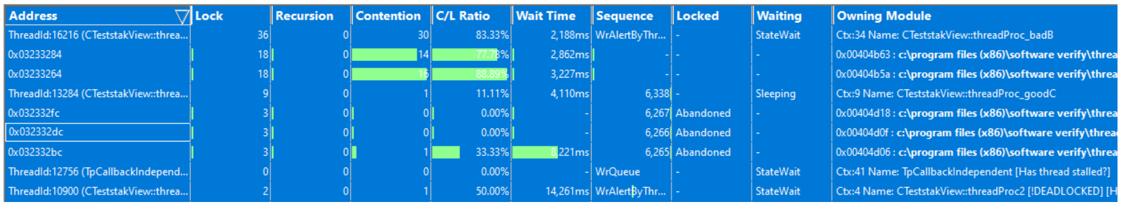
- clear all selected items in a table or tree
- copy selected items (and relevant data where applicable) to the clipboard
- select all the items available

Selected data is formatted into one line per row with a single tab-space used to separate column data (tsv format).



Select All will include the header row as well as the data, and **Copy** will include the column titles.

For example, after running the example application, **Select All** on the Locks Per Thread tab might show:



Address	Lock	Recursion	Contention	C/L Ratio	Wait Time	Sequence	Locked	Waiting	Owning Module
ThreadId:16216 (CTeststakView::threa...	36	0	30	83.33%	2,188ms	WrAlertByThr...	-	StateWait	Ctx:34 Name: CTeststakView::threadProc_badB
0x03233284	18	0	14	77.78%	2,862ms	-	-	-	0x00404b63 : **c:\program files (x86)\software verify\threa**
0x03233264	18	0	16	88.89%	3,227ms	-	-	-	0x00404b5a : **c:\program files (x86)\software verify\threa**
ThreadId:13284 (CTeststakView::threa...	9	0	1	11.11%	4,110ms	6,338	-	Sleeping	Ctx:9 Name: CTeststakView::threadProc_goodC
0x032332fc	3	0	0	0.00%	-	6,267	Abandoned	-	0x00404d18 : **c:\program files (x86)\software verify\threa**
0x032332dc	3	0	0	0.00%	-	6,266	Abandoned	-	0x00404d0f : **c:\program files (x86)\software verify\threa**
0x032332bc	3	0	1	33.33%	8,221ms	6,265	Abandoned	-	0x00404d06 : **c:\program files (x86)\software verify\threa**
ThreadId:12756 (TpCallbackIndepend...	0	0	0	0.00%	-	WrQueue	-	StateWait	Ctx:41 Name: TpCallbackIndependent [Has thread stalled?]
ThreadId:10900 (CTeststakView::threa...	2	0	1	50.00%	14,261ms	WrAlertByThr...	-	StateWait	Ctx:4 Name: CTeststakView::threadProc2 [!DEADLOCKED] [H

This would result in the following tab separated content being copied to clipboard:

```
Address       Lock   Recursion     Contention    C/L Ratio     Wait Time     Sequence      Lo
ThreadId:16216 (CTeststakView::threadProc_badB) 36     0      30     83.33% 2,188ms       Wr
0x03233284    18     0      14     77.78% 2,862ms       -      -      -      0x00404b63 : c:\
0x03233264    18     0      16     88.89% 3,227ms       -      -      -      0x00404b5a : c:\
ThreadId:13284 (CTeststakView::threadProc_goodC)        9      0      1      11.11% 4,110ms
0x032332fc    3      0      0      0.00%  -      6,267  Abandoned     -      0x00404d18 : c:\
0x032332dc    3      0      0      0.00%  -      6,266  Abandoned     -      0x00404d0f : c:\
0x032332bc    3      0      1      33.33% 8,221ms       6,265  Abandoned     -      0x00404d0
ThreadId:12756 (TpCallbackIndependent)    0      0      0      0.00%  -      WrQueue       -
ThreadId:10900 (CTeststakView::threadProc2)      2      0      1      50.00% 14,261ms      Wr
```

3.2.4 Settings menu

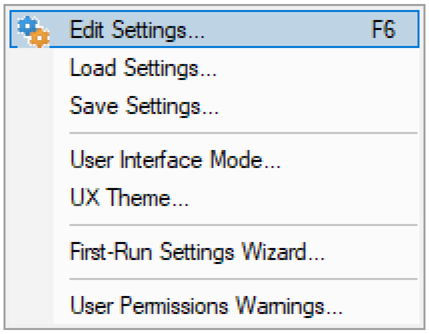
Settings and UI mode

The **Settings** menu allows you to:

- choose the user interface mode (wizards or dialogs)
- change settings for global data and how it is displayed

Global settings are also accessible via the session toolbar.

Click on an item in the menu below to find out more:



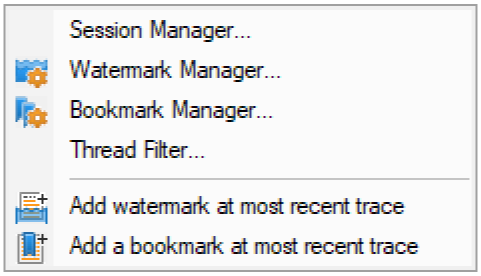
3.2.5 Managers menu

Managers and markers

The **Managers** menu provides tools for

- managing sessions including limiting the number that can be open
- managing watermarks and bookmarks
- exclude thread data using thread filters

Click on the menu item in the picture to find out more:



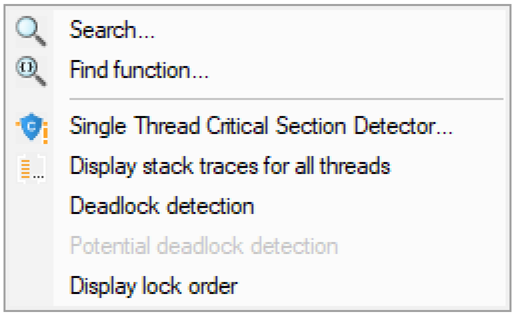
3.2.6 Query menu

Query and Search

The query and search tools enable you to find particular thread data collected in each session.

Some of these options are also available from the Query toolbar.

Click on an item in the picture below to find out more:


Search...
Find function...
Single Thread Critical Section Detector...
Display stack traces for all threads
Deadlock detection
Potential deadlock detection
Display lock order

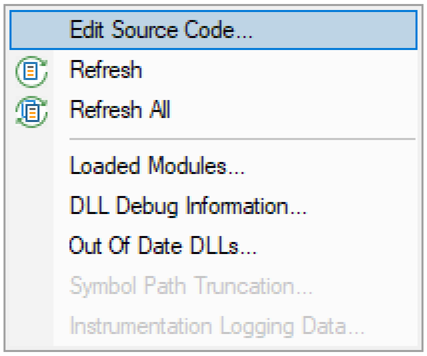
3.2.7 Tools menu

Tools and Information

The Tools menu provides access to a few different tools including a couple not found on the Tools toolbar:

- A list of the modules loaded by your target application
- A list of the debug information status of modules loaded by your application

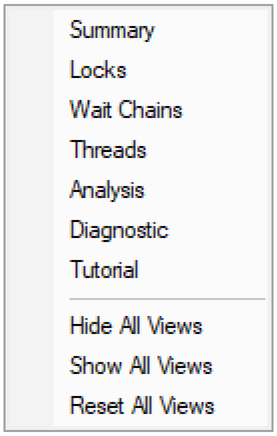
Click on an item in the picture below to find out more:



3.2.8 Data Views menu

Data Views

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

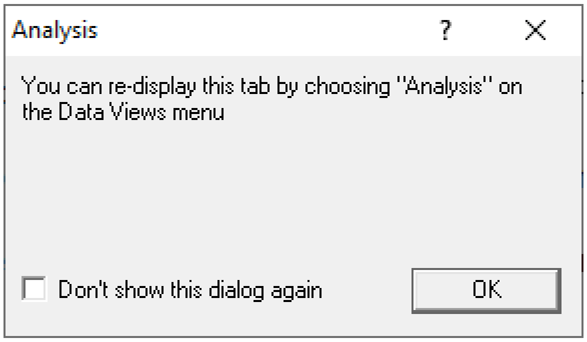


Selecting any of the items shows the relevant tab (if it's not visible already), and makes it the current selected tab.

- **Hide All Views ›** hides all tabs *except* the one that's currently visible
- **Show All Views ›** shows *all* the listed tabs, and in their normal order
- **Reset All Views ›** shows only the most *popular* tabs, so excludes the Analysis, and the Line Times tabs

  This is the default setting when you first use the software

When you hide a tab (by clicking the cross on the right of the tab header), you'll initially be reminded of where to go to show it again, but you can choose not to keep seeing this reminder.



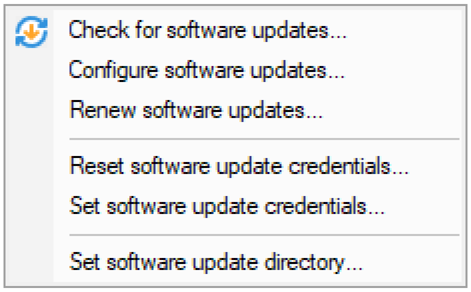
You can re-display this tab by choosing "Analysis" on the Data Views menu

Hidden views are remembered between sessions.

3.2.9 Software Updates menu

Software updates

All three items in this menu are covered in the Software Updates topic.

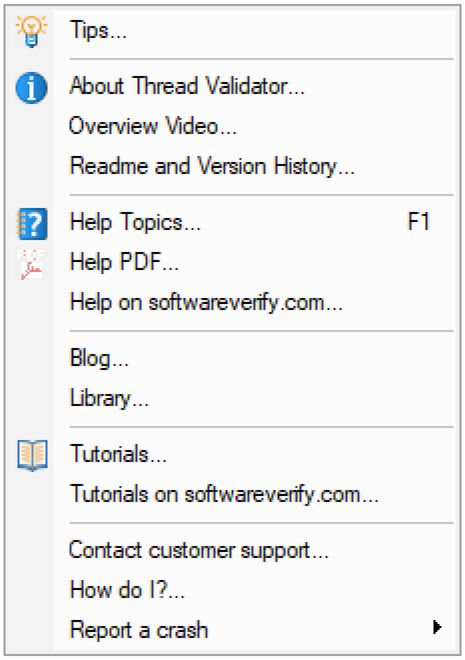

Check for software updates...
Configure software updates...
Renew software updates...
Reset software update credentials...
Set software update credentials...
Set software update directory...

3.2.10 Help menu

Help

The help menu provides access to this manual in different formats as well as tips and tutorials.

Click on an item in the picture below to find out more about each item in the Help topic:

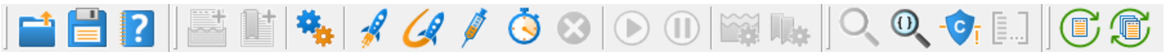

Tips...
About Thread Validator...
Overview Video...
Readme and Version History...
Help Topics... F1
Help PDF...
Help on softwareverify.com...
Blog...
Library...
Tutorials...
Tutorials on softwareverify.com...
Contact customer support...
How do I?...
Report a crash

➡ Check out the Frequently Asked Questions as well!

3.3 Toolbar Reference

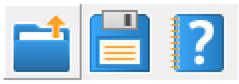
This reference section lists the various toolbars in Thread Validator, with quick links to their own section of the help manual.

The items are listed in left to right order.



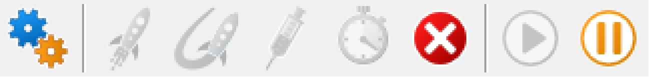
You can click on any part of the pictures below to jump straight to the topic:

Standard toolbar



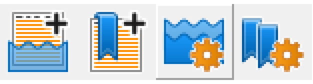
- Load session
- Save session
- Help

Session toolbar



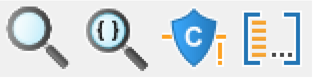
- Settings
- Inject into application
- Launch application
- Relaunch application
- Wait for application to start
- Stop application
- Enable collecting data
- Disable collecting data

Watermarks



- Watermark manager
- Bookmark manager
- Add watermark at most recent trace
- Add bookmark at most recent trace

Query



- Search synchronization objects
- Find function
- Single thread critical section detector
- Display stack traces for all threads in application

Tools

- Refresh view
- Refresh all views

3.4 The status bar

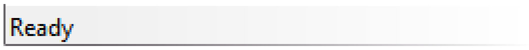
Elements of the status bar

The status bar has three main sections, from left to right:

- the message line
- program information
- data collection statistics

The message line

Most of the time, you'll just see this:



Ready

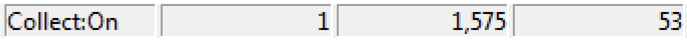
When you hover your mouse over a toolbar button or a menu item for a short time, a help message appears in the status bar describing the button's action.

Data collection statistics

The data statistics counts give a guideline indicator of how data is being collected by the stub and sent to Thread Validator.

This collection data has a few counters and a collection status:

- Status indicating whether collection is currently on or off
- Data items sent from stub that are waiting to be processed
- Number of data items processed
- Number of data items not yet resolved



Collect:On	1	1,575	53

The boxes stay gray when the values are static, but will be coloured for a few seconds when the value changes:

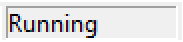
🟥 The value increased

🟪 The value decreased

Profiling status

The status of the flow trace indicates what is currently happening.

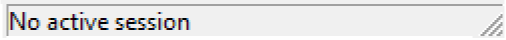
- **Ready**. Waiting to start a run, or a run has finished and is waiting for you to analyze the data.
- **Starting**. Starting a run (hooks being installed etc).
- **Running**. Target executable is hooked and running.
- **Terminating**. Target executable has entered ExitProcess but has not yet finished executing.
- **Post Processing**. Target executable has finished executing. There is data that still needs to be processed.



Running

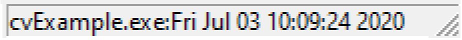
Target program name

This displays **No active session** when there is no session running, terminating or loaded.



No active session

When a session is running, terminated or loaded, this displays the name of the target program followed by a timestamp.



cvExample.exe:Fri Jul 03 10:09:24 2020

3.5 Keyboard Shortcuts

Keyboard shortcuts

The following shortcuts are available:

Ctrl + A Select All

Ctrl + C Copy

Ctrl + O Open session

Ctrl + S Save session

F1 Help (contextual for current view or dialog)

F2 Wait for application

F3 Inject into process

F4 Start application

F5 Restart application

F6 Monitor a Service

F7 Monitor IIS and ISAPI

3.6 Icons

Some of the displays include an icon on the left border of the scrolled list/tree to indicate the type of data that is present on that line. The icons are shown below, with an explanation.

Icons used in settings, views and editors

- ☑ Option enabled
- □ Option disabled
- ⇨ Source code line indicator
- 📄 Source code

Icons used in the data displays

Some of the displays include an icon on the left border of the scrolled list/tree to indicate the type of data that is present on that line.

- 💬 Watermark placed by the user
- H Critical section allocation
- H Critical section deallocation
- T Trace message, or `OutputDebugString()` message
- E Trace from an exception
- A Trace from an `ASSERT`

3.7 The main display

The tab windows

The main display of Thread Validator consists of tabbed windows. Not all the tabs may be visible - see the Data Views menu to show any hidden tabs.

Each tab allows the data collected to be viewed, inspected and queried in different and complementary ways.

Typical usage might be to use the Summary, or various Locks tabs to monitor the thread activity in the target program, and then use another view to gain insights at a different granularity or about related functions and callstacks.

Click on an item in the picture below to find out more about each of the tabbed windows, or use the list further below:

- Summary
- Locks
- Wait Chains
- Threads
- Analysis
- Diagnostic
- Tutorial

Hiding and showing tabs

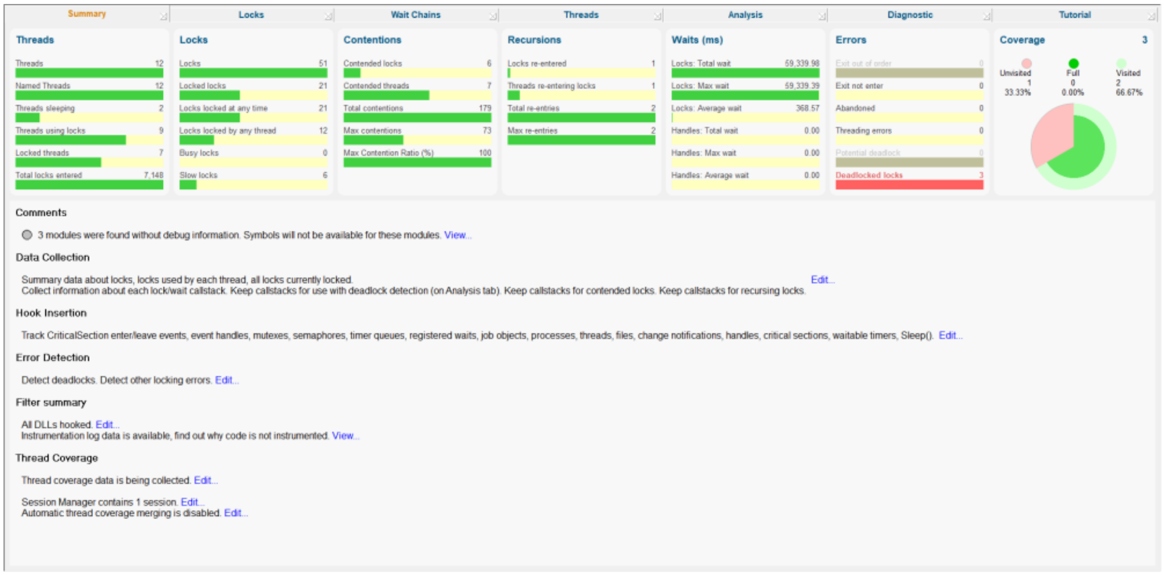
Each tabbed window can be closed by clicking the small [x] on the right hand side of the tab. The window can be redisplayed from the Data Views menu.

Icons

Most windows use a small number of icons to indicate different types of data.

3.7.1 Summary

The **Summary** tab displays a dashboard showing high level information about the threading behaviour of the current application.



Threads
Threads 12
Named Threads 12
Threads sleeping 2
Threads using locks 9
Locked threads 7
Total locks entered 7,148

Locks
Locks 51
Locked locks 21
Locks locked at any time 21
Locks locked by any thread 12
Busy locks 0
Slow locks 6

Contentions
Contended locks 6
Contended threads 7
Total contentions 179
Max contentions 73
Max Contention Ratio (%) 100

Recursions
Locks re-entered 1
Threads re-entering locks 1
Total re-entries 2
Max re-entries 2

Waits (ms)
Locks: Total wait 59,339.98
Locks: Max wait 59,339.39
Locks: Average wait 368.57
Handles: Total wait 0.00
Handles: Max wait 0.00
Handles: Average wait 0.00

Errors
Exit out of order 0
Exit not enter 0
Abandoned 0
Threading errors 0
Potential deadlock 0
Deadlocked locks 3

Coverage 3
Unvisited 1 33.33%
Full 0 0.00%
Visited 2 66.67%

Comments

○ 3 modules were found without debug information. Symbols will not be available for these modules. View...

Data Collection

Summary data about locks, locks used by each thread, all locks currently locked. Edit...
Collect information about each lock/wait callstack. Keep callstacks for use with deadlock detection (on Analysis tab). Keep callstacks for contended locks. Keep callstacks for recursing locks.

Hook Insertion

Track CriticalSection enter/leave events, event handles, mutexes, semaphores, timer queues, registered waits, job objects, processes, threads, files, change notifications, handles, critical sections, waitable timers, Sleep(). Edit...

Error Detection

Detect deadlocks. Detect other locking errors. Edit...

Filter summary

All DLLs hooked. Edit...
Instrumentation log data is available, find out why code is not instrumented. View...

Thread Coverage

Thread coverage data is being collected. Edit...

Session Manager contains 1 session. Edit...
Automatic thread coverage merging is disabled. Edit...

The Summary tab

The summary shows a visual overview with statistics of threads and locks with percentages where appropriate.

Each bar or dial summarises the thread, lock, wait or error information found in the main tabs.

- Threads
- Locks
- Contentions
- Recursions
- Waits
- Errors
- Coverage

Clicking on a bar or dial takes you to the corresponding main tab to explore in more detail.

Most of the bars take you to the All Locks summary tab, with each one sorting on a specific column related to the data at hand.

Examples:

- Clicking on the *Contended locks* bar shows the All Locks tab sorted by Contention Count
- The *Max Contention Ratio* bar shows the same tab sorted by Contention Ratio
- The Recursions *Locks re-entered* bar shows the All Locks tab sorted by Recursion Count
- The *Coverage* dial shows the Coverage tab

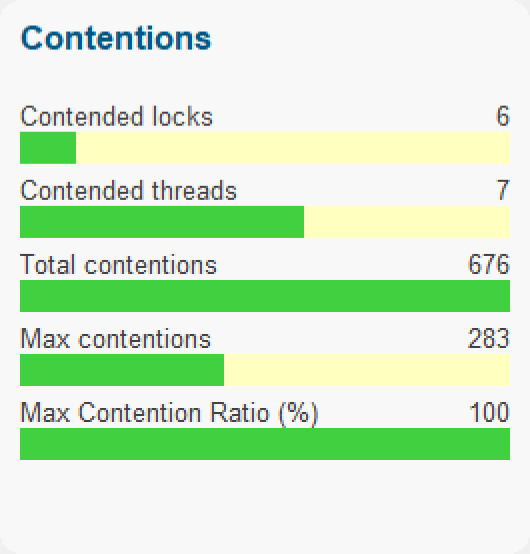
Understanding the bars

Each bar displays the name of the statistic underneath with its value.

Bars are filled to represent the fraction of the value it is related to.

In the example below, the bars show that:

- the number of contended locks is 6, shown as a fraction of the number of locks (in this case 51 which was displayed separately on the *Locks* panel)
- the number of contended threads is 7, shown as a fraction of the 12 threads being monitored in the application (displayed on the *Threads* panel)
- the total number of contentions is 676, of which one of the locks is responsible for 283 of those contentions
- the maximum contention ratio is 100%.



Contentions

Contended locks	6
Contended threads	7
Total contentions	676
Max contentions	283
Max Contention Ratio (%)	100

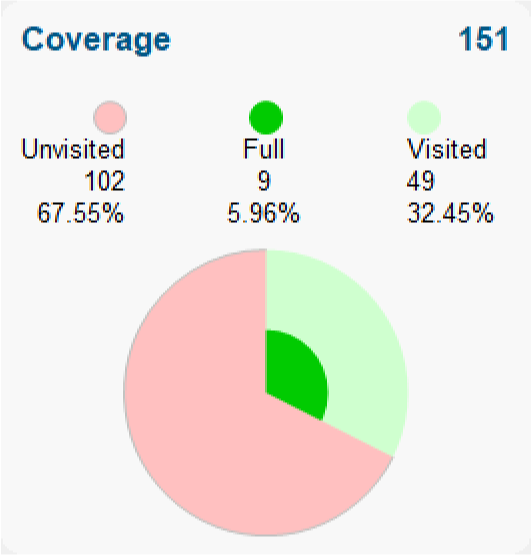
Understanding the dial

The Coverage dial displays:

- numeric statistics on visited and unvisited items, and those items with 100% coverage
- the unvisited/visited information as angular data
- the 100% coverage as inner radial data
- the partial coverage distribution as outer radial data

Example:

The following dial summarises thread coverage data on a total of 151 known files in a complex target program



The radius of the inner area may grow or shrink as the target program runs, since the proportion of visited functions that have 100% coverage can go up or down.

Status summary area

Below the dials is a status area showing any comments or special notices related to the current session.

Underneath the comments you'll find the status of any data collection, error detection, filters, or session merging.

Clicking **Edit...** or **View...** opens a dialog to edit or view the relevant settings.

In most cases the settings shown are identical to the relevant page of the global settings dialog.

Comments:

- Debug modules **View...** › Module debug information dialog

Data Collection:

- Locks and callstacks **Edit...** › Collect settings

Hook Insertion:

- Locks and handles **Edit...** › Hook Insertion Settings

Error Detection:

• Threading errors	**Edit...** > Detect settings

Filter Summary:

• DLLs Hooked	**Edit...** > Hooked DDLs settings
• Instrumentation logging	**Edit...** > Instrumentation Logging settings (If logging is off)
	View... > Instrumentation Log dialog (If logging is on)

Thread Coverage:

• Thread coverage collection	**Edit...** > Thread Coverage Settings
• Session manager status	**Edit...** > Session Chooser dialog
• Session merging status	**Edit...** > Auto Merge settings

3.7.2 Locks

The Locks tab contains three different sub tabs, each providing a different view onto the Critical Sections in use and the Waits that are taking place in the program.

All Locks

Displays all locks in use by the program. Also displays any waits in progress.

Locks Per Thread

Displays all locks in use by the program. Also displays any waits in progress. Data is grouped by thread.

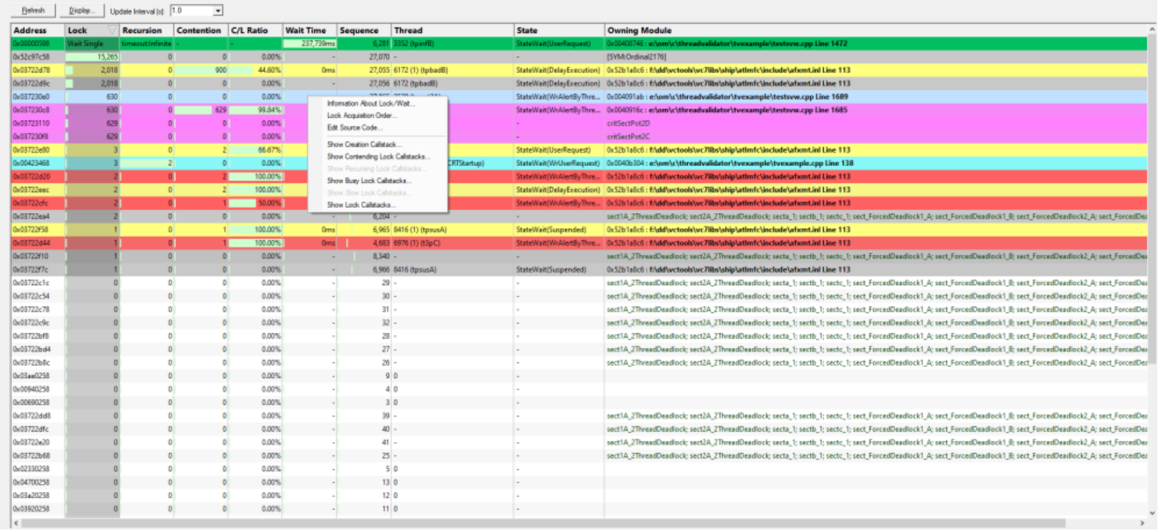
Active Locks

Displays all locks that are currently locked by the program. Also displays any waits in progress.

3.7.2.1 All Locks

The **All Locks** tab displays information about all critical sections, and `WaitForXXXObject` calls in the application being monitored.

Read on, or click a part of the image below to jump straight to the help for that area.


Refresh
Display...
Information About Lock/Wait...
Lock Acquisition Order...
Edit Source Code...
Show Creation Callstack...
Show Contending Lock Callstacks...
Show Recursing Lock Callstacks...
Show Busy Lock Callstacks...
Show Slow Lock Callstacks...
Show Lock Callstacks...

Update Interval (s): 1.0

Address	Lock	Recursion	Contention	C/L Ratio	Wait Time	Sequence	Thread	State	Owning Module
0x00000598	Wait Single	timeout:Infinite	-	-	237,739ms	6,281	3352 (tpinfB)	StateWait(UserRequest)	0x00408746 : e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1472
0x52c97c58	15,265	0	0	0.00%	-	27,070	-	-	[SYM:Ordinal2176]
0x03722d78	2,018	0	900	44.60%	0ms	27,055	6172 (1) (tpbadB)	StateWait(DelayExecution)	0x52b1a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x03722d9c	2,018	0	0	0.00%	-	27,056	6172 (tpbadB)	StateWait(DelayExecution)	0x52b1a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x037230e0	630	0	0	0.00%				StateWait(WrAlertByThre...	0x004091ab : e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1689
0x037230c8	630	0	629	99.84%				StateWait(WrAlertByThre...	0x0040916c : e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1685
0x03723110	629	0	0	0.00%				-	critSectPot2D
0x037230f8	629	0	0	0.00%				-	critSectPot2C
0x03722e80	3	0	2	66.67%				StateWait(UserRequest)	0x52b1a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x00423468	3	2	0	0.00%			CRTStartup)	StateWait(WrUserRequest)	0x0040b304 : e:\om\c\threadvalidator\tvexample\tvexample.cpp Line 138
0x03722d20	2	0	2	100.00%				StateWait(WrAlertByThre...	0x52b1a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x03722eec	2	0	2	100.00%				StateWait(DelayExecution)	0x52b1a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x03722cfc	2	0	1	50.00%				StateWait(WrAlertByThre...	0x52b1a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x03722ea4	2	0	0	0.00%	-	6,204	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722f58	1	0	1	100.00%	0ms	6,965	8416 (1) (tpsusA)	StateWait(Suspended)	0x52b1a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x03722d44	1	0	1	100.00%	0ms	4,683	6976 (1) (t3pC)	StateWait(WrAlertByThre...	0x52b1a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x03722f10	1	0	0	0.00%	-	8,340	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722f7c	1	0	0	0.00%	-	6,966	8416 (tpsusA)	StateWait(Suspended)	0x52b1a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x03722c1c	0	0	0	0.00%	-	29	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722c54	0	0	0	0.00%	-	30	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722c78	0	0	0	0.00%	-	31	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722c9c	0	0	0	0.00%	-	32	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722bf8	0	0	0	0.00%	-	28	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722bd4	0	0	0	0.00%	-	27	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722b8c	0	0	0	0.00%	-	26	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03ae0258	0	0	0	0.00%	-	9	0	-	
0x00940258	0	0	0	0.00%	-	4	0	-	
0x00690258	0	0	0	0.00%	-	3	0	-	
0x03722dd8	0	0	0	0.00%	-	39	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722dfc	0	0	0	0.00%	-	40	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722e20	0	0	0	0.00%	-	41	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x03722b68	0	0	0	0.00%	-	25	-	-	sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1; sectc_1; sect_ForcedDeadlock1_A; sect_ForcedDeadlock1_B; sect_ForcedDeadlock2_A; sect_ForcedDea
0x02330258	0	0	0	0.00%	-	5	0	-	
0x04700258	0	0	0	0.00%	-	13	0	-	
0x03a20258	0	0	0	0.00%	-	12	0	-	
0x03920258	0	0	0	0.00%	-	11	0	-	

The Locks View

The Locks view shows information about each critical section used by the target application.

Each critical section is shown once and the display can be sorted by different criteria by clicking on the appropriate column header.

The table highlights error and status conditions using the user defined lock colours. In the example above, red 🟥 indicates a deadlock, yellow 🟨 contention and cyan ▨ recursion.

Lines highlighted in grey denote an active status, meaning that the critical section has been entered, while the white lines show sections where nothing of interest has happened yet.

What data is available?

The following table lists all the available columns of data:

Attribute	Description
• **Address**	Address of the critical section, the handle for a `WaitForSingleObject` call, or the number of handles for a `WaitForMultipleObjects` call.
• **Lock**	Number of times the critical section has been locked by any thread.

- **Recursion** Number of times the critical section has been re-entered whilst locked by the current thread.

- **Contention** Number of attempts a thread has made to acquire the critical section but has had to wait because it is owned by another thread.

- **Wait Time** Total time spent waiting by all threads to gain access to this critical section.

- **Sequence** The sequence id is an internal monotonic integer that is incremented for each synchronization object event. This integer allows you to identify the order in which events occurred.

  Note that this can be misleading if the windows scheduler switches threads at the wrong time.

- **Thread** Thread id (in decimal notation) of the thread that currently owns the critical section.
  A number in brackets is the number of other threads waiting to gain access.

  If the thread has been given a name, it is displayed.

- **State** Current state of the thread, obtained by polling the operating system for data at regular intervals.
  This information may not match the other data at all times, but it's very useful for identifying thread conditions when thread states don't change for long periods, e.g. long wait states and deadlocks.

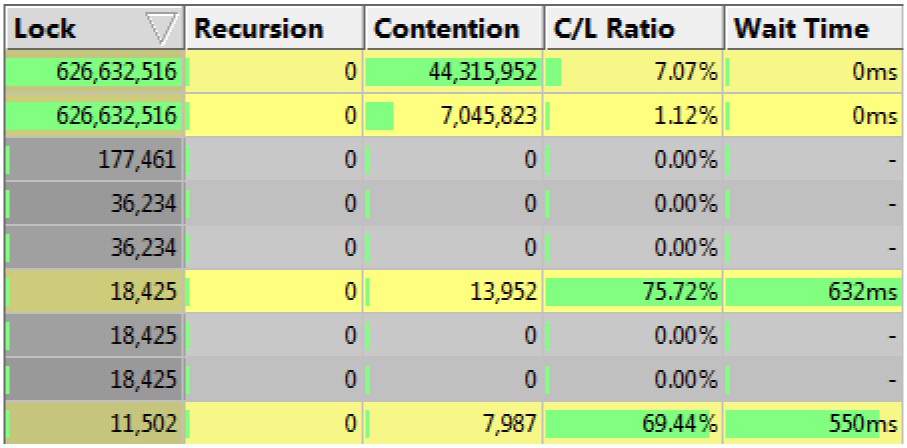
- **Owning Module** One or more of the following values if available, shown in order of priority with most important first:

  - Symbol name (within 256 bytes of the critical section address) for the critical section
  - Filename and line number of the code location that acquired the lock, determined from the callstack (if collected)
  - Filename and line number of the critical section
  - Name of the owning module, if the critical section is static data inside a module (DLL or EXE)
  - Data indicating dynamic memory or stack space

  Some of these details can be optionally hidden using the Display options.

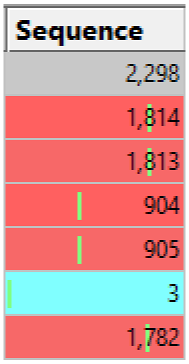
Percentage bars and markers

All the cells in the table that show timing or count data have a percentage bar indicating the item's value relative to the maximum value in the column.



Lock	Recursion	Contention	C/L Ratio	Wait Time
626,632,516	0	44,315,952	7.07%	0ms
626,632,516	0	7,045,823	1.12%	0ms
177,461	0	0	0.00%	-
36,234	0	0	0.00%	-
36,234	0	0	0.00%	-
18,425	0	13,952	75.72%	632ms
18,425	0	0	0.00%	-
18,425	0	0	0.00%	-
11,502	0	7,987	69.44%	550ms

Not all bars are percentages. In the Sequence column, the markers indicate the relative position of the sequence id (see table above) with respect to the first sequence id and most recent sequence id.



Sequence
2,298
1,814
1,813
904
905
3
1,782

Updating the display

- **Update Interval ›** automatically updates the display at your choice of interval, from 0.1 to 60 seconds

  This should be set depending on the complexity of your application.

An update interval that is too short may mean Thread Validator spends too much time updating the display.

- **Refresh ›** updates the display - as does the button on the Tools menu and toolbar

Sorting the data

You can sort the data in the table by clicking in the table header.

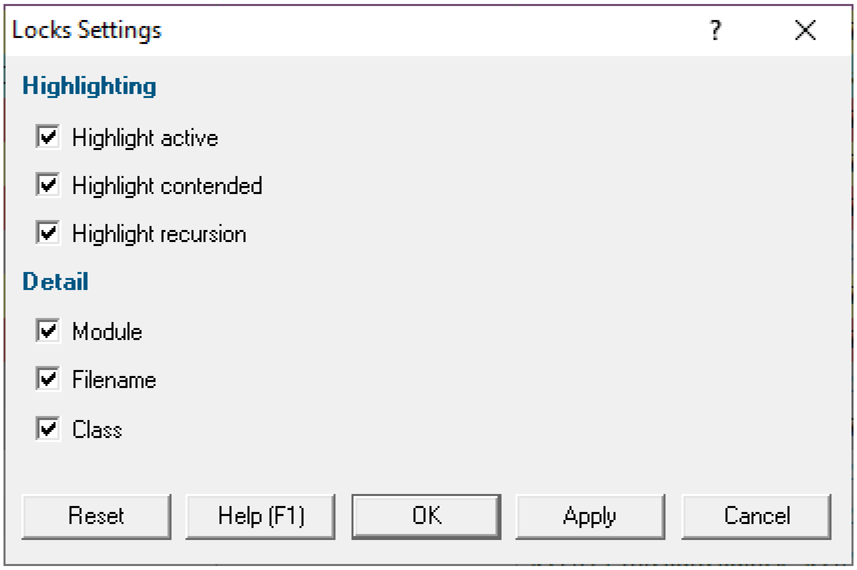
See what data is available for descriptions of each column.

Note that while your application is executing, the sorted data is live. Sorting may not complete correctly as the data may change during the sort. Only when your program has finished executing is the sorted data guaranteed accurate.

Display settings

The Locks Settings dialog controls a small number of highlighting and display options.

- **Display...** › show the Locks Settings dialog:



Locks Settings

Highlighting

☑ Highlight active

☑ Highlight contended

☑ Highlight recursion

Detail

☑ Module

☑ Filename

☑ Class

Highlighting settings:

- **Highlight active** › initialised (entered) critical sections are shown in the active colour (grey ■ by default)

- **Highlight contended** › critical sections that have had to wait are shown in the contending colour (yellow 🟨)

- **Highlight recursion** › re-entered critical sections are shown in the recursion colour (cyan 🟦)

The next three display settings control the display of **Module**, **Filename** and **Class** in the **Owning Module** data column.

Other options include:

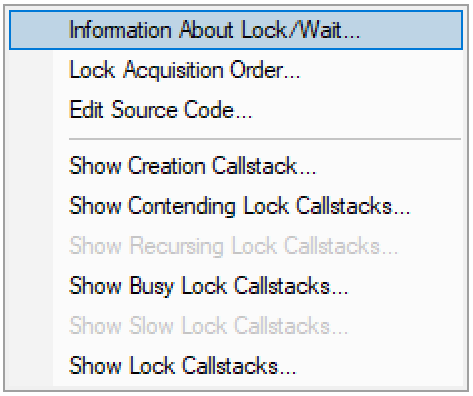
- **Reset** › resets these values to their startup values (all checked)

- **Apply** › apply the settings to the Locks view without closing the dialog

Locks menu options

The following popup menu is available over the data area to add filters, examine more details or edit code.

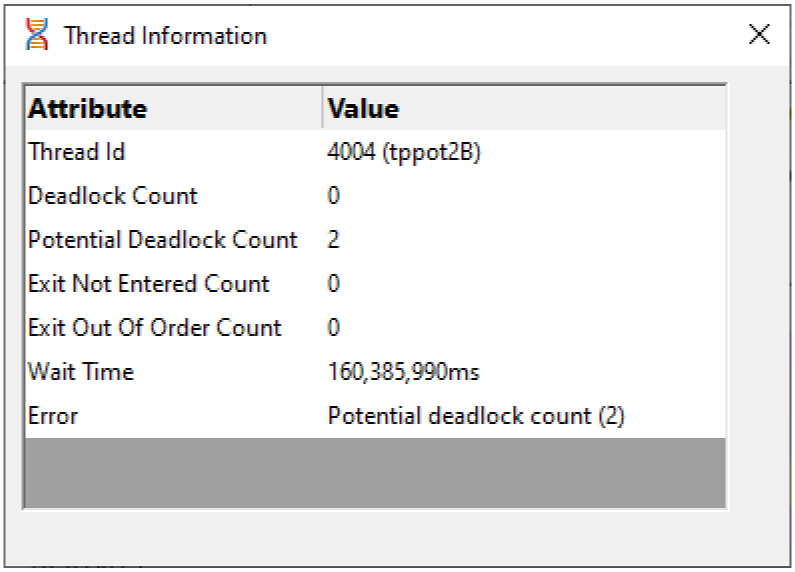
Menu actions apply to the function for the row at the menu-click location.


Information About Lock/Wait...
Lock Acquisition Order...
Edit Source Code...
Show Creation Callstack...
Show Contending Lock Callstacks...
Show Recursing Lock Callstacks...
Show Busy Lock Callstacks...
Show Slow Lock Callstacks...
Show Lock Callstacks...

Menu options: Information about lock or wait

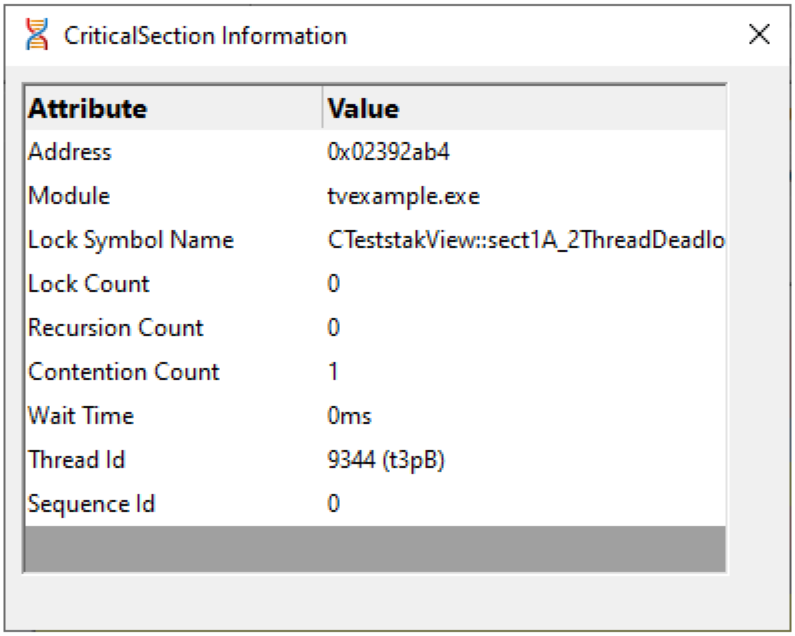
- **Information about lock/wait...** › shows the relevant information dialog from those below, and depending on the type of item selected

  These information dialogs do not block the application so you can show as many as you need, either from this tab or others, and leave them open to compare or investigate later.



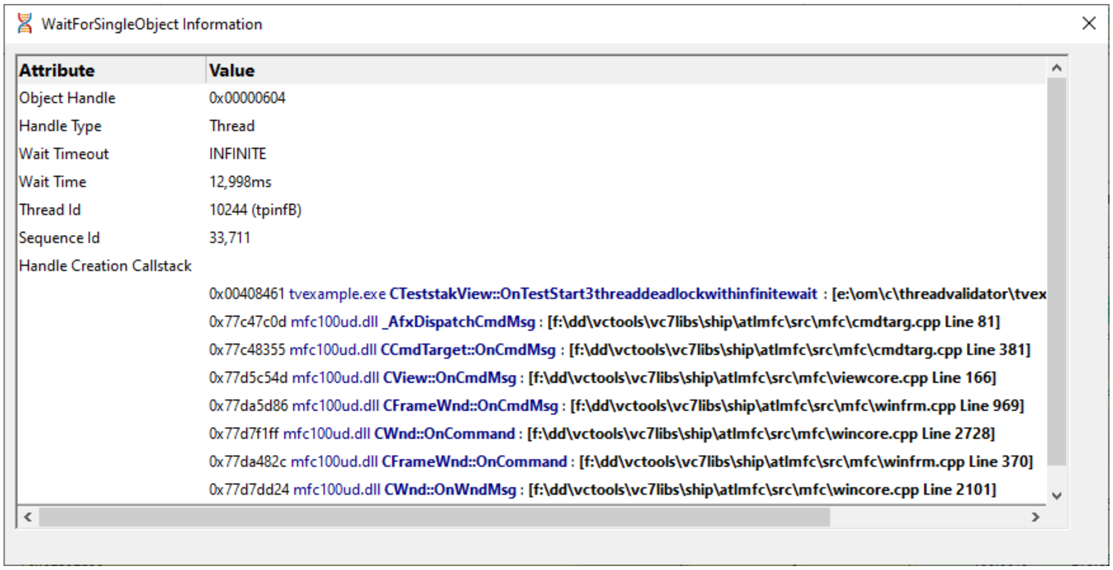
Thread Information

Attribute	Value
Thread Id	4004 (tppot2B)
Deadlock Count	0
Potential Deadlock Count	2
Exit Not Entered Count	0
Exit Out Of Order Count	0
Wait Time	160,385,990ms
Error	Potential deadlock count (2)



CriticalSection Information

Attribute	Value
Address	0x02392ab4
Module	tvexample.exe
Lock Symbol Name	CTeststakView::sect1A_2ThreadDeadlo
Lock Count	0
Recursion Count	0
Contention Count	1
Wait Time	0ms
Thread Id	9344 (t3pB)
Sequence Id	0



WaitForSingleObject Information

Attribute	Value
Object Handle	0x00000604
Handle Type	Thread
Wait Timeout	INFINITE
Wait Time	12,998ms
Thread Id	10244 (tpinfB)
Sequence Id	33,711
Handle Creation Callstack	
	0x00408461 tvexample.exe CTeststakView::OnTestStart3threaddeadlockwithinfinitewait : [e:\om\c\threadvalidator\tvex
	0x77c47c0d mfc100ud.dll _AfxDispatchCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 81]
	0x77c48355 mfc100ud.dll CCmdTarget::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 381]
	0x77d5c54d mfc100ud.dll CView::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\viewcore.cpp Line 166]
	0x77da5d86 mfc100ud.dll CFrameWnd::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 969]
	0x77d7f1ff mfc100ud.dll CWnd::OnCommand : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\wincore.cpp Line 2728]
	0x77da482c mfc100ud.dll CFrameWnd::OnCommand : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 370]
	0x77d7dd24 mfc100ud.dll CWnd::OnWndMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\wincore.cpp Line 2101]

Menu options: Lock acquisition order

- **Lock Acquisition Order >** shows the Locks and Waits in Sequence Order dialog

  This dialog displays the order...

  - in which critical sections are locked and waited upon
  - that waits are entered into

- in which threads sleep and are suspended

Menu option: editing source code

- **Edit Source Code...** › opens the default or preferred editor to edit the source code

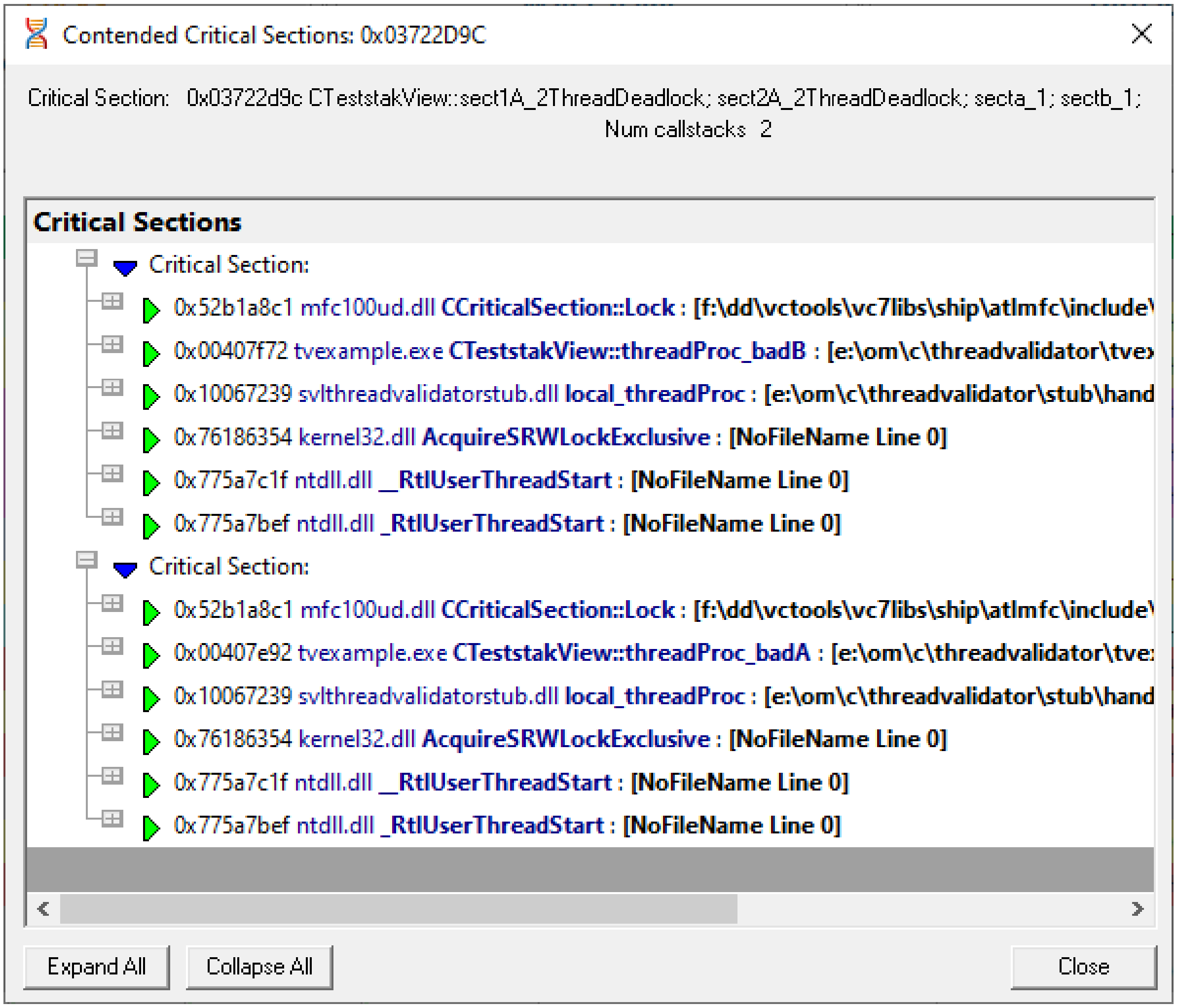
Menu options: creation callstack

- **Show Creation Callstack...** › shows the Critical Section Callstack dialog for the creation of this item, i.e. the locked or waiting critical section and thread

Menu options: contended / recursion callstack

The following menu item is only available over *contended* critical sections (highlighted in yellow 🟨 by default)

- **Show contended callstacks...** › shows the Contended Critical Sections dialog for this item



Contended Critical Sections: 0x03722D9C

Critical Section: 0x03722d9c CTeststakView::sect1A_2ThreadDeadlock; sect2A_2ThreadDeadlock; secta_1; sectb_1;
Num callstacks 2

Critical Sections

```
Critical Section:
    0x52b1a8c1 mfc100ud.dll CCriticalSection::Lock : [f:\dd\vctools\vc7libs\ship\atlmfc\include\
    0x00407f72 tvexample.exe CTeststakView::threadProc_badB : [e:\om\c\threadvalidator\tvex
    0x10067239 svlthreadvalidatorstub.dll local_threadProc : [e:\om\c\threadvalidator\stub\hand
    0x76186354 kernel32.dll AcquireSRWLockExclusive : [NoFileName Line 0]
    0x775a7c1f ntdll.dll __RtlUserThreadStart : [NoFileName Line 0]
    0x775a7bef ntdll.dll _RtlUserThreadStart : [NoFileName Line 0]
Critical Section:
    0x52b1a8c1 mfc100ud.dll CCriticalSection::Lock : [f:\dd\vctools\vc7libs\ship\atlmfc\include\
    0x00407e92 tvexample.exe CTeststakView::threadProc_badA : [e:\om\c\threadvalidator\tve:
    0x10067239 svlthreadvalidatorstub.dll local_threadProc : [e:\om\c\threadvalidator\stub\hand
    0x76186354 kernel32.dll AcquireSRWLockExclusive : [NoFileName Line 0]
    0x775a7c1f ntdll.dll __RtlUserThreadStart : [NoFileName Line 0]
    0x775a7bef ntdll.dll _RtlUserThreadStart : [NoFileName Line 0]
```

This dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

The following menu item is only available over *recursing* critical sections (highlighted in light blue 🟦 by default)

- **Show recursion callstacks...** › shows the Recursing Critical Sections dialog for this item

  This dialog is very similar to the one for contentions, above.

  As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

- **Show busy lock callstacks...** › shows the busy Critical Sections dialog for this item

  This dialog is very similar to the one for contentions, above.

  As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

- **Show slow lock callstacks...** › shows the slow Critical Sections dialog for this item

This dialog is very similar to the one for contentions, above.

As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

- **Show lock callstacks...** › shows the Critical Sections dialog for this item

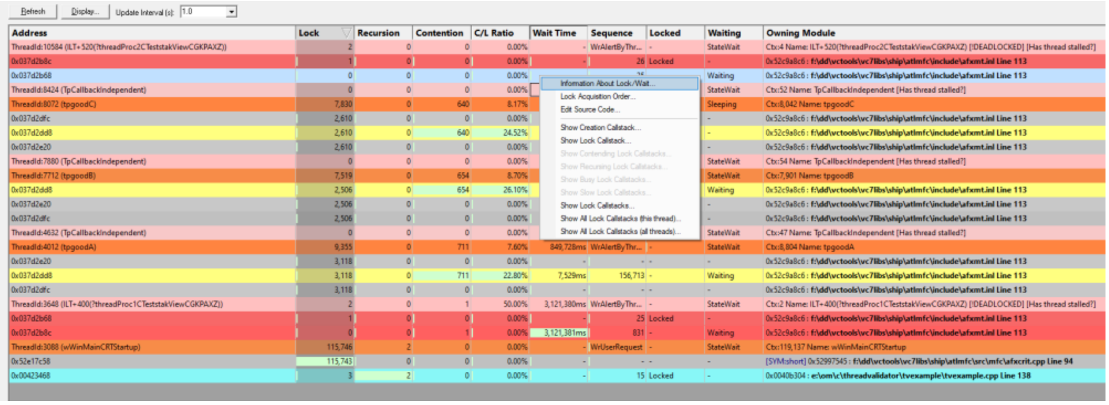
This dialog is very similar to the one for contentions, above.

As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

3.7.2.2 Locks Per Thread

The **Locks Per Thread** displays information about the critical sections and `WaitForXXXObject` calls used by each thread in the application being monitored.

Read on, or click a part of the image below to jump straight to the help for that area.


Refresh
Display...
Information About Lock/Wait...
Lock Acquisition Order...
Edit Source Code...
Show Creation Callstack...
Show Lock Callstack...
Show Contending Lock Callstacks...
Show Recursing Lock Callstacks...
Show Busy Lock Callstacks...
Show Slow Lock Callstacks...
Show Lock Callstacks...
Show All Lock Callstacks (this thread)...
Show All Lock Callstacks (all threads)...

Update Interval (s): 1.0

Address	Lock	Recursion	Contention	C/L Ratio	Wait Time	Sequence	Locked	Waiting	Owning Module
ThreadId:10584 (ILT+520(?threadProc2CTeststakViewCGKPAXZ))	2	0	0	0.00%	-	WrAlertByThr...	-	StateWait	Ctx:4 Name: ILT+520(?threadProc2CTeststakViewCGKPAXZ) [!DEADLOCKED] [Has thread stalled?]
0x037d2b8c	1	0	0	0.00%	-	26	Locked	-	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x037d2b68	0	0	0	0.00%				Waiting	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
ThreadId:8424 (TpCallbackIndependent)	0	0	0	0.00%				StateWait	Ctx:52 Name: TpCallbackIndependent [Has thread stalled?]
ThreadId:8072 (tpgoodC)	7,830	0	640	8.17%				Sleeping	Ctx:8,042 Name: tpgoodC
0x037d2dfc	2,610	0	0	0.00%				-	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x037d2dd8	2,610	0	640	24.52%				-	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x037d2e20	2,610	0	0	0.00%				-	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
ThreadId:7880 (TpCallbackIndependent)	0	0	0	0.00%				StateWait	Ctx:54 Name: TpCallbackIndependent [Has thread stalled?]
ThreadId:7712 (tpgoodB)	7,519	0	654	8.70%				StateWait	Ctx:7,901 Name: tpgoodB
0x037d2dd8	2,506	0	654	26.10%				Waiting	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x037d2e20	2,506	0	0	0.00%				-	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x037d2dfc	2,506	0	0	0.00%				-	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
ThreadId:4632 (TpCallbackIndependent)	0	0	0	0.00%				StateWait	Ctx:47 Name: TpCallbackIndependent [Has thread stalled?]
ThreadId:4012 (tpgoodA)	9,355	0	711	7.60%	849,728ms	WrAlertByThr...	-	StateWait	Ctx:8,804 Name: tpgoodA
0x037d2e20	3,118	0	0	0.00%	-	-	-	-	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x037d2dd8	3,118	0	711	22.80%	7,529ms	156,713	-	Waiting	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x037d2dfc	3,118	0	0	0.00%	-	-	-	-	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
ThreadId:3648 (ILT+400(?threadProc1CTeststakViewCGKPAXZ))	2	0	1	50.00%	3,121,380ms	WrAlertByThr...	-	StateWait	Ctx:2 Name: ILT+400(?threadProc1CTeststakViewCGKPAXZ) [!DEADLOCKED] [Has thread stalled?]
0x037d2b68	1	0	0	0.00%	-	25	Locked	-	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x037d2b8c	0	0	1	0.00%	3,121,381ms	831	-	Waiting	0x52c9a8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
ThreadId:3088 (wWinMainCRTStartup)	115,746	2	0	0.00%	-	WrUserRequest	-	StateWait	Ctx:119,137 Name: wWinMainCRTStartup
0x52e17c58	115,743	0	0	0.00%	-	-	-	-	[SYM:short] 0x52997545 : **f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\afxcrit.cpp Line 94**
0x00423468	3	2	0	0.00%	-	15	Locked	-	0x0040b304 : **e:\om\c\threadvalidator\tvexample\tvexample.cpp Line 138**

The Locks Per Thread View

The view shows information about critical sections used by each thread in the target application.

In the example above, information for six threads are displayed, with each thread having with a summary header row identified by the `ThreadId:nnn` in the Address field.

An additional three threads (TpCallbackIndependent) are shown but have no associated locks or waits.

Thread rows will be highlighted orange 🟧 (thread information colour), or pale red ▨ (the stalled thread colour) unless you customise these colours.

Each relevant critical section is shown once underneath each thread and the display can be sorted by different criteria by clicking on the appropriate column header.

The table highlights error and status conditions using the user defined lock colours. In the example above, red 🟥 indicates a deadlock, and cyan 🟦 for recursion.

Lines highlighted in grey ⬜ denote an active status, meaning that the critical section has been entered.

Unlike the All Locks tab, critical sections where nothing of interest has happened are not displayed.

Any `WaitForXXXObject` calls used by each thread in the application are also listed.

What data is available?

The following tables list all the available columns of data.

Note that the thread header rows display different information to the critical section rows.

For critical section rows:

Attribute	Description
Address	Address of the critical section, the handle for a `WaitForSingleObject` call, or the number of handles for a `WaitForMultipleObjects` call.
Lock	Number of times the critical section has been locked by any thread.
Recursion	Number of times the critical section has been re-entered whilst locked by the current thread.
Contention	Number of attempts a thread has made to acquire the critical section but has had to wait because it is owned by another thread.
Wait Time	Total time spent waiting by all threads to gain access to this critical section.
Sequence	The sequence id is an internal monotonic integer that is incremented for each synchronization object event. This integer allows you to identify the order in which events occurred. Note that this can be misleading if the windows scheduler switches threads at the wrong time.
Locked	Shows "Locked" if the critical section is locked.
Waiting	Shows "Waiting" if the critical section is waiting.
Owning Module	One or more of the following values if available, shown in order of priority with most important first:

- Thread name only if the thread has been given a name, the thread name is also displayed
- Filename and line number of the code location that acquired the lock, determined from the callstack (if collected)
- Filename and line number of the critical section
- Name of the owning module, if the critical section is static data inside a module (DLL or EXE)
- Data indicating dynamic memory or stack space

Some of these details can be optionally hidden using the Display options.

For thread header rows, the information is typically a total for the thread as a whole, or an indication of thread state.

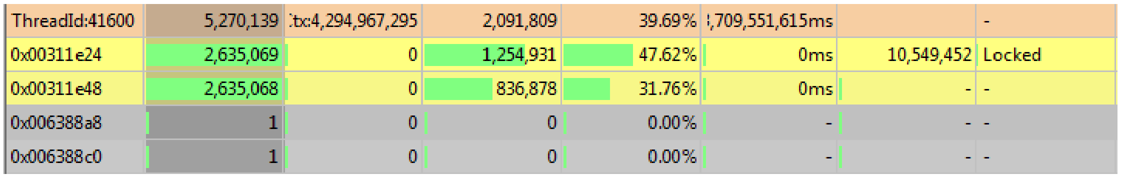
Attribute	Description
• **Address**	Thread Id
• **Lock**	The total number of locks acquired by this thread
• **Recursion**	The total number of context switches performed by this thread
• **Contention**	The total number of critical section contentions for this thread
• **C/L Ratio**	The percentage of the total contentions over the total number of locks for this thread
• **Wait Time**	The total number time spent waiting for access to critical sections
• **Sequence**	Not used
• **Locked**	Not used
• **Waiting**	The current state of the thread - as obtained by polling the operating system for data at regular intervals. As such this information may not match the other data at all times. This information is very useful for identifying thread conditions when thread states do not change for long time periods (long wait states and deadlocks).
• **Owning Module**	Information about thread errors. As an example, in the image at the top of this page the thread entry lines (starting with `ThreadId::nnn` in the Address column), you can see that the data for deadlocked threads reads [!DEADLOCKED] and the data for threads with recursions reads [!Locking errors], indicating serious errors for these threads.

Other messages might indicate that the thread is suspended, sleeping, stalled or has had an exit from a critical section that has not been entered.

If nothing of interest is happening the cell will just be left blank.

Percentage bars and markers

All the cells in the table that show timing or count data for critical sections have a percentage bar indicating the item's value relative to the maximum value in the column.



ThreadId:41600	5,270,139	:tx:4,294,967,295	2,091,809	39.69%	;,709,551,615ms		-
0x00311e24	2,635,069	0	1,254,931	47.62%	0ms	10,549,452	Locked
0x00311e48	2,635,068	0	836,878	31.76%	0ms	-	-
0x006388a8	1	0	0	0.00%	-	-	-
0x006388c0	1	0	0	0.00%	-	-	-

Not all bars are percentages. In the Sequence column, the markers indicate the relative position of the sequence id (see table above) with respect to the first sequence id and most recent sequence id. Thread header rows show the marker position for the highest sequence id of any locks in that thread.

Updating the display

- **Update Interval** › automatically updates the display at your choice of interval, from 0.1 to 60 seconds

  This should be set depending on the complexity of your application.

An update interval that is too short may mean Thread Validator spends too much time updating the display.

- **Refresh** › updates the display - as does the button on the Tools menu and toolbar

Sorting the data

You can sort the data in the table by clicking in the table header.

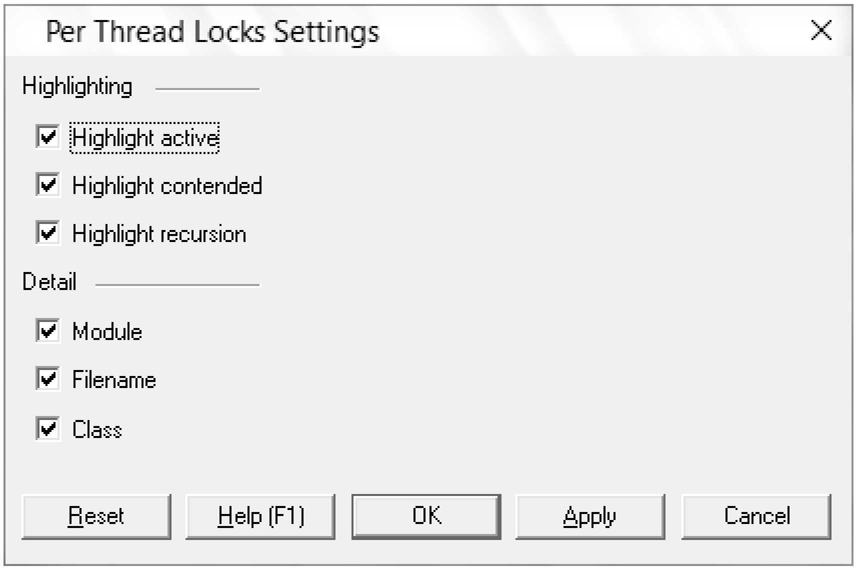
See what data is available for descriptions of each column

Note that while your application is executing, the sorted data is live. Sorting may not complete correctly as the data may change during the sort. Only when your program has finished executing is the sorted data guaranteed accurate.

Display settings

The Locks Settings dialog controls a small number of highlighting and display options.

- **Display...** › show the Locks Per Thread Settings dialog:



Per Thread Locks Settings

Highlighting

☑ Highlight active

☑ Highlight contended

☑ Highlight recursion

Detail

☑ Module

☑ Filename

☑ Class

Highlighting settings:

- **Highlight active** › initialised (entered) critical sections are shown in the active colour (grey ⬜ by default)

- **Highlight contended** › critical sections that have had to wait are shown in the contending colour (yellow 🟨)

- **Highlight recursion** › re-entered critical sections are shown in the recursion colour (cyan 🟦)

The next three display settings control the display of **Module**, **Filename** and **Class** in the **Owning Module** data column.

Other options include:

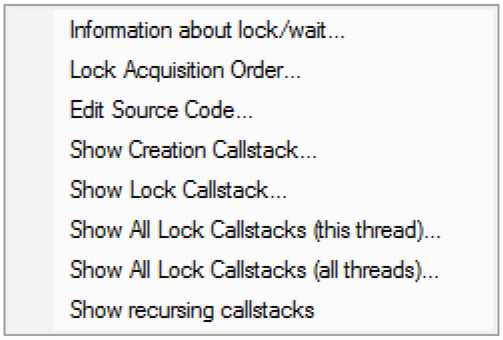
- **Reset** › resets these values to their startup values (all checked)

- **Apply** › apply the settings to the Locks view without closing the dialog

Locks Per Thread menu options

The following popup menu is available over the data area to add filters, examine more details or edit code.

Menu actions apply to the function for the row at the menu-click location.


Information about lock/wait...
Lock Acquisition Order...
Edit Source Code...
Show Creation Callstack...
Show Lock Callstack...
Show All Lock Callstacks (this thread)...
Show All Lock Callstacks (all threads)...
Show recursing callstacks

Menu options: Information about lock or wait

- **Information about lock/wait...** › shows the relevant information dialog from those shown for the same menu option on the All Locks tab

Menu options: Lock acquisition order

- **Lock Acquisition Order** › shows the Locks and Waits in Sequence Order dialog

  This dialog displays the order...

  - in which critical sections are locked and waited upon
  - that waits are entered into
  - in which threads sleep and are suspended

Menu option: editing source code

- **Edit Source Code...** › opens the default or preferred editor to edit the source code

Menu options: show ... callstacks

Each of these four options show the Critical Section Callstack dialog for the selected item's callstack

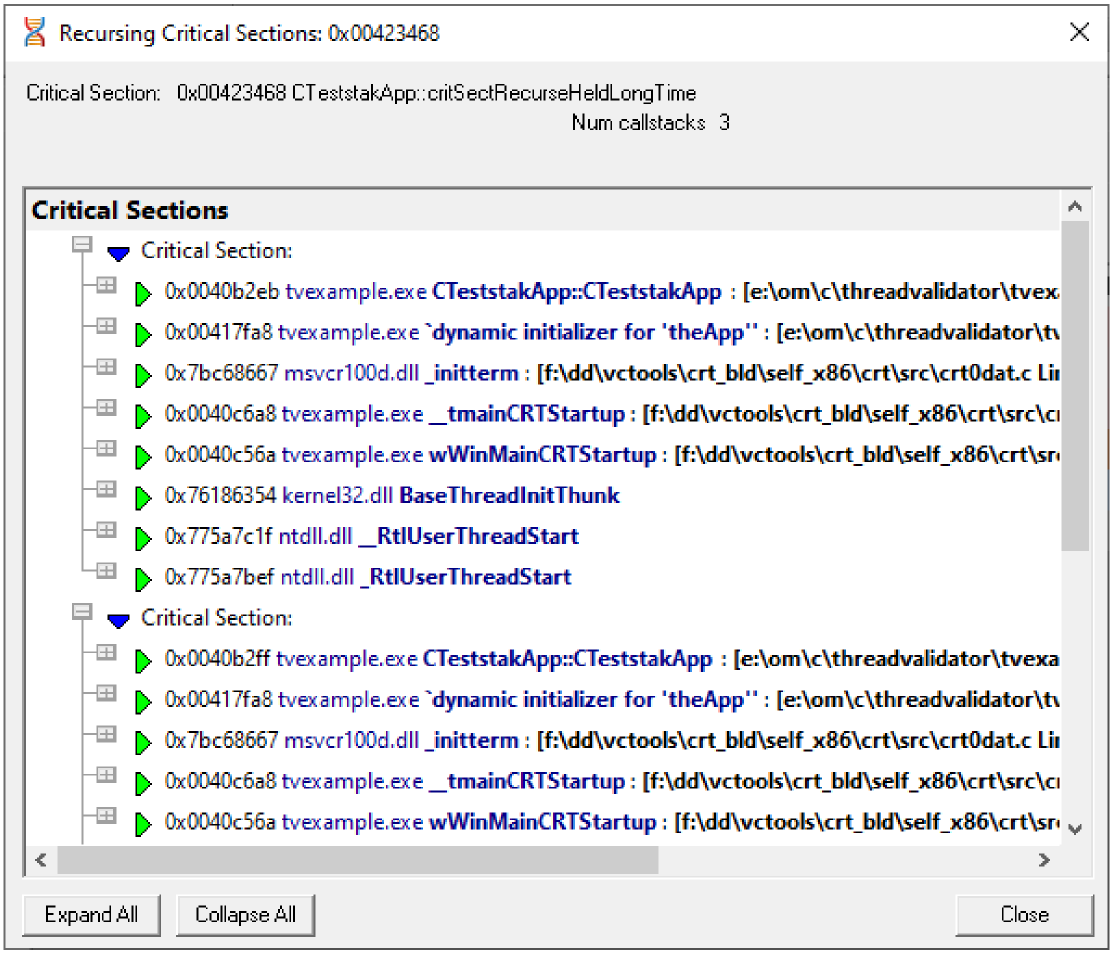
- **Show Creation Callstack...** › shows the callstack for the creation of this item, i.e. the locked or waiting critical section and thread
- **Show Lock Callstack...** › shows the callstack for the locked or waiting critical section and thread
- **Show All Lock Callstacks (this thread)...** › shows all callstacks for this critical section and *only* this thread
- **Show All Lock Callstacks (all threads)...** › as above but for *any* thread

The following menu item is only available over *recursing* critical sections (highlighted in light blue 🟦 by default)

- **Show recursing callstack...** › shows the Recursing Critical Sections dialog for this item

The critical section in question is shown at the top.

This dialog can only be shown if the callstack information is still available. If not you'll hear a beep.



Recursing Critical Sections: 0x00423468

Critical Section: 0x00423468 CTeststakApp::critSectRecurseHeldLongTime

Num callstacks 3

Critical Sections

```
Critical Section:
  0x0040b2eb tvexample.exe CTeststakApp::CTeststakApp : [e:\om\c\threadvalidator\tvex
  0x00417fa8 tvexample.exe `dynamic initializer for 'theApp'' : [e:\om\c\threadvalidator\tv
  0x7bc68667 msvcr100d.dll _initterm : [f:\dd\vctools\crt_bld\self_x86\crt\src\crt0dat.c Lin
  0x0040c6a8 tvexample.exe __tmainCRTStartup : [f:\dd\vctools\crt_bld\self_x86\crt\src\cr
  0x0040c56a tvexample.exe wWinMainCRTStartup : [f:\dd\vctools\crt_bld\self_x86\crt\sr
  0x76186354 kernel32.dll BaseThreadInitThunk
  0x775a7c1f ntdll.dll __RtlUserThreadStart
  0x775a7bef ntdll.dll _RtlUserThreadStart
Critical Section:
  0x0040b2ff tvexample.exe CTeststakApp::CTeststakApp : [e:\om\c\threadvalidator\tvexa
  0x00417fa8 tvexample.exe `dynamic initializer for 'theApp'' : [e:\om\c\threadvalidator\tv
  0x7bc68667 msvcr100d.dll _initterm : [f:\dd\vctools\crt_bld\self_x86\crt\src\crt0dat.c Lin
  0x0040c6a8 tvexample.exe __tmainCRTStartup : [f:\dd\vctools\crt_bld\self_x86\crt\src\cr
  0x0040c56a tvexample.exe wWinMainCRTStartup : [f:\dd\vctools\crt_bld\self_x86\crt\sr
```

The following menu item is only available over *contended* critical sections (highlighted in yellow 🟨 by default)

- **Show contended callstacks...** › shows the Contended Critical Sections dialog for this item

This dialog is very similar to the one for recursions, above.

As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

The following menu item is only available over *recursing* critical sections (highlighted in light blue 🟦 by default)

- **Show recursion callstacks...** › shows the Recursing Critical Sections dialog for this item

  This dialog is very similar to the one for contentions, above.

  As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

- **Show busy lock callstacks...** › shows the busy Critical Sections dialog for this item

  This dialog is very similar to the one for contentions, above.

  As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

- **Show slow lock callstacks...** › shows the slow Critical Sections dialog for this item

  This dialog is very similar to the one for contentions, above.

  As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

- **Show lock callstacks...** › shows the Critical Sections dialog for this item

  This dialog is very similar to the one for contentions, above.

  As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

- **Show all lock callstacks (this thread)...** › shows the Critical Sections dialog for this item

  This dialog is very similar to the one for contentions, above.

  As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

- **Show all lock callstacks (all threads)...** › shows the Critical Sections dialog for this item

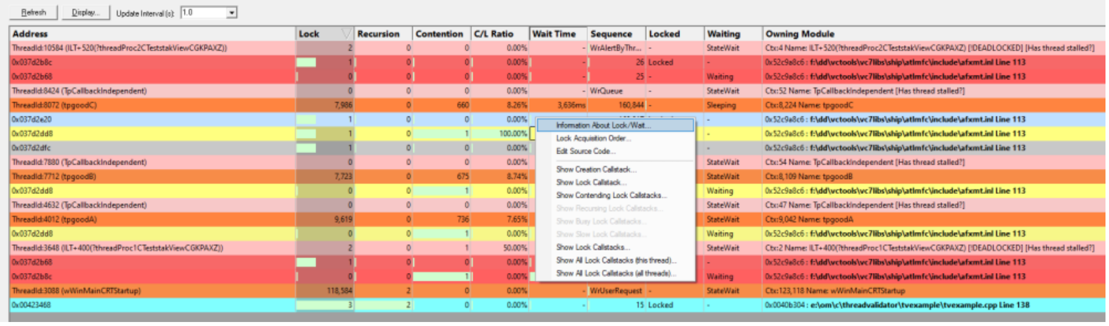
  This dialog is very similar to the one for contentions, above.

  As above, this dialog can only be shown if the callstack information is still available. If not you'll hear a beep.

3.7.2.3 Active Locks

The **Active Locks** displays information about the *currently active* critical sections and `WaitForXXXObject` calls in each thread of the application being monitored.

Read on, or click a part of the image below to jump straight to the help for that area.


Refresh
Display...
Information About Lock/Wait...
Lock Acquisition Order...
Edit Source Code...
Show Creation Callstack...
Show Lock Callstack...
Show Contending Lock Callstacks...
Show Recursing Lock Callstacks...
Show Busy Lock Callstacks...
Show Slow Lock Callstacks...
Show Lock Callstacks...
Show All Lock Callstacks (this thread)...
Show All Lock Callstacks (all threads)...

Update Interval (s): 1.0

Address	Lock	Recursion	Contention	C/L Ratio	Wait Time	Sequence	Locked	Waiting	Owning Module
ThreadId:10584 (ILT+520(?threadProc2CTeststakViewCGKPAXZ))	2	0	0	0.00%	-	WrAlertByThr...	-	StateWait	Ctx:4 Name: ILT+520(?threadProc2CTeststakViewCGKPAXZ) [!DEADLOCKED] [Has thread stalled?]
0x037d2b8c	1	0	0	0.00%	-	26	Locked	-	0x52c9a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x037d2b68	0	0	0	0.00%	-	25	-	Waiting	0x52c9a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
ThreadId:8424 (TpCallbackIndependent)	0	0	0	0.00%	-	WrQueue	-	StateWait	Ctx:52 Name: TpCallbackIndependent [Has thread stalled?]
ThreadId:8072 (tpgoodC)	7,986	0	660	8.26%	3,636ms	160,844	-	Sleeping	Ctx:8,224 Name: tpgoodC
0x037d2e20	1	0	0	0.00%				-	0x52c9a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x037d2dd8	1	0	1	100.00%				-	0x52c9a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x037d2dfc	1	0	0	0.00%				-	0x52c9a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
ThreadId:7880 (TpCallbackIndependent)	0	0	0	0.00%				StateWait	Ctx:54 Name: TpCallbackIndependent [Has thread stalled?]
ThreadId:7712 (tpgoodB)	7,723	0	675	8.74%				StateWait	Ctx:8,109 Name: tpgoodB
0x037d2dd8	0	0	1	0.00%				Waiting	0x52c9a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
ThreadId:4632 (TpCallbackIndependent)	0	0	0	0.00%				StateWait	Ctx:47 Name: TpCallbackIndependent [Has thread stalled?]
ThreadId:4012 (tpgoodA)	9,619	0	736	7.65%				StateWait	Ctx:9,042 Name: tpgoodA
0x037d2dd8	0	0	1	0.00%				Waiting	0x52c9a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
ThreadId:3648 (ILT+400(?threadProc1CTeststakViewCGKPAXZ))	2	0	1	50.00%				StateWait	Ctx:2 Name: ILT+400(?threadProc1CTeststakViewCGKPAXZ) [!DEADLOCKED] [Has thread stalled?]
0x037d2b68	1	0	0	0.00%				-	0x52c9a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
0x037d2b8c	0	0	1	0.00%				Waiting	0x52c9a8c6 : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113
ThreadId:3088 (wWinMainCRTStartup)	118,584	2	0	0.00%	-	WrUserRequest	-	StateWait	Ctx:123,118 Name: wWinMainCRTStartup
0x00423468	3	2	0	0.00%	-	15	Locked	-	0x0040b304 : e:\om\c\threadvalidator\tvexample\tvexample.cpp Line 138

The Active Locks View

The view shows information about critical sections currently locked or waiting for each known thread in the target application.

The display and controls are the same as the Locks Per Thread tab, except that here only *currently active* locks are listed, rather than *all* previously active critical sections.

In the example above information for seven threads are displayed, with each thread having with a summary header row identified by the `ThreadId:nnn` in the Address field.

Thread rows will be highlighted in the thread information colour (orange 🟧 by default), or pale red ■, the stalled thread colour unless you customise these colours.

Each relevant critical section is shown once underneath each thread and the display can be sorted by different criteria by clicking on the appropriate column header.

The table highlights error and status conditions using the user defined lock colours. In the example above, yellow 🟨 indicates a contention, and cyan ■ for recursion.

Lines highlighted in grey ■ denote an active status, meaning that the critical section has been entered.

Unlike the All Locks tab, critical sections where nothing of interest has happened are not displayed.

Any `WaitForXXXObject` calls used by each thread in the application are also listed.

There may not be any critical sections listed for a given thread.

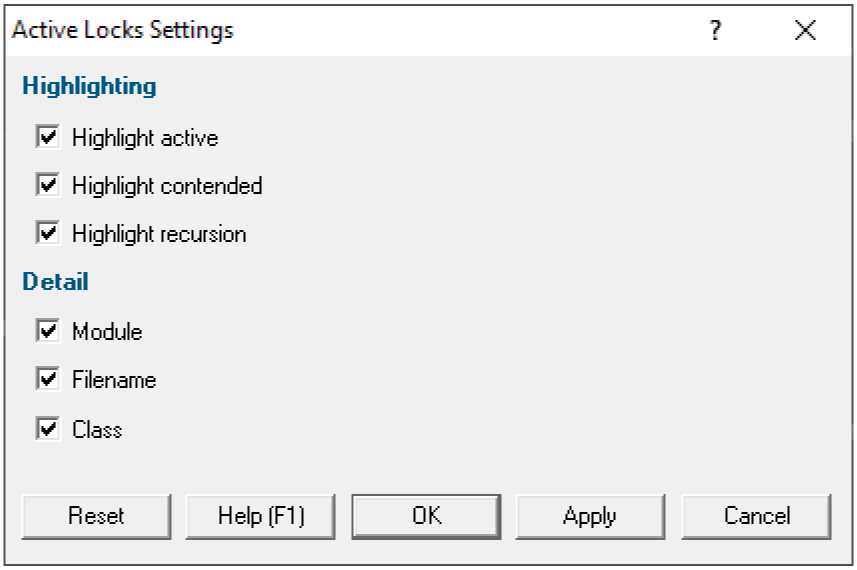
What data is available?

The columns displayed in the Active Locks view are identical to the columns on the Locks Per Thread tab.

Note that like the Locks Per Thread, the thread header rows display different information to the critical section rows.

Display settings

The Active Locks Settings dialog below, is identical to the Display Settings on the Locks Per Thread tab.



Active Locks Settings

Highlighting

☑ Highlight active

☑ Highlight contended

☑ Highlight recursion

Detail

☑ Module

☑ Filename

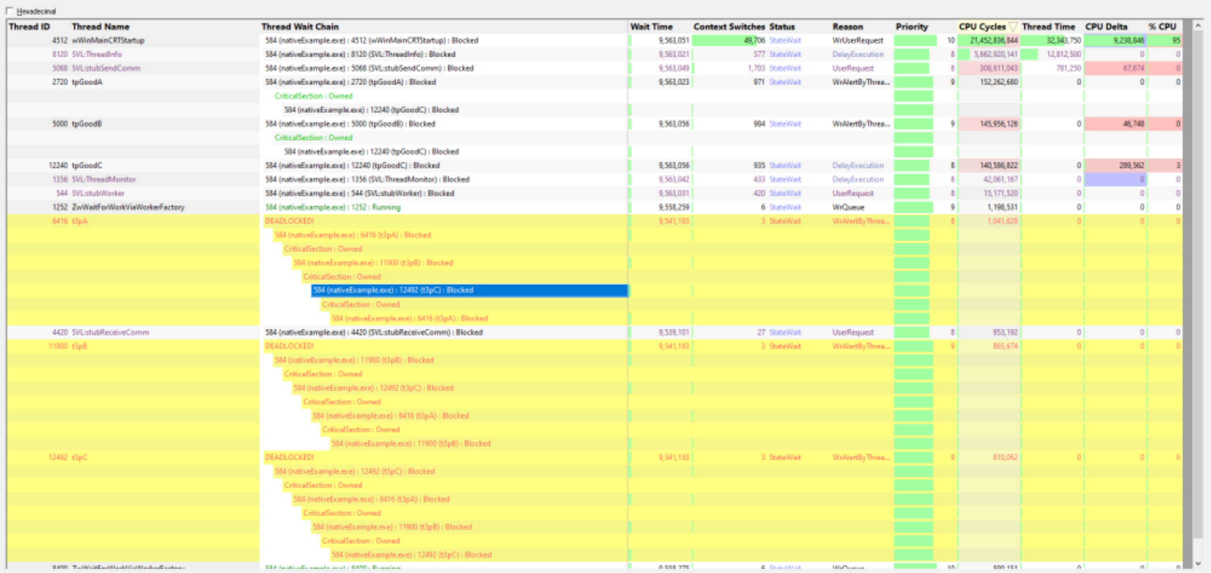
☑ Class

Active Locks menu options

Like the display settings, the menu options are also the same as those on the Locks Per Thread tab.

3.7.3 Wait Chains

The **Wait Chains** tab displays information about all the wait chains for the application being monitored.



☐ Hexadecimal

Thread ID	Thread Name	Thread Wait Chain	Wait Time	Context Switches	Status	Reason	Priority	CPU Cycles	Thread Time	CPU Delta	% CPU
4512	wWinMainCRTStartup	584 (nativeExample.exe) : 4512 (wWinMainCRTStartup) : Blocked	9,563,051	49,706	StateWait	WrUserRequest	10	21,452,836,844	32,343,750	9,238,846	95
8120	SVL:ThreadInfo	584 (nativeExample.exe) : 8120 (SVL:ThreadInfo) : Blocked	9,563,021	577	StateWait	DelayExecution	8	5,662,920,141	12,812,500	0	0
5068	SVL:stubSendComm	584 (nativeExample.exe) : 5068 (SVL:stubSendComm) : Blocked	9,563,049	1,703	StateWait	UserRequest	8	308,611,043	781,250	67,674	0
2720	tpGoodA	584 (nativeExample.exe) : 2720 (tpGoodA) : Blocked	9,563,023	971	StateWait	WrAlertByThrea...	9	152,262,680	0	0	0
		CriticalSection : Owned									
		584 (nativeExample.exe) : 12240 (tpGoodC) : Blocked									
5000	tpGoodB	584 (nativeExample.exe) : 5000 (tpGoodB) : Blocked	9,563,056	984	StateWait	WrAlertByThrea...	9	145,956,126	0	46,748	0
		CriticalSection : Owned									
		584 (nativeExample.exe) : 12240 (tpGoodC) : Blocked									
12240	tpGoodC	584 (nativeExample.exe) : 12240 (tpGoodC) : Blocked	9,563,056	935	StateWait	DelayExecution	8	140,586,822	0	289,562	3
1356	SVL:ThreadMonitor	584 (nativeExample.exe) : 1356 (SVL:ThreadMonitor) : Blocked	9,563,042	433	StateWait	DelayExecution	8	42,061,167	0	0	0
544	SVL:stubWorker	584 (nativeExample.exe) : 544 (SVL:stubWorker) : Blocked	9,563,031	420	StateWait	UserRequest	8	15,171,520	0	0	0
1252	ZwWaitForWorkViaWorkerFactory	584 (nativeExample.exe) : 1252 : Running	9,558,259	6	StateWait	WrQueue	9	1,198,531	0	0	0
6416	t3pA	DEADLOCKED!	9,541,183	3	StateWait	WrAlertByThrea...	8	1,041,628	0	0	0
		584 (nativeExample.exe) : 6416 (t3pA) : Blocked									
		CriticalSection : Owned									
		584 (nativeExample.exe) : 11900 (t3pB) : Blocked									
		CriticalSection : Owned									
		584 (nativeExample.exe) : 12492 (t3pC) : Blocked									
		CriticalSection : Owned									
		584 (nativeExample.exe) : 6416 (t3pA) : Blocked									
4420	SVL:stubReceiveComm	584 (nativeExample.exe) : 4420 (SVL:stubReceiveComm) : Blocked	9,539,101	27	StateWait	UserRequest	8	953,192	0	0	0
11900	t3pB	DEADLOCKED!	9,541,183	3	StateWait	WrAlertByThrea...	9	865,674	0	0	0
		584 (nativeExample.exe) : 11900 (t3pB) : Blocked									
		CriticalSection : Owned									
		584 (nativeExample.exe) : 12492 (t3pC) : Blocked									
		CriticalSection : Owned									
		584 (nativeExample.exe) : 6416 (t3pA) : Blocked									
		CriticalSection : Owned									
		584 (nativeExample.exe) : 11900 (t3pB) : Blocked									
12492	t3pC	DEADLOCKED!	9,541,183	3	StateWait	WrAlertByThrea...	9	810,062	0	0	0
		584 (nativeExample.exe) : 12492 (t3pC) : Blocked									
		CriticalSection : Owned									
		584 (nativeExample.exe) : 6416 (t3pA) : Blocked									
		CriticalSection : Owned									
		584 (nativeExample.exe) : 11900 (t3pB) : Blocked									
		CriticalSection : Owned									
		584 (nativeExample.exe) : 12492 (t3pC) : Blocked									
8400	ZwWaitForWorkViaWorkerFactory	584 (nativeExample.exe) : 8400 : Running	9,558,275	6	StateWait	WrQueue	10	590,151	0	0	0

The Display

Wait chain information is colour coded. Errors are displayed in red, healthy statuses in black and green. Software Verify threads are displayed in purple so that they are easily identifiable as "not your threads".

For any selected thread, that thread and related threads are shown with a yellow background. This is useful so easily identifying threads that are interacting with each other (in a deadlock or a complex wait), or which are not interacting with any other thread at all.

Information that is displayed about each thread:

Thread id
This is the numeric identifier assigned to the thread by the Windows operating system.

Thread Name
This is the name of the thread if the thread has been given a name using the `SetThreadDescription()` API. This is only available on Windows 10.

If a thread description is not available we use any thread name that is provided using the Microsoft Thread Naming Exception.

If a thread description is not available by either of the above two methods we attempt to provide a name for this thread by querying the thread's start address, and if successful we try to turn this start address into a useful symbolic name. Depending on the process and the operating system these operations may success, in which we display a name, or they may fail in which case we display nothing.

Thread Wait Chain
This is the wait chain for this thread. For each entry we show the process id and the thread id and if the object being waited upon has a name, we show that too. If process names and thread names have been enabled we show these as well. Some wait chains rely on waiting for other processes, which are more readily identified by process name than process id. Any deadlocks that are detected are shown in red.

Wait Time
This is how long the thread has been waiting.

Context switches
This is the number of context switches this thread has been involved in.

Status
The status of this thread. Is the thread running, waiting, sleeping, etc.

Reason
The reason for the status of this thread if the thread is waiting.

Priority
The thread priority.

Cpu Cycles
The number of cpu cycles spent executing this thread.

Thread Time
The amount of time spent executing this thread (returned by the `GetThreadTimes()` API).

CAUTION! This value can be misleading as it only counts whole thread quanta that complete, any thread that switches before it's quanta is complete doesn't add to this count.

Cpu Delta
The amount of cpu cycles spent executing this thread since the last time we updated the display.

CPU
The percentage of CPU that is being used by this thread (percentage relative to this process, not relative to all processes).

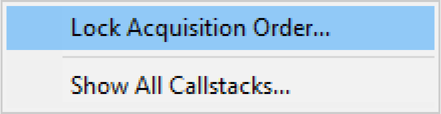
Controls

There is only one control for this display

- **Hexadecimal** › displays values in hexadecimal when selected, decimal when not selected

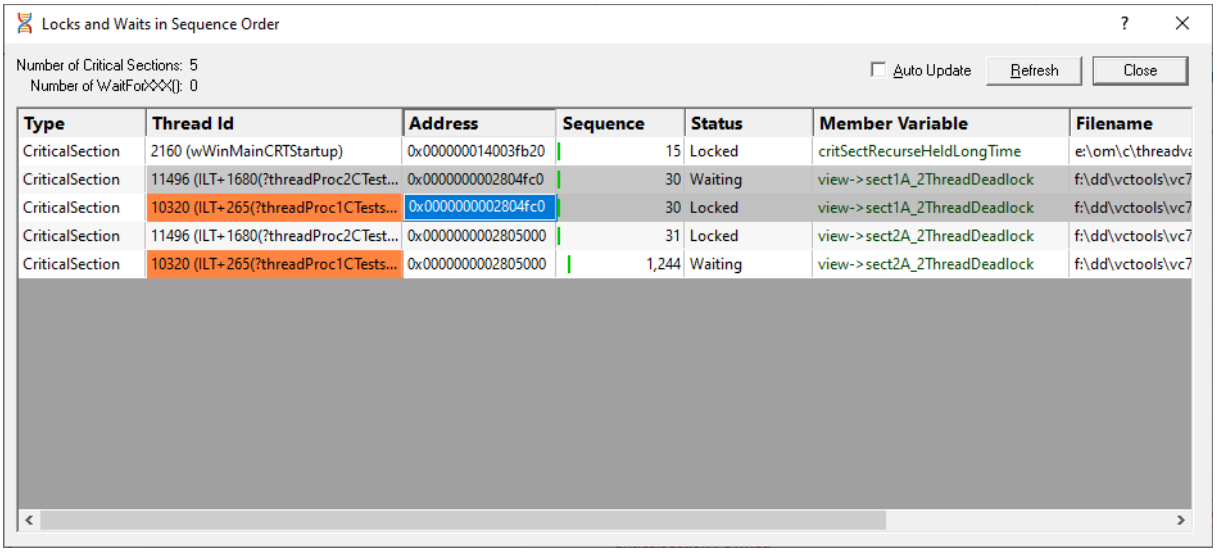
Context Menu

If you right click on the grid, a context menu is displayed.



- **Lock Acquisition Order...** › displays a dialog to allow you to view the order the locks in a wait chain were acquired.

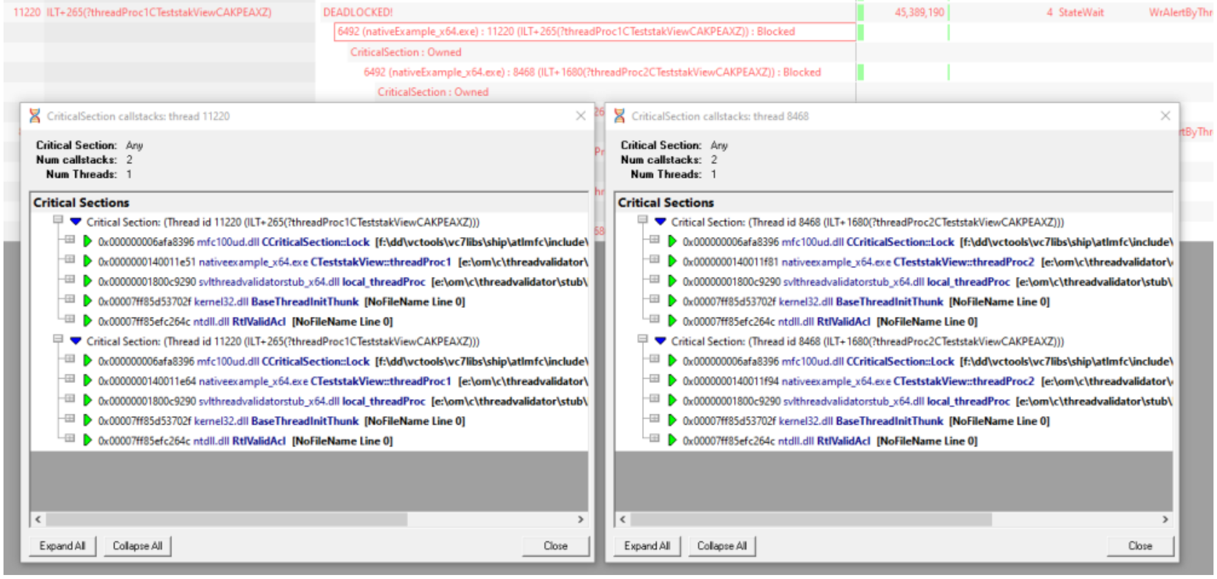
In the image below, the locks are all involved in a deadlock over which context menu was displayed.



Number of Critical Sections: 5
Number of WaitForXXX(): 0

Type	Thread Id	Address	Sequence	Status	Member Variable	Filename
CriticalSection	2160 (wWinMainCRTStartup)	0x000000014003fb20	15	Locked	critSectRecurseHeldLongTime	e:\om\c\threadva
CriticalSection	11496 (ILT+1680(?threadProc2CTest...	0x0000000002804fc0	30	Waiting	view->sect1A_2ThreadDeadlock	f:\dd\vctools\vc7
CriticalSection	10320 (ILT+265(?threadProc1CTests...	0x0000000002804fc0	30	Locked	view->sect1A_2ThreadDeadlock	f:\dd\vctools\vc7
CriticalSection	11496 (ILT+1680(?threadProc2CTest...	0x0000000002805000	31	Locked	view->sect2A_2ThreadDeadlock	f:\dd\vctools\vc7
CriticalSection	10320 (ILT+265(?threadProc1CTests...	0x0000000002805000	1,244	Waiting	view->sect2A_2ThreadDeadlock	f:\dd\vctools\vc7

- **Show All Callstacks...** ❯ for the selected lock (or all locks involved in a wait chain) the callstack for that lock is displayed.


CriticalSection callstacks: thread 11220
Expand All
Collapse All
Close
CriticalSection callstacks: thread 8468
Expand All
Collapse All
Close

```
11220  ILT+265(?threadProc1CTeststakViewCAKPEAXZ)   DEADLOCKED!                                                          45,389,190   4  StateWait   WrAlertByThr
  6492 (nativeExample_x64.exe) : 11220 (ILT+265(?threadProc1CTeststakViewCAKPEAXZ)) : Blocked
    CriticalSection : Owned
      6492 (nativeExample_x64.exe) : 8468 (ILT+1680(?threadProc2CTeststakViewCAKPEAXZ)) : Blocked
        CriticalSection : Owned
```

CriticalSection callstacks: thread 11220

Critical Section: Any
Num callstacks: 2
Num Threads: 1

Critical Sections

```
Critical Section: (Thread id 11220 (ILT+265(?threadProc1CTeststakViewCAKPEAXZ)))
    0x000000006afa8396 mfc100ud.dll CCriticalSection::Lock  [f:\dd\vctools\vc7libs\ship\atlmfc\include\
    0x0000000140011e51 nativeexample_x64.exe CTeststakView::threadProc1  [e:\om\c\threadvalidator\
    0x00000001800c9290 svlthreadvalidatorstub_x64.dll local_threadProc  [e:\om\c\threadvalidator\stub\
    0x00007ff85d53702f kernel32.dll BaseThreadInitThunk  [NoFileName Line 0]
    0x00007ff85efc264c ntdll.dll RtlValidAcl  [NoFileName Line 0]
Critical Section: (Thread id 11220 (ILT+265(?threadProc1CTeststakViewCAKPEAXZ)))
    0x000000006afa8396 mfc100ud.dll CCriticalSection::Lock  [f:\dd\vctools\vc7libs\ship\atlmfc\include\
    0x0000000140011e64 nativeexample_x64.exe CTeststakView::threadProc1  [e:\om\c\threadvalidator\
    0x00000001800c9290 svlthreadvalidatorstub_x64.dll local_threadProc  [e:\om\c\threadvalidator\stub\
    0x00007ff85d53702f kernel32.dll BaseThreadInitThunk  [NoFileName Line 0]
    0x00007ff85efc264c ntdll.dll RtlValidAcl  [NoFileName Line 0]
```

CriticalSection callstacks: thread 8468

Critical Section: Any
Num callstacks: 2
Num Threads: 1

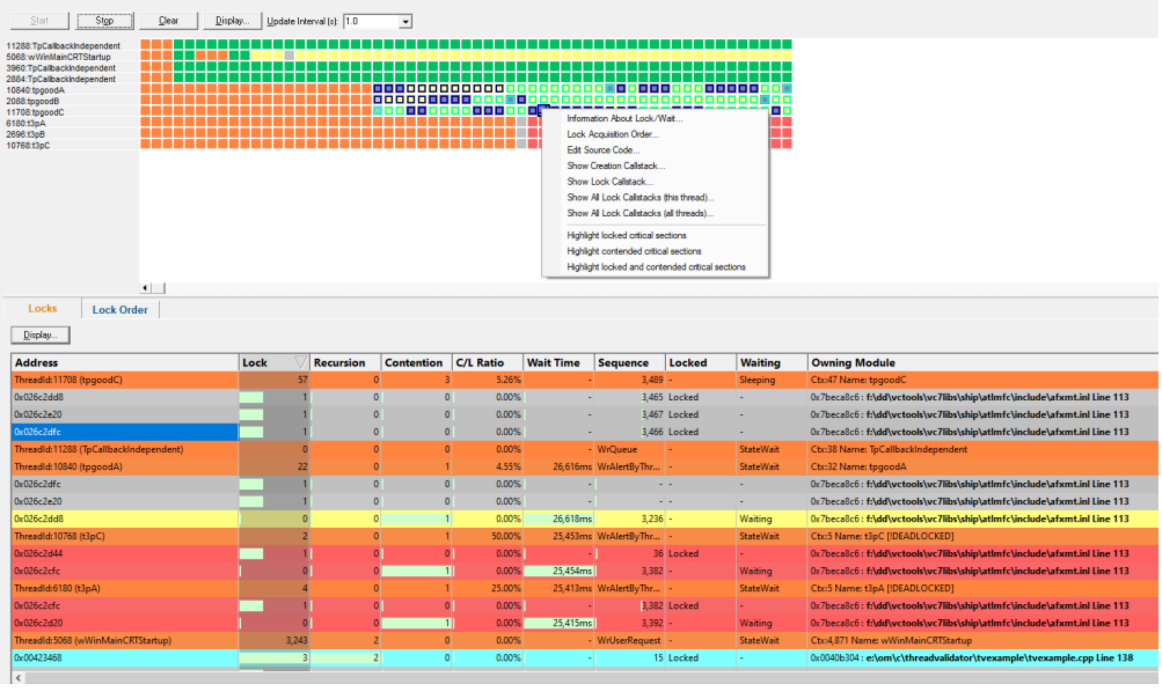
Critical Sections

```
Critical Section: (Thread id 8468 (ILT+1680(?threadProc2CTeststakViewCAKPEAXZ)))
    0x000000006afa8396 mfc100ud.dll CCriticalSection::Lock  [f:\dd\vctools\vc7libs\ship\atlmfc\include\
    0x0000000140011f81 nativeexample_x64.exe CTeststakView::threadProc2  [e:\om\c\threadvalidator\
    0x00000001800c9290 svlthreadvalidatorstub_x64.dll local_threadProc  [e:\om\c\threadvalidator\stub\
    0x00007ff85d53702f kernel32.dll BaseThreadInitThunk  [NoFileName Line 0]
    0x00007ff85efc264c ntdll.dll RtlValidAcl  [NoFileName Line 0]
Critical Section: (Thread id 8468 (ILT+1680(?threadProc2CTeststakViewCAKPEAXZ)))
    0x000000006afa8396 mfc100ud.dll CCriticalSection::Lock  [f:\dd\vctools\vc7libs\ship\atlmfc\include\
    0x0000000140011f94 nativeexample_x64.exe CTeststakView::threadProc2  [e:\om\c\threadvalidator\
    0x00000001800c9290 svlthreadvalidatorstub_x64.dll local_threadProc  [e:\om\c\threadvalidator\stub\
    0x00007ff85d53702f kernel32.dll BaseThreadInitThunk  [NoFileName Line 0]
    0x00007ff85efc264c ntdll.dll RtlValidAcl  [NoFileName Line 0]
```

3.7.4 Threads

The **Threads** tab lets you monitor the history of lock states in each thread in relation to other threads.

Read on, or click a part of the image below to jump straight to the help for that area.


Start
Stop
Clear
Display...
Update Interval (s): 1.0
11288:TpCallbackIndependent
5068:wWinMainCRTStartup
3960:TpCallbackIndependent
2884:TpCallbackIndependent
10840:tpgoodA
2088:tpgoodB
11708:tpgoodC
6180:t3pA
2696:t3pB
10768:t3pC
Information About Lock/Wait...
Lock Acquisition Order...
Edit Source Code...
Show Creation Callstack...
Show Lock Callstack...
Show All Lock Callstacks (this thread)...
Show All Lock Callstacks (all threads)...
Highlight locked critical sections
Highlight contended critical sections
Highlight locked and contended critical sections
Locks
Lock Order
Display...

Address	Lock	Recursion	Contention	C/L Ratio	Wait Time	Sequence	Locked	Waiting	Owning Module
ThreadId:11708 (tpgoodC)	57	0	3	5.26%	-	3,489	-	Sleeping	Ctx:47 Name: tpgoodC
0x026c2dd8	1	0	0	0.00%	-	3,465	Locked	-	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x026c2e20	1	0	0	0.00%	-	3,467	Locked	-	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x026c2dfc	1	0	0	0.00%	-	3,466	Locked	-	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
ThreadId:11288 (TpCallbackIndependent)	0	0	0	0.00%	-	WrQueue	-	StateWait	Ctx:38 Name: TpCallbackIndependent
ThreadId:10840 (tpgoodA)	22	0	1	4.55%	26,616ms	WrAlertByThr...	-	StateWait	Ctx:32 Name: tpgoodA
0x026c2dfc	1	0	0	0.00%	-	-	-	-	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x026c2e20	1	0	0	0.00%	-	-	-	-	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x026c2dd8	0	0	1	0.00%	26,618ms	3,236	-	Waiting	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
ThreadId:10768 (t3pC)	2	0	1	50.00%	25,453ms	WrAlertByThr...	-	StateWait	Ctx:5 Name: t3pC [!DEADLOCKED]
0x026c2d44	1	0	0	0.00%	-	36	Locked	-	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x026c2cfc	0	0	1	0.00%	25,454ms	3,382	-	Waiting	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
ThreadId:6180 (t3pA)	4	0	1	25.00%	25,413ms	WrAlertByThr...	-	StateWait	Ctx:5 Name: t3pA [!DEADLOCKED]
0x026c2cfc	1	0	0	0.00%	-	3,382	Locked	-	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x026c2d20	0	0	1	0.00%	25,415ms	3,392	-	Waiting	0x7beca8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
ThreadId:5068 (wWinMainCRTStartup)	3,243	2	0	0.00%	-	WrUserRequest	-	StateWait	Ctx:4,871 Name: wWinMainCRTStartup
0x00423468	3	2	0	0.00%	-	15	Locked	-	0x0040b304 : **e:\om\c\threadvalidator\tvexample\tvexample.cpp Line 138**

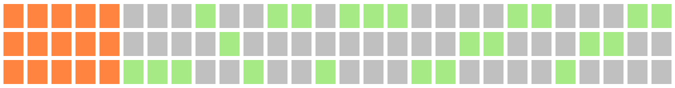
The Thread History View

The view is split vertically into two halves.

The top half monitors a history of lock activity in known threads on a horizontal time-line using sampled snapshots of the thread states.

The state of each thread is displayed as a coloured block, as defined by the lock colour settings.

For example, below is a sequence of 28 samples of three threads taking part in a good locking strategy which starts after the fifth sample.



Newest samples appear on the right of the display.

Selecting any one of the blocks displays the thread and critical section information in the Locks and Lock Order tabs in the lower half of the view.

When selecting a block, any cells that are related by critical section address or waited Win32 handle are also highlighted.

The interval at which thread states are sampled can be changed (see Update Interval in the next section).

The sampled data is stored in a buffer, the size of which can be specified, but which defaults to 1000 points.

The display can be zoomed from one cell per data item to displaying the entire data buffer on the screen.

Managing the time-line data

The options above the time-line graph (initially blank) let you controlling the recording and display of the sampled data.

- **Start / Stop** > control the recording of the sampled thread states
- **Clear** > remove all samples from the display and from the sample buffer

  You don't have to stop recording before clearing the data

- **Update Interval** > automatically updates the display at your choice of interval, from 0.1 to 60 seconds

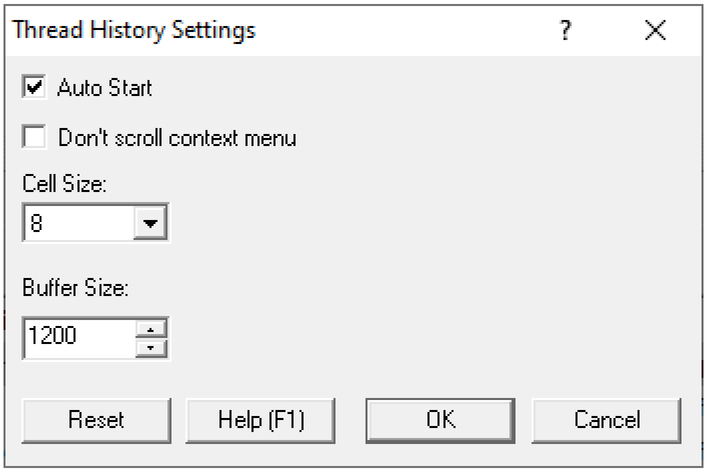
  This should be set depending on the complexity of your application.

  An update interval that is too short may mean Thread Validator spends too much time updating the display.

Display settings

The Thread History Settings dialog controls a small number of recording and display options.

- **Display...** > show the Thread History Settings dialog:



Thread History Settings

☑ Auto Start

□ Don't scroll context menu

Cell Size: 8

Buffer Size: 1200

- **Auto Start** > automatically start recording thread samples when you launch an application
- **Don't scroll context menu** > prevent the time-line from updating while its popup menu is open

  Data samples are still recorded and are displayed when the menu is closed.

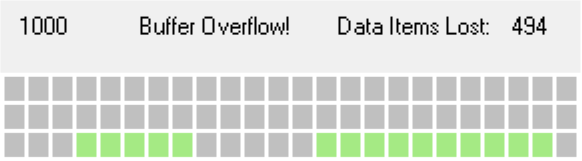
- **Cell Size** > choose the size of each cell displayed on the display

The name or id of the thread on the left hand side are also affected.

- **Buffer Size** › set the number of samples to keep track of in the time-line (default is 1000)

  The buffer operates on a first-in first-out basis, so when the buffer is full, oldest samples are discarded in favour of new ones.

  When the buffer is full an overflow warning and a count of lost samples is displayed as below:



1000 Buffer Overflow! Data Items Lost: 494

- **Reset** › reset the above values to their defaults

Thread history menu options

Most of the menu options the same information and callstack options as on the Locks Per Thread tab and several others.

In this thread history view, the menu options apply to the selected sample block in the time-line display.

Three additional options are available, to highlight related blocks within the samples displayed.

- **Highlight locked critical sections** › highlight other blocks that *locked* critical sections or waits used in the selected block
- **Highlight contended critical sections** › highlight other blocks that *contended* critical sections or waits
- **Highlight locked and contended...** › highlight other blocks that *locked or contended* critical sections or waits

The Locks tab

The Locks tab in the bottom half of the view shows information about locked or waiting critical sections for each known thread in the target application.

Only the lock information at the selected sample point in the top graph is displayed. Selecting blocks at different points along the time-line will change the data shown.

The set of displayed information is the same as the Active Locks, except that here only locks active at the selected sample point are listed.

Thread rows will be highlighted in the thread information colour (orange 🟧 by default) or pale red 🟥, the stalled thread colour unless you customise these colours.

Each relevant critical section is shown once underneath each thread and the display can be sorted by different criteria by clicking on the appropriate column header.

The table highlights error and status conditions using the user defined lock colours. In the example above, yellow 🟨 indicates a contention, and cyan 🟦 for recursion.

Lines highlighted in grey ◻ denote an active status, meaning that the critical section has been entered.

Unlike the All Locks tab, critical sections where nothing of interest has happened are not displayed.

Any `WaitForXXXObject` calls used by each thread in the application are also listed.

What data is available in the Locks tab?

The columns displayed in the Lock tab view are identical to the columns on the Locks Per Thread tab (and the Active Locks tab).

Note that like the Locks Per Thread, the thread header rows display different information to the critical section rows.

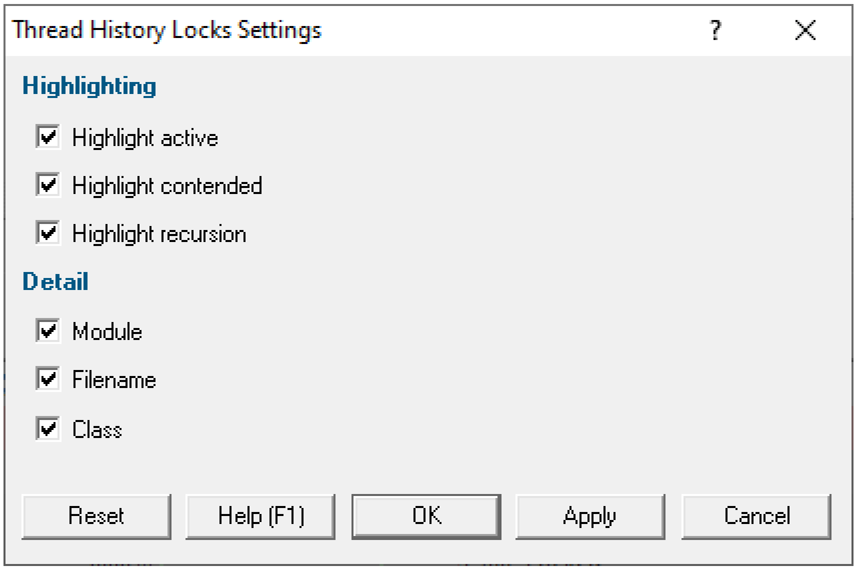
Locks tab menu options

The menu options are also the same as those on the Locks Per Thread tab.

Locks tab display settings

The thread history display settings are the same as the Locks Per Thread tab display settings.

- **Display...** > shows the Thread History Locks Settings dialog:



Thread History Locks Settings

Highlighting

☑ Highlight active

☑ Highlight contended

☑ Highlight recursion

Detail

☑ Module

☑ Filename

☑ Class

Highlighting related time-line data from the Locks tab

Selecting rows in the Locks tab of the lower view highlights related data in the sampled time-line view.

Selecting a row for a...

- Thread › highlights blocks if they use critical sections and Win32 handles used in the selected thread
- Critical section › highlights blocks if they use critical sections with the same address
- Single or multiple wait › highlights blocks if they use Win32 handles with the same handle value(s)

The Lock Order tab

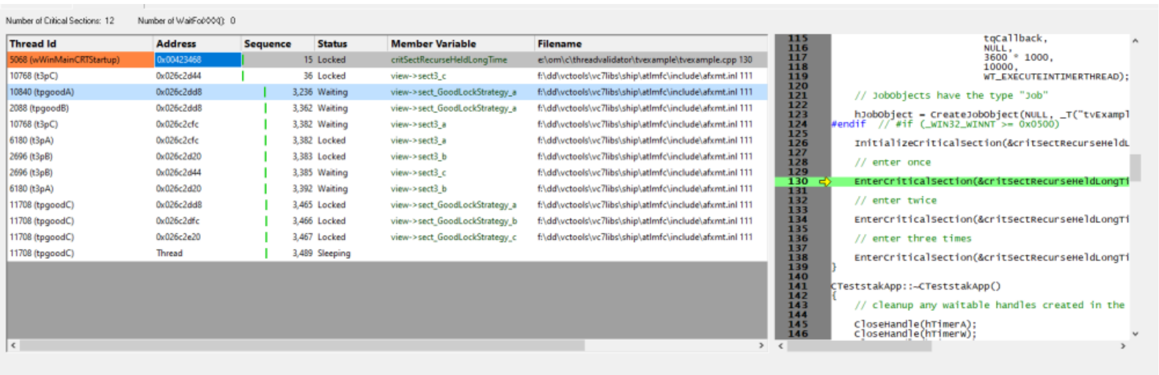
The Lock Order tab in the bottom half of the view shows locks and waits in the sample in sequence number order.

The dialog displays the order...

- in which critical sections are locked and waited upon
- that waits are entered into
- in which threads sleep and are suspended

The data and menu options are very similar to the Locks and Waits in Sequence Order dialog, except that here the entries shown are those at the selected sample point in the time-line.

Selecting an entry (lock, wait or thread) will highlight the other locks, waits and threads that are related. For example selecting a lock will highlight all locks entries that have the same address, regardless of the thread they are used in. Also any other entries related to the same thread are also highlighted. This allows you to see where you hold locks and where the same lock is waiting. The source code showing where the lock is locked or waiting is shown on the right of the display.



Number of Critical Sections: 12   Number of WaitForXXX(): 0

Thread Id	Address	Sequence	Status	Member Variable	Filename
5068 (wWinMainCRTStartup)	0x00423468	15	Locked	critSectRecurseHeldLongTime	e:\om\c\threadvalidator\tvexample\tvexample.cpp 130
10768 (t3pC)	0x026c2d44	36	Locked	view->sect3_c	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
10840 (tpgoodA)	0x026c2dd8	3,236	Waiting	view->sect_GoodLockStrategy_a	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
2088 (tpgoodB)	0x026c2dd8	3,362	Waiting	view->sect_GoodLockStrategy_a	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
10768 (t3pC)	0x026c2cfc	3,382	Waiting	view->sect3_a	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
6180 (t3pA)	0x026c2cfc	3,382	Locked	view->sect3_a	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
2696 (t3pB)	0x026c2d20	3,383	Locked	view->sect3_b	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
2696 (t3pB)	0x026c2d44	3,385	Waiting	view->sect3_c	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
6180 (t3pA)	0x026c2d20	3,392	Waiting	view->sect3_b	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
11708 (tpgoodC)	0x026c2dd8	3,465	Locked	view->sect_GoodLockStrategy_a	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
11708 (tpgoodC)	0x026c2dfc	3,466	Locked	view->sect_GoodLockStrategy_b	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
11708 (tpgoodC)	0x026c2e20	3,467	Locked	view->sect_GoodLockStrategy_c	f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl 111
11708 (tpgoodC)	Thread	3,489	Sleeping		

```
                                  tqCallback,
                                  NULL,
                                  3600 * 1000,
                                  10000,
                                  WT_EXECUTEINTIMERTHREAD);

    // JobObjects have the type "Job"

    hJobObject = CreateJobObject(NULL, _T("tvExampl
#endif  // #if (_WIN32_WINNT >= 0x0500)

    InitializeCriticalSection(&critSectRecurseHeldL

    // enter once

    EnterCriticalSection(&critSectRecurseHeldLongTi

    // enter twice

    EnterCriticalSection(&critSectRecurseHeldLongTi

    // enter three times

    EnterCriticalSection(&critSectRecurseHeldLongTi
}

CTeststakApp::~CTeststakApp()
{
    // cleanup any waitable handles created in the

    CloseHandle(hTimerA);
    CloseHandle(hTimerW);
```

3.7.5 Analysis

The Analysis tab contains four different sub tabs, each providing a different ability to query threading data that isn't necessarily about Critical Sections or Waits (you can query those as well).

Query

The ability to query to find synchronization objects by address, object type, function (class and method), source file, module and thread.

Coverage

Code coverage for synchronization primitives - are your tests hitting all the right places to test your thread synchronization?

Active Objects

All the synchronization objects and waitable handles allocated by the application, including callstack and additional data about each handle.

Object Handles

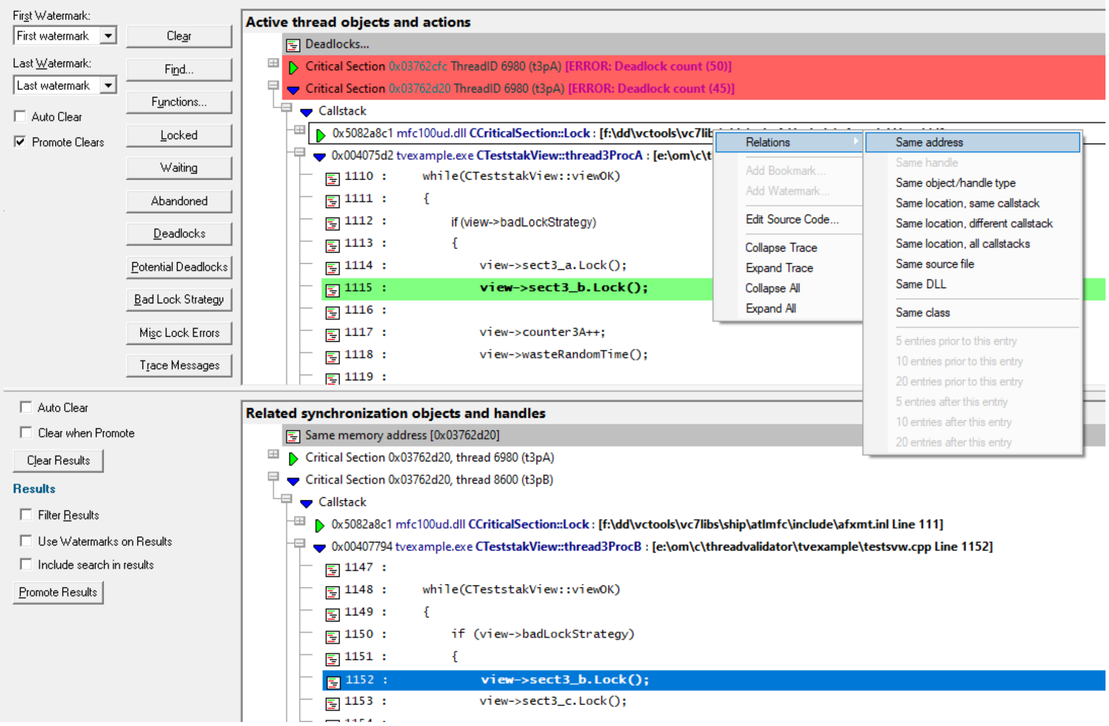
All the synchronization objects allocated by the application.

3.7.5.1 Query

The **Analysis** tab shows groups of search results and allows you to find many types of related data for different synchronization objects.

If the Analysis tab isn't visible, use the Data Views menu to set which views are shown.

Read on, or click a part of the image below to jump straight to the help for that area.



Active thread objects and actions

```
Deadlocks...
Critical Section 0x03762cfc ThreadID 6980 (t3pA) [ERROR: Deadlock count (50)]
Critical Section 0x03762d20 ThreadID 6980 (t3pA) [ERROR: Deadlock count (45)]
  Callstack
    0x5082a8c1 mfc100ud.dll CCriticalSection::Lock : [f:\dd\vctools\vc7lib
    0x004075d2 tvexample.exe CTeststakView::thread3ProcA : [e:\om\c\t
      1110 :     while(CTeststakView::viewOK)
      1111 :     {
      1112 :         if (view->badLockStrategy)
      1113 :         {
      1114 :             view->sect3_a.Lock();
      1115 :             view->sect3_b.Lock();
      1116 :
      1117 :             view->counter3A++;
      1118 :             view->wasteRandomTime();
      1119 :
```

Related synchronization objects and handles

```
Same memory address [0x03762d20]
Critical Section 0x03762d20, thread 6980 (t3pA)
Critical Section 0x03762d20, thread 8600 (t3pB)
  Callstack
    0x5082a8c1 mfc100ud.dll CCriticalSection::Lock : [f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 111]
    0x00407794 tvexample.exe CTeststakView::thread3ProcB : [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1152]
      1147 :
      1148 :     while(CTeststakView::viewOK)
      1149 :     {
      1150 :         if (view->badLockStrategy)
      1151 :         {
      1152 :             view->sect3_b.Lock();
      1153 :             view->sect3_c.Lock();
```

The Analysis Data

The Analysis tab has two parts separated by a resizing handle.

The top view is where the initial data to work with is displayed - i.e. the result of manual or predefined searches.

The lower display shows any related synchronization objects found using the popup menu options - e.g. other calls with similar addresses or callstacks.

Manual searches allow information about synchronization objects to be displayed based on:

- type (e.g. GDI handle object type)
- file
- function
- DLL
- address or handle value
- size

Predefined searches display:

- locked or waiting critical sections
- abandoned critical sections
- deadlocks or potential deadlocks
- bad lock strategies
- miscellaneous locking errors
- trace messages

The text on each line is the same and indicates:

- datatype (if known)
- address/handle value
- source file and line number (if available) for the function

To edit the source code, double click on any part of the lines of source code displayed or use **Edit Source Code...**

The upper window - working data

The upper window will contain any results of queries made via the manual or predefined query options on the left.

These results become your 'working data', and will grow (with a header line between each group) as each set of results is added.

Using the relations option on the popup menu, you can then find related allocations or objects which are displayed as separate results in the lower window.

Collected data is displayed in a tree, optionally constrained by watermarks:

- the text on the line indicates the data type (if known), its size, address/handle value, and the source file and line number (if available) where the event occurred

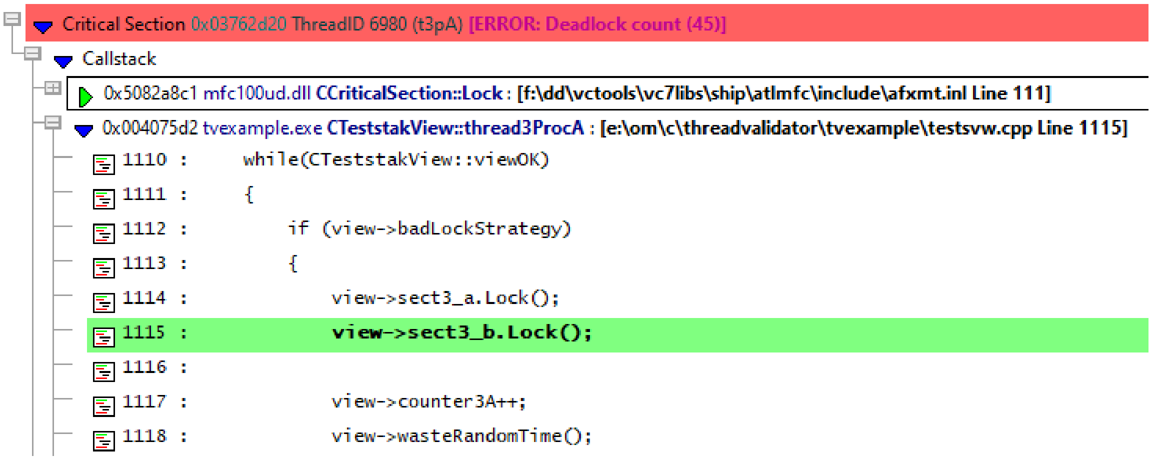
- the colour of the background for the line indicates the status of the data, such as the thread is deadlocked 🟥, potentially deadlocked 🟪, or involved in other errors such as a bad lock strategy 🟪

  See the section on predefined queries for example views

Each item can be expanded showing some information about the item, such as thread id, timestamp and the callstack for the item.

Each line of the callstack shows the instruction address, module name, undecorated C++ function name and the source file and line number (if available) for that function.

The callstack itself can be expanded (see example below) to display the source code which can also be edited.



```
Critical Section 0x03762d20 ThreadID 6980 (t3pA) [ERROR: Deadlock count (45)]
  Callstack
    0x5082a8c1 mfc100ud.dll CCriticalSection::Lock : [f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 111]
    0x004075d2 tvexample.exe CTeststakView::thread3ProcA : [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1115]
      1110 :     while(CTeststakView::viewOK)
      1111 :     {
      1112 :         if (view->badLockStrategy)
      1113 :         {
      1114 :             view->sect3_a.Lock();
      1115 :             view->sect3_b.Lock();
      1116 :
      1117 :             view->counter3A++;
      1118 :             view->wasteRandomTime();
```

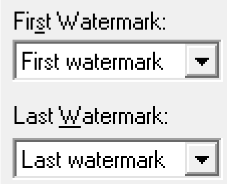
Please note that not all data for the queries is acquired in the same manner. This is because of the measure taken to reduce the amount of memory needed to store data about the locking activity of the application. As a result, **queries that you may expect to yield results may sometimes fail to do so** because the information required for the query was not collected or was collected in such a manner that it is not available for the query.

The lower window - results within results

Having obtained related data in the lower window you can inspect it in the same way, filter it and promote it back up to the top window.

Analysis tab options - upper window

At the far left of the window are the same Watermark options as on the Active Objects tab.



First Watermark: First watermark

Last Watermark: Last watermark

Choose two watermarks to restrict the displayed data to lie between them.

- **First Watermark** › Choose a watermark from the list › **Last Watermark** › Choose another watermark › **Refresh** › updates the data shown in the display

  There are two permanent default watermarks, called *First watermark* (before anything else) and *Last watermark* (after anything else).

  Attempting to choose a first watermark later than the last watermark, or vice-versa will result in the alternate watermark automatically updating.

Analysis tab options - clearing data

The working data in the upper display can be cleared manually or automatically in different ways:

- **Clear** › manually remove all working data from the upper window only (the lower window has its own clear button)
- **Auto Clear** › clears any existing working data on each new search
- **Promote Clears** › automatically clears the working data each time data from the lower window is promoted to the top view

Running data queries

From the buttons at the left you can run some common or very specific queries to search for synchronization objects, thread errors, and messages.

Custom queries

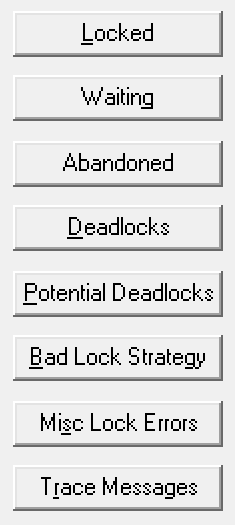
Several of the main tabs have some comprehensive methods of querying synchronization objects or functions, and which are accessible from the main query menu and query toolbar.

The Analysis tab has dedicated buttons for two of these queries:

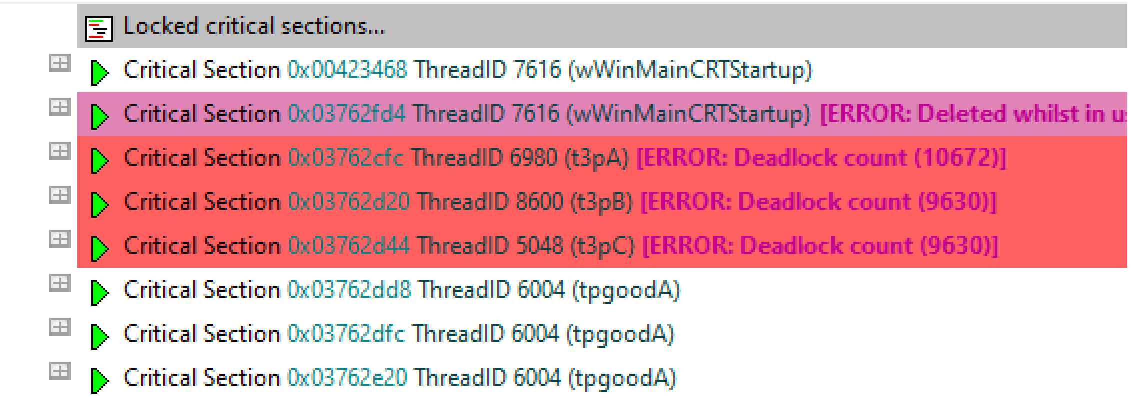
- **Find...** › Shows the Find Synchronization Objects dialog to use a variety of search criteria to search for objects and display them in the Analysis tab
- **Functions...** › Shows the Find Functions dialog but displays the results here in the analysis tab

Predefined queries

There are a selection of common predefined queries available at a single click:

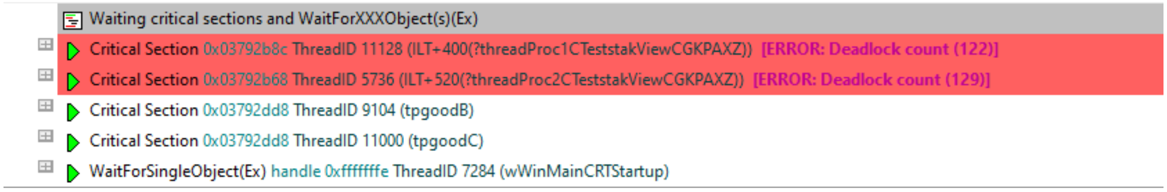


- **Locked** › finds all locked critical sections



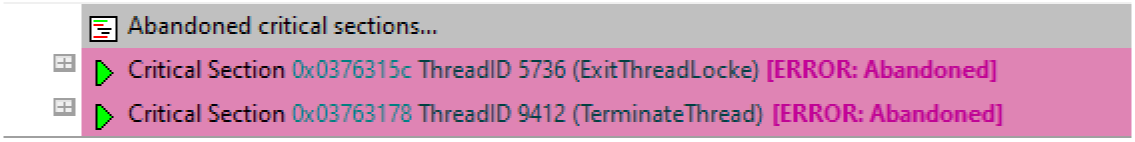
```
Locked critical sections...
Critical Section 0x00423468 ThreadID 7616 (wWinMainCRTStartup)
Critical Section 0x03762fd4 ThreadID 7616 (wWinMainCRTStartup) [ERROR: Deleted whilst in u
Critical Section 0x03762cfc ThreadID 6980 (t3pA) [ERROR: Deadlock count (10672)]
Critical Section 0x03762d20 ThreadID 8600 (t3pB) [ERROR: Deadlock count (9630)]
Critical Section 0x03762d44 ThreadID 5048 (t3pC) [ERROR: Deadlock count (9630)]
Critical Section 0x03762dd8 ThreadID 6004 (tpgoodA)
Critical Section 0x03762dfc ThreadID 6004 (tpgoodA)
Critical Section 0x03762e20 ThreadID 6004 (tpgoodA)
```

- **Waiting** › all waiting critical sections and waiting handles



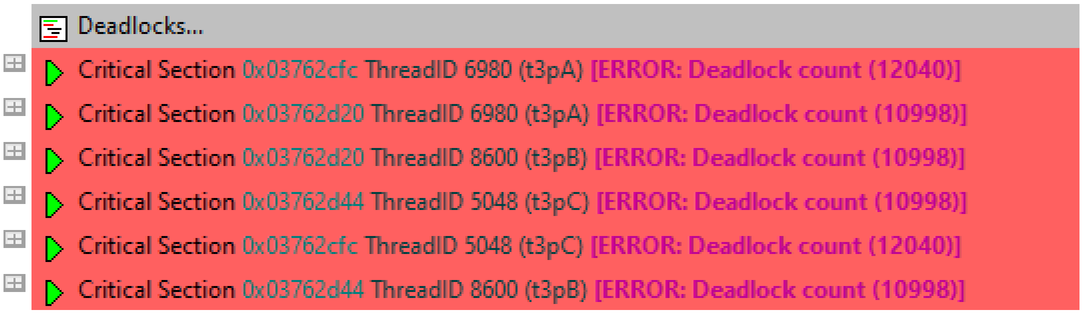
```
Waiting critical sections and WaitForXXXObject(s)(Ex)
Critical Section 0x03792b8c ThreadID 11128 (ILT+400(?threadProc1CTeststakViewCGKPAXZ)) [ERROR: Deadlock count (122)]
Critical Section 0x03792b68 ThreadID 5736 (ILT+520(?threadProc2CTeststakViewCGKPAXZ)) [ERROR: Deadlock count (129)]
Critical Section 0x03792dd8 ThreadID 9104 (tpgoodB)
Critical Section 0x03792dd8 ThreadID 11000 (tpgoodC)
WaitForSingleObject(Ex) handle 0xfffffffe ThreadID 7284 (wWinMainCRTStartup)
```

- **Abandoned** › finds abandoned critical sections (critical sections locked but not owned by any thread)



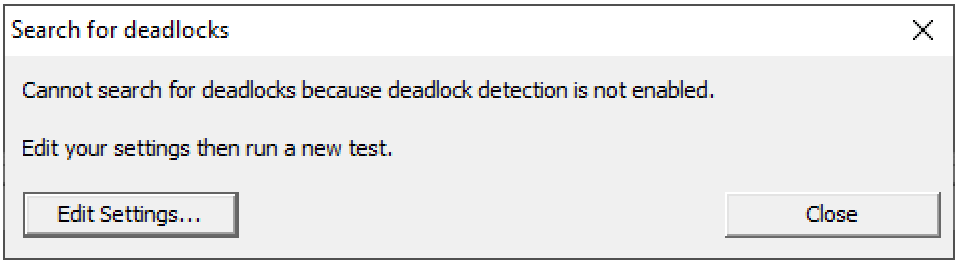
```
Abandoned critical sections...
Critical Section 0x0376315c ThreadID 5736 (ExitThreadLocke) [ERROR: Abandoned]
Critical Section 0x03763178 ThreadID 9412 (TerminateThread) [ERROR: Abandoned]
```

- **Deadlocked** › finds deadlock locations



```
Deadlocks...
Critical Section 0x03762cfc ThreadID 6980 (t3pA) [ERROR: Deadlock count (12040)]
Critical Section 0x03762d20 ThreadID 6980 (t3pA) [ERROR: Deadlock count (10998)]
Critical Section 0x03762d20 ThreadID 8600 (t3pB) [ERROR: Deadlock count (10998)]
Critical Section 0x03762d44 ThreadID 5048 (t3pC) [ERROR: Deadlock count (10998)]
Critical Section 0x03762cfc ThreadID 5048 (t3pC) [ERROR: Deadlock count (12040)]
Critical Section 0x03762d44 ThreadID 8600 (t3pB) [ERROR: Deadlock count (10998)]
```

You'll need to enable the detection of deadlocks otherwise you'll just see this warning, with the option to edit the settings.

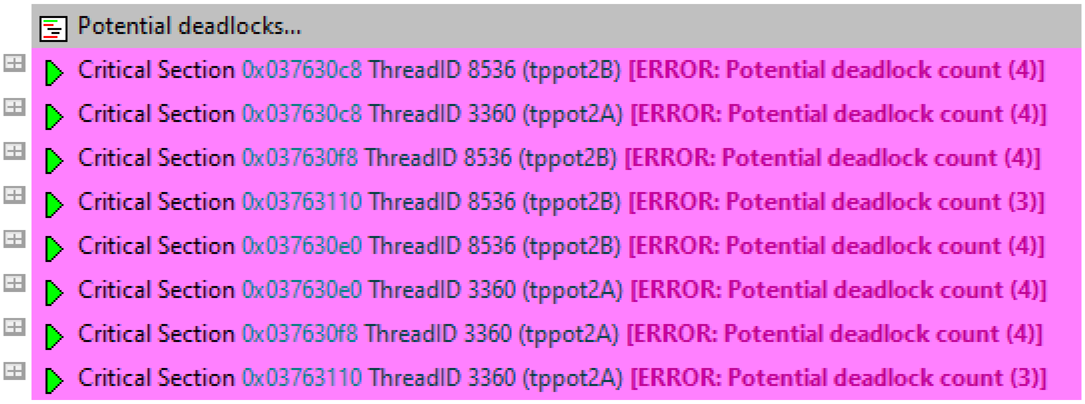


Search for deadlocks

Cannot search for deadlocks because deadlock detection is not enabled.

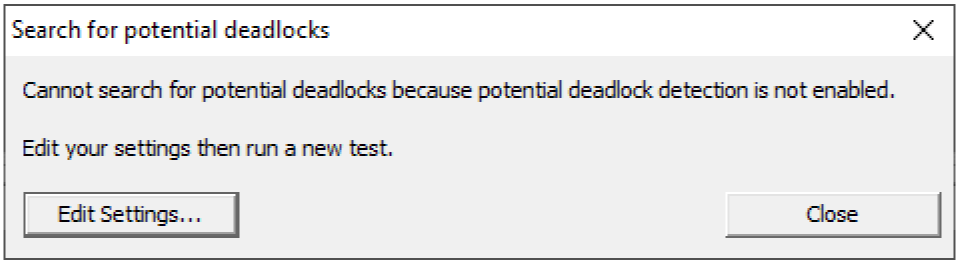
Edit your settings then run a new test.

- **Potential Deadlocked** › finds locations for potential deadlocks



```
Potential deadlocks...
Critical Section 0x037630c8 ThreadID 8536 (tppot2B) [ERROR: Potential deadlock count (4)]
Critical Section 0x037630c8 ThreadID 3360 (tppot2A) [ERROR: Potential deadlock count (4)]
Critical Section 0x037630f8 ThreadID 8536 (tppot2B) [ERROR: Potential deadlock count (4)]
Critical Section 0x03763110 ThreadID 8536 (tppot2B) [ERROR: Potential deadlock count (3)]
Critical Section 0x037630e0 ThreadID 8536 (tppot2B) [ERROR: Potential deadlock count (4)]
Critical Section 0x037630e0 ThreadID 3360 (tppot2A) [ERROR: Potential deadlock count (4)]
Critical Section 0x037630f8 ThreadID 3360 (tppot2A) [ERROR: Potential deadlock count (4)]
Critical Section 0x03763110 ThreadID 3360 (tppot2A) [ERROR: Potential deadlock count (3)]
```

You'll need to enable the detection of potential deadlocks otherwise you'll just see this warning, with the option to edit the settings.

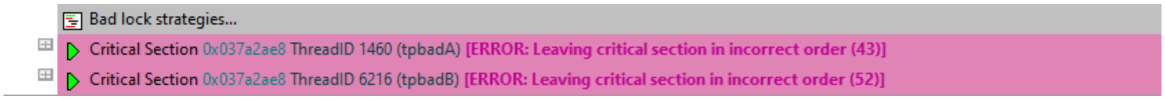


Search for potential deadlocks

Cannot search for potential deadlocks because potential deadlock detection is not enabled.

Edit your settings then run a new test.

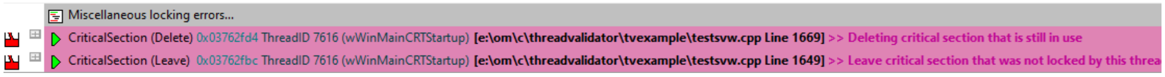
- **Bad Lock Strategy** › finds all critical sections involved in bad lock strategies



```
Bad lock strategies...
Critical Section 0x037a2ae8 ThreadID 1460 (tpbadA) [ERROR: Leaving critical section in incorrect order (43)]
Critical Section 0x037a2ae8 ThreadID 6216 (tpbadB) [ERROR: Leaving critical section in incorrect order (52)]
```

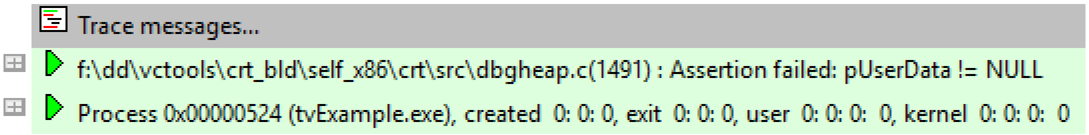
To ensure this reports the maximum number of errors you'll need to enable the detection out of order critical sections and other locking errors.

- **Misc Lock Errors** › finds a variety of other synchronization errors not covered by the other predefined searches



```
Miscellaneous locking errors...
CriticalSection (Delete) 0x03762fd4 ThreadID 7616 (wWinMainCRTStartup) [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1669] >> Deleting critical section that is still in use
CriticalSection (Leave)  0x03762fbc ThreadID 7616 (wWinMainCRTStartup) [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1649] >> Leave critical section that was not locked by this threa
```

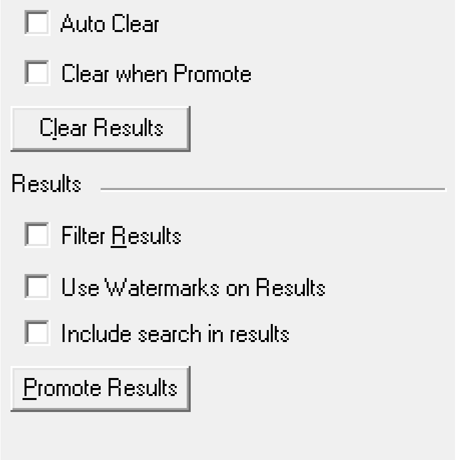
- **Trace Messages** › shows any trace messages if you've switched the trace hook setting on



```
Trace messages...
f:\dd\vctools\crt_bld\self_x86\crt\src\dbgheap.c(1491) : Assertion failed: pUserData != NULL
Process 0x00000524 (tvExample.exe), created  0: 0: 0, exit  0: 0: 0, user  0: 0: 0:  0, kernel  0: 0: 0:  0
```

Analysis tab options - lower window

The lower window has its own comparatively simple set of options:



□ Auto Clear

□ Clear when Promote

Results

□ Filter Results

□ Use Watermarks on Results

□ Include search in results

- **Auto Clear** › clears the lower window before adding a new relations search from the upper window
- **Clear when Promote** › clears the lower window when promoting these results to the upper window
- **Clear Results** › simply empties the lower window
- **Filter Results** › optionally filters the lower window data using the same filters as the upper window

- **Use Watermarks on Results** › applies the upper window's watermark settings to the lower window

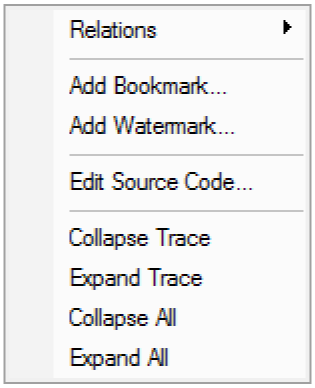
- **Include search in results** › brings the upper window item that initiated the relations search along with the results

- **Promote Results** › pushes all or selected lower window results into the upper window.

  This optionally adds to or replaces what's in the upper window already, depending on the state of the Promote Clears option in the upper window.

Analysis view popup menu

The following popup menu is available over the upper window.



Menu option: relations

The relations menu has a sub-menu with many different options for choosing a set of related data to display in the lower analysis window.

Think of this as a sub-query on the working data - like searching for friends of friends on a social network!

Given an entry in the upper window, available relations are as follows:

• **Same address**	Finds any other critical sections using the same memory address
• **Same handle** • **Same object/handle type**	Finds any other WaitForSingleObject using the same resource handle Critical sections and wait calls that have the same object type or handle type.
• **Same location, same callstack**	Finds other critical sections that have the same usage location and:

• **Same address**	Finds any other critical sections using the same memory address
different callstack **all callstacks**	using the same callstack using different callstacks regardless of callstack
• **Same source file** • **Same DLL**	All critical sections in the same source file... or the same DLL
• **Same class**	All critical sections from the same C++ class
• **Entries prior** • **Entries after**	For critical section events, finds the previous 5, 10 or 20 events or the next 5, 10 or 20 events

Menu option: bookmarks and watermarks

Bookmarks allow you to find a data item easily at a later date, while watermarks are used above to show only those items between two points in time

- **Add Bookmark...** › adds a bookmark for the selected item
- **Add Watermark...** › adds a watermark for the selected item

Menu option: editing source code

- **Edit Source Code...** › opens the default or preferred editor to edit the source code

Menu options: collapse / expand trace

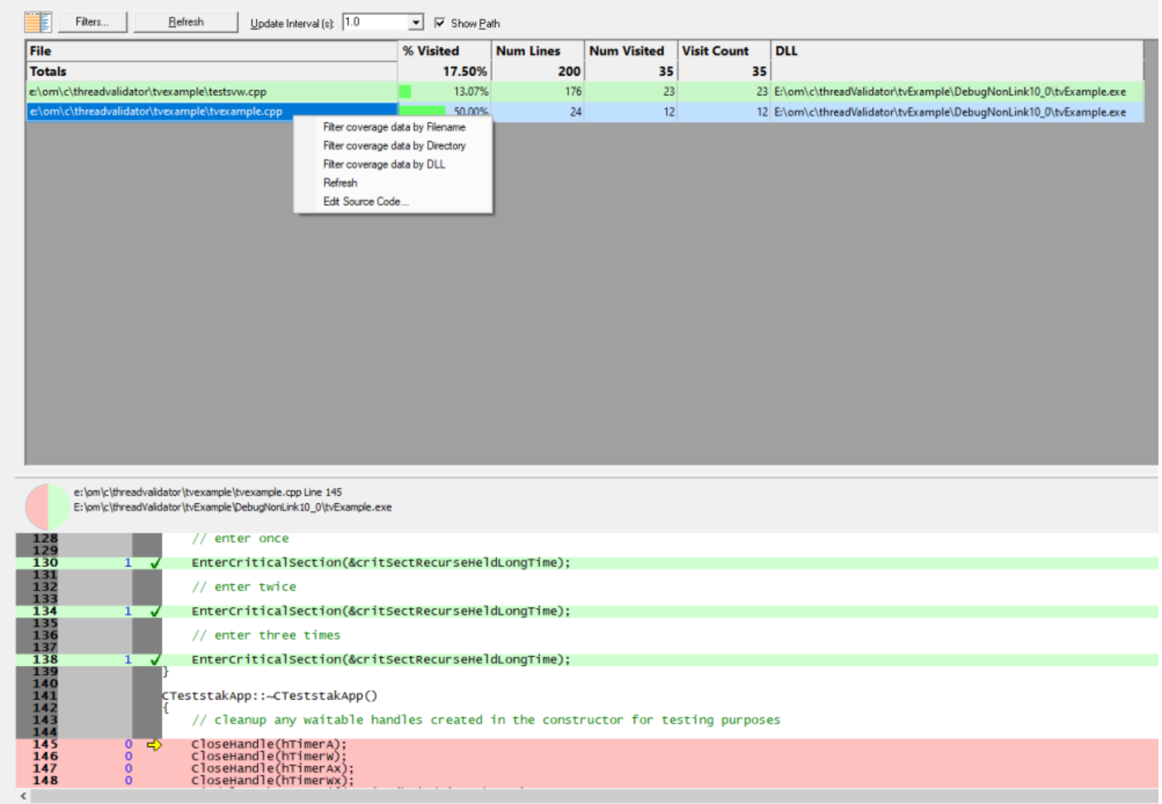
- **Collapse** or **Expand Trace** › simply shows and hides data item information, the same as using the ⊞ or ⊟ buttons
- **Collapse All** › completely collapses all data items in the upper window, including any source code views that were open
- **Expand All** › expands all data items down to but not including the source code snapshots

3.7.5.2 Coverage

The **Coverage** tab lets you find untested parts of your program's thread synchronization code.

If the Coverage tab isn't visible, use the Data Views menu to set which views are shown.

Read on, or click a part of the image below to jump straight to the help for that area.


Filters...
Refresh
Update Interval (s): 1.0
☑ Show Path
Filter coverage data by Filename
Filter coverage data by Directory
Filter coverage data by DLL
Refresh
Edit Source Code...

File	% Visited	Num Lines	Num Visited	Visit Count	DLL
Totals	17.50%	200	35	35	
e:\om\c\threadvalidator\tvexample\testsvw.cpp	13.07%	176	23	23	E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe
e:\om\c\threadvalidator\tvexample\tvexample.cpp	50.00%	24	12	12	E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe

e:\om\c\threadvalidator\tvexample\tvexample.cpp Line 145
E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe

```
128           // enter once
129
130   1  ✓    EnterCriticalSection(&critSectRecurseHeldLongTime);
131
132           // enter twice
133
134   1  ✓    EnterCriticalSection(&critSectRecurseHeldLongTime);
135
136           // enter three times
137
138   1  ✓    EnterCriticalSection(&critSectRecurseHeldLongTime);
139       }
140
141       CTeststakApp::~CTeststakApp()
142       {
143           // cleanup any waitable handles created in the constructor for testing purposes
144
145   0  ⇨    CloseHandle(hTimerA);
146   0       CloseHandle(hTimerW);
147   0       CloseHandle(hTimerAx);
148   0       CloseHandle(hTimerWx);
```

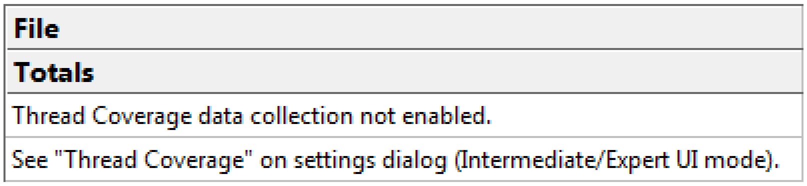
Using the coverage information

Inspecting the coverage tells you which thread synchronization locations are or are not being tested.

Understanding the thread coverage helps you plan and improve your regression tests to include areas of code that perform thread synchronization but are not yet being visited.

The display shows two resizable panes, one with the coverage data, and the other shows source code when you click on a row in the table.

In order to gather coverage statistics, you'll need to switch on the thread coverage setting, otherwise you'll just see the reminders below:



File
Totals
Thread Coverage data collection not enabled.
See "Thread Coverage" on settings dialog (Intermediate/Expert UI mode).

Colours used in the displays

Each file's row is coloured according to whether it has:

🟥 no lines visited

🟨 some lines visited

🟦 all lines visited

⬜ been filtered out for subsequent sessions (see below)

The % Visited column and the source code view uses the following colours:

🟩 for the percentage of lines visited, or visited lines of source code

🟦 for unvisited lines

Coverage data

The data in each column is summarised below

- **File** the statistics are gathered for each source code file found
- **Num Lines** number of lines in each file that allocate, reallocate or free memory and handles
- **Num Visited** the number of those lines that have been visited
- **Visit Count** total number of visits to those lines
- **% Visited** the percentage of relevant lines visited (Num Visited / Num Lines)
- **DLL** the DLL responsible for the file

The Visit Count may be equal to the Num Visited if you have opted to keep the default thread coverage setting of counting visits to each allocation only once. You can change this setting on the fly to start counting multiple visits right away.

At the top of the table is a **Totals** line showing combined results for all files.

The statistics here only cover lines that affect *thread synchronization*, unlike SoftwareVerify's sister tool Coverage Validator which determines complete code coverage.

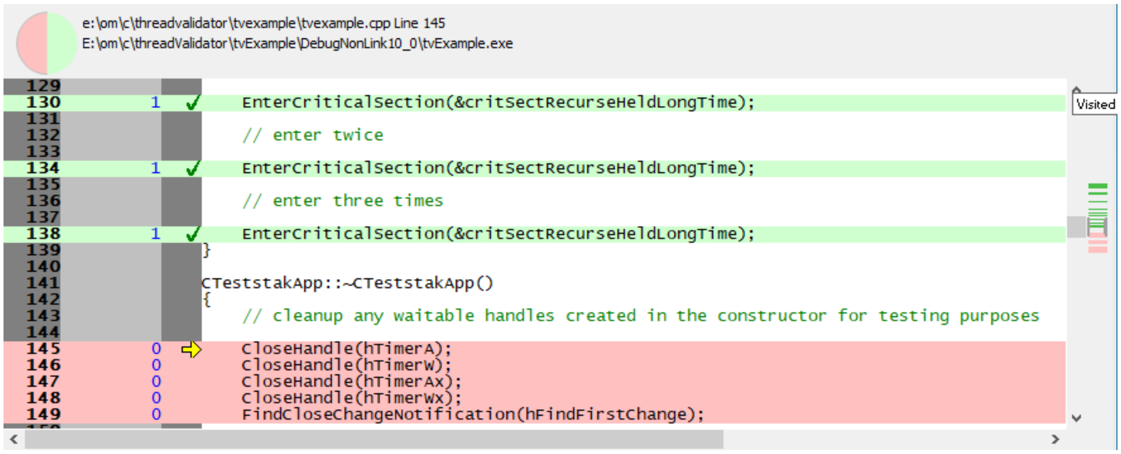
Sorting columns

Sorted columns are highlighted yellow in the table header. Click on the header to change the sorting column or its sort direction order.

Source code view

The source code view is syntax-highlighted with green visited and pale red unvisited lines.

The columns at the left show line numbers (in black) and visit counts (blue) for each thread synchronization line. Visit counts are also available via a tooltip.

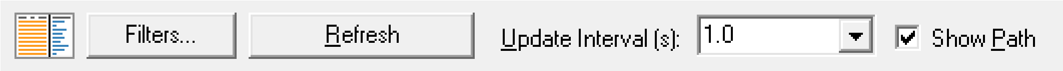


e:\om\c\threadvalidator\tvexample\tvexample.cpp Line 145
E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe

```
1 ✓    EnterCriticalSection(&critSectRecurseHeldLongTime);

       // enter twice

1 ✓    EnterCriticalSection(&critSectRecurseHeldLongTime);

       // enter three times

1 ✓    EnterCriticalSection(&critSectRecurseHeldLongTime);
     }

     CTeststakApp::~CTeststakApp()
     {
       // cleanup any waitable handles created in the constructor for testing purposes
0      CloseHandle(hTimerA);
0      CloseHandle(hTimerW);
0      CloseHandle(hTimerAx);
0      CloseHandle(hTimerWx);
0      FindCloseChangeNotification(hFindFirstChange);
```

Coverage options

The following controls are displayed to the left of the coverage results



Update Interval (s): 1.0

☑ Show Path

Filters

Unwanted results in the coverage can be excluded via the filters according to filename, directory or DLL.

Filters can be managed in the filters dialog or added via the menu options below.

- **Filters...** › shows the Thread Coverage Filters dialog

  The filters dialog is the same one as found on the thread coverage page of the global settings dialog where it is described in detail.

Window orientation

The data and source code panes can be arranged horizontally or vertically with the orientation button.

Updating the display

- **Update Interval (s)** › automatically updates the display at your choice of interval between 0.1 and 60 seconds - or not at all!

  Adjust this depending on the complexity of your application.

- **Refresh** › updates the display - as does the button on the Tools menu and toolbar, and the button on the popup menu

    With an update interval set to No Update, you'll need to use this Refresh button to update the display when you wish.

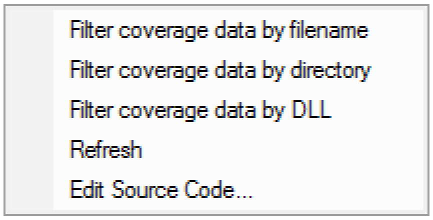
Display settings

On the Coverage tab, there's only the one display option:

- **Show Path** › shows the short file name or the longer file path in the **File** column of the data

Coverage view popup menu

The following popup menu is available over the data area to add filters, refresh the view or edit source code.


Filter coverage data by filename
Filter coverage data by directory
Filter coverage data by DLL
Refresh
Edit Source Code...

Menu options: Filtering coverage

The first three menu options let you add filters at different levels of granularity.

Filters become effective at the start of the next session. Adding a filter during a session will show the row in grey so that you can see which files are filtered, but the coverage results will continue to be included for the rest of the session.

- **Filter coverage data by filename** › adds a new filter to the Filters dialog, excluding the selected file from the results of a subsequent session
- **Filter coverage data by directory** › excludes all files in the same directory as the selected file
- **Filter coverage data by DLL** › excludes all files belonging to the same DLL as the selected file

Menu option: Refresh

- **Refresh...** › updates the statistics displayed in the table

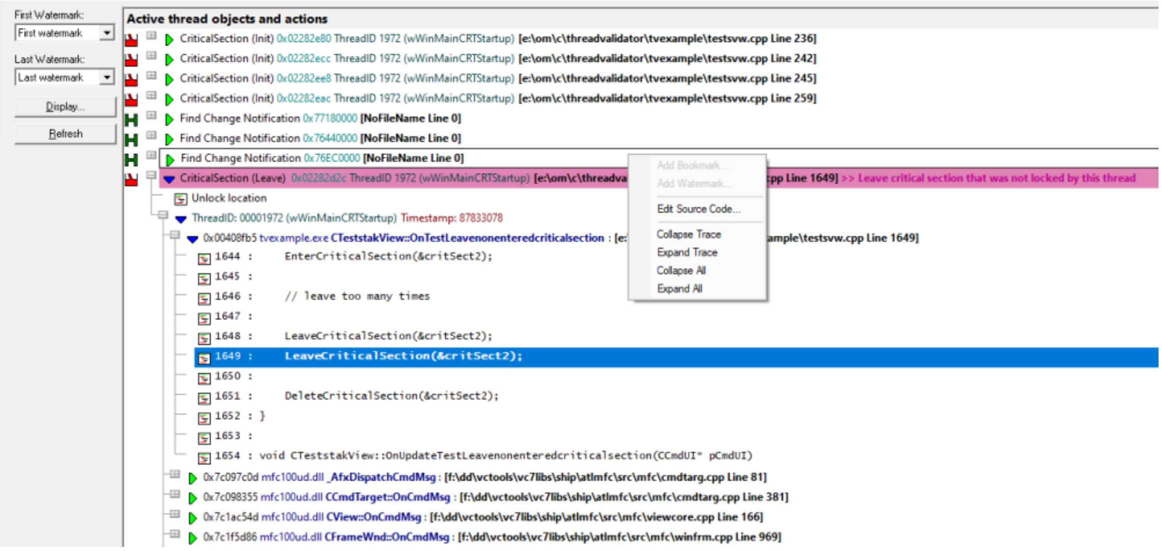
Menu option: Editing source code

- **Edit Source Code...** ❯ opens the default or preferred editor to edit the source code

3.7.5.3 Active Objects

The **Active Objects** tab allows you to view and examine synchronization objects that are currently active.

Read on, or click a part of the image below to jump straight to the help for that area.



```
Active thread objects and actions
CriticalSection (Init) 0x02282e80 ThreadID 1972 (wWinMainCRTStartup) [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 236]
CriticalSection (Init) 0x02282ecc ThreadID 1972 (wWinMainCRTStartup) [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 242]
CriticalSection (Init) 0x02282ee8 ThreadID 1972 (wWinMainCRTStartup) [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 245]
CriticalSection (Init) 0x02282eac ThreadID 1972 (wWinMainCRTStartup) [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 259]
Find Change Notification 0x77180000 [NoFileName Line 0]
Find Change Notification 0x76440000 [NoFileName Line 0]
Find Change Notification 0x76EC0000 [NoFileName Line 0]
CriticalSection (Leave) 0x02282d2c ThreadID 1972 (wWinMainCRTStartup) [e:\om\c\threadva              cpp Line 1649] >> Leave critical section that was not locked by this thread
    Unlock location
    ThreadID: 00001972 (wWinMainCRTStartup) Timestamp: 87833078
      0x00408fb5 tvexample.exe CTeststakView::OnTestLeavenonenteredcriticalsection : [e:              ample\testsvw.cpp Line 1649]
        1644 :     EnterCriticalSection(&critSect2);
        1645 :
        1646 :     // leave too many times
        1647 :
        1648 :     LeaveCriticalSection(&critSect2);
        1649 :     LeaveCriticalSection(&critSect2);
        1650 :
        1651 :     DeleteCriticalSection(&critSect2);
        1652 : }
        1653 :
        1654 : void CTeststakView::OnUpdateTestLeavenonenteredcriticalsection(CCmdUI* pCmdUI)
      0x7c097c0d mfc100ud.dll _AfxDispatchCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 81]
      0x7c098355 mfc100ud.dll CCmdTarget::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 381]
      0x7c1ac54d mfc100ud.dll CView::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\viewcore.cpp Line 166]
      0x7c1f5d86 mfc100ud.dll CFrameWnd::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 969]
```

The Active Objects View

The view shows information about all active synchronization objects.

Each object is listed with its callstack to enable you to drill down to the source code for the call.

When Thread Validator identifies miscellaneous error conditions, data about the conditions and callstack is displayed in this window.

That data can be filtered and restricted by watermarks to fine tune the displayed data.

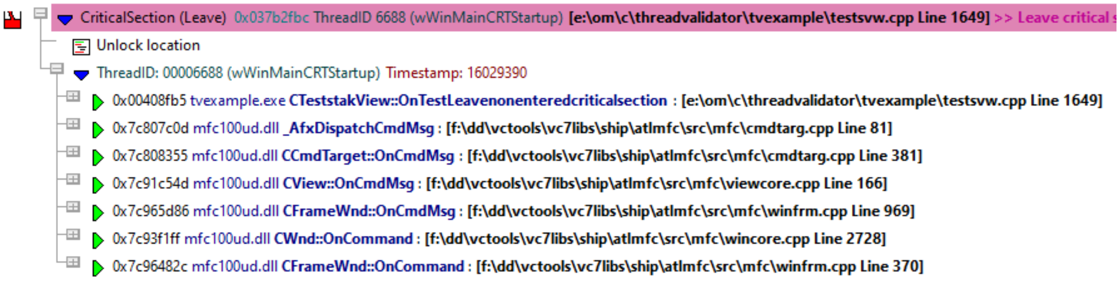
Examining a data item

Each item can be expanded with the ⊞ button (and then collapsed with the ⊟) to show more detailed information:

- thread id and the name, if assigned
- timestamp
- the callstack for the item

Each line of the callstack shows:

- instruction address
- module name
- undecorated C++ function name
- source file and line number (if available) for the function



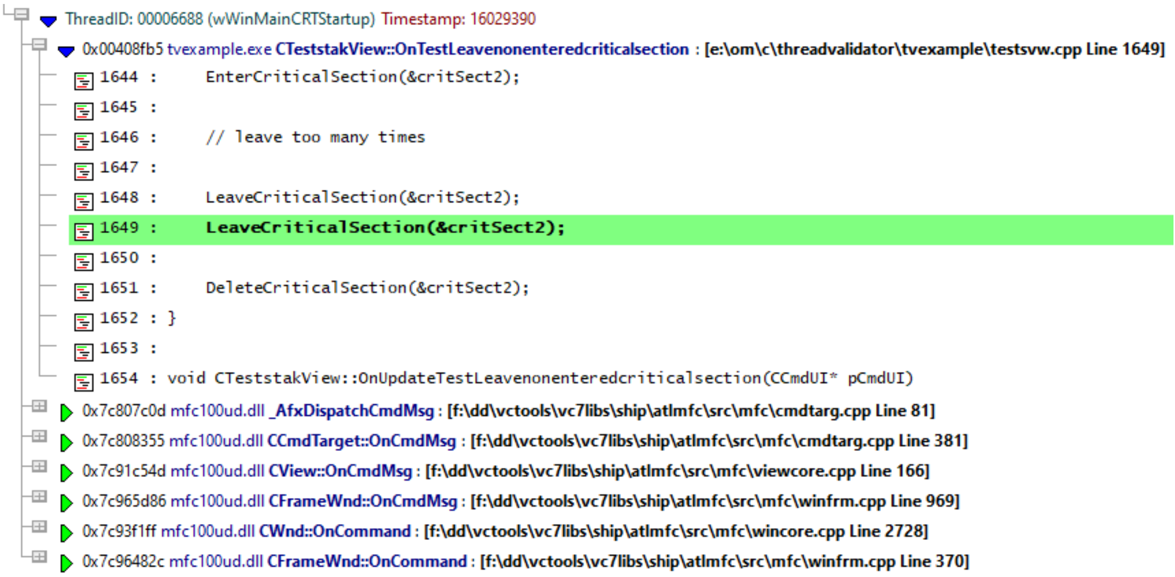
```
CriticalSection (Leave) 0x037b2fbc ThreadID 6688 (wWinMainCRTStartup) [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1649] >> Leave critical
  Unlock location
  ThreadID: 00006688 (wWinMainCRTStartup) Timestamp: 16029390
    0x00408fb5 tvexample.exe CTeststakView::OnTestLeavenonenteredcriticalsection : [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1649]
    0x7c807c0d mfc100ud.dll _AfxDispatchCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 81]
    0x7c808355 mfc100ud.dll CCmdTarget::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 381]
    0x7c91c54d mfc100ud.dll CView::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\viewcore.cpp Line 166]
    0x7c965d86 mfc100ud.dll CFrameWnd::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 969]
    0x7c93f1ff mfc100ud.dll CWnd::OnCommand : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\wincore.cpp Line 2728]
    0x7c96482c mfc100ud.dll CFrameWnd::OnCommand : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 370]
```

Examining the callstack and code

One or more parts of the callstack can be expanded or collapsed using the ⊞ or ⊟ to show the source code around the relevant line in the associated file.

If the source code can't be found, or the file location is invalid you'll be prompted for the file.

The line on which the allocation occurred is highlighted, e.g. green in this example:

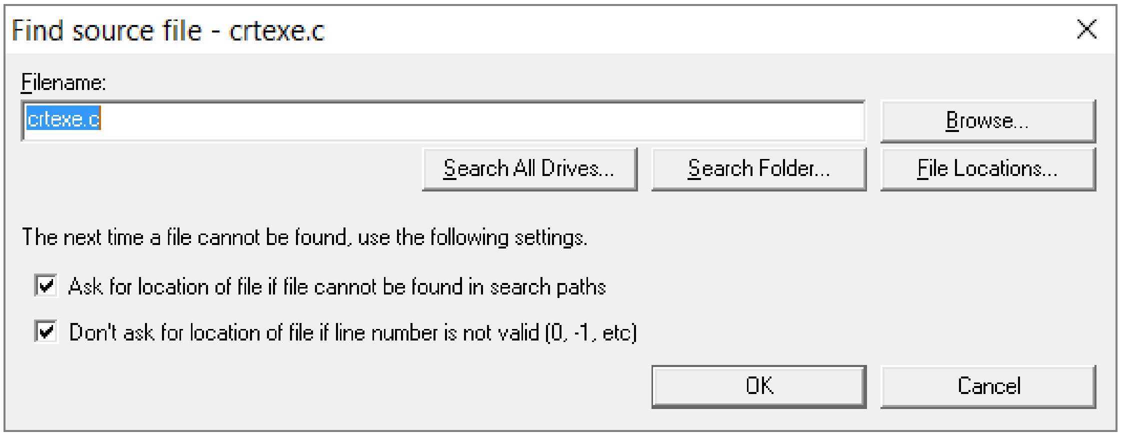


```
ThreadID: 00006688 (wWinMainCRTStartup) Timestamp: 16029390
  0x00408fb5 tvexample.exe CTeststakView::OnTestLeavenonenteredcriticalsection : [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1649]
    1644 :     EnterCriticalSection(&critSect2);
    1645 :
    1646 :     // leave too many times
    1647 :
    1648 :     LeaveCriticalSection(&critSect2);
    1649 :     LeaveCriticalSection(&critSect2);
    1650 :
    1651 :     DeleteCriticalSection(&critSect2);
    1652 : }
    1653 :
    1654 : void CTeststakView::OnUpdateTestLeavenonenteredcriticalsection(CCmdUI* pCmdUI)
  0x7c807c0d mfc100ud.dll _AfxDispatchCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 81]
  0x7c808355 mfc100ud.dll CCmdTarget::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 381]
  0x7c91c54d mfc100ud.dll CView::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\viewcore.cpp Line 166]
  0x7c965d86 mfc100ud.dll CFrameWnd::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 969]
  0x7c93f1ff mfc100ud.dll CWnd::OnCommand : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\wincore.cpp Line 2728]
  0x7c96482c mfc100ud.dll CFrameWnd::OnCommand : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 370]
```

To edit the source code, double click on any part of the lines of source code displayed or use **Edit Source Code...**

Source file not found automatically?

If the source file isn't found automatically, you'll be prompted to provide the location manually with the Find Source File dialog



Find source file - crtexe.c

Filename: crtexe.c

The next time a file cannot be found, use the following settings.

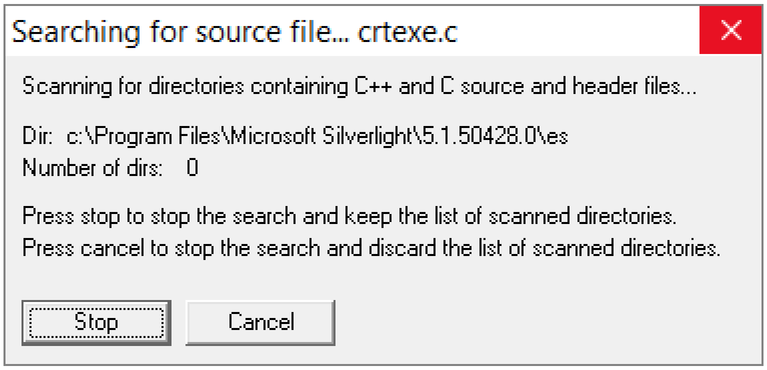
☑ Ask for location of file if file cannot be found in search paths

☑ Don't ask for location of file if line number is not valid (0, -1, etc)

You can scan, search or browse for the source location depending on how much of an idea you have of the location:

- **Browse...** › uses an explorer to search manually
- **Search All Drives...** › does a full scan of your computer, showing the Searching For Source Files dialog

You can stop the search at any time



Searching for source file... crtexe.c

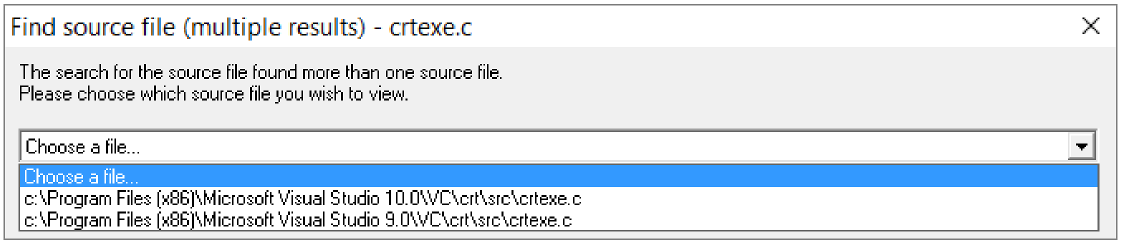
Scanning for directories containing C++ and C source and header files...

Dir: c:\Program Files\Microsoft Silverlight\5.1.50428.0\es
Number of dirs: 0

Press stop to stop the search and keep the list of scanned directories.
Press cancel to stop the search and discard the list of scanned directories.

If a file is found, the filename is entered at the top of the Find Source File dialog.

If multiple results are found, pick the best one from the results dialog that appears:



Find source file (multiple results) - crtexe.c

The search for the source file found more than one source file.
Please choose which source file you wish to view.

Choose a file...

Choose a file...
c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\crt\src\crtexe.c
c:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\crt\src\crtexe.c

- **Search Folder...** › prompts for a folder, and scans that using the same Searching For Source Files dialog as above

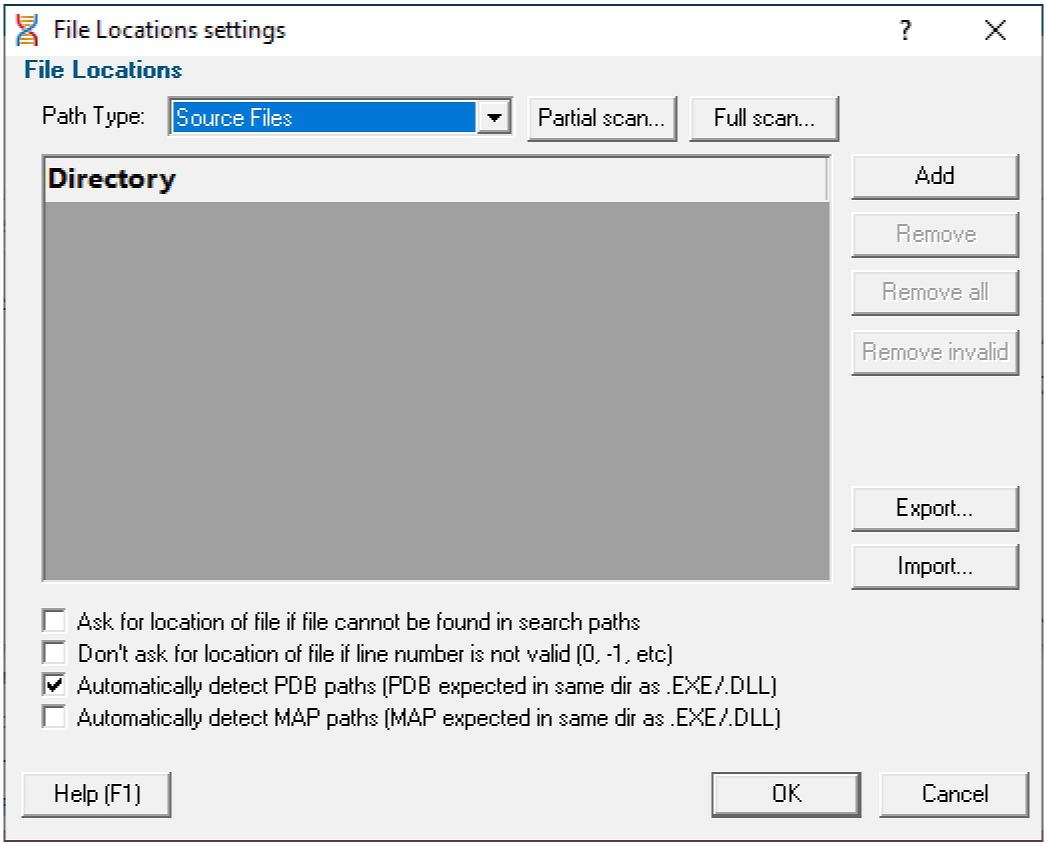
  If multiple results are found, pick the best one from the results dialog (above)

Rather than repeatedly searching manually for locations, it's recommended to modify the automatic source file search paths:

- **File Locations...** › shows the File Locations Settings dialog so you can change the automatic search paths

  Changing the search paths to include additional source locations means you'll get prompted less.

  The file locations settings dialog is identical to the File Locations page of the global settings dialog.



Path Type: Source Files

□ Ask for location of file if file cannot be found in search paths
□ Don't ask for location of file if line number is not valid (0, -1, etc)
☑ Automatically detect PDB paths (PDB expected in same dir as .EXE/.DLL)
□ Automatically detect MAP paths (MAP expected in same dir as .EXE/.DLL)

If you don't want to be prompted with this dialog, then uncheck the first option below

- **Ask for location of file if file cannot be found in search paths** › shows this dialog each time you try to open a source file where the location is unknown

- **Don't ask for location of file if line number is not valid** › stops this dialog from showing when line numbers are invalid, e.g. zero or negative

  The default is *not* to ask in this case.

Watermarks

The amount of data in the main display can be reduced by watermarks.

Here you can choose two watermarks allowing only the data between them to be displayed.

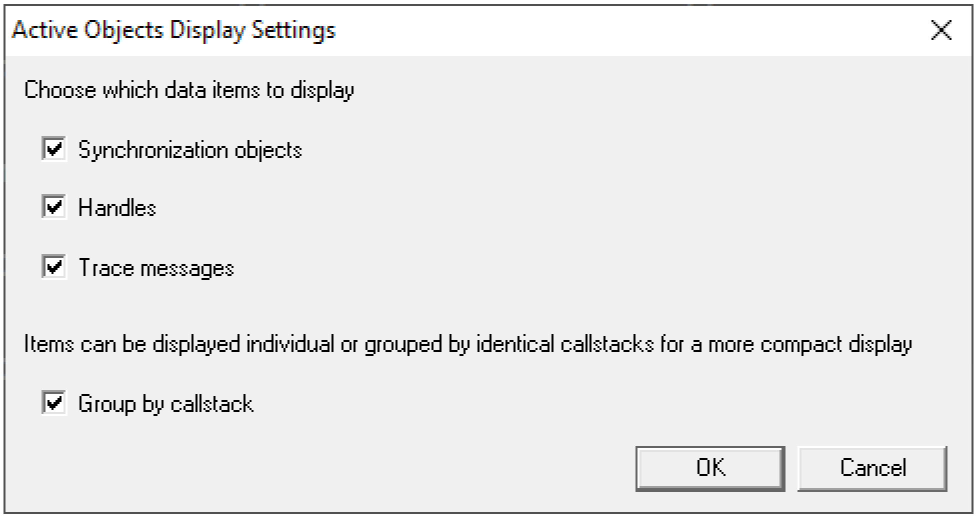
- **First Watermark** › Choose a watermark from the list › **Last Watermark** › Choose another watermark › **Refresh** › updates the data shown in the display

  There are two permanent default watermarks, called *First watermark* (before anything else) and *Last watermark* (after anything else).

  Attempting to choose a first watermark later than the last watermark, or vice-versa will result in the alternate watermark automatically updating.

Updating the display

- **Display...** › show the Active Objects Display Settings dialog



Active Objects Display Settings

Choose which data items to display

☑ Synchronization objects

☑ Handles

☑ Trace messages

Items can be displayed individual or grouped by identical callstacks for a more compact display

☑ Group by callstack

Choose the types of data to display and whether the data should be grouped or displayed individually.

- **Refresh** › refresh the list manually when you need to

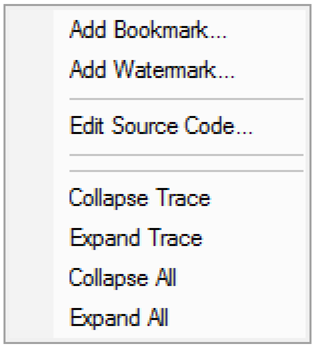
- **Group by callstack** › groups information by related callstack (default)

  The number of items in an ungrouped table can get very long when there are many items with the same callstack.

Active objects menu options

The following popup menu is available over the data area to add bookmarks, watermarks and edit source code.

Menu actions apply to the function for the row at the menu-click location.



Menu option: bookmarks and watermarks

Bookmarks allow you to find a data item easily at a later date, while watermarks are used above to show only those items between two points in time.

- **Add Bookmark...** › adds a bookmark for the selected item
- **Add Watermark...** › adds a watermark for the selected item

Menu option: editing source code

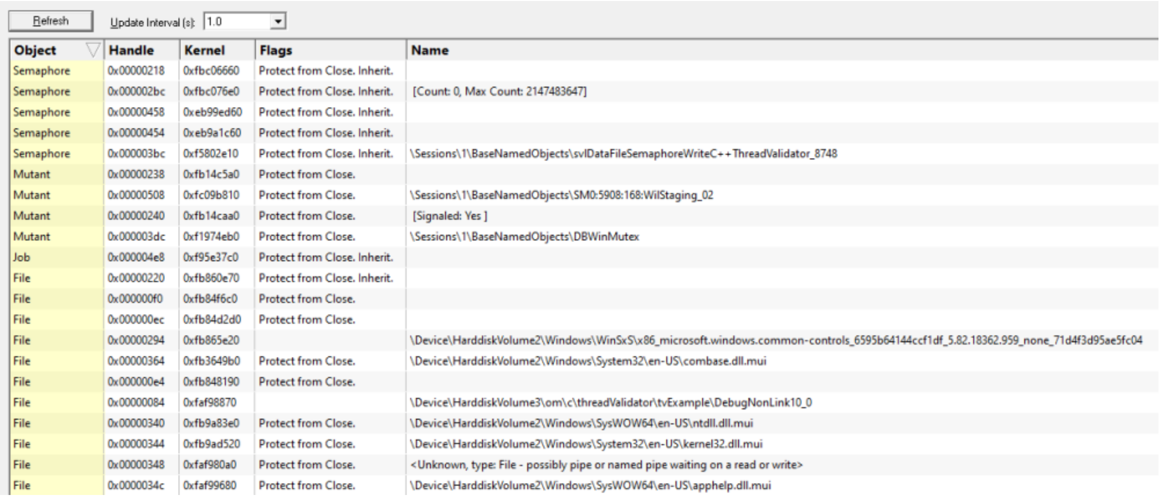
- **Edit Source Code...** › opens the default or preferred editor to edit the source code

Menu options: collapse / expand trace

- **Collapse** or **Expand Trace** › simply shows and hides data item information, the same as using the ⊞ or ⊟ buttons
- **Collapse** or **Expand All** › hides or shows all the available callstack information for every listed item

3.7.5.4 Object Handles

The **Objects** view displays a list of all the waitable handles the target program is using (except those used by Thread Validator).



Update Interval (s): 1.0

Object	Handle	Kernel	Flags	Name
Semaphore	0x00000218	0xfbc06660	Protect from Close. Inherit.	
Semaphore	0x000002bc	0xfbc076e0	Protect from Close. Inherit.	[Count: 0, Max Count: 2147483647]
Semaphore	0x00000458	0xeb99ed60	Protect from Close. Inherit.	
Semaphore	0x00000454	0xeb9a1c60	Protect from Close. Inherit.	
Semaphore	0x000003bc	0xf5802e10	Protect from Close. Inherit.	\Sessions\1\BaseNamedObjects\svlDataFileSemaphoreWriteC++ThreadValidator_8748
Mutant	0x00000238	0xfb14c5a0	Protect from Close.	
Mutant	0x00000508	0xfc09b810	Protect from Close.	\Sessions\1\BaseNamedObjects\SM0:5908:168:WilStaging_02
Mutant	0x00000240	0xfb14caa0	Protect from Close.	[Signaled: Yes]
Mutant	0x000003dc	0xf1974eb0	Protect from Close.	\Sessions\1\BaseNamedObjects\DBWinMutex
Job	0x000004e8	0xf95e37c0	Protect from Close. Inherit.	
File	0x00000220	0xfb860e70	Protect from Close. Inherit.	
File	0x000000f0	0xfb84f6c0	Protect from Close.	
File	0x000000ec	0xfb84d2d0	Protect from Close.	
File	0x00000294	0xfb865e20		\Device\HarddiskVolume2\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_5.82.18362.959_none_71d4f3d95ae5fc04
File	0x00000364	0xfb3649b0	Protect from Close.	\Device\HarddiskVolume2\Windows\System32\en-US\combase.dll.mui
File	0x000000e4	0xfb848190	Protect from Close.	
File	0x00000084	0xfaf98870		\Device\HarddiskVolume3\om\c\threadValidator\tvExample\DebugNonLink10_0
File	0x00000340	0xfb9a83e0	Protect from Close.	\Device\HarddiskVolume2\Windows\SysWOW64\en-US\ntdll.dll.mui
File	0x00000344	0xfb9ad520	Protect from Close.	\Device\HarddiskVolume2\Windows\System32\en-US\kernel32.dll.mui
File	0x00000348	0xfaf980a0	Protect from Close.	<Unknown, type: File - possibly pipe or named pipe waiting on a read or write>
File	0x0000034c	0xfaf99680	Protect from Close.	\Device\HarddiskVolume2\Windows\SysWOW64\en-US\apphelp.dll.mui

The Objects View

The display is a scrolled list displaying:

- type (e.g. GDI handle object type)
- handle value
- kernel address
- object flags
- object name (if available)

The list can be sorted in increasing or decreasing by clicking in the header of any column.

Updating the display

- **Update Interval (s)** › automatically updates the display at your choice of interval between 0.1 and 60 seconds - or not at all!

  Adjust this depending on the complexity of your application.

- **Refresh** › updates the display - as does the button on the Tools menu and toolbar

  With an update interval set to No Update, you'll need to use this Refresh button to update the display when you wish.

Running 32 bit applications on 64 bit Operating Systems

When running 32 bit applications on 64 bit Operating Systems you may find that the Objects view does not display much information.

If there's little or no information, this is likely because it is not possible to get complete object information in this environment.

There are two solutions to this:

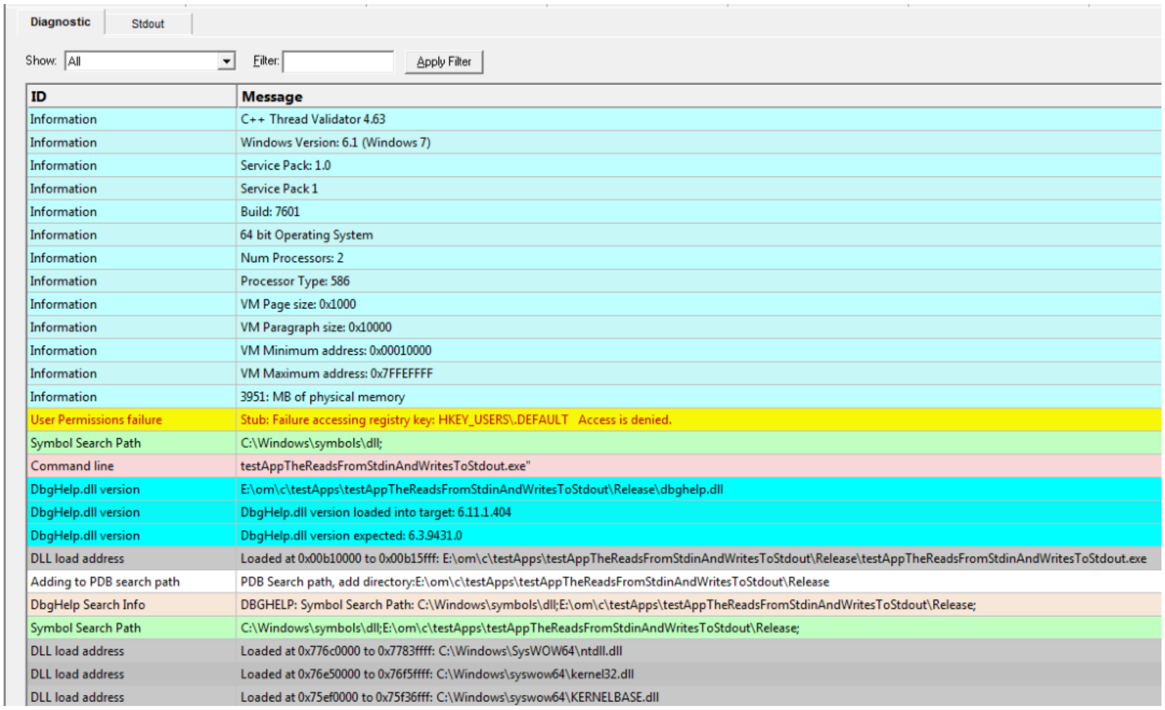
- Run your 32 bit application (and Thread Validator) on a 32 bit operating system
- Run your 64 bit application (and Thread Validator x64) on a 64 bit operating system.

3.7.6 Diagnostic

The **Diagnostic** view displays all diagnostic information collected from the stub.

There are two subtabs. One for Diagnostic information and one for displaying any data captured from stdout and stderr.

Diagnostic



Show: All

Filter:

ID	Message
Information	C++ Thread Validator 4.63
Information	Windows Version: 6.1 (Windows 7)
Information	Service Pack: 1.0
Information	Service Pack 1
Information	Build: 7601
Information	64 bit Operating System
Information	Num Processors: 2
Information	Processor Type: 586
Information	VM Page size: 0x1000
Information	VM Paragraph size: 0x10000
Information	VM Minimum address: 0x00010000
Information	VM Maximum address: 0x7FFEFFFF
Information	3951: MB of physical memory
User Permissions failure	Stub: Failure accessing registry key: HKEY_USERS\.DEFAULT Access is denied.
Symbol Search Path	C:\Windows\symbols\dll;
Command line	testAppTheReadsFromStdinAndWritesToStdout.exe"
DbgHelp.dll version	E:\om\c\testApps\testAppTheReadsFromStdinAndWritesToStdout\Release\dbghelp.dll
DbgHelp.dll version	DbgHelp.dll version loaded into target: 6.11.1.404
DbgHelp.dll version	DbgHelp.dll version expected: 6.3.9431.0
DLL load address	Loaded at 0x00b10000 to 0x00b15fff: E:\om\c\testApps\testAppTheReadsFromStdinAndWritesToStdout\Release\testAppTheReadsFromStdinAndWritesToStdout.exe
Adding to PDB search path	PDB Search path, add directory:E:\om\c\testApps\testAppTheReadsFromStdinAndWritesToStdout\Release
DbgHelp Search Info	DBGHELP: Symbol Search Path: C:\Windows\symbols\dll;E:\om\c\testApps\testAppTheReadsFromStdinAndWritesToStdout\Release;
Symbol Search Path	C:\Windows\symbols\dll;E:\om\c\testApps\testAppTheReadsFromStdinAndWritesToStdout\Release;
DLL load address	Loaded at 0x776c0000 to 0x7783ffff: C:\Windows\SysWOW64\ntdll.dll
DLL load address	Loaded at 0x76e50000 to 0x76f5ffff: C:\Windows\syswow64\kernel32.dll
DLL load address	Loaded at 0x75ef0000 to 0x75f36fff: C:\Windows\syswow64\KERNELBASE.dll

Diagnostic information

When Thread Validator's stub is injected into the target program, it logs diagnostic information to the main window for inspection.

Examples of diagnostic data collected are below, and may be displayed with a message, although you may not see some of these if all is well:

Hooking information

- Ordinal hook found
- Hook C++ constructor / destructor

Other information

- DLL load address
- DbgHelp searching

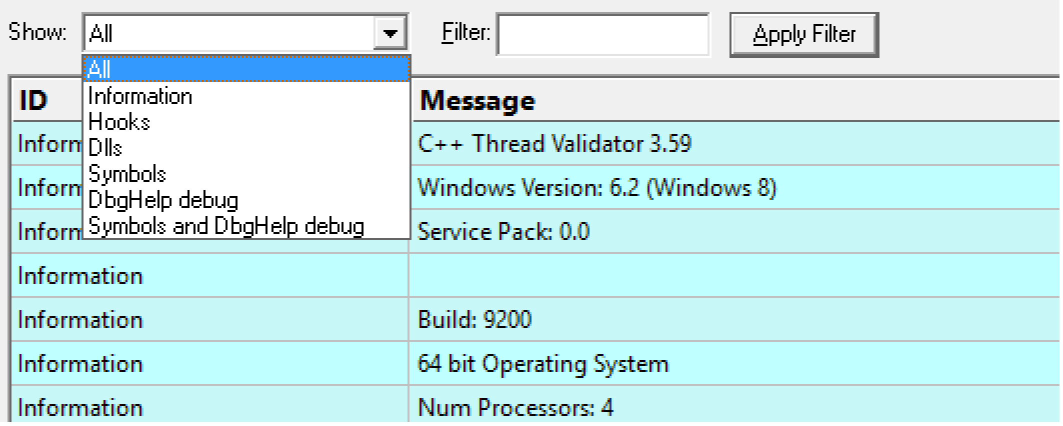
- Function hook success or failure
- Delay loaded function hooked
- Possible hook found
- Function already hooked
- Hook at address
- Image source line
- Unknown instruction found
- Disassembly of troublesome code
- Failed to find Release/Debug CRT heap
- Symbol reader status

The locations of loaded DLLs are also displayed in the window for each `LoadLibrary()`, `LoadLibraryEx()` and `FreeLibrary()` in the target program.

If for whatever reason, you don't want to collect diagnostic information, you can switch it off in the General > Instrumentation Logging page of the settings dialog

Filtering diagnostic information

By default, all information is displayed, but you can filter the messages to show only one type:



ID	Message
Inform	C++ Thread Validator 3.59
Inform	Windows Version: 6.2 (Windows 8)
Inform	Service Pack: 0.0
Information	
Information	Build: 9200
Information	64 bit Operating System
Information	Num Processors: 4

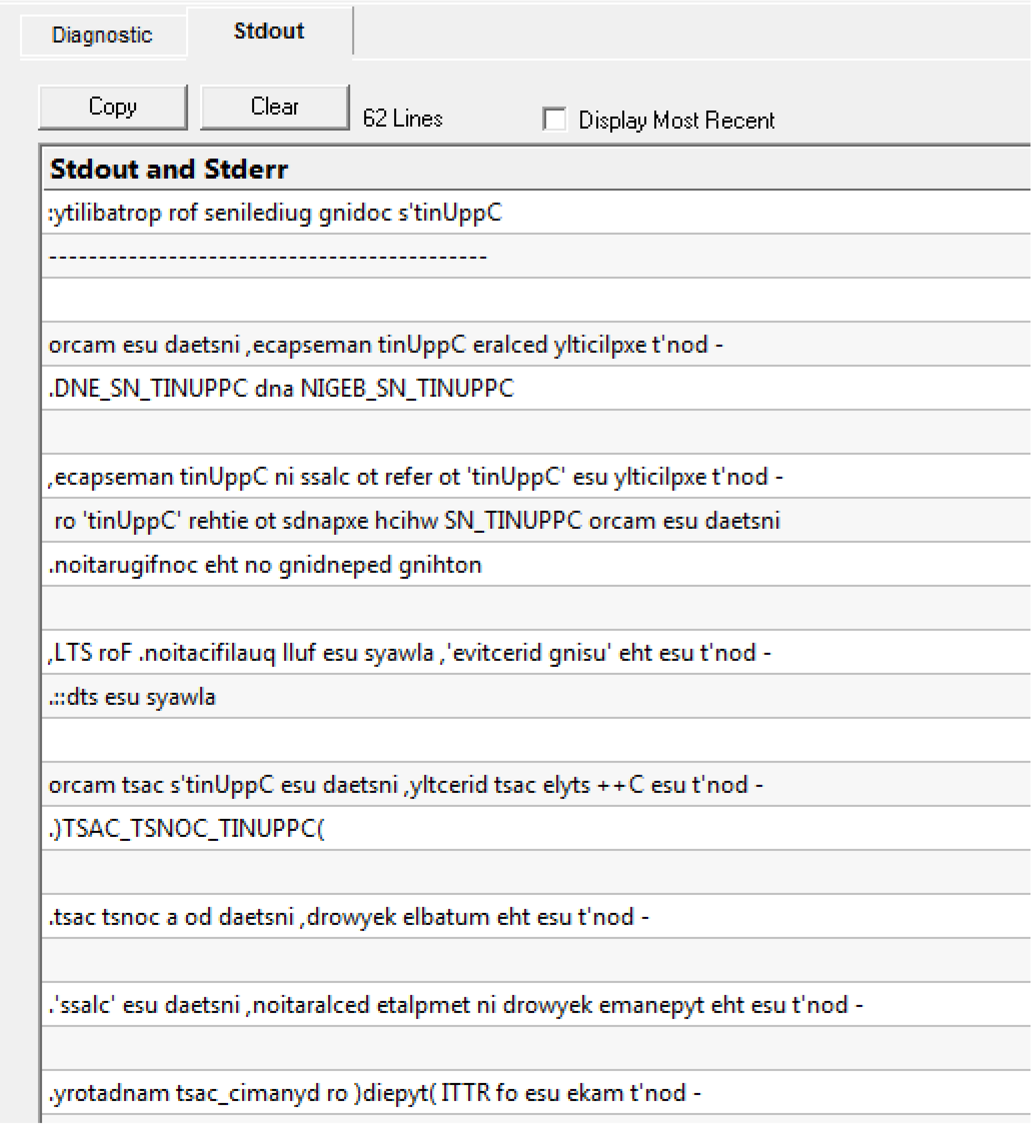
- **All** › the default option is to show everything
- **Information** › operating system and environment information, etc
- **Error** › notification of denied access and other error messages
- **Hooks** › hooking success and failure messages
- **DLLs** › DLL related information
- **Symbols** › symbol loading status messages
- **DbgHelp debug** › messages from DbgHelp.dll about the DLL symbol search processes
- **Symbols and DbgHelp debug** › both the previous two

As well as filtering different *types* of lines, you can also search for specific terms:

- **Filter** › enter some text and **Apply Filter** to show lines with the term in the Message column

When identifying why symbols aren't loading, you'll find it's much easier when showing only the **DbgHelp debug** information.

Stdout and Stderr



```
:ytilibatrop rof senilediug gnidoc s'tinUppC
------------------------------------------

orcam esu daetsni ,ecapseman tinUppC eralced ylticilpxe t'nod -
.DNE_SN_TINUPPC dna NIGEB_SN_TINUPPC

,ecapseman tinUppC ni ssalc ot refer ot 'tinUppC' esu ylticilpxe t'nod -
 ro 'tinUppC' rehtie ot sdnapxe hcihw SN_TINUPPC orcam esu daetsni
.noitarugifnoc eht no gnidneped gnihton

,LTS roF .noitacifilauq lluf esu syawla ,'evitcerid gnisu' eht esu t'nod -
.::dts esu syawla

orcam tsac s'tinUppC esu daetsni ,yltcerid tsac elyts ++C esu t'nod -
.)TSAC_TSNOC_TINUPPC(

.tsac tsnoc a od daetsni ,drowyek elbatum eht esu t'nod -

.'ssalc' esu daetsni ,noitaralced etalpmet ni drowyek emanepyt eht esu t'nod -

.yrotadnam tsac_cimanyd ro )diepyt( ITTR fo esu ekam t'nod -
```

The **Stdout** tab displays any data collected from stdout and stderr. ➡ The option to enable this data collection is specified on the launch dialog/wizard.

The above image shows some data collected from a program that reverses the characters in each line passed to it.

- **Copy** › copy all data from the display on to the clipboard. For large amounts of data this can be time consuming.

- **Clear** › clear the display of any captured data.

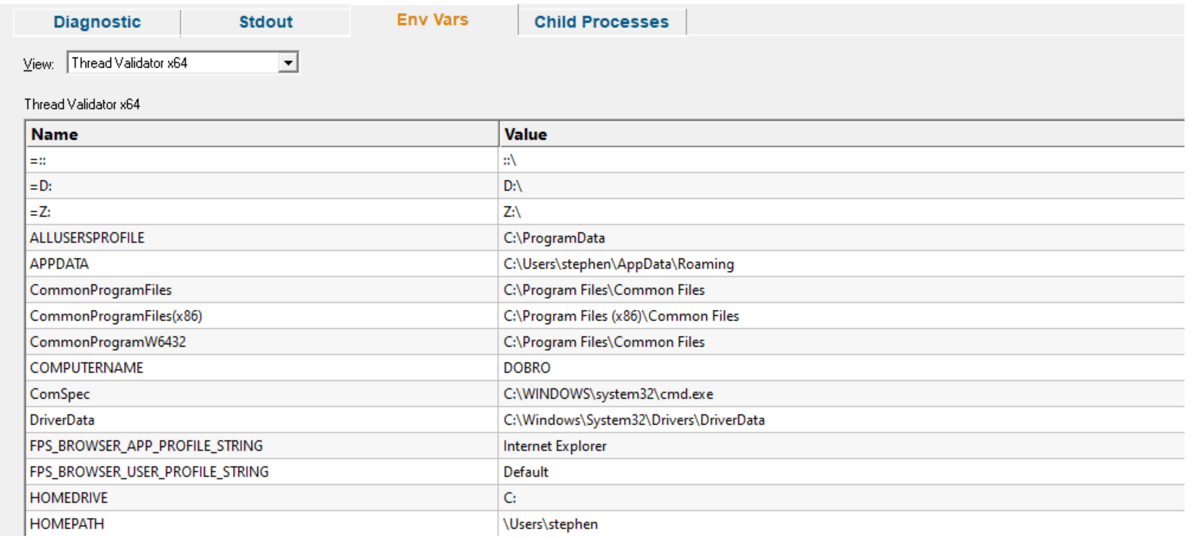
- **Display Most Recent** › the display will be scrolled to ensure the most recently captured data is displayed.

Environment Variables

Environment variables tab displays environment variables from Thread Validator, environment variables from the program under test and environment variable substitution errors.

Choose which data you wish to view using the combo box at the top left of the tab.

Thread Validator environment variables



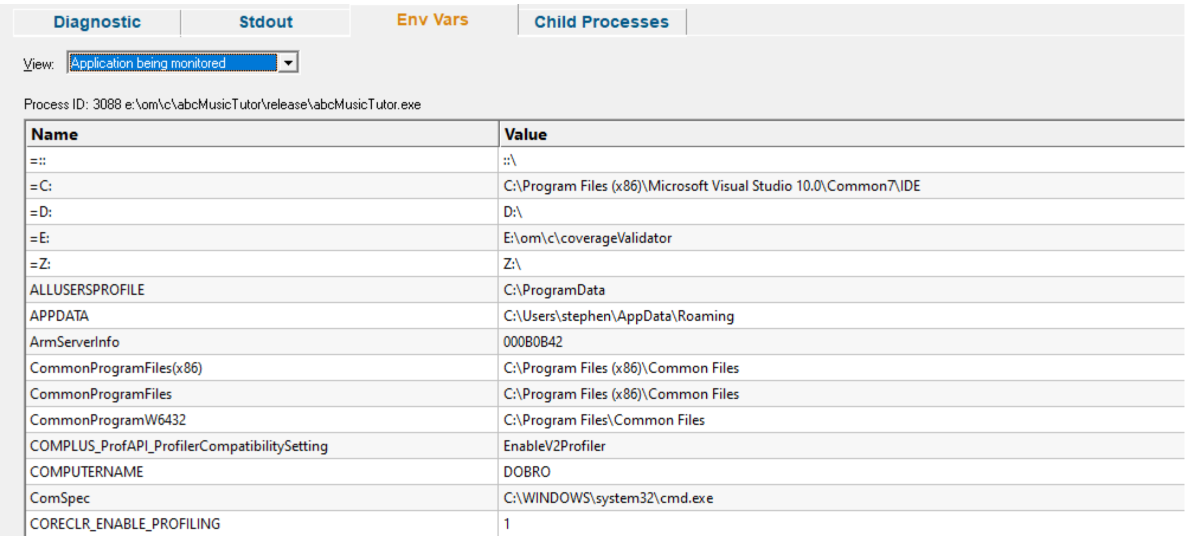
Thread Validator x64

Name	Value
=::	::\
=D:	D:\
=Z:	Z:\
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\stephen\AppData\Roaming
CommonProgramFiles	C:\Program Files\Common Files
CommonProgramFiles(x86)	C:\Program Files (x86)\Common Files
CommonProgramW6432	C:\Program Files\Common Files
COMPUTERNAME	DOBRO
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
FPS_BROWSER_APP_PROFILE_STRING	Internet Explorer
FPS_BROWSER_USER_PROFILE_STRING	Default
HOMEDRIVE	C:
HOMEPATH	\Users\stephen

Target application environment variables

If you launched the target application from Thread Validator the target application’s environment variables will be similar to those in Thread Validator, but with some additional env vars to control .Net profilers and and some other SVL_ prefixed env vars to communicate various data to Software Verify components that are loaded.

If you launched the target application as a standalone application, or service and used one of our APIs to connect to Thread Validator, the environment variables shown will reflect those in force at the time the application/service was started, and the account that application/service is running on.



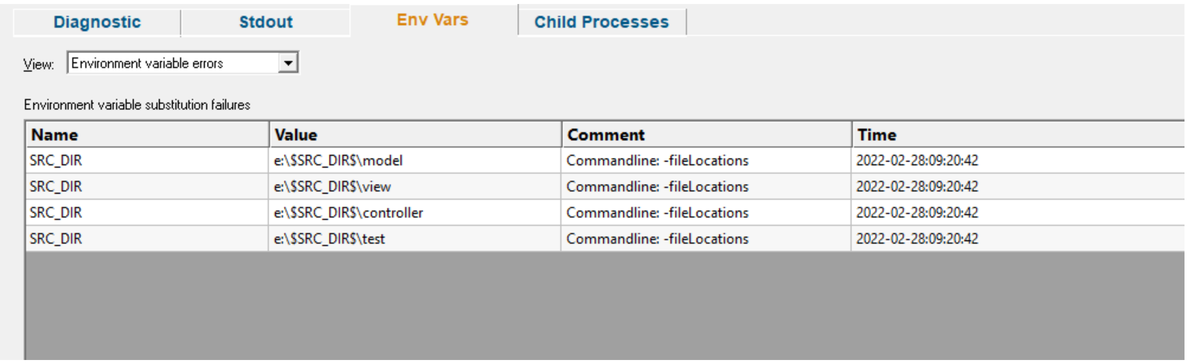
View: Application being monitored

Process ID: 3088 e:\om\c\abcMusicTutor\release\abcMusicTutor.exe

Name	Value
=::	::\
=C:	C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE
=D:	D:\
=E:	E:\om\c\coverageValidator
=Z:	Z:\
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\stephen\AppData\Roaming
ArmServerInfo	000B0B42
CommonProgramFiles(x86)	C:\Program Files (x86)\Common Files
CommonProgramFiles	C:\Program Files (x86)\Common Files
CommonProgramW6432	C:\Program Files\Common Files
COMPLUS_ProfAPI_ProfilerCompatibilitySetting	EnableV2Profiler
COMPUTERNAME	DOBRO
ComSpec	C:\WINDOWS\system32\cmd.exe
CORECLR_ENABLE_PROFILING	1

Environment variable errors

The environment variable errors display shows the name of the environment variable that could not be found, the string containing the environment variable, a comment indicating where the string came from (in this example, the command line), and a timestamp.



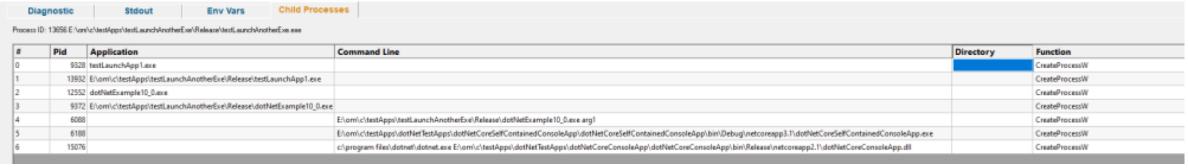
View: Environment variable errors

Environment variable substitution failures

Name	Value	Comment	Time
SRC_DIR	e:\SRC_DIR\model	Commandline: -fileLocations	2022-02-28:09:20:42
SRC_DIR	e:\SRC_DIR\view	Commandline: -fileLocations	2022-02-28:09:20:42
SRC_DIR	e:\SRC_DIR\controller	Commandline: -fileLocations	2022-02-28:09:20:42
SRC_DIR	e:\SRC_DIR\test	Commandline: -fileLocations	2022-02-28:09:20:42

Child Processes

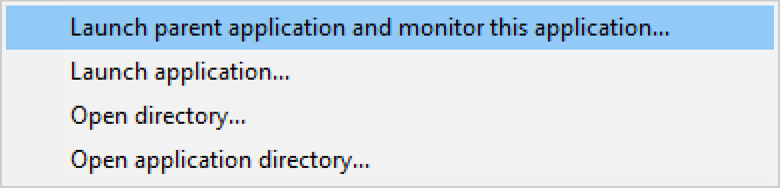
Information about child processes, and the appropriate launch parameters passed to CreateProcess (and related functions) are displayed on this tab.



Process ID: 13656 E:\om\c\testApps\testLaunchAnotherExe\Release\testLaunchAnotherExe.exe

#	Pid	Application	Command Line	Directory	Function
0	9328	testLaunchApp1.exe			CreateProcessW
1	13932	E:\om\c\testApps\testLaunchAnotherExe\Release\testLaunchApp1.exe			CreateProcessW
2	12552	dotNetExample10_0.exe			CreateProcessW
3	9372	E:\om\c\testApps\testLaunchAnotherExe\Release\dotNetExample10_0.exe			CreateProcessW
4	6088		E:\om\c\testApps\testLaunchAnotherExe\Release\dotNetExample10_0.exe arg1		CreateProcessW
5	6188		E:\om\c\testApps\dotNetTestApps\dotNetCoreSelfContainedConsoleApp\dotNetCoreSelfContainedConsoleApp\bin\Debug\netcoreapp3.1\dotNetCoreSelfContainedConsoleApp.exe		CreateProcessW
6	15076		c:\program files\dotnet\dotnet.exe E:\om\c\testApps\dotNetTestApps\dotNetCoreConsoleApp\dotNetCoreConsoleApp\bin\Release\netcoreapp2.1\dotNetCoreConsoleApp.dll		CreateProcessW

A context menu is provided to allow you to perform some actions with the launched application data.


Launch parent application and monitor this application...
Launch application...
Open directory...
Open application directory...

- **Launch parent application and monitor this application...** › the launch application dialog is displayed configured to launch the parent application and monitor this application

- **Launch application...** › the launch application dialog is displayed configured to launch and monitor this application

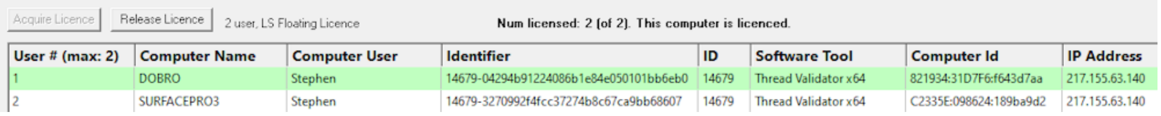
- **Open directory...** › Windows Explorer is launched to view the contents of the launch directory (the directory field is empty nothing will be shown)

- **Open application directory...** › Windows Explorer is launched to view the directory that contains the application (if the application specification has no path nothing will be shown)

3.7.7 Floating Licence

The **Floating Licence** view displays information about the computers using the floating licence.

This view is only displayed if a floating licence has been purchased. Evaluation users will not see this view.



2 user, LS Floating Licence

Num licensed: 2 (of 2). This computer is licenced.

User # (max: 2)	Computer Name	Computer User	Identifier	ID	Software Tool	Computer Id	IP Address
1	DOBRO	Stephen	14679-04294b91224086b1e84e050101bb6eb0	14679	Thread Validator x64	821934:31D7F6:f643d7aa	217.155.63.140
2	SURFACEPRO3	Stephen	14679-3270992f4fcc37274b8c67ca9bb68607	14679	Thread Validator x64	C2335E:098624:189ba9d2	217.155.63.140

The screenshot above show two computers using the same 2 user floating licence, that has maintenance id 14679. Both computer users are licenced and can use the software.

On startup the software automatically checks to see if a floating licence is available, and acquires the licence if possible. This takes a few seconds to process, after startup of the software

Global Floating Licences

An internet connection is required for floating licences to work. The licence server is managed and run by Software Verify.

Local Network Floating Licences

An internet connection is not required for local network floating licences to work. No licence server is required. The licences are automatically managed by the computers on the local network.

Licence information

The information show in this display allows you to identify which of your colleagues are using the software and which versions of the software are in use.

- User
  The user id (1 to number of licensed users).

- Computer Name
  The name of the computer

- Computer User
  The login name of the user of the computer.

- Identifier
  The unique identifier for this licence, used on the licence server.

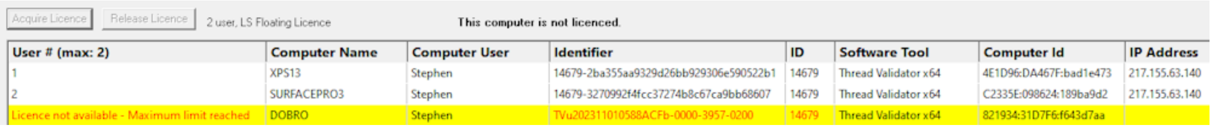
- ID
  The maintenance id for the software.

- Software Tool
  The software tool and version of the software that is running on that computer.

- Computer ID
  The unique id for this computer.

- IP Address
  This computer's IP address.

Unlicenced users



2 user, LS Floating Licence

This computer is not licenced.

User # (max: 2)	Computer Name	Computer User	Identifier	ID	Software Tool	Computer Id	IP Address
1	XPS13	Stephen	14679-2ba355aa9329d26bb929306e590522b1	14679	Thread Validator x64	4E1D96:DA467F:bad1e473	217.155.63.140
2	SURFACEPRO3	Stephen	14679-3270992f4fcc37274b8c67ca9bb68607	14679	Thread Validator x64	C2335E:098624:189ba9d2	217.155.63.140
Licence not available - Maximum limit reached	DOBRO	Stephen	TVu20231101058 8ACFb-0000-3957-0200	14679	Thread Validator x64	821934:31D7F6:f643d7aa	

If any additional users are trying to get a licence for the software, but there are not enough licences, they will also be shown in the display, but with red text on a yellow background.

Please note that on the machine of an unlicensed user the status information will be different.

The software checks to see if a licence has been released on a periodic basis, so that if a licence is released by another user, it can be acquired by the next waiting user.

Releasing a licence

If you have finished using a licence and wish to let a team mate use the software, you have two choices.

You can close Thread Validator, releasing the licence as it closes.

Or you can keep Thread Validator running by manually releasing the licence. Do this by clicking the **Release Licence** button.

Acquiring a licence

If you have released a licence you will need to actively reclaim a licence when you wish to use Thread Validator again. You can start this procedure by clicking the **Acquire Licence** button.

3.8 User Interface Mode

Setting the user interface mode - Wizards or Dialogs?

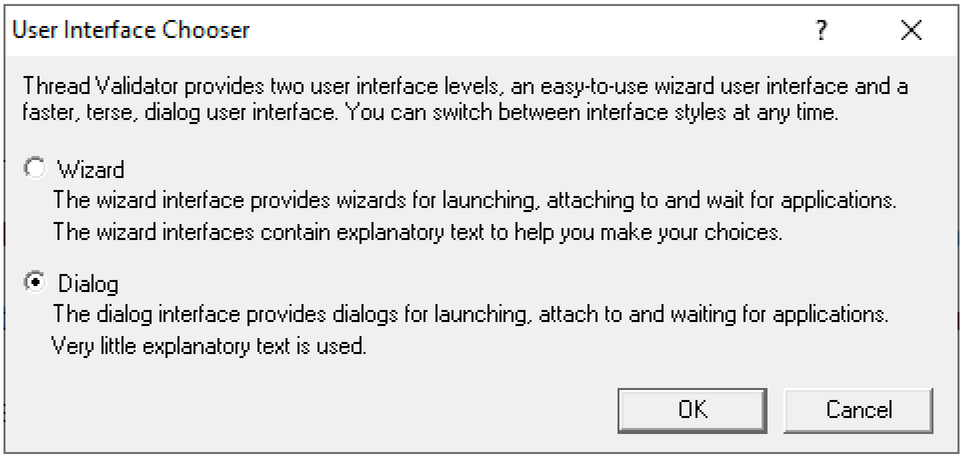
For some key tasks, there are two user interface modes controlling the way in which options are presented to you:

- **Wizard mode** › guides you through the tasks in a linear fashion
- **Dialog mode** › all options are contained in a single dialog

  Experienced users will find this mode quicker to use.

To set the user interface mode:

☰ **Settings** menu › **User Interface Mode...** › select the desired mode in the **User Interface Chooser** dialog:



User Interface Chooser

Thread Validator provides two user interface levels, an easy-to-use wizard user interface and a faster, terse, dialog user interface. You can switch between interface styles at any time.

○ Wizard
The wizard interface provides wizards for launching, attaching to and wait for applications.
The wizard interfaces contain explanatory text to help you make your choices.

● Dialog
The dialog interface provides dialogs for launching, attach to and waiting for applications.
Very little explanatory text is used.

The user interface mode affects the following tasks:

- Attaching to an application (Injection)
- Launching an application
- Wait for application to start

3.9 UX Theme

The user interface provides three UX themes.

- Modern. The look and feel of current Software Verify tools.
- Classic. The look and feel of previous Software Verify tools.
- High Contrast. A higher contrast version of the Modern theme.

Setting the UX theme

To set the UX theme

Settings menu › **UX Theme...** › shows the UX Theme chooser dialog

Changing the UX theme will update some of the colours that you can modify with the colour settings dialog.

3.10 Settings

Thread Validator allows extensive control over which data is collected and how that data is displayed. Additional options control the way the application behaves.

These settings can generally be considered as being either Global settings or Local settings.

Global and local settings

Global settings affect all data collected and its display throughout Thread Validator.

Global settings are changed via the Settings Dialog and the following pages describe each available group of settings.

Local settings apply to controls and data displayed in each of the main display windows.

Local settings are found at the top of each relevant tab.

Other settings

There are a few more settings not included in the global settings dialog such as:

- User interface mode
- Session settings

- User permissions warnings

3.10.1 Global Settings

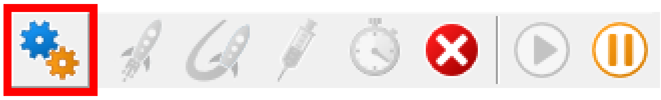
The Data Collection Settings dialog allows you to control all the global settings in Thread Validator that affect the way data is collected and displayed. There are also local display options on most of the main tabs.

This section has a warning about use of the Reset button.

Opening the settings dialog

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

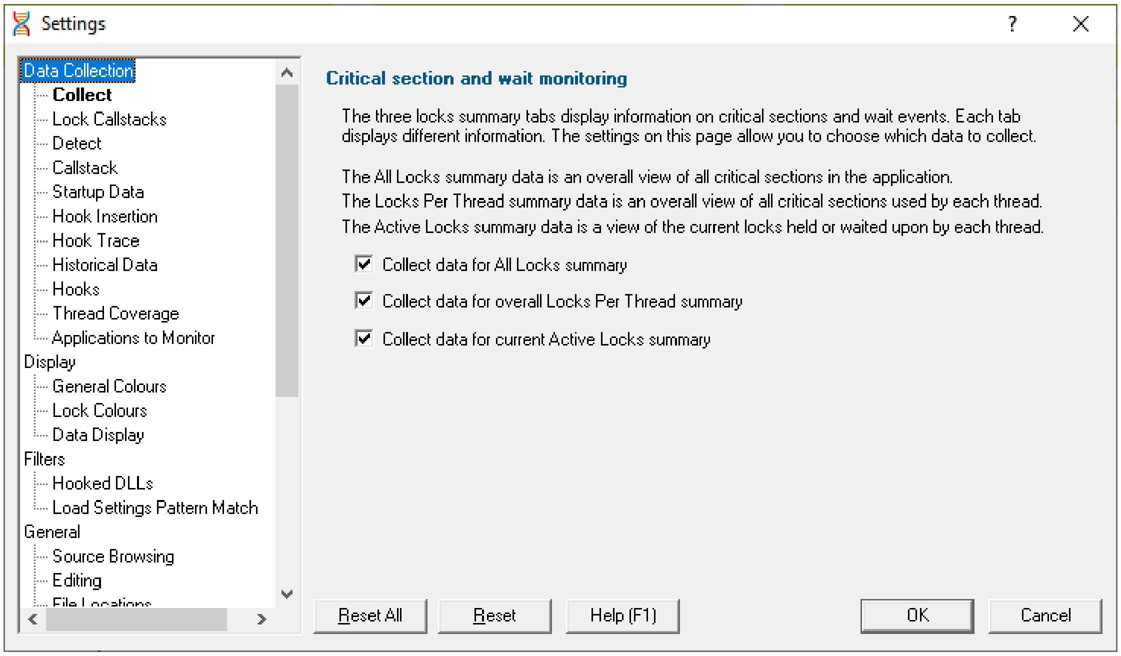
Or use the option on the Session Toolbar:



Using the settings dialog

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

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



Critical section and wait monitoring

The three locks summary tabs display information on critical sections and wait events. Each tab displays different information. The settings on this page allow you to choose which data to collect.

The All Locks summary data is an overall view of all critical sections in the application.
The Locks Per Thread summary data is an overall view of all critical sections used by each thread.
The Active Locks summary data is a view of the current locks held or waited upon by each thread.

☑ Collect data for All Locks summary

☑ Collect data for overall Locks Per Thread summary

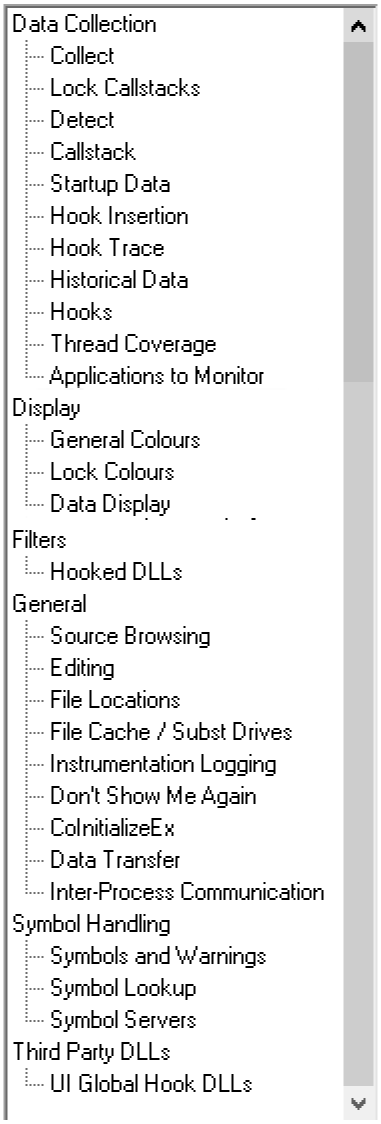
☑ Collect data for current Active Locks summary

After selecting a topic, you can also use the cursor up and down arrow keys to change the selected item.

Entering a character on the keyboard cycles though topics starting with that letter.

Too many settings? It may seem that there is an overwhelming number of settings to worry about. **Don't panic!** The good news is that for new users, very few (if any) settings actually need to be changed to use the application in most cases, and even for experienced users, many groups of settings will not be needed. However, Thread Validator remains flexible for all our users in many different scenarios.

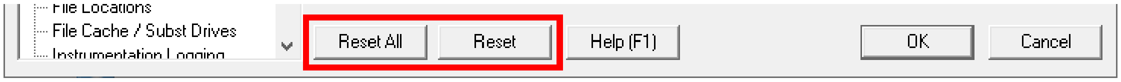
Click on any item in the picture below to find out more about the settings for that group.



- Data Collection
  - Collect
  - Lock Callstacks
  - Detect
  - Callstack
  - Startup Data
  - Hook Insertion
  - Hook Trace
  - Historical Data
  - Hooks
  - Thread Coverage
  - Applications to Monitor
- Display
  - General Colours
  - Lock Colours
  - Data Display
- Filters
  - Hooked DLLs
- General
  - Source Browsing
  - Editing
  - File Locations
  - File Cache / Subst Drives
  - Instrumentation Logging
  - Don't Show Me Again
  - CoInitializeEx
  - Data Transfer
  - Inter-Process Communication
- Symbol Handling
  - Symbols and Warnings
  - Symbol Lookup
  - Symbol Servers
- Third Party DLLs
  - UI Global Hook DLLs

Restoring the default settings

The settings dialog has **Reset All** and **Reset** buttons near the bottom left of the dialog which you can use to reset all global settings back to their default values.



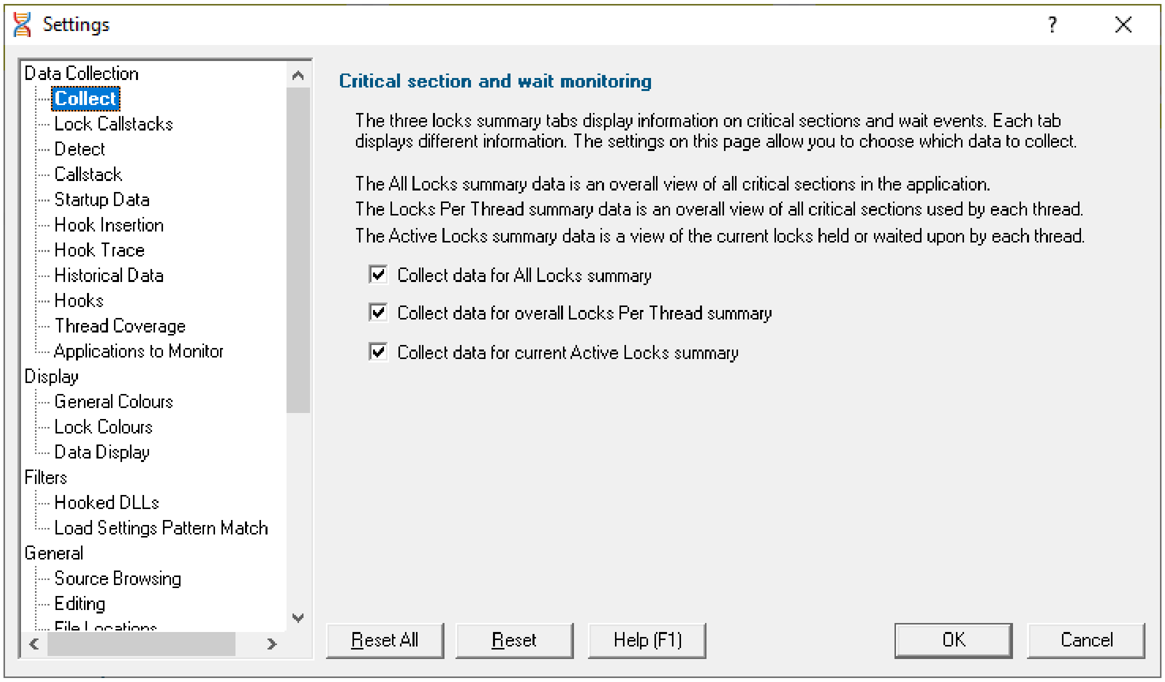
This **Reset All** button resets nearly **all global settings** in Thread Validator, not just the settings visible on the current tab of the dialog.

The **Reset** button resets just the settings visible on the current tab of the dialog.

3.10.1.1 Data Collection

3.10.1.1.1 Collect

The **Collect** tab allows you to specify the types of data that will be collected for display by Thread Validator.



Critical section and wait monitoring

The three locks summary tabs display information on critical sections and wait events. Each tab displays different information. The settings on this page allow you to choose which data to collect.

The All Locks summary data is an overall view of all critical sections in the application.
The Locks Per Thread summary data is an overall view of all critical sections used by each thread.
The Active Locks summary data is a view of the current locks held or waited upon by each thread.

☑ Collect data for All Locks summary

☑ Collect data for overall Locks Per Thread summary

☑ Collect data for current Active Locks summary

Critical sections and Wait Monitoring

The first few tabs on the main window display information about critical sections and wait events.

Data for these tabs is collected by default, but collection can be turned off using the appropriate option below.

- **Collect data for All Locks summary** › collect data for the Summary and Locks Summary tab

- **Collect data for overall Locks Per Thread summary** › collect data for the Locks Per Thread tab
- **Collect data for current Active Locks summary** › collect data for the Active Locks (per thread) tab

Even with all the options switched off you may still see thread 'header' rows, but no lock information under each one.

Instrumentation logging

The logging of DLLs, source files, classes, methods and functions that are not instrumented can help you understand the reason why part of your code isn't getting the coverage information you expect.

Once enabled, and a session has started, you can view a list of items that have not been instrumented via the Tools menu.

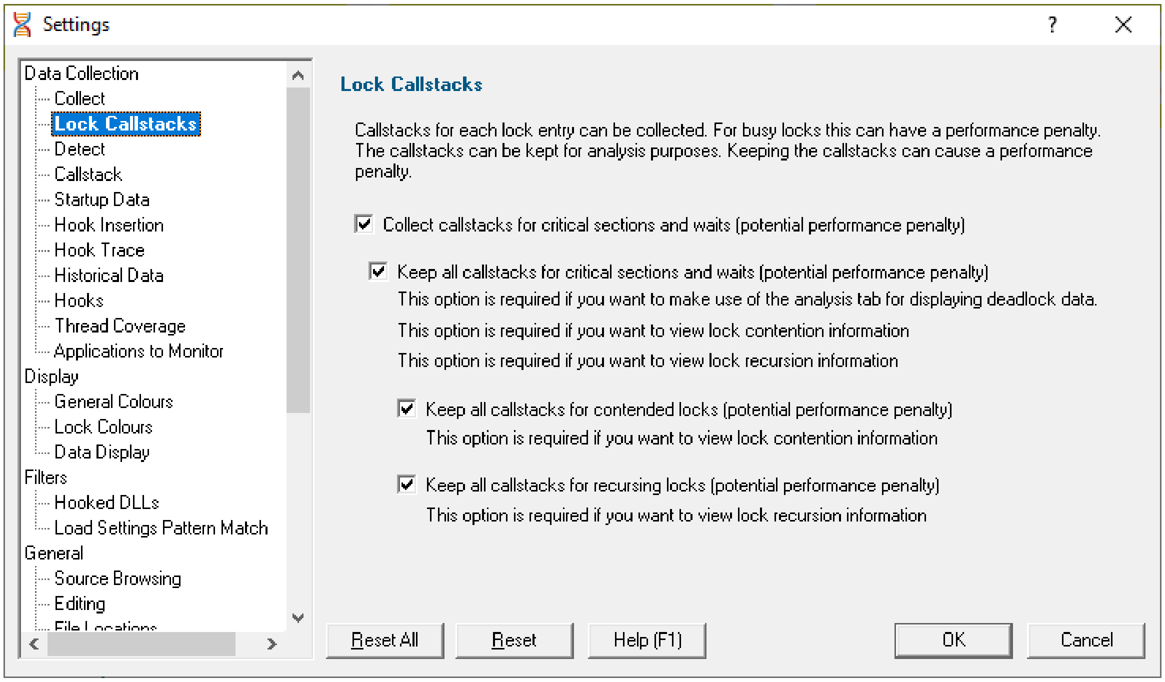
- **Enable instrumentation logging** › check to enable logging once the next session starts

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

Reset - Resets the settings on the current page.

3.10.1.1.2 Lock Callstacks

The **Lock Callstacks** tab allows you to specify the types of data that will be collected for display by Thread Validator.



Lock Callstacks

Callstacks for each lock entry can be collected. For busy locks this can have a performance penalty. The callstacks can be kept for analysis purposes. Keeping the callstacks can cause a performance penalty.

☑ Collect callstacks for critical sections and waits (potential performance penalty)

☑ Keep all callstacks for critical sections and waits (potential performance penalty)
This option is required if you want to make use of the analysis tab for displaying deadlock data.
This option is required if you want to view lock contention information
This option is required if you want to view lock recursion information

☑ Keep all callstacks for contended locks (potential performance penalty)
This option is required if you want to view lock contention information

☑ Keep all callstacks for recursing locks (potential performance penalty)
This option is required if you want to view lock recursion information

Lock Callstacks

The summary and overview information gathered with the Collect settings is very useful and has a relatively low impact upon the performance of the target application.

In some circumstances the information collected is enough to diagnose various threading problems, but other times you'll need more data.

Many threading problems are much easier to diagnose if a callstack is also associated with the data. Callstacks allow source code inspection and a greater level of understanding of the exact nature of the problem.

Thread Validator can store the *most recent* callstack or *all* callstacks associated with critical sections and wait events, contentions and recursions.

Information is stored on a per thread basis. If for example, a deadlock occurs, you can see which critical section was locked or waiting, and where this happened.

- **Collect callstacks for critical sections and waits** › collect the *most recent* callstacks for each of the checked items below, or not at all.

  Collecting callstacks can cause significant drops in performance depending on the nature of the target application.

- **Keep all callstacks for...**

  - **critical sections and waits** › keep *all* callstacks per critical section so that analysis of the various callers can be performed

- **contended locks** › collect all lock contention callstacks
- **recursing locks** › collect all lock recursion callstacks

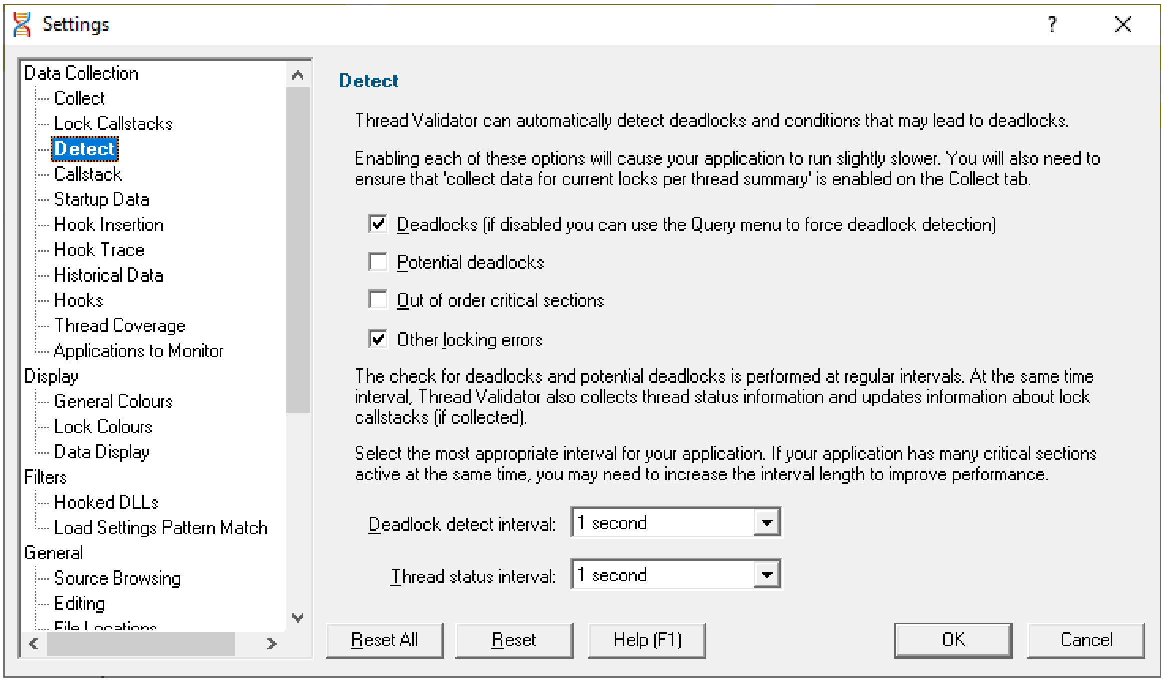
Keeping all callstacks causes an even larger performance drop and increase in memory usage. It's recommended that all these option are only checked if you have a need for the information about *all* callstacks.

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

Reset - Resets the settings on the current page.

3.10.1.1.3 Detect

The **Detect** tab allows you to specify how the automatic detection of errors is handled.



Detect

Thread Validator can automatically detect deadlocks and conditions that may lead to deadlocks.

Enabling each of these options will cause your application to run slightly slower. You will also need to ensure that 'collect data for current locks per thread summary' is enabled on the Collect tab.

☑ Deadlocks (if disabled you can use the Query menu to force deadlock detection)

□ Potential deadlocks

□ Out of order critical sections

☑ Other locking errors

The check for deadlocks and potential deadlocks is performed at regular intervals. At the same time interval, Thread Validator also collects thread status information and updates information about lock callstacks (if collected).

Select the most appropriate interval for your application. If your application has many critical sections active at the same time, you may need to increase the interval length to improve performance.

Deadlock detect interval: 1 second

Thread status interval: 1 second

Automatic detection of error conditions

Thread Validator regularly scans for various error conditions using the same data collected by the options on the Collect tab.

At the same time, thread status information from the operating system is collected to update symbolic and callstack information for use by the user interface.

- **Deadlocks** › automatically detect deadlocks

If your application has many threads, disable this option to prevent Thread Validator from using too much processor time.

When disabled, the **Query** menu option Deadlock Detection is enabled.

- **Potential deadlocks** › automatically detect potential deadlocks

If your application has many threads, disable this option to prevent Thread Validator from using too much processor time.

When disabled, the **Query** menu option Potential Deadlock Detection is *also* disabled (unlike Deadlocks above).

- **Out of order critical sections** › automatically detect out-of-order critical sections
- **Other locking errors** › automatically detect various other locking errors

Detection Intervals

Deadlock detection and thread status happens at regular intervals:

- **Deadlock detect interval** › the interval at which the automatic error detection will take place

  Deadlock detection can be set for intervals of 1 to 60 seconds - or *Never*

- **Thread status interval** › the interval at which the status data collection will take place

  Thread status can be determined at intervals of 1 to 60 seconds

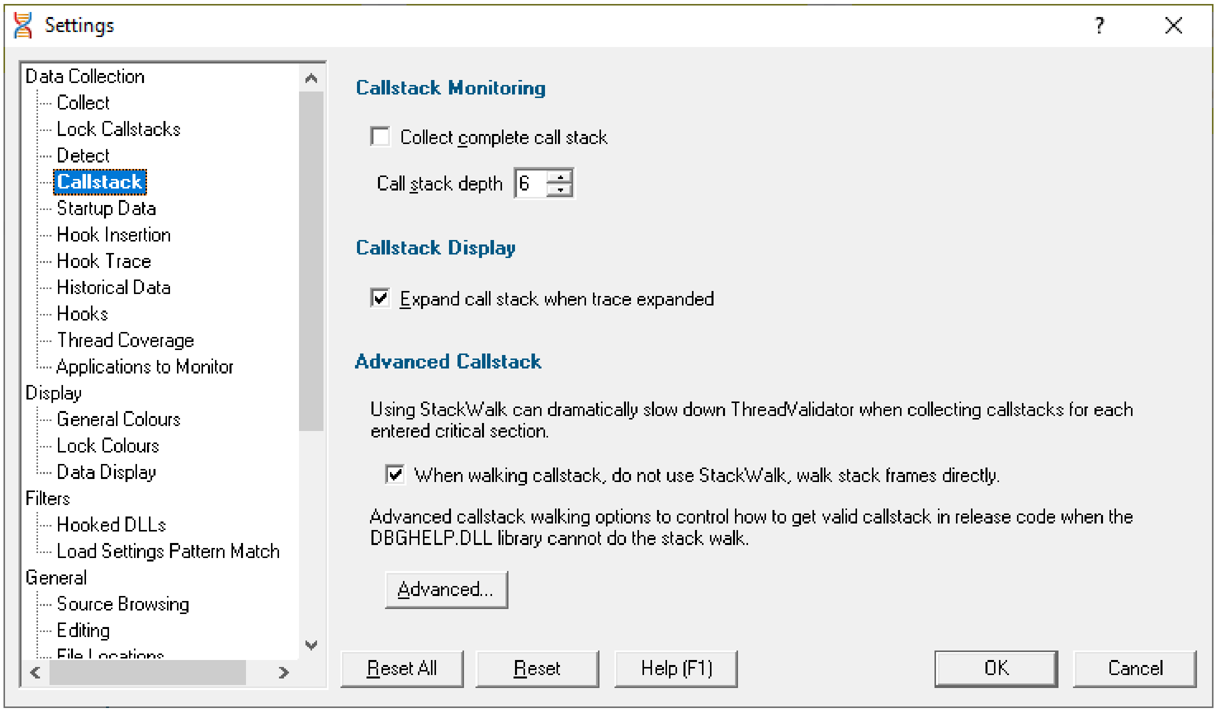
Automatic detection of these conditions may cause your application to run slightly slower.

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

Reset - Resets the settings on the current page.

3.10.1.1.4 Callstack

The **Callstack** tab allows you to specify how the callstack is collected, and how information about the callstack is displayed.



Callstack Monitoring

□ Collect complete call stack

Call stack depth 6

Callstack Display

☑ Expand call stack when trace expanded

Advanced Callstack

Using StackWalk can dramatically slow down ThreadValidator when collecting callstacks for each entered critical section.

☑ When walking callstack, do not use StackWalk, walk stack frames directly.

Advanced callstack walking options to control how to get valid callstack in release code when the DBGHELP.DLL library cannot do the stack walk.

Advanced...

Callstack Monitoring

When Thread Validator stub collects callstacks for each hooked function, the stack traces can be very long.

Collecting longer stack traces means:

- collecting stack traces takes longer
- converting all the addresses in the stack trace to symbol names takes more time
- the target program runs slower
- more memory is consumed in the Thread Validator user interface

For this reason, you're able to specify whether you want complete or partial stack traces.

- **Collect complete call stack** › check for complete traces, uncheck to collect partial traces

  Specify the exact depth you wish to capture with the **Call stack depth** option

Callstack Display

Data items recorded by Thread Validator can be displayed with parameter names as well as being automatically expanded.

- **Show parameter names** › check to show parameter names shown with the function name

- **Expand call stack when trace expanded** › check to automatically expand the entire callstack when examining a data item

Advanced Callstack

As the title suggests, these callstack options are advanced and not usually necessary to change unless you are experiencing incomplete stack traces.

Thread Validator collects callstacks by walking along the stack frames. When no more stack frames can be walked, Thread Validator uses DbgHelp `StackWalk()` to attempt to walk any remaining stack frames.

`StackWalk()` is much slower than Thread Validator's direct stack walking. Calling `StackWalk()` to attempt to walk any remaining stack frames may not result in any extra stack frames being walked, but will result in slower execution of your application.

On the other hand, not using `StackWalk()` *may* result in shorter callstacks being collected.

- **When walking callstack, do not use StackWalk...** › prevent Thread Validator using `StackWalk()`

  You may have problems collecting some callstacks in release mode programs, and in some special cases in debug programs depending on your program.

In addition to the optimisation described above, Thread Validator provides three different methods of collecting callstacks for functions. These methods are in the Advanced section described below and are provided to allow you to tailor callstack collection to the task at hand.

As the title below suggests, the remaining options are for *advanced* use! An example might be if you find your program has a problem that can be identified, but for which the callstack cannot be collected properly.

Advanced callstack settings

- **Callstack walk helper size** › Specify the size of the cache used to optimize callstack walking

  Thread Validator uses a cache to help it optimize the callstack walking process. The cache is used to avoid calling operating system functions to walk the callstack when the result has been previously calculated.

  For applications generating many unique callstacks, this size may need to be increased.

  All sizes are prime numbers, and the default size is 100003 which is large enough for most applications. **If in doubt leave it at the default value of 100003**

Callstack walking

Thread Validator provides three different methods of collecting callstacks for functions which you can choose to tailor callstack collection to the task at hand.

1) The standard Microsoft® DbgHelp StackWalk() function

This function is optimised for walking standard Intel i386 stack frames where the EBP register is pushed on to the stack at function entry and popped from the stack when the function exits. This is the typical stack frame for a program built in debug mode.

The DbgHelp `StackWalk()` function is also capable of reading frame pointer omission data (FPO_DATA) included in a PE file. FPO_DATA is included in optimised binaries that do not use the EBP register to identify the stack frame - this is typical of a program built in release mode.

⊟ Missing data in your callstack? Although Microsoft have provided a very capable stack walking function, there are occasions when the StackWalk() function cannot continue walking along the stack, from one frame pointer to the next. When this happens collected stack traces often appear to have data missing, or look "too short". You may have noticed this when debugging release mode programs in Visual Studio®.

2) Alternative (custom) StackWalk() function

This proprietary method, although slower than Microsoft's stack walker, does not use stack frames to walk the stack, and so enables the stack walker to walk callstacks that DbgHelp StackWalk() cannot.

⊟ What's different about this method? A detailed technical discussion of how this algorithm works is not appropriate here, but suffice to say that all addresses found on the stack are checked for validity, both for code sections, likelihood of CALL instruction taking place, target and source addresses of CALL instructions, removal of duplicate data, and so forth. The resulting callstacks often contain some bogus stack entries, which are obvious to the end user, but not possible to detect by the callstack verification algorithm (this is often due to CALL instructions relying on indirect indexes held in registers which have been changed by the time the stack walker has walked to this point in the callstack - such entries must be taken at face value because they may be valid).

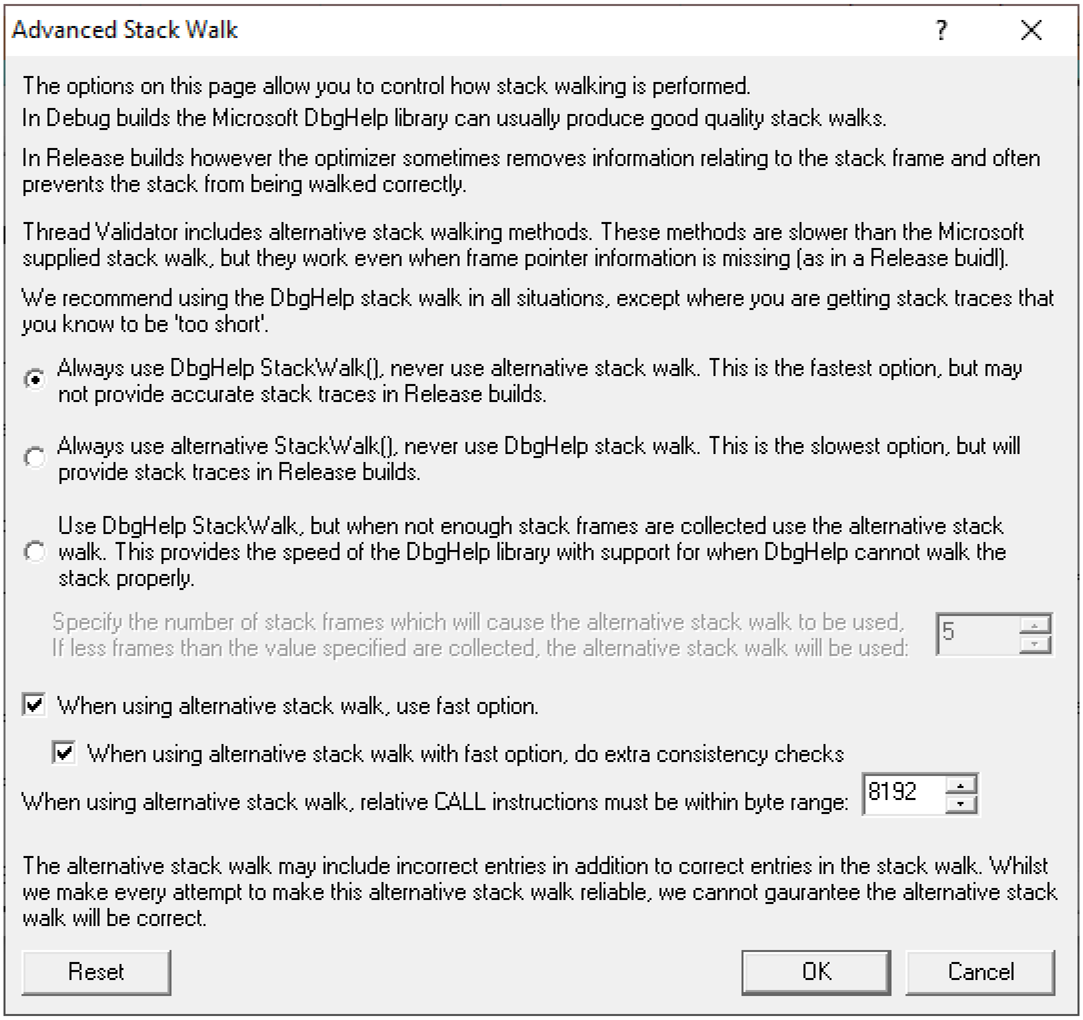
3) Hybrid of the two

The third stack walk type is a hybrid of the other two.

The first method is used to collect all callstacks. Any callstacks that are too short (defined by a callstack length threshold) then have the callstack collected by the second method.

This provides the speed and power of the standard Microsoft stack walker, with the flexibility to collect callstacks that would otherwise be uncollectible when DbgHelp StackWalk() fails to collect the callstack.

The Advanced Stack Walk dialog shown below is accessed via the **Advanced...** button and is used to choose one of the three callstack collection options above:



Advanced Stack Walk

The options on this page allow you to control how stack walking is performed.
In Debug builds the Microsoft DbgHelp library can usually produce good quality stack walks.

In Release builds however the optimizer sometimes removes information relating to the stack frame and often prevents the stack from being walked correctly.

Thread Validator includes alternative stack walking methods. These methods are slower than the Microsoft supplied stack walk, but they work even when frame pointer information is missing (as in a Release buidl).

We recommend using the DbgHelp stack walk in all situations, except where you are getting stack traces that you know to be 'too short'.

● Always use DbgHelp StackWalk(), never use alternative stack walk. This is the fastest option, but may not provide accurate stack traces in Release builds.

○ Always use alternative StackWalk(), never use DbgHelp stack walk. This is the slowest option, but will provide stack traces in Release builds.

○ Use DbgHelp StackWalk, but when not enough stack frames are collected use the alternative stack walk. This provides the speed of the DbgHelp library with support for when DbgHelp cannot walk the stack properly.

Specify the number of stack frames which will cause the alternative stack walk to be used. If less frames than the value specified are collected, the alternative stack walk will be used: 5

☑ When using alternative stack walk, use fast option.

☑ When using alternative stack walk with fast option, do extra consistency checks

When using alternative stack walk, relative CALL instructions must be within byte range: 8192

The alternative stack walk may include incorrect entries in addition to correct entries in the stack walk. Whilst we make every attempt to make this alternative stack walk reliable, we cannot gaurantee the alternative stack walk will be correct.

- **DbgHelp StackWalk** ❯ use the DbgHelp.dll `StackWalk()` function to walk all callstacks

  You may have problems collecting some callstacks in release mode programs, and in some special cases in debug programs depending on your program.

  - **Alternative StackWalk** ❯ use the alternative stack walking function to walk all callstacks

  All callstacks will be collected in both debug and release mode programs, but you may find that some callstacks contain incorrect entries.

- **Hybrid StackWalk** ❯ use the hybrid method outline above

  When the alternative stack walk is used you may find that some callstacks contain incorrect entries.

  To specify when to use the alternative stack walking function, set a **callstack depth**.

  Any standard callstacks that are *shorter* than the depth specified, will be collected using the alternative stack walking function.

➡ See also - recommended usage below

Alternative stack walk method - fast or slow?

- **When using alternative stack walk, use fast option** › uses a faster address verification scheme (recommended) or a slower one

Do extra consistency checks › When using the *fast* option, consistency checks can optionally be performed (recommended) to reduce the amount of incorrect addresses included in the callstack.

Alternative stack walk method - range of relative addresses

When the alternative stack walk is used, relative address CALL instructions have their target address computed to test if the target address is within a threshold of the previous callstack address.

This provides a form of source address to target address integrity to prevent invalid addresses be placed in the callstack.

- **Relative CALL instruction byte range** › set the threshold which the target address must be within, to fine tune the stack walk

A *larger* threshold reduces the accuracy of the stack walk by allowing too many invalid addresses into the stack.

A *smaller* threshold reduces the accuracy of the stack walk by rejecting valid addresses from being placed in the callstack.

The default is a reasonably large 8192 bytes.

Alternative stack walk method - caveats

When the alternative stack walk is used you *may* notice some unusual data on the display:

- **<UNKNOWN> symbols in the middle of call stacks**

This happens rarely, because the address is not valid but for some reason was not rejected by the alternative stack walk.

- **Symbols in the middle of callstacks that you know cannot be correct**

This may happen because the address is valid, but not for this position in the callstack, and the address passed the alternative stack walk address verification tests - this address was most probably the target of an indirect CALL instruction, and as such, could not be verified.

- **Callstacks for data that make no sense**

This again is rare, but occurs due to Thread Validator monitoring its own behaviour (which can happen in a few limited circumstances). These callstacks are filtered in both stack walk methods, correctly in the standard one, but not perfectly in the alternative method!

Recommend usage

We recommend that in all situations the stack walking method used is either **DbgHelp StackWalk** or **Hybrid StackWalk.**

Only if you find your program has a problem that can be identified, but for which the callstack cannot be collected, do we recommend using **Hybrid StackWalk** or **Alternative StackWalk** as appropriate.

Reset

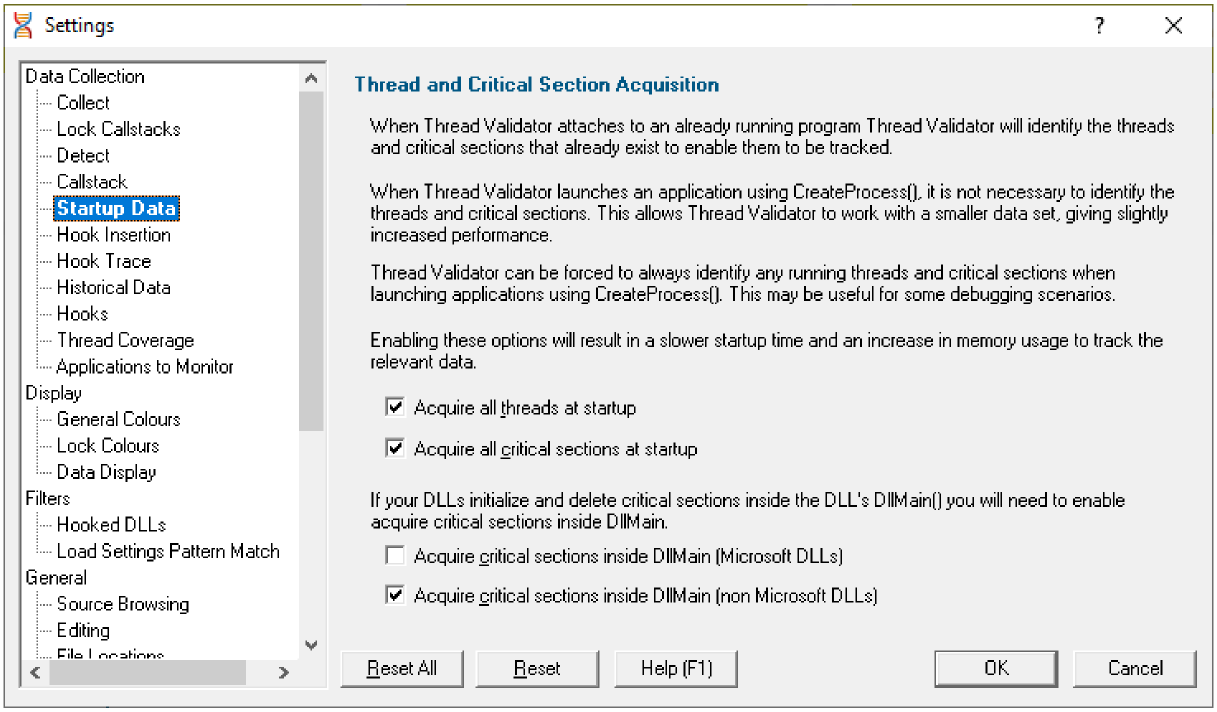
The Advanced Stack Walk dialog has a button labeled **Reset** at the bottom left of the dialog. This resets *only* the settings on this dialog back to their default values.

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

Reset - Resets the settings on the current page.

3.10.1.1.5 Startup Data

The **Startup Data** tab controls how to manage Critical Sections in DLLs loaded prior to Thread Validator's stub being loaded.



Thread and Critical Section Acquisition

When Thread Validator attaches to an already running program Thread Validator will identify the threads and critical sections that already exist to enable them to be tracked.

When Thread Validator launches an application using CreateProcess(), it is not necessary to identify the threads and critical sections. This allows Thread Validator to work with a smaller data set, giving slightly increased performance.

Thread Validator can be forced to always identify any running threads and critical sections when launching applications using CreateProcess(). This may be useful for some debugging scenarios.

Enabling these options will result in a slower startup time and an increase in memory usage to track the relevant data.

☑ Acquire all threads at startup

☑ Acquire all critical sections at startup

If your DLLs initialize and delete critical sections inside the DLL's DllMain() you will need to enable acquire critical sections inside DllMain.

☐ Acquire critical sections inside DllMain (Microsoft DLLs)

☑ Acquire critical sections inside DllMain (non Microsoft DLLs)

DLLs that are already loaded

When Thread Validator attaches to an already running process or is injected at the start of an application's execution, other DLLs have already been loaded into the target application's address space.

Any of those DLLs that use critical sections will have initialized and may even have entered those critical sections.

Thread Validator monitors the initialization and destruction of critical sections and depends on knowing about their existence prior to usage in order to detect many of its thread errors.

If Thread Validator doesn't see the initialization but then does see the critical section being used, a possible error condition will be raised.

To prevent this you can acquire information about all critical sections when Thread Validator starts the target application (or attaches to it). For consistency, information about all threads in the application can be obtained at the same time.

Typically, when launching the target application in the normal way, you don't need to acquire all critical sections at startup. However, if attaching to an already running application (using inject or wait for application), you should acquire all critical sections at startup.

Determining threads and critical sections existing at startup

Two options allow Thread Validator to determine all the threads or critical sections that exist when Thread Validator attaches to the target application:

- **Acquire all threads at startup** › check to determine all the threads that exist
- **Acquire all critical sections at startup** › check to determine all the critical sections that exist

Critical sections inside DllMain

If your DLLs initialize or destroy critical sections inside DllMain, there are two options for tracking these critical sections.

One method deals with Microsoft DLLs and the other with non-Microsoft DLLs.

- **Acquire critical sections inside DllMain...** › check to determine all the critical sections initialised when a DLL is loaded using `LoadLibrary`

  (Microsoft DLLs) › when a *Microsoft* DLL is loaded

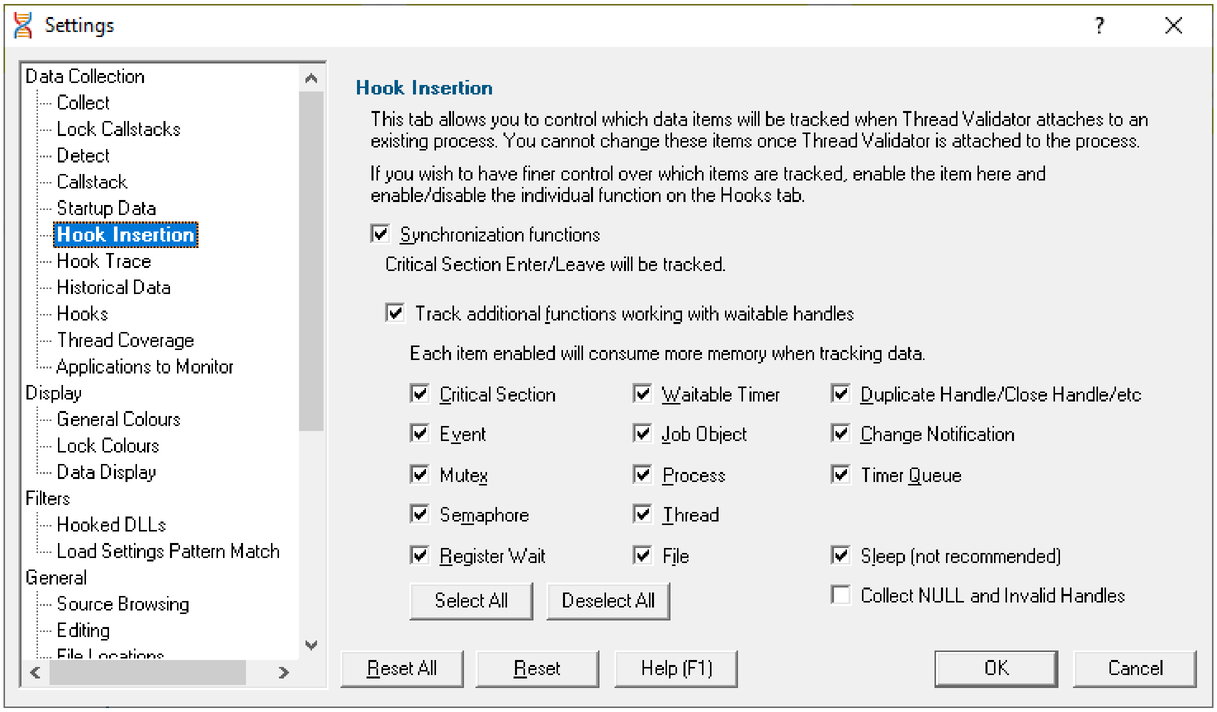
  (non Microsoft DLLs) › when a *non-Microsoft* DLL is loaded

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

Reset - Resets the settings on the current page.

3.10.1.1.6 Hook Insertion

The **Hook Insertion** tab allows you to control which functions are monitored by Thread Validator.



Hook Insertion

This tab allows you to control which data items will be tracked when Thread Validator attaches to an existing process. You cannot change these items once Thread Validator is attached to the process.

If you wish to have finer control over which items are tracked, enable the item here and enable/disable the individual function on the Hooks tab.

☑ Synchronization functions

Critical Section Enter/Leave will be tracked.

☑ Track additional functions working with waitable handles

Each item enabled will consume more memory when tracking data.

☑ Critical Section ☑ Waitable Timer ☑ Duplicate Handle/Close Handle/etc

☑ Event ☑ Job Object ☑ Change Notification

☑ Mutex ☑ Process ☑ Timer Queue

☑ Semaphore ☑ Thread

☑ Register Wait ☑ File ☑ Sleep (not recommended)

□ Collect NULL and Invalid Handles

Change hooks before a session

Before a session starts you can set which data items are hooked.

Hooking changes made mid session will come into effect on the next session, but you can enable or disable the data from installed hooks at any time using the hooks tab.

Hook Insertion

Two check boxes control which groups of hooks are inserted into your application.

- **Synchronization functions** ❯ inserts hooks for tracking synchronization objects into the target program

  If enabled, the default behaviour is that critical section enter and leave events will be tracked. Additional functions can then be specified below.

- **Functions working with waitable handles** ❯ hooks (below) for tracking functions that create, manage and destroy waitable handles will be inserted into the target program

  A few additional functions (Sleep, and critical section related functions) are also monitored.

  Specify which groups of functions should have data collected from them using the individual check boxes. (See note about memory below).

  Any collected data can be viewed on the Active Objects and Analysis tabs.

Groups of items	Functions that...
• **Critical Section**	initialize, modify and destroy
• **Event**	create, open and destroy
• **Mutex**	create, open and destroy
• **Semaphore**	create, open and destroy
• **Register Wait**	register and unregister
• **Waitable Timer**	create, open and destroy
• **Job Object**	create, open and destroy
• **Process**	create, open and destroy
• **Thread**	create, open and destroy
• **File**	create, open and destroy files (of any type)
• **Duplicate Handle**	miscellaneous handle functions
• **Change Notification**	create, open and destroy file change notifications
• **Timer Queue**	create, open and destroy timer queues and timer queue timers

Other options:

- **Sleep** `Sleep` and `SleepEx`

  For many applications, enabling `Sleep` will generate a lot of data, as various threads sleep from time to time.

- **Collect NULL and Invalid Handles** information about failed handle creation

  When collecting waitable handles, you may also want to collect information about failed handle creation using this option.

- **Select/Deselect All** switches all the above functions allocating waitable handles - i.e. not `Sleep` and invalid handles

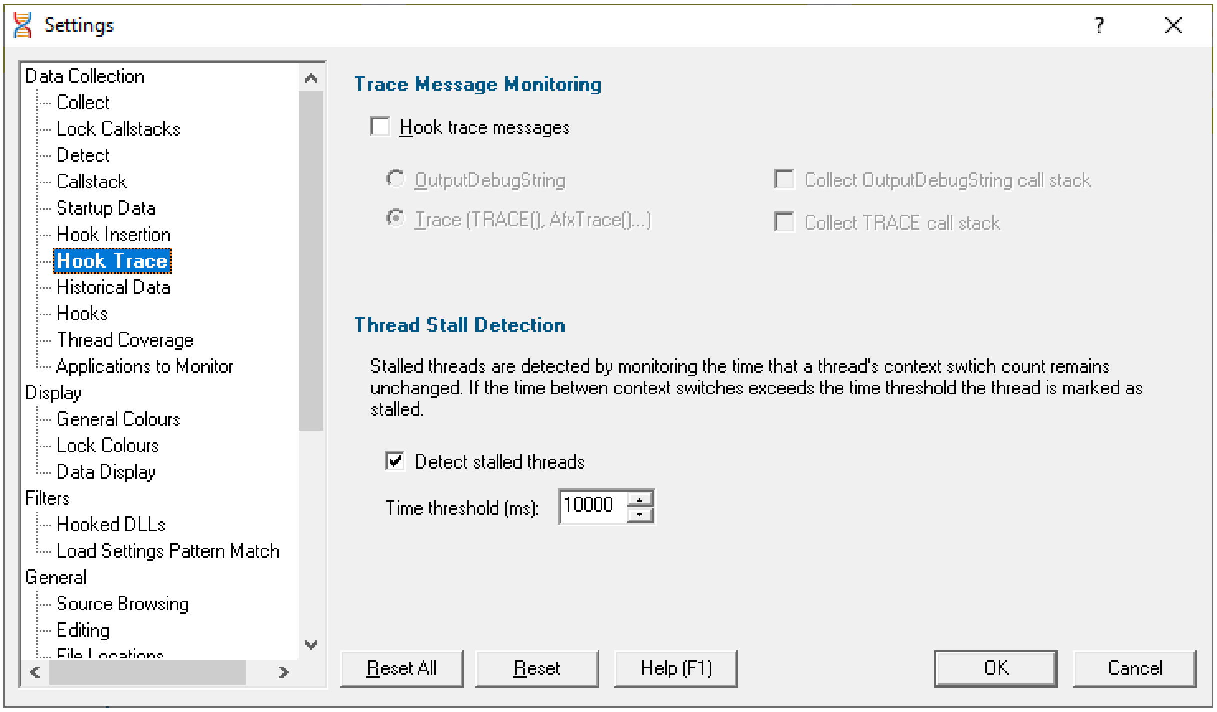
It's recommended that options in this section are only enabled if you need to monitor the data. If you subsequently find that memory consumption is very high, consider changing the settings here, or on the Historical Data tab.

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

Reset - Resets the settings on the current page.

3.10.1.1.7 Hook Trace

The **Hook Trace** tab allows you to control which debug or trace functions are monitored by Thread Validator.



Trace Message Monitoring

□ Hook trace messages

○ OutputDebugString □ Collect OutputDebugString call stack

● Trace (TRACE(), AfxTrace()...) □ Collect TRACE call stack

Thread Stall Detection

Stalled threads are detected by monitoring the time that a thread's context swtich count remains unchanged. If the time betwen context switches exceeds the time threshold the thread is marked as stalled.

☑ Detect stalled threads

Time threshold (ms): 10000

Trace Message Monitoring

Thread Validator can collect the output from various debug and trace functions or macros, including:

- `AfxTrace()`
- `TRACE()`
- `OutputDebugString()`

The `TRACE()` macros ultimately get routed via `OutputDebugString()`, so to avoid duplicate output Thread Validator doesn't allow both to be collected at the same time.

Collection is enabled via the main hook option, with output available on the Analysis tab.

- **Hook trace messages** ❯ collect `TRACE()` or `OutputDebugString()` messages

Once hooking is enabled (not the default), choose either:

- **OutputDebugString** ❯ collect `OutputDebugString()` messages

  Collect OutputDebugString call stack ❯ optionally also collect the callstack for each message

- **Trace (TRACE(), AfxTrace()...)** ❯ collect `TRACE()` messages

Collect TRACE call stack › optionally also collect the callstack for each message

Thread Stall Detection

Thread Validator can detect stalled threads by monitoring the time a thread's context switch count is unchanged.

When the time for which a thread's context switch count is unchanged exceeds a threshold, the thread is considered stalled.

In reality, there may be many reasons why a thread may be (or appear to be) stalled. It may be due to:

- a time consuming wait operation on a handle object
- a wait on an owned critical section
- a call to `Sleep()` or `SleepEx()`
- the thread having been suspended
- the thread being in a deadlock

Once a thread has been marked as stalled, then if the context switch count starts to change the thread will be marked as non-stalled.

Stalled threads are displayed using a different colour 🟧, as defined on the Lock Colours settings dialog.

- **Detect stalled threads** › enable thread stall detection

- **Time threshold (mS)** › enter the threshold time (in milliseconds)

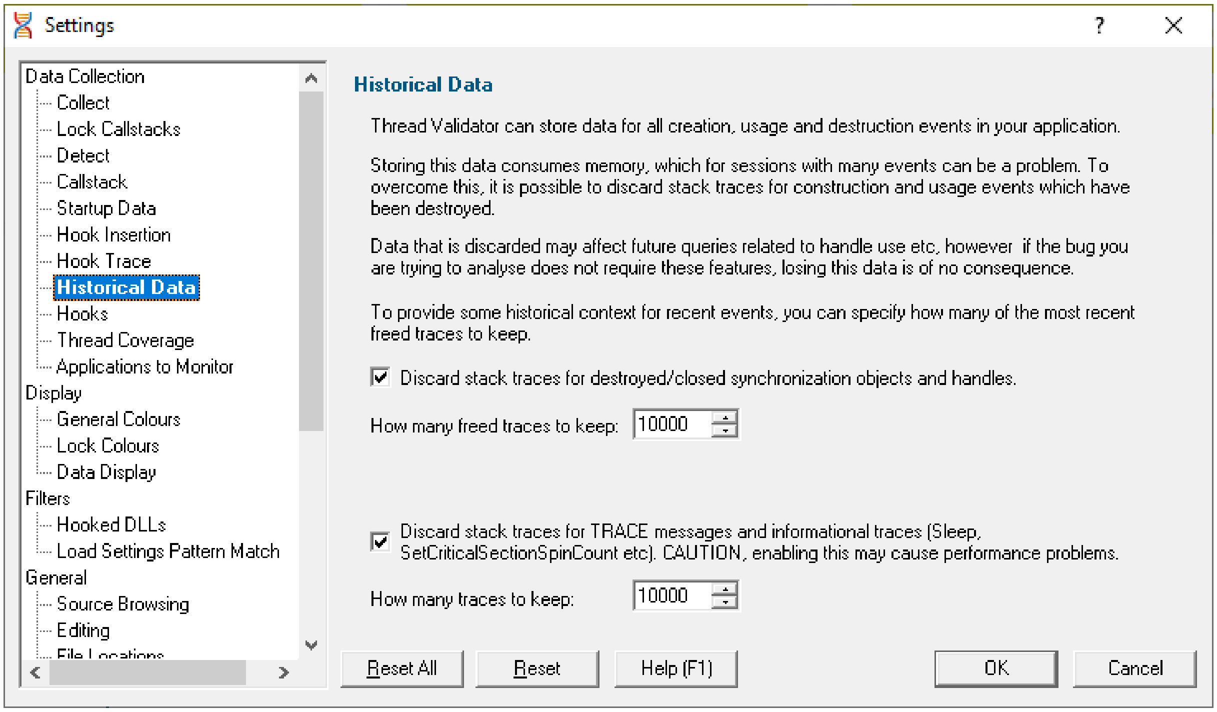
    The default is 10 seconds (10000ms)

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

Reset - Resets the settings on the current page.

3.10.1.1.8 Historical Data

The **Historical Data** tab allows you to control how many of the most recent stack traces are kept by Thread Validator.



Historical Data

Thread Validator can store data for all creation, usage and destruction events in your application.

Storing this data consumes memory, which for sessions with many events can be a problem. To overcome this, it is possible to discard stack traces for construction and usage events which have been destroyed.

Data that is discarded may affect future queries related to handle use etc, however if the bug you are trying to analyse does not require these features, losing this data is of no consequence.

To provide some historical context for recent events, you can specify how many of the most recent freed traces to keep.

☑ Discard stack traces for destroyed/closed synchronization objects and handles.

How many freed traces to keep: 10000

☑ Discard stack traces for TRACE messages and informational traces (Sleep, SetCriticalSectionSpinCount etc). CAUTION, enabling this may cause performance problems.

How many traces to keep: 10000

Historical Data

There may be occasions when you want control over when to keep or discard information on older data. This may depend on:

- your computer's RAM capacity
- virtual memory storage
- the task you are trying to complete
- the target program being inspected

Two options allow you to **Discard stack traces for...**

- **...destroyed/closed synchronization objects and handles** › discard information about these freed handles

  When selected you can specify **How many freed traces to keep**. The default is 10,000.

  Data is discarded on a first deallocated, first discarded basis, so most recent deallocations are always kept.

- **...TRACE messages and informational traces** › discard older information about these items, including Sleep()

  When selected you can specify **How many traces to keep**. The default is 10,000.

Note that the less information kept about deallocated data, the smaller the memory requirements placed on your computer.

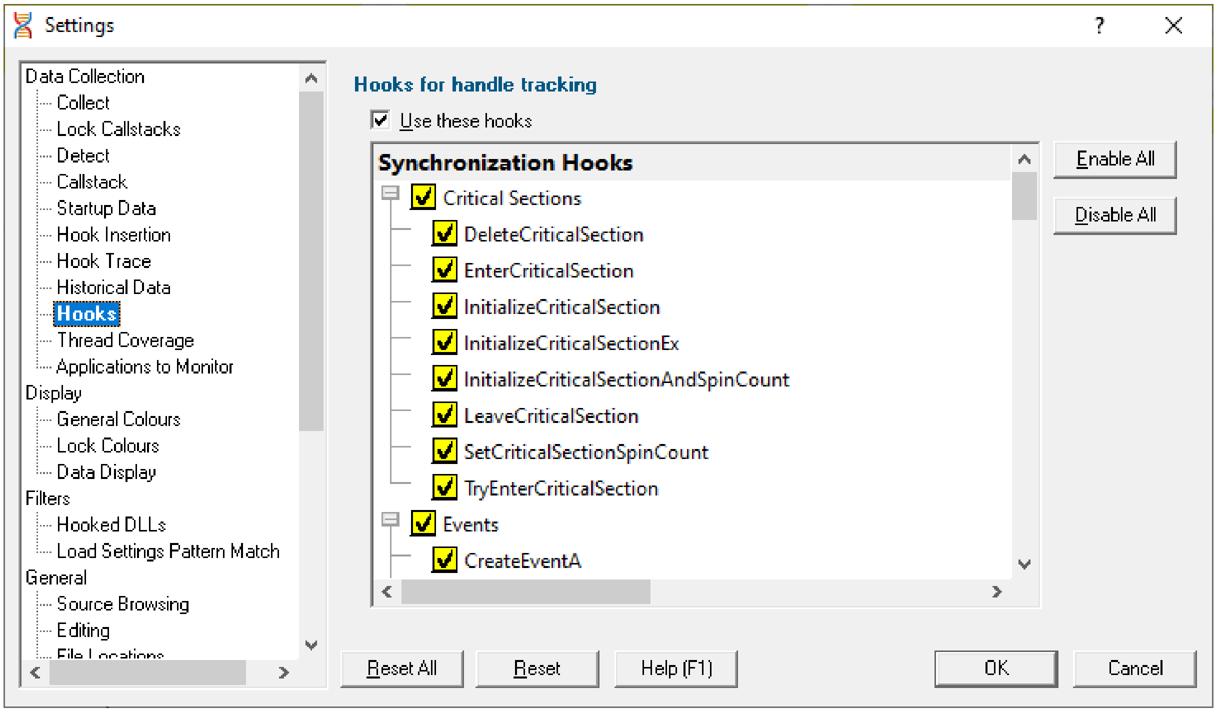
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.1.9 Hooks

The **Hooks** tab allows you to enable and disable every hook that Thread Validator uses (except for buffer checking hooks).

All the hooks can be enabled and disabled as a whole.



Hook Groups

The picture above shows the hook groups expanded to show that all their functions are enabled (ticked).

Tick or untick any synchronization hook to enable or disable its status. Use the top level items (Critical Sections, Events, etc) to set all the items in the corresponding group.

Enabling or disabling these hooks after the session has started *doesn't* modify behaviour during the current session.

- **Enable All** › enables all the hooks in all the groups

- **Disable All ›** disables all the hooks in all groups

For a full list of the hooks, see the hook reference.

- **Use these hooks ›** toggle the use of all selected hooks without losing the checked states

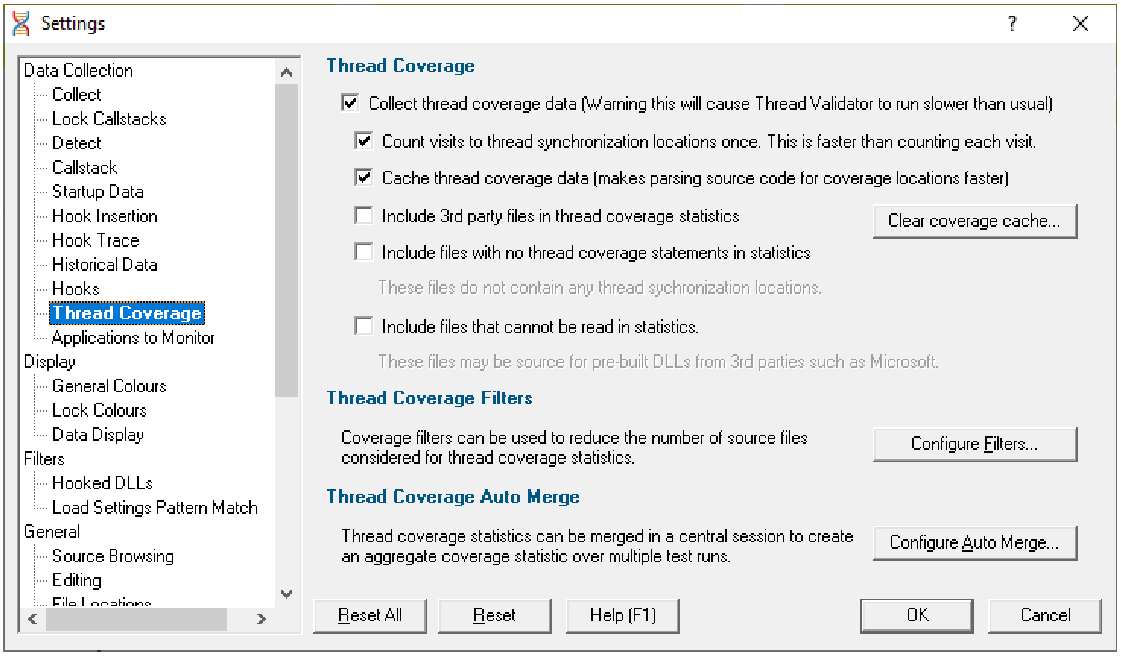
Useful if you've set up a mixture of hook states and want to temporarily disable them for a session.

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

Reset - Resets the settings on the current page.

3.10.1.1.10 Thread Coverage

The **Thread Coverage** tab allows you to control collection of coverage statistics for synchronization functions.



Thread Coverage

☑ Collect thread coverage data (Warning this will cause Thread Validator to run slower than usual)

☑ Count visits to thread synchronization locations once. This is faster than counting each visit.

☑ Cache thread coverage data (makes parsing source code for coverage locations faster)

☐ Include 3rd party files in thread coverage statistics

☐ Include files with no thread coverage statements in statistics

These files do not contain any thread sychronization locations.

☐ Include files that cannot be read in statistics.

These files may be source for pre-built DLLs from 3rd parties such as Microsoft.

Thread Coverage Filters

Coverage filters can be used to reduce the number of source files considered for thread coverage statistics.

Thread Coverage Auto Merge

Thread coverage statistics can be merged in a central session to create an aggregate coverage statistic over multiple test runs.

What is thread coverage?

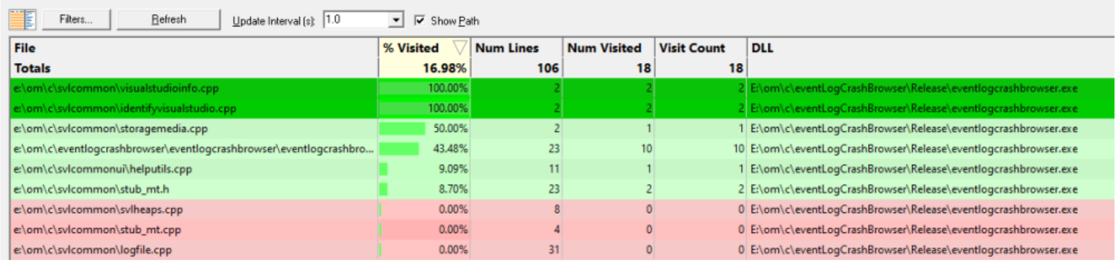
Calculating thread coverage involves parsing your source code to determine all thread synchronization locations and then matching that data with information from the callstacks of each allocation in your session.

Some of these activities are time consuming, so the default is *not* to collect thread coverage statistics, but to let you optionally control the behaviour as below.

Controlling thread coverage data collection

- **Collect thread coverage data** › enables the collection of thread coverage data

The Coverage tab (example shown below) shows the thread coverage statistics for each file that contains thread synchronization statements. This allows you to check how well you're testing your thread synchronization code.



Update Interval (s): 1.0 ☑ Show Path

File	% Visited	Num Lines	Num Visited	Visit Count	DLL
Totals	**16.98%**	**106**	**18**	**18**	
e:\om\c\svlcommon\visualstudioinfo.cpp	100.00%	2	2	2	E:\om\c\eventLogCrashBrowser\Release\eventlogcrashbrowser.exe
e:\om\c\svlcommon\identifyvisualstudio.cpp	100.00%	2	2	2	E:\om\c\eventLogCrashBrowser\Release\eventlogcrashbrowser.exe
e:\om\c\svlcommon\storagemedia.cpp	50.00%	2	1	1	E:\om\c\eventLogCrashBrowser\Release\eventlogcrashbrowser.exe
e:\om\c\eventlogcrashbrowser\eventlogcrashbrowser\eventlogcrashbro...	43.48%	23	10	10	E:\om\c\eventLogCrashBrowser\Release\eventlogcrashbrowser.exe
e:\om\c\svlcommonui\helputils.cpp	9.09%	11	1	1	E:\om\c\eventLogCrashBrowser\Release\eventlogcrashbrowser.exe
e:\om\c\svlcommon\stub_mt.h	8.70%	23	2	2	E:\om\c\eventLogCrashBrowser\Release\eventlogcrashbrowser.exe
e:\om\c\svlcommon\svlheaps.cpp	0.00%	8	0	0	E:\om\c\eventLogCrashBrowser\Release\eventlogcrashbrowser.exe
e:\om\c\svlcommon\stub_mt.cpp	0.00%	4	0	0	E:\om\c\eventLogCrashBrowser\Release\eventlogcrashbrowser.exe
e:\om\c\svlcommon\logfile.cpp	0.00%	31	0	0	E:\om\c\eventLogCrashBrowser\Release\eventlogcrashbrowser.exe

See the section at the end of this page about tracking waitable handles.

When the collection of thread coverage data is enabled, the are some additional options to tune the performance.

- **Count visits to synchronization locations once** › only count line visits once per line when this is selected (default)

  Normally each line is counted just once (faster).

  Although this option means they can be counted for all visits to each line, this will be noticeably slower for large applications.

- **Cache thread coverage data** › enable caching of coverage statement information (default)

  Calculating the coverage statements for each file can be time consuming, especially for large applications as it requires parsing the source code each time the application is run to determine where the thread synchronization locations are.

  Caching the coverage statement information improves this by only recalculating it when the source code is modified.

- **Include 3rd party files in thread coverage statistics** › include 3rd party files as specified in File Locations

  For most activities you'll want to keep this option deselected.

- **Include files with no thread coverage statements in statistics** › includes relevant source code files

Some source code files do not perform any thread synchronization operations. As such these files may be viewed as unwanted data on the coverage report, so the default is not to include these.

- **Include files that cannot be read in statistics** › includes source code filenames referencing files that do not exist on your computer

  Typically these are third party files in pre-built DLLs. By default they are not included in the statistics.

- **Clear Coverage Cache...** › delete any existing thread coverage cache files stored on your computer

Thread coverage filters

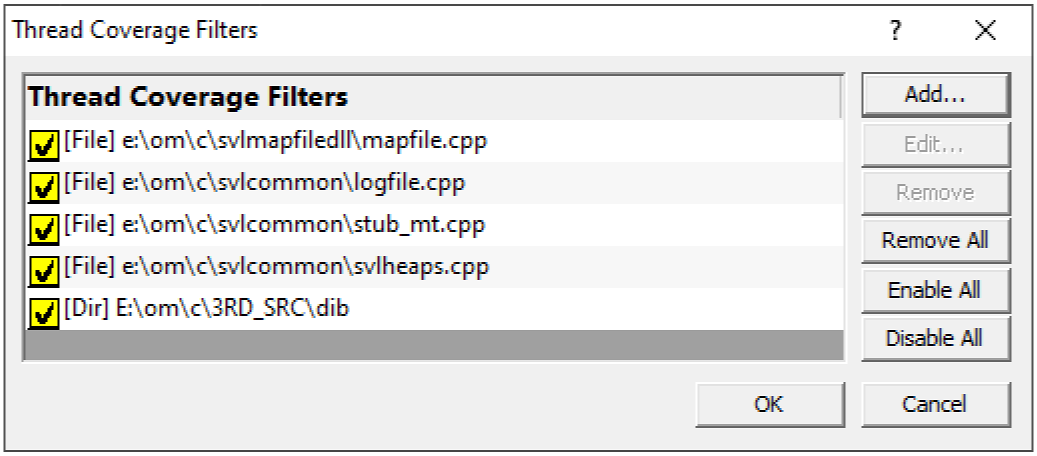
Thread coverage calculation is quite likely to include some third party source files or header files that are out of your control, e.g. from STL, third party files in pre-built DLLs, etc.

There may also be some files that create waitable handles but which you know will never be used in a thread synchronization operation.

If you don't want results to include such files, you can specify a number of filters to exclude them.

Coverage filters need to be set *before* the session starts in order to affect results.

- **Configure Filters...** › shows the Thread Coverage Filters dialog that lets you manage the list of filters...



Thread Coverage Filters

Thread Coverage Filters

☑ [File] e:\om\c\svlmapfiledll\mapfile.cpp
☑ [File] e:\om\c\svlcommon\logfile.cpp
☑ [File] e:\om\c\svlcommon\stub_mt.cpp
☑ [File] e:\om\c\svlcommon\svlheaps.cpp
☑ [Dir] E:\om\c\3RD_SRC\dib

This dialog is the same one accessed via the local Filters button on the Coverage tab.

The Filters dialog has some basic controls for managing the filters:

- **Add...** › adds a new filter, as in the example above
- **Edit...** or double click an item in the list › opens the Thread Coverage Filter dialog to modify the enabled state, type or target of the selected filter item

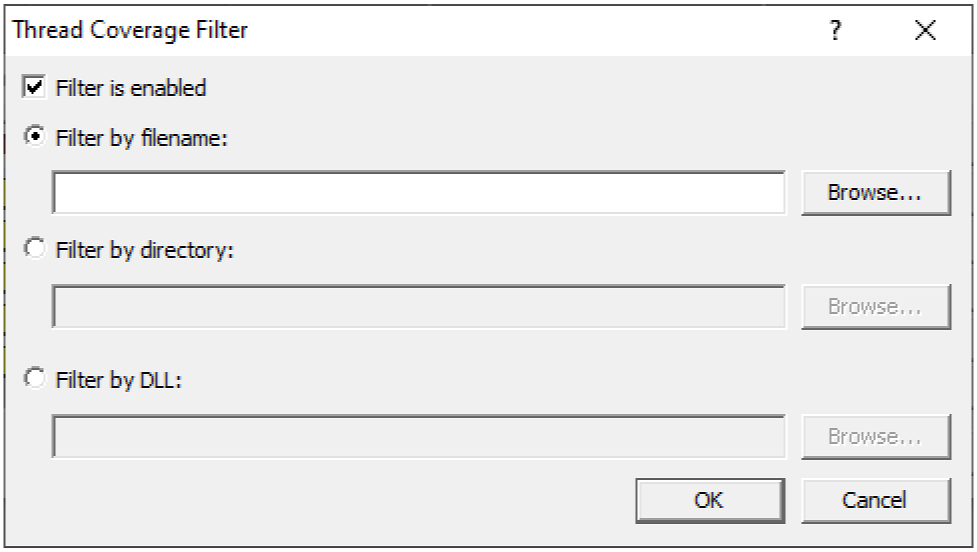
  You can also enable/disable an item via the yellow checkbox on each item in the list.

- **Remove** › deletes all *selected* filters in the list, or press Del
- **Remove All** › clears the list of filters
- **Enable All** › sets all filters active
- **Disable All** › sets all filters inactive (but does not remove them from the list)

Each filter can only be one of the three types: *filename*, *directory* or *DLL*. However, you can mix and match multiple filters of any type.

For example, to add a new filter:

- **Configure Filters...** › **Add...** › choose a filter type **filename**, **directory** or **DLL** › **Browse...** (or enter a path directly) › choose a relevant item to add › **OK** › **OK**



Thread Coverage Filter

☑ Filter is enabled

● Filter by filename:

○ Filter by directory:

○ Filter by DLL:

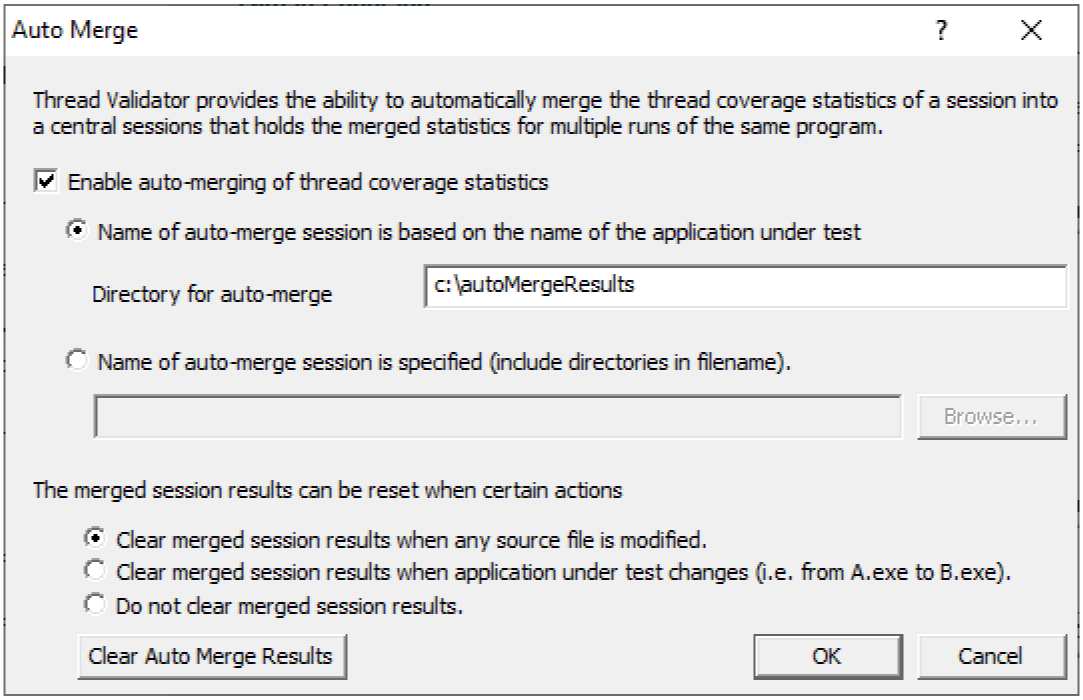
To help recognise types of filters in the list, each item is prefixed by some bracketed flags as follows

[File] - file
[Dir] - directory
[DLL] - DLL

Thread coverage auto merge

Different runs of your application may execute different parts of your application, in which case you might want to merge the results of one run with the results of another.

- **Configure Auto Merge...** › shows the Auto Merge dialog that lets you configure merging of thread coverage results



Auto Merge

Thread Validator provides the ability to automatically merge the thread coverage statistics of a session into a central sessions that holds the merged statistics for multiple runs of the same program.

☑ Enable auto-merging of thread coverage statistics

● Name of auto-merge session is based on the name of the application under test

Directory for auto-merge: c:\autoMergeResults

○ Name of auto-merge session is specified (include directories in filename).

The merged session results can be reset when certain actions

● Clear merged session results when any source file is modified.
○ Clear merged session results when application under test changes (i.e. from A.exe to B.exe).
○ Do not clear merged session results.

The automatic merging works by merging the results of each individual thread coverage session into a central session.

The central session is stored on disk in a file you specify, or in a file using the name of the session, e.g. `TestThis.exe` would get saved in `TestThis.tvm` in the same directory as Thread Validator resides.

- **Enable auto-merging of thread coverage statistics** › switches the merging feature on (default is off)

Depending on how many applications you are performing thread coverage on, you may want your thread coverage data to go to one central location or to a different location for each application under test.

- **Name of auto-merge session is based on the name of the application under test** › saves the central session in a file named according to the application under test in this session (the default)

  By default, the auto-merge session will be stored in the same directory as Thread Validator, but you can change this:

  Directory for auto-merge session › saves the auto-merge session in the specified directory

  For example, if you run the application `nativeExample.exe`, and specify a central session directory of `e:\threadCoverageResults`, the central session will be saved in a file named `e:\threadCoverageResults\nativeExample.tvm`.

- **Name of central session is specified** › save the auto-merge session in a filename and path of your choice (enter or **Browse...** to a file)

Auto-merge session reset

The auto-merge results can be automatically cleared by certain triggers, or not cleared at all.

When performing thread coverage analysis sometimes you will uncover a bug in your software and need to modify the software, and/or run different executables. When this happens, line numbers and/or files often change, and you usually wouldn't want to merge thread coverage data from the modified software with existing coverage data.

The triggers for clearing the merged session results are:

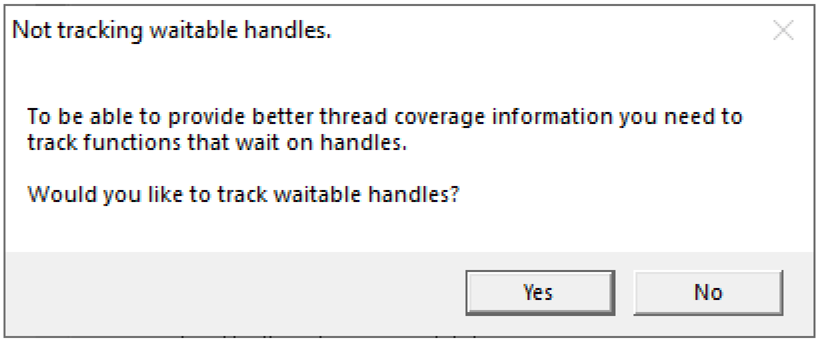
- When any **source file** is modified (the default)
- When the **application** under test changes
- **No clearing** of merged session results occurs under any circumstance

If you don't want an automatic trigger, there's also a manual trigger:

- **Clear Auto Merge Results** › click to clear auto-merged results at any time of your choosing

Not tracking waitable handles?

If you choose to collect thread coverage data, and close the settings dialog (or switch to another settings page), you may get the following offer to improve coverage by tracking waitable handles.



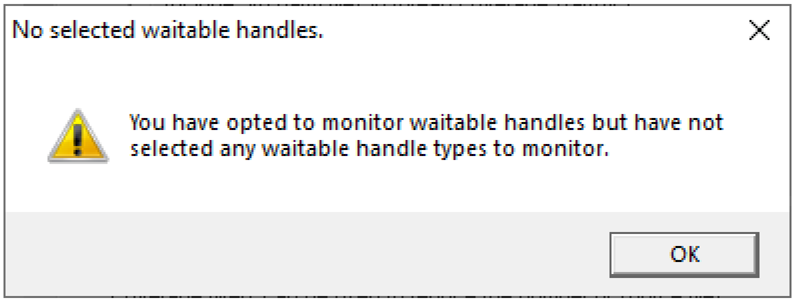
Not tracking waitable handles.

To be able to provide better thread coverage information you need to track functions that wait on handles.

Would you like to track waitable handles?

- **Yes** › switches on the Hook Insertion setting for tracking functions working with waitable handles

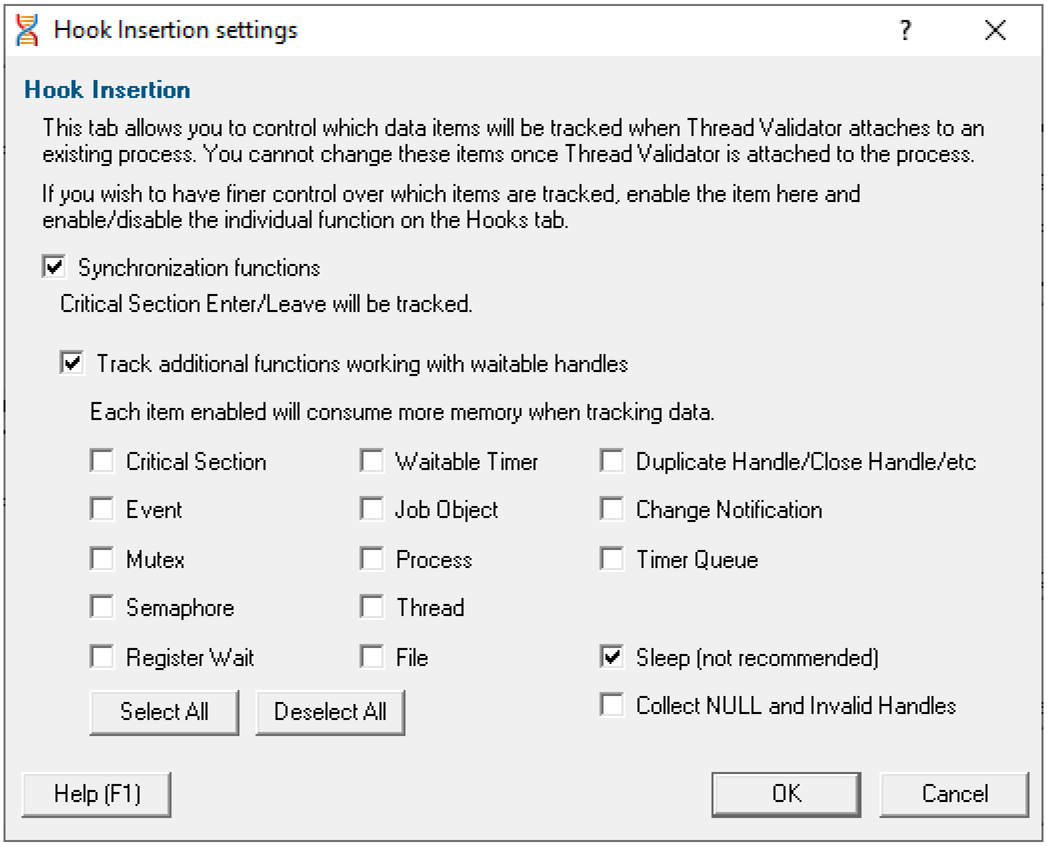
  If you haven't actually enabled any of the waitable handles, a warning will appear:


No selected waitable handles.

You have opted to monitor waitable handles but have not selected any waitable handle types to monitor.

OK › displays the Hook Insertion settings to choose which waitable handles you want to track

Pick which waitable handle functions you're intertested in for coverage and click **OK**


Hook Insertion settings

Hook Insertion

This tab allows you to control which data items will be tracked when Thread Validator attaches to an existing process. You cannot change these items once Thread Validator is attached to the process.

If you wish to have finer control over which items are tracked, enable the item here and enable/disable the individual function on the Hooks tab.

☑ Synchronization functions

Critical Section Enter/Leave will be tracked.

☑ Track additional functions working with waitable handles

Each item enabled will consume more memory when tracking data.

☐ Critical Section
☐ Event
☐ Mutex
☐ Semaphore
☐ Register Wait
☐ Waitable Timer
☐ Job Object
☐ Process
☐ Thread
☐ File
☐ Duplicate Handle/Close Handle/etc
☐ Change Notification
☐ Timer Queue
☑ Sleep (not recommended)
☐ Collect NULL and Invalid Handles

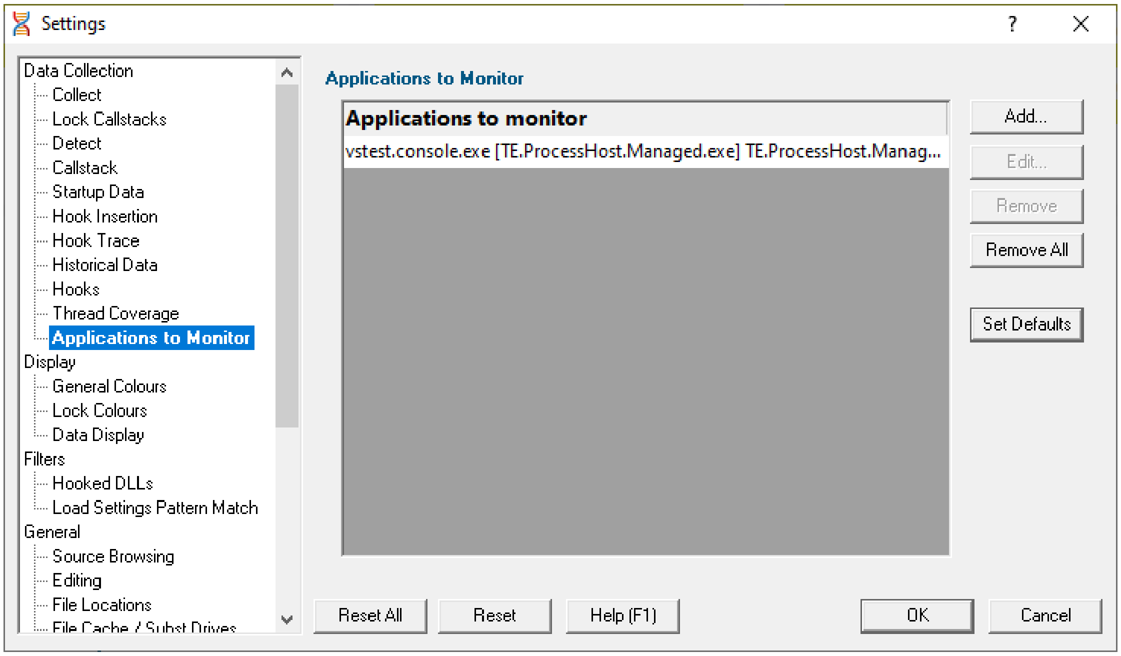
Alternatively if you don't want to get coverage after all, **Cancel** and then uncheck **Collect thread coverage data** to prevent these warnings.

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

Reset - Resets the settings on the current page.

3.10.1.1.11 Applications to Monitor

If your target program launches other child applications then the **Applications to Monitor** page lets you choose which ones to monitor.



Applications to monitor
vstest.console.exe [TE.ProcessHost.Managed.exe] TE.ProcessHost.Manag...

Monitoring child applications

You may have a case where the program you need to start is not the one you are interested in.

Your program may launch child applications and it may be one of *those* that you want to monitor with Thread Validator.

An example might be for unit testing where a test program spawns one or more child applications, or it might launch the same application multiple times.

The applications to monitor

The main list of **Applications to monitor** shows programs you may want to launch and the child applications they subsequently start - i.e. the you may be interested in monitoring.

Once a definition has been added, you can then use the Application to Monitor setting on the Launch Dialog or wizard to choose which of these child applications you actually want to monitor in a given session.

Managing the applications to monitor

The list contains a set of definitions - each one being for a different launch program.

For each launch program you can set the child applications you might want to monitor later.

An application is defined by its type (native and .Net, or .Net Core), the application executable name, and for .Net Core applications an additional application DLL that is used to identify the application.

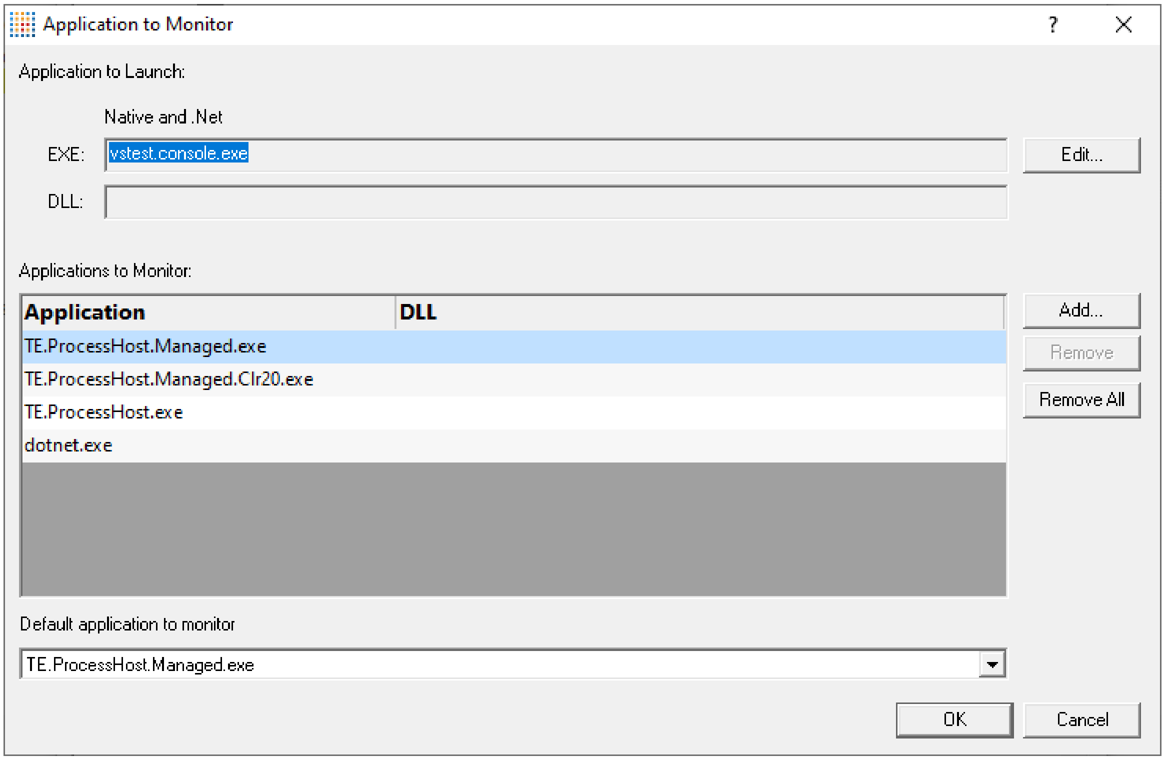
- **Add** › add a new module definition using the **Application to Monitor** dialog below
- **Edit** › modify a selected definition in the list, using the **Application to Monitor** dialog again
- **Remove** › removes any selected definitions in the list
- **Remove All** › clears the list
- **Set Defaults** › reset the list of known applications to those as configured with a new install of Thread Validator

  The defaults are currently setup for Microsoft's Visual Test software `vstest.console.exe`.

The Application to Monitor dialog

The **Application to Monitor** dialog lets you define or edit a launch program and it's child applications.

The values you specify here are the ones used on the launch dialog and launch wizard to customize which application actually gets monitored.



Application to Monitor

Application to Launch:

Native and .Net

EXE: vstest.console.exe

DLL:

Applications to Monitor:

Application	DLL
TE.ProcessHost.Managed.exe	
TE.ProcessHost.Managed.Clr20.exe	
TE.ProcessHost.exe	
dotnet.exe	

Default application to monitor

TE.ProcessHost.Managed.exe

- **Application to Launch** › **Edit...** to select the initial starting application that will be *launching* the applications you want to monitor

  Any executable names found in the selected program will automatically be displayed in the list of **Applications to Monitor**.

  If you don't wish to use these automatic names you can **Remove** them.

- **Add** › add an additional application that you know will be started by the launch program

  Child applications that you add are used *without* the path.

  Excluding the path gives more scope for matching launched application names if they are launched with a different path.

- **Remove** › removes any selected applications in the list

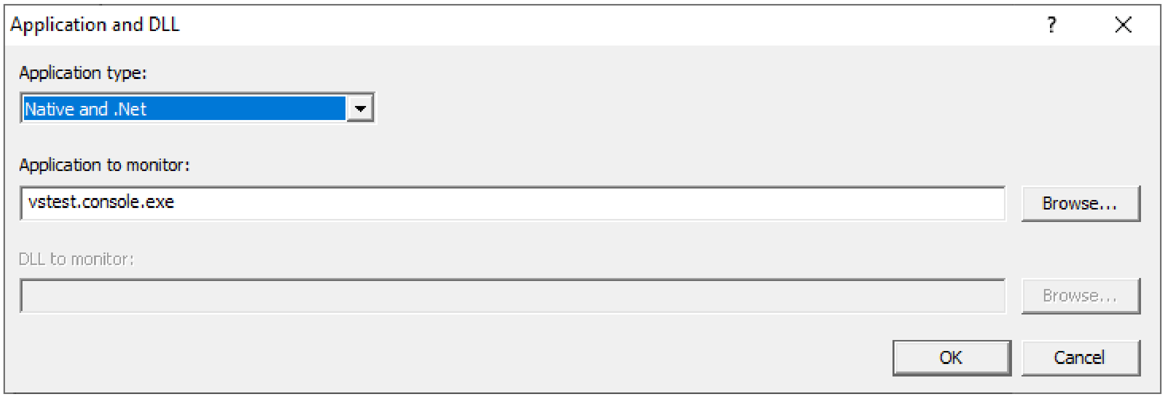
- **Remove All** › clears the list

- **Default application to monitor** › choose the appropriate item to be the default item

  The default application will be selected on the launch dialog (or wizard) whenever the *start* program is specified as the one at the top of this dialog.

The Application and DLL dialog

The **Application and DLL** dialog lets you define or edit a launch program and a launch DLL.



Application and DLL

Application type: Native and .Net

Application to monitor: vstest.console.exe

DLL to monitor:

- **Application type** › choose the type of application

  - Native and .Net
  - .Net Core (Framework Dependent)
  - .Net Core (Self Contained)

- **Application to monitor** › edit or **Browse...** the application EXE to monitor.

This can be an executable name or the full path to the executable. For example **test.exe** or **c:\unitTests\test.exe**.

For native applications this is the application executable.

For .Net Framework applications this is the application executable.

For .Net Core Framework-dependent applications this is most likely going to be **c:\program files\dotnet\dotnet.exe**.

For .Net Core Self-contained applications this is the application executable.

- **DLL to monitor** › edit or **Browse...** the application DLL to monitor. This field is only needed for .Net Core applications.

  This can be an executable name or the full path to the executable. For example **test.dll** or **c:\unitTests\test.dll**.

  For native applications this is not used.

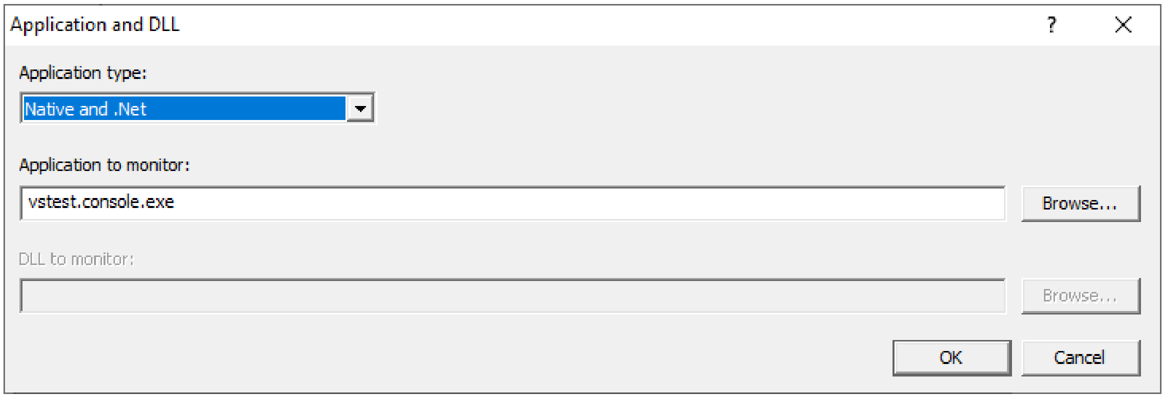
  For .Net Framework applications this is not used.

  For .Net Core Framework-dependent applications this is the application dll. (the name of the dll that you would pass to dotnet.exe on the command line).

  For .Net Core Self-contained applications this is the dll that has the same name as the application executable. (for theApp.exe, the dll name is theApp.dll).

Example Dialogs

Native



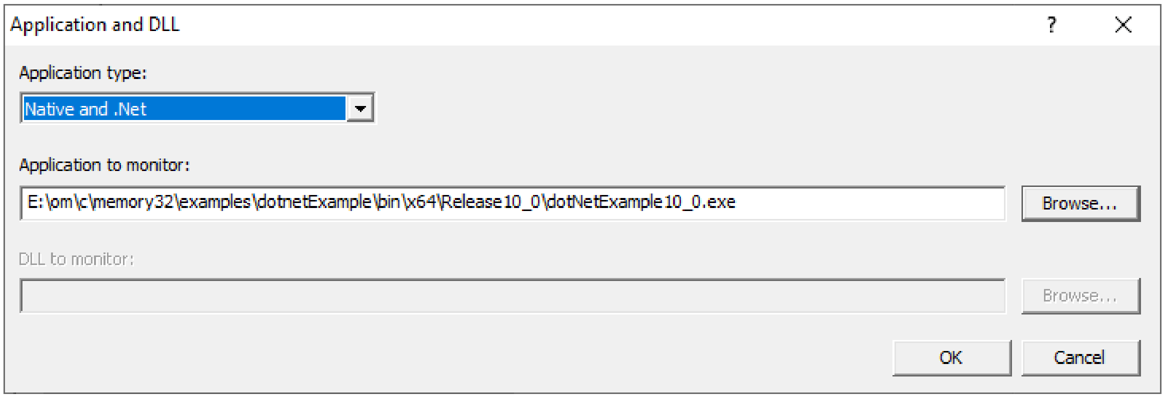
Application and DLL

Application type: Native and .Net

Application to monitor: vstest.console.exe

DLL to monitor:

.Net

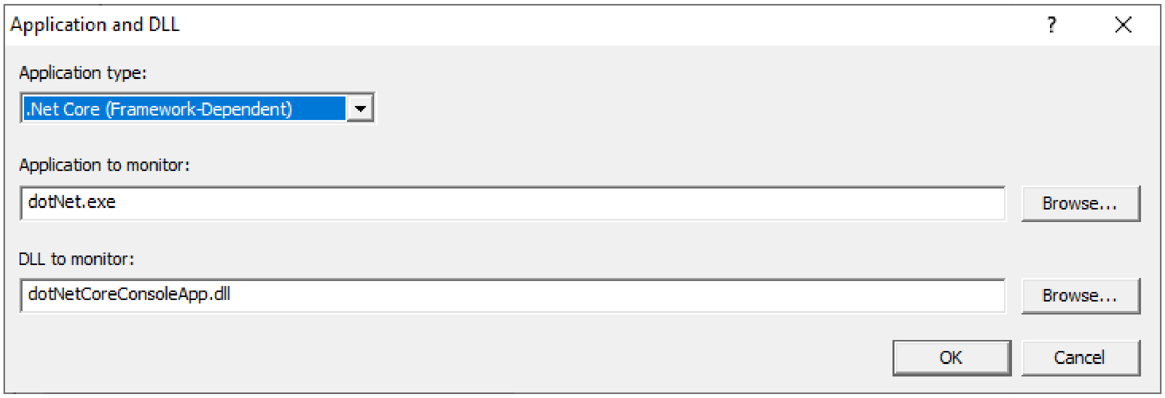


Application type: Native and .Net

Application to monitor: E:\om\c\memory32\examples\dotnetExample\bin\x64\Release10_0\dotNetExample10_0.exe

DLL to monitor:

.Net Core (Framework-dependent)

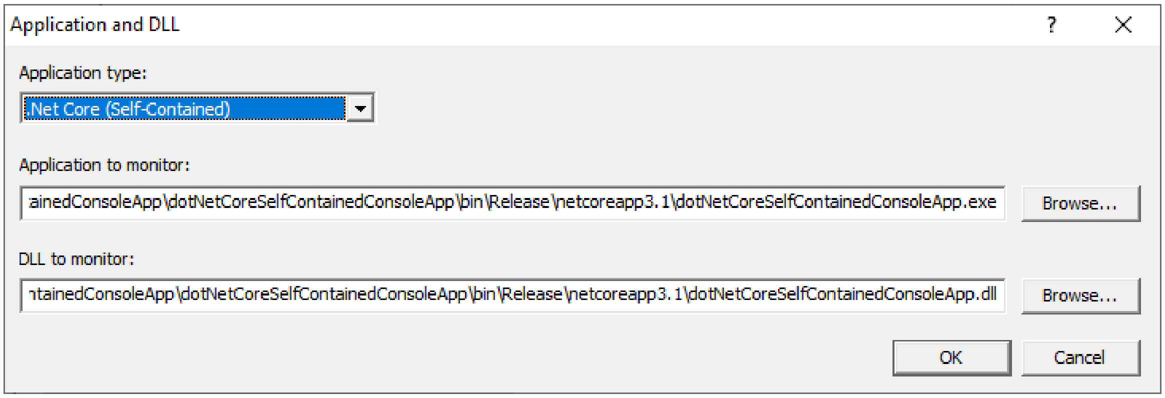


Application type: .Net Core (Framework-Dependent)

Application to monitor: dotNet.exe

DLL to monitor: dotNetCoreConsoleApp.dll

.Net Core (Self-contained)



Application type: .Net Core (Self-Contained)

Application to monitor: ainedConsoleApp\dotNetCoreSelfContainedConsoleApp\bin\Release\netcoreapp3.1\dotNetCoreSelfContainedConsoleApp.exe

DLL to monitor: ntainedConsoleApp\dotNetCoreSelfContainedConsoleApp\bin\Release\netcoreapp3.1\dotNetCoreSelfContainedConsoleApp.dll

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

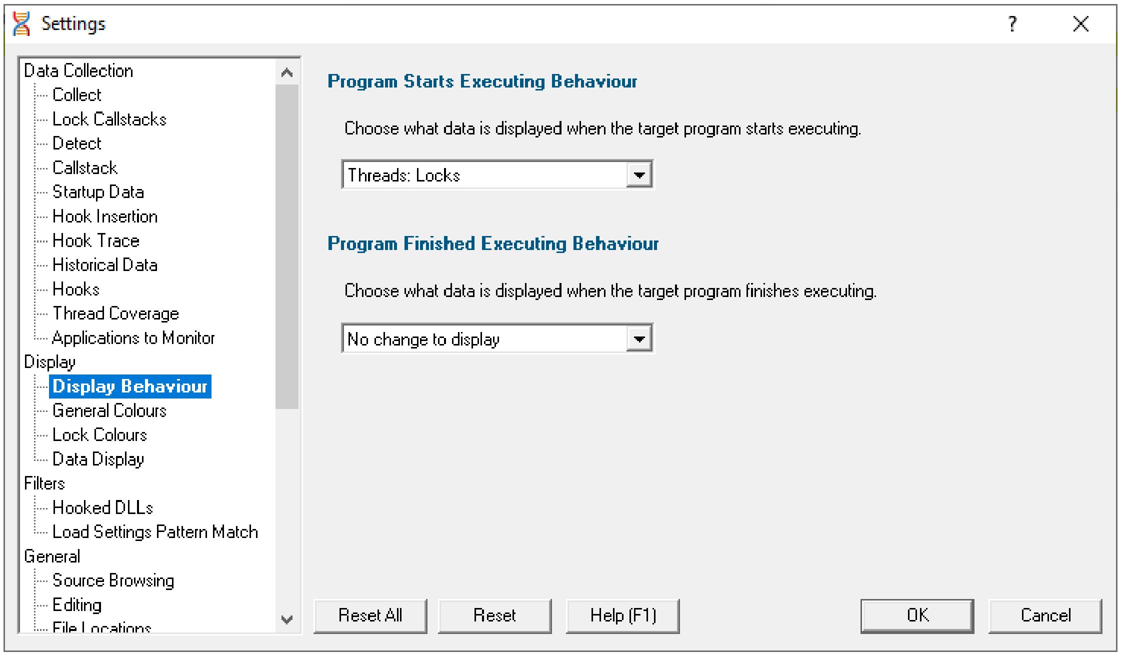
Reset - Resets the settings on the current page.

3.10.1.2 Display

3.10.1.2.1 Display Behaviour

The **Display Behaviour** tab allows you control how which displays are shown when a program starts executing and when a program finishes executing.

The default options are shown below:



Program Starts Executing Behaviour

Choose what data is displayed when the target program starts executing.

Threads: Locks

Program Finished Executing Behaviour

Choose what data is displayed when the target program finishes executing.

No change to display

Thread Validator can change the current display to any of the following displays.

- **Summary** › the main display
- **Locks : All Locks** › all locks in the application
- **Locks : Locks per Thread** › locks displayed per thread that uses them
- **Locks : Active Locks** › locks that are currently locked
- **Wait Chains** › locks and waits and their relations to each other
- **Threads : Locks** › thread history, plus locks for a given time segment
- **Threads : Lock Order** › thread history, plus the order locks were acquired for a given time segment
- **Analysis : Query** › perform queries on the locking data
- **Analysis : Coverage** › code coverage for locking primitives
- **Analysis : Active Objects** › locking primitives and handles and the callstack for the creation of the handle
- **Analysis : Object Handles** › handles open in the target application
- **Diagnostic : Diagnostic** › diagnostic information
- **Diagnostic : Stdout** › text collected from stdout
- **Diagnostic : Environment Variables** › environment variables from the program under test
- **Diagnostic : Child Processes** › processes launched by the program under test

Program Starts Executing Behaviour

When Thread Validator starts monitoring the behaviour of an application the current display can automatically be switched to any of the displays listed above.

There is also the option not to change the display.

Program Finished Executing Behaviour

When Thread Validator has finished processing all the information from the target application the current display can automatically be switched to any of the displays listed above.

There is also the option not to change the display.

The type of display that may interesting for collected data depends on the type of program that was executed. Native, .Net or Mixed mode. To accommodate this we provide one setting for each of the three program types.

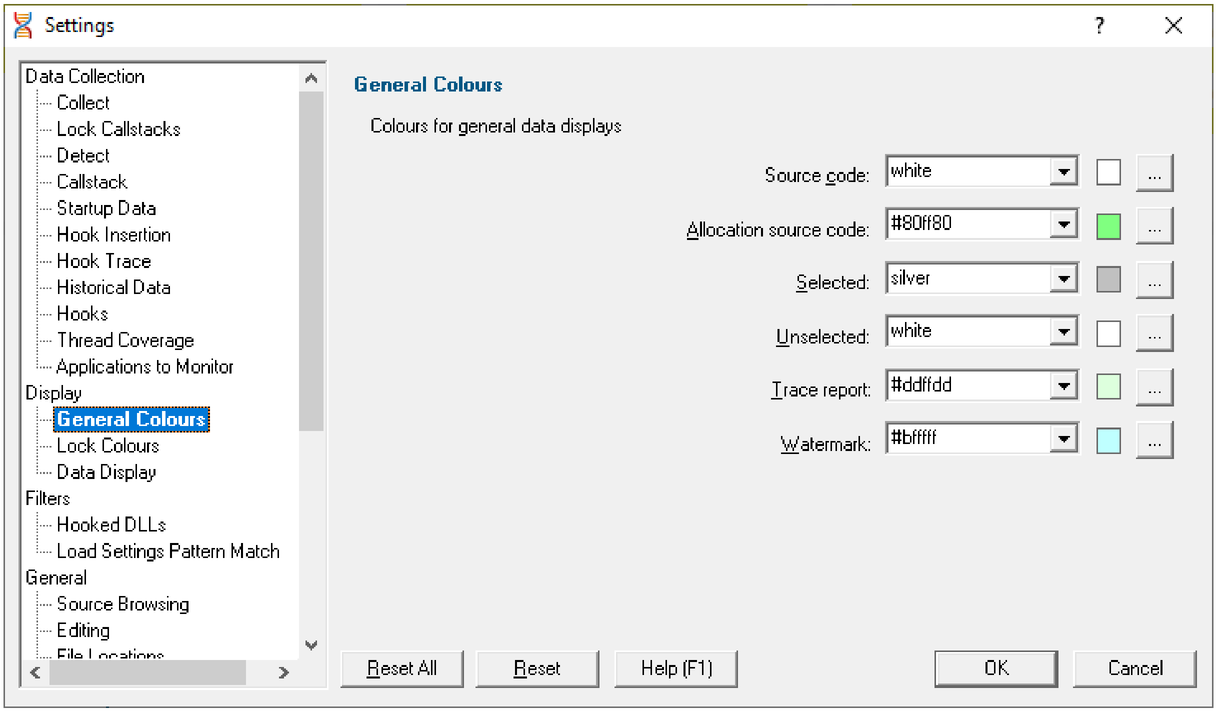
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.2.2 General Colours

The **General Colours** tab allows you to specify the colours that will be used to display source code and non-error data.

The default colours are shown below:



General Colours

Colours for general data displays

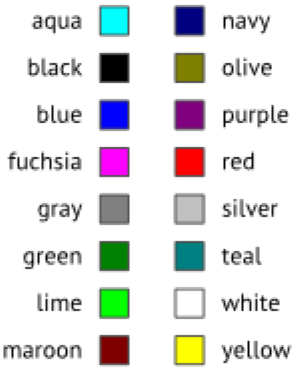
Source code:	white	□	...
Allocation source code:	#80ff80	🟩	...
Selected:	silver	■	...
Unselected:	white	□	...
Trace report:	#ddffdd	🟩	...
Watermark:	#bfffff	🟦	...

The colours are used for highlighting the source code on most of the main tabs, as well as in some of the statistics.

Changing display colours

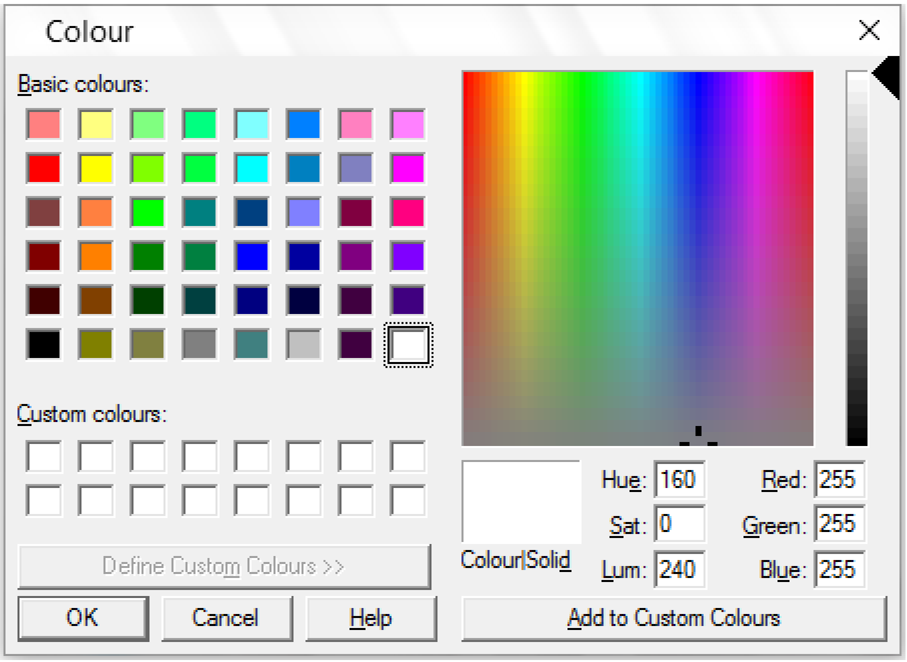
For each colour you can choose a predefined colour or make your own:

- Use the drop-down list white ▼ › pick one of 16 predefined colours below



aqua	🟦	■	navy
black	■	■	olive
blue	🟦	🟪	purple
fuchsia	🟪	🟥	red
gray	■	■	silver
green	🟩	■	teal
lime	🟩	□	white
maroon	🟥	🟨	yellow

- Click the ... button › edit the colour using the standard colour dialog:



Hue: 160
Sat: 0
Lum: 240
Red: 255
Green: 255
Blue: 255

Highlighting data that won't be shown on the next run

You can use some settings, such as the Thread Coverage settings to exclude files or directories of source code from being analyzed or displayed.

If you've already got coverage data being displayed, and you filter out some of that data, it is not removed from statistics in the current session.

However, the data that would be excluded is highlighted using the *Selected* colour (light grey in the above example).

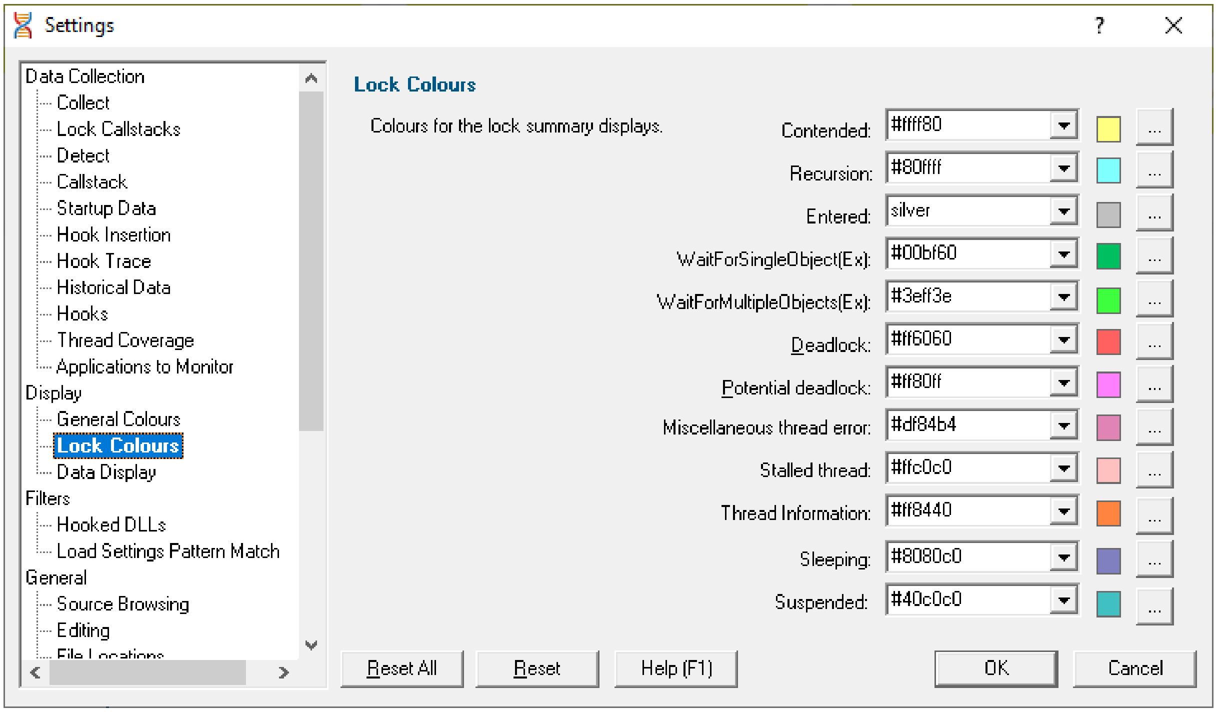
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.2.3 Lock Colours

The **Lock Colours** tab allows you to specify the colours that will be used to represent each type of data item collected by Thread Validator.

The default colours are shown below:



Lock Colours

Colours for the lock summary displays.

Contended:	#ffff80	🟨
Recursion:	#80ffff	🟦
Entered:	silver	⬜
WaitForSingleObject(Ex):	#00bf60	🟩
WaitForMultipleObjects(Ex):	#3eff3e	🟩
Deadlock:	#ff6060	🟥
Potential deadlock:	#ff80ff	🟪
Miscellaneous thread error:	#df84b4	🟪
Stalled thread:	#ffc0c0	🟥
Thread Information:	#ff8440	🟧
Sleeping:	#8080c0	🟦
Suspended:	#40c0c0	🟦

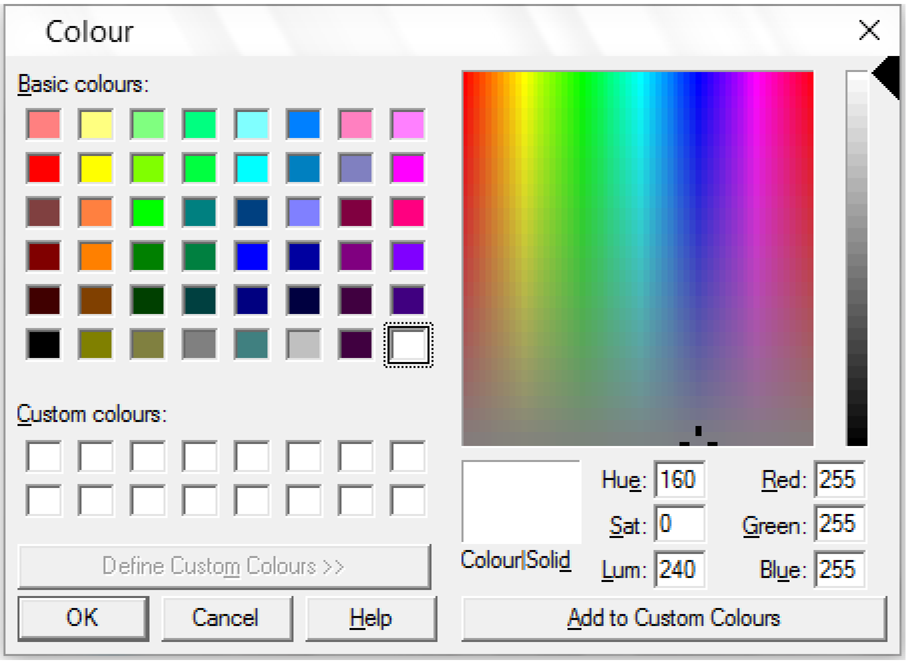
Changing lock colours

For each colour you can choose a predefined colour or make your own:

- Use the drop-down list white ▼ > pick one of 16 predefined colours below

aqua	🟦	🟦	navy
black	⬛	🟫	olive
blue	🟦	🟪	purple
fuchsia	🟪	🟥	red
gray	◼	⬜	silver
green	🟩	🟦	teal
lime	🟩	⬜	white
maroon	🟥	🟨	yellow

- Click the ... button > edit the colour using the standard colour dialog:



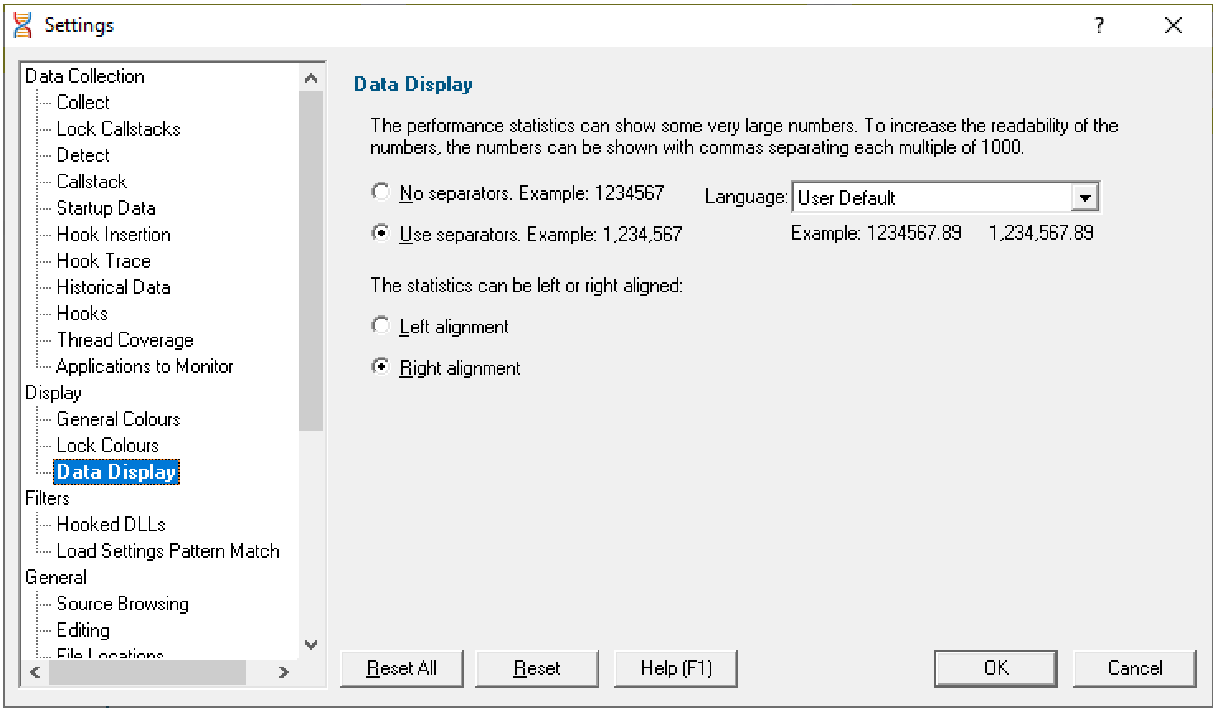
Hue: 160
Sat: 0
Lum: 240
Red: 255
Green: 255
Blue: 255

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

Reset - Resets the settings on the current page.

3.10.1.2.4 Data Display

The **Data Display** page allows you to specify how numeric data is formatted in the various displays.



Data Display

The performance statistics can show some very large numbers. To increase the readability of the numbers, the numbers can be shown with commas separating each multiple of 1000.

○ No separators. Example: 1234567

● Use separators. Example: 1,234,567

Language: User Default

Example: 1234567.89 1,234,567.89

The statistics can be left or right aligned:

○ Left alignment

● Right alignment

Numerical data format

Numeric data on the displays can contain some pretty big values, up to 2^64-1 if needed!

Long values are hard to read without grouping digits. To improve readability Thread Validator can delimit each group of three digits, so 1234567 becomes 1,234,567 for example.

- **No separators / Use separators** › Choose whether to group digits

The format used to delimit digit groups is set to **User Default**, which uses your computer's current locale, but you can change the format to suit another language if you wish.

- **Language** › Format numbers according to the default locale, or choose another language

Numerical data alignment

Numeric data on the displays can be aligned left or right, with right aligned numbers being more easily compared.

- **Left alignment / Right alignment** › Choose preferred alignment

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

Reset - Resets the settings on the current page.

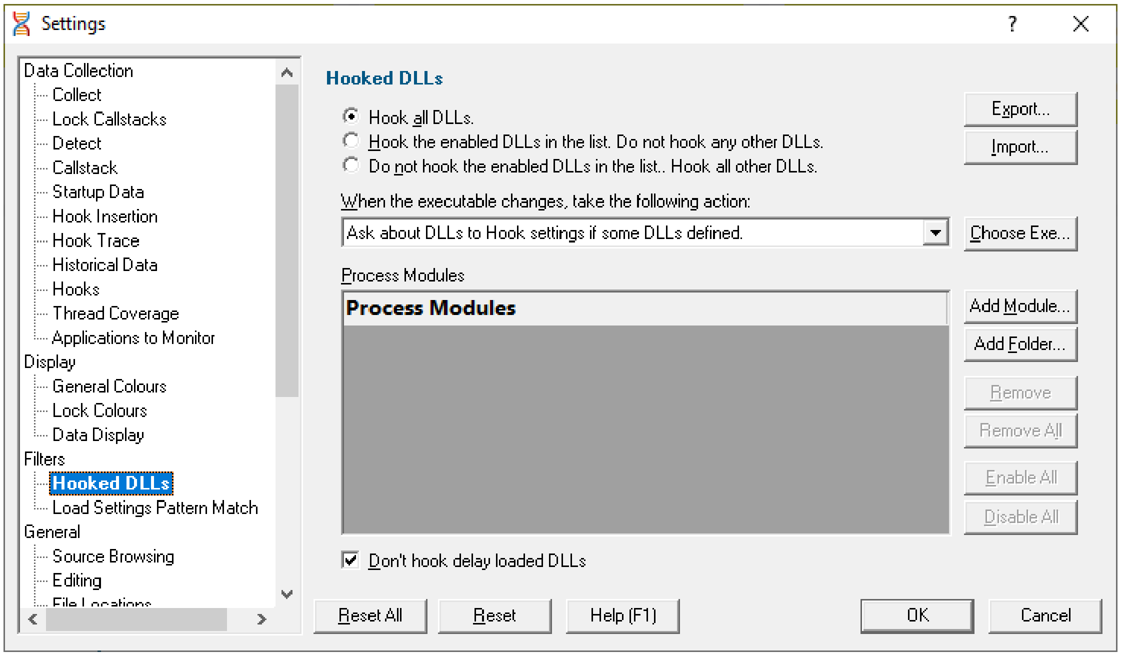
3.10.1.3 Filters

3.10.1.3.1 Hooked DLLs

The **Hooked DLLs** tab allows you to specify which DLLs should be hooked or not.

The default is simply to hook everything.

Read on, or click on a setting in the picture below to find out more.



Hooked DLLs

● Hook all DLLs.
○ Hook the enabled DLLs in the list. Do not hook any other DLLs.
○ Do not hook the enabled DLLs in the list.. Hook all other DLLs.

When the executable changes, take the following action:
Ask about DLLs to Hook settings if some DLLs defined.

Process Modules

Process Modules

☑ Don't hook delay loaded DLLs

Which DLLs to hook - the hooking rule

By default, Thread Validator will try to hook all DLLs and .EXEs used by your application, but you can choose to list only those which should be included or excluded:

- **Hook all DLLs** > hook everything - ignoring the settings in the list
- **Hook the enabled DLLs in the list** > hook only the ticked modules listed
- **Do not hook the enabled DLLs in the list** > *ignore* all the ticked modules in the list, and hook everything else

Populating the Process Modules list

The **Process Modules** list should specify the following items to be included or excluded from hooking in the target application

- DLLs
- .EXEs
- folders *containing* DLLs and .EXEs

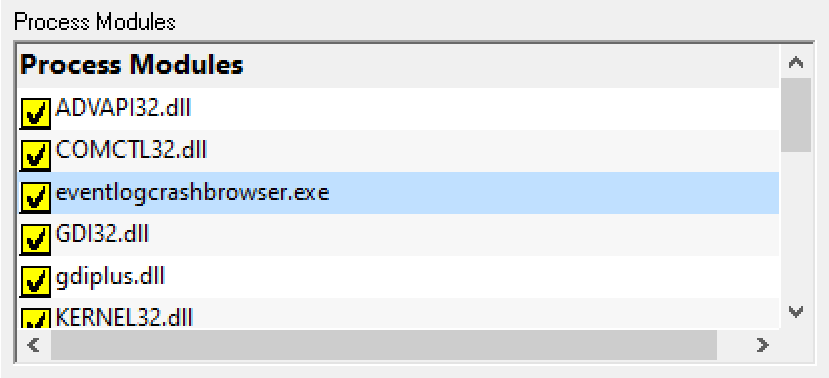
Initially the list is empty as the default option is to hook all DLLs and ignore the list. You can add modules to the list by:

- automatically adding modules on which your application is dependent
- manually adding modules or folders
- editing modules or folders already in the list

Automatic module addition

You can automatically populate the list with all the dependent modules for your application:

- **Choose Exe...** › navigate to your application and click **Open** › all the process modules appear in the list



Process Modules

Process Modules
☑ ADVAPI32.dll
☑ COMCTL32.dll
☑ eventlogcrashbrowser.exe
☑ GDI32.dll
☑ gdiplus.dll
☑ KERNEL32.dll

Manual module addition

You can also manually add one or more modules or a folder to the list.

- **Add Module** › navigate to the DLL or EXE and click **Open** › all the selected items are added

- **Add Folder** › navigate to the folder and click **OK** › the folder is added to the list

Manual addition might be useful for example if you use `LoadLibrary()` to load a DLL rather than linking it, as this would not be picked up automatically by the **Choose Exe...** method.

By default, all the modules are ticked in the yellow checkboxes.

Any DLLs in the list override the DLL Hook Insertion settings on the Hook Insertion tab.

Note that ticked modules or folders are either **in**cluded or **ex**cluded depending on the hooking rule above.

Altering existing module names

Although you can't add blank entries to the list and edit them, you *can* edit existing items in the list by double clicking on an entry:

- enter only the module name, not the path
- you can use wild-cards like MFC*.dll, but only for DLLs, not folders

Managing the process modules list

The usual controls apply for removing or changing the enabled state of items in the list:

- **Remove** › removes selected items in the list

- **Remove All** › removes all items, clearing the list

- **Enable All** › ticks all items in the list for applying to the hooking rule

- **Disable All** › unticks all items in the list, meaning they won't apply to the hooking rule

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

Exporting and importing

Since the list of hooked DLLs (and the rule being applied) can be quite complicated to set up and optimise, you can export the settings to a file and import them again later. This is useful when switching between different target applications.

- **Export...** › **choose or enter** a filename › **Save** › outputs the hooking rule and the list of modules to the file

- **Import...** › **navigate** to an existing *.tvx file › **Open** › loads the hooking rule and the list of modules

Optionally hooking delay loaded DLLs

- **Don't hook delay loaded DLLs** › prevents hooking of delay loaded DLLs. The default is to hook these.

  What is 'delay loading'?

  > Delay loading a DLL is when it is implicitly linked, but not actually loaded until your code references a symbol contained in the DLL.
  >
  > Delay loading can speed up startup time, but unhandled exceptions may cause your program to terminate if the DLL can't be found when needed during the run time.

Launching new Applications

When specifying DLLs to hook, and launching different applications, it can be quite easy to forget to change the hooked DLLs for the new program. This might be the case when performing unit tests, for example.

Using the wrong list of hooked DLLs for a program will likely cause incorrect thread data results, so you can opt to be warned about the DLLs being hooked whenever the target application changes between sessions (using the dialog below).

The choices in the drop down list are only applicable when the application changes:

- **Ask about DLLs to Hook settings if some DLLs defined**

  You'll only be asked about the settings if you defined some DLLs in the list *and* if the hooking rule is *not* set to **Hook all DLLs**.

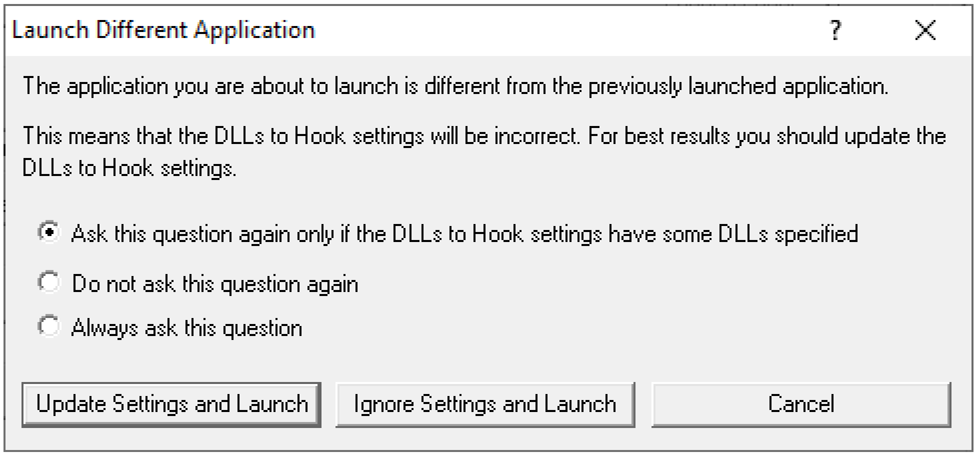
- **Always ask about DLLs to Hook settings**

  You'll always be asked about the settings - *whatever* the other settings are.

- **Never ask about DLLs to Hook settings**

The 'Launch Different Application' dialog

When being asked about the hooked DLL settings, you'll see the following dialog:



Launch Different Application

The application you are about to launch is different from the previously launched application.

This means that the DLLs to Hook settings will be incorrect. For best results you should update the DLLs to Hook settings.

● Ask this question again only if the DLLs to Hook settings have some DLLs specified

○ Do not ask this question again

○ Always ask this question

You can update the settings; ignore them and launch anyway, or just cancel the launch:

- **Update Settings and Launch** › **edit** the settings › click **OK** › the application will be launched
- **Ignore Settings and Launch** › the application will be launched without updating the settings
- **Cancel** › won't launch the application

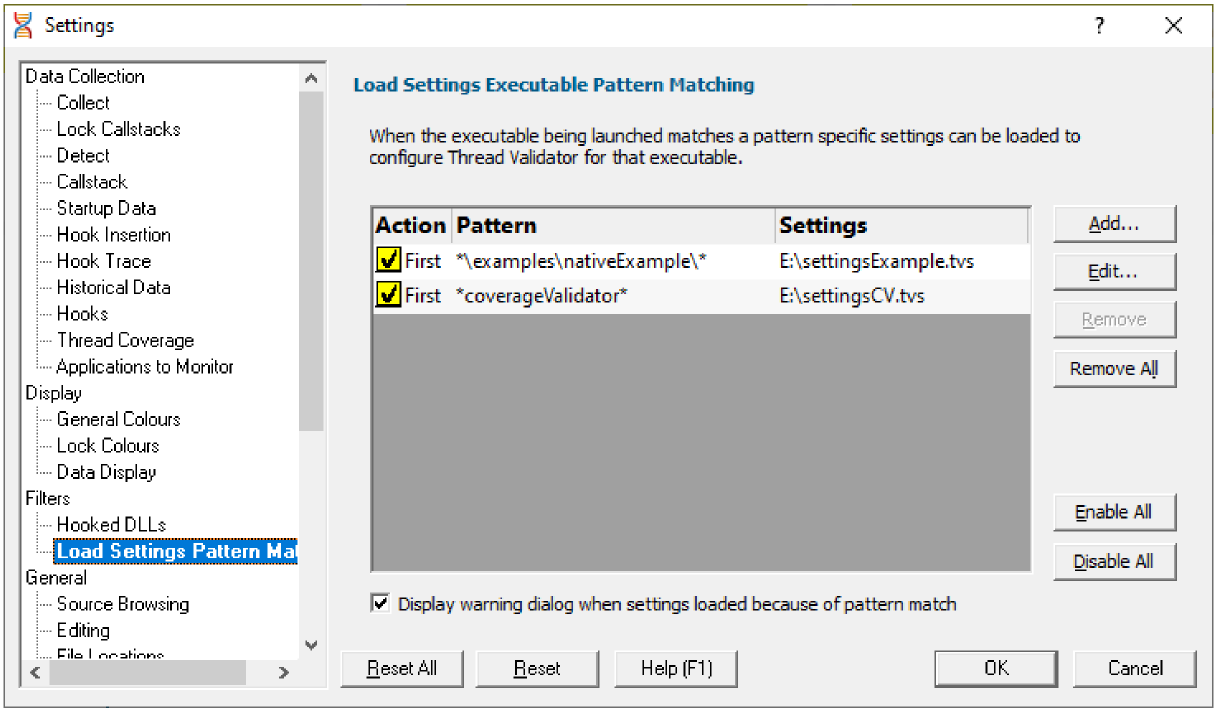
To change when you are asked this question, just choose the appropriate option in the dialog.

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

Reset - Resets the settings on the current page.

3.10.1.3.2 Load Settings Pattern Match

The **Load Settings Pattern Match** tab allows you to configure loading of different settings depending on the executable being launched (or relaunched).



Settings

Load Settings Executable Pattern Matching

When the executable being launched matches a pattern specific settings can be loaded to configure Thread Validator for that executable.

Action	Pattern	Settings
☑ First	*\examples\nativeExample\*	E:\settingsExample.tvs
☑ First	*coverageValidator*	E:\settingsCV.tvs

☑ Display warning dialog when settings loaded because of pattern match

The grid shows one pattern match per line.

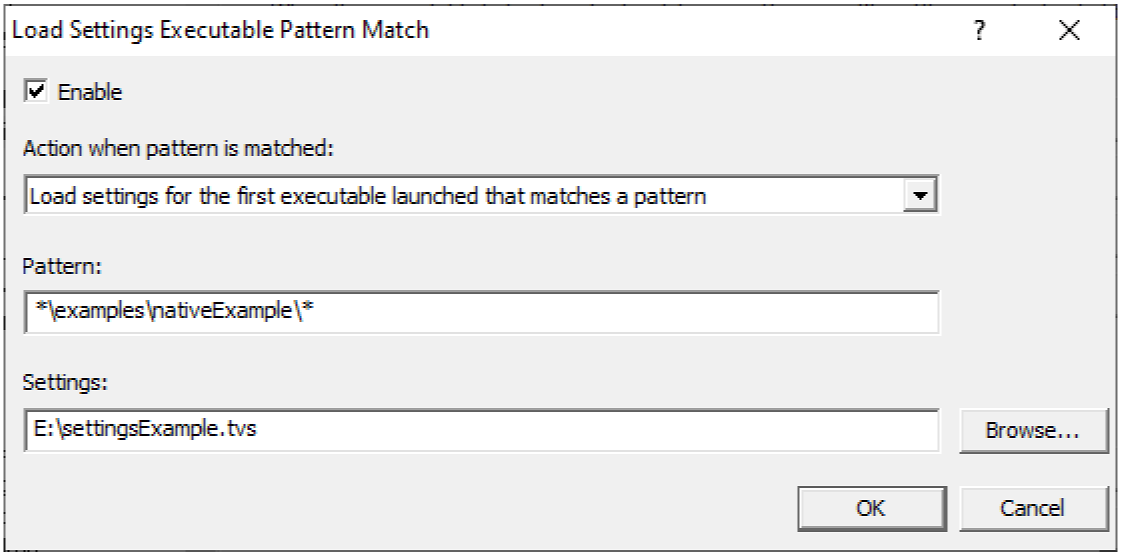
The buttons alongside allow you to Add, Edit and Remove patterns that you have created. You can also enable and disable them all.

- **Add...** › display the pattern match dialog to create a pattern to match.
- **Edit...** › display the pattern match dialog to edit the selected pattern.
- **Remove...** › delete the selected pattern.
- **Remove All...** › delete all selected patterns.

- **Enable All...** › enable all patterns.
- **Disable All...** › disable all patterns.

Pattern Match Dialog

The pattern match dialog allows you to create and edit pattern matches.


Load Settings Executable Pattern Match

☑ Enable

Action when pattern is matched: Load settings for the first executable launched that matches a pattern

Pattern: *\examples\nativeExample\*

Settings: E:\settingsExample.tvs

- **Enable** › enable or disable this pattern.
- **Action** › how to evaluate if this pattern is matched.
  - **Load settings for first executable...** › the first executable that matches a pattern causes the settings to be loaded.
  - **Load settings for each executable...** › each executable that matches a pattern causes the settings to be loaded provided it is not the same executable that previously loaded the settings.
  - **Load settings for every executable...** › every executable that matches a pattern causes the settings to be loaded.

  When a different pattern matches the first/each status is reset.

- **Pattern** › a text pattern, including the * wildcard, to match an executable path.

  Examples:
  ```
  *\examples\nativeExample\*
  c:\tests\*
  e:\dev\myProject\release\*.exe
  ```

- **Settings** › the full path to the settings you want to load if the action and pattern match an executable.

How does the pattern matching work?

It is probably easiest to demonstrate how pattern matching works with some examples.

Let's assume we have two patterns:

`*\examples\nativeExample\*` that will load `e:\settingsExamples.tvs`
`*coverageValidator*` that will load `e:\settingsCV.tvs`.

We'll cover each of the possible action criteria for a sequence of application launches, showing which settings are loaded and why.

- **Load settings for first executable...** › the first executable that matches a pattern causes the settings to be loaded.

Launched application	Settings loaded	Reason
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.tvs	1st application, new pattern
e:\examples\nativeExample\release\nativeExample.exe		repeat application, same pattern
e:\examples\nativeExample\debug\nativeExample.exe		2nd application, same pattern
c:\program files (x86)\software verify\coverage validator x86\coverageValidator.exe	e:\settingsCV.tvs	1st application, new pattern
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.tvs	1st application, new pattern

- **Load settings for each executable...** › each executable that matches a pattern causes the settings to be loaded provided it is not the same executable that previously loaded the settings.

Launched application	Settings loaded	Reason
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.tvs	new application, new pattern
e:\examples\nativeExample\release\nativeExample.exe		repeat application, same pattern
e:\examples\nativeExample\debug\nativeExample.exe	e:\settingsExamples.tvs	new application, same pattern
c:\program files (x86)\software verify\coverage validator x86\coverageValidator.exe	e:\settingsCV.tvs	new application, new pattern
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.tvs	new application, new pattern

- **Load settings for every executable...** › every executable that matches a pattern causes the settings to be loaded.

Launched application	Settings loaded	Reason
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.tvs	Every application

e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.tvs	Every application
e:\examples\nativeExample\debug\nativeExample.exe	e:\settingsExamples.tvs	Every application
c:\program files (x86)\software verify\coverage validator x86\coverageValidator.exe	e:\settingsCV.tvs	Every application
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.tvs	Every application

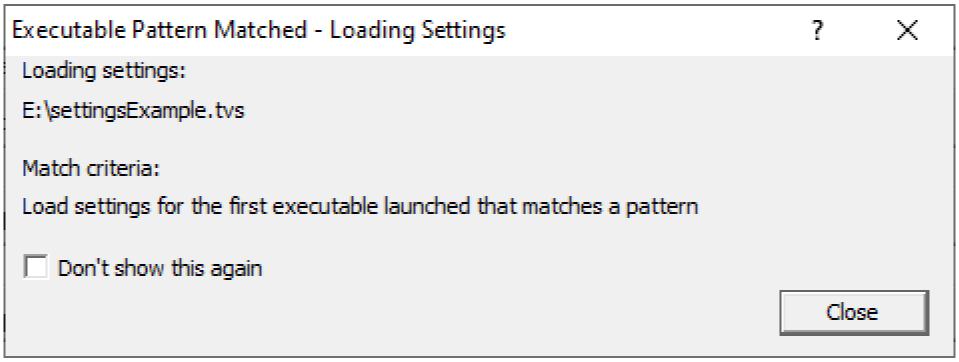
Warning

When a pattern is matched and the action criteria are satisfied the specified settings will be loaded.

A warning can be displayed at this point to remind you that the settings are being changed.

- **Display warning dialog...** > the warning dialog will be displayed when the pattern match criteria are met.

The warning dialog looks like this:



Executable Pattern Matched - Loading Settings

Loading settings:

E:\settingsExample.tvs

Match criteria:

Load settings for the first executable launched that matches a pattern

☐ Don't show this again

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

Reset - Resets the settings on the current page.

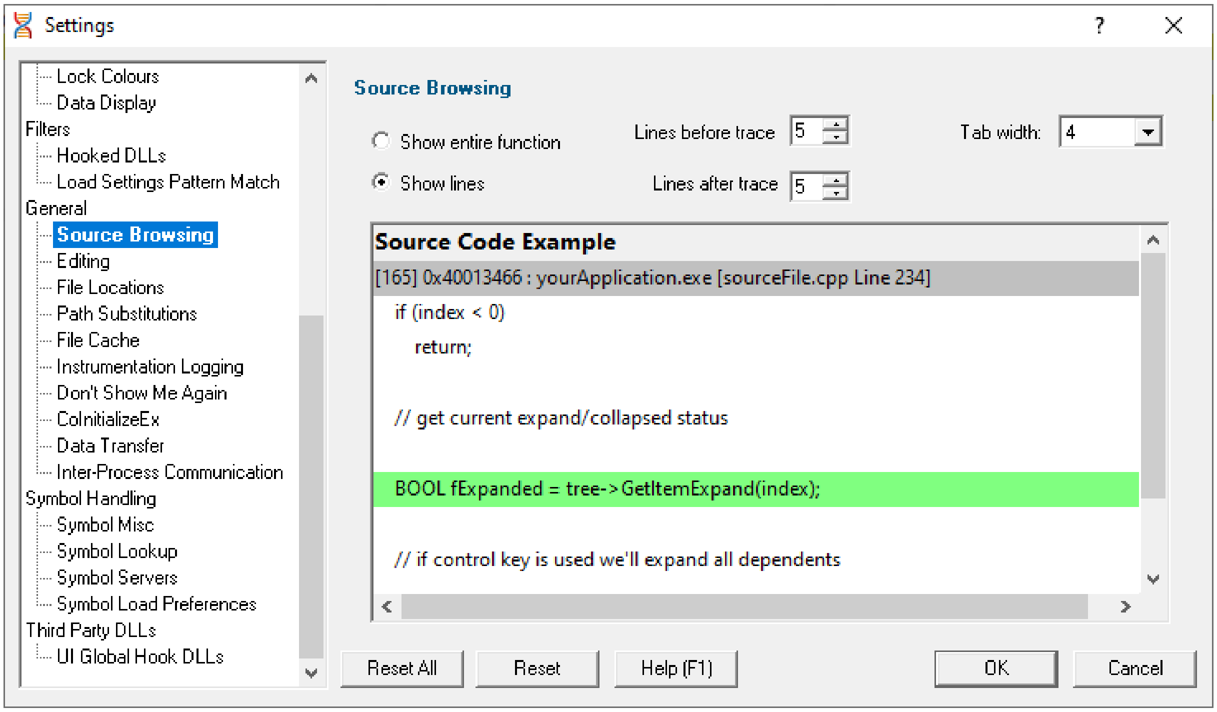
3.10.1.4 General

3.10.1.4.1 Source Browsing

There a few areas in Thread Validator where you can view snippets of source code, such as in the Active Objects tab, Analysis tab or Find Function query results.

The **Source Browsing** tab allows you control how much source code is displayed and the indentation.

The default options are shown below:



Source Browsing

○ Show entire function

● Show lines

Lines before trace 5

Lines after trace 5

Tab width: 4

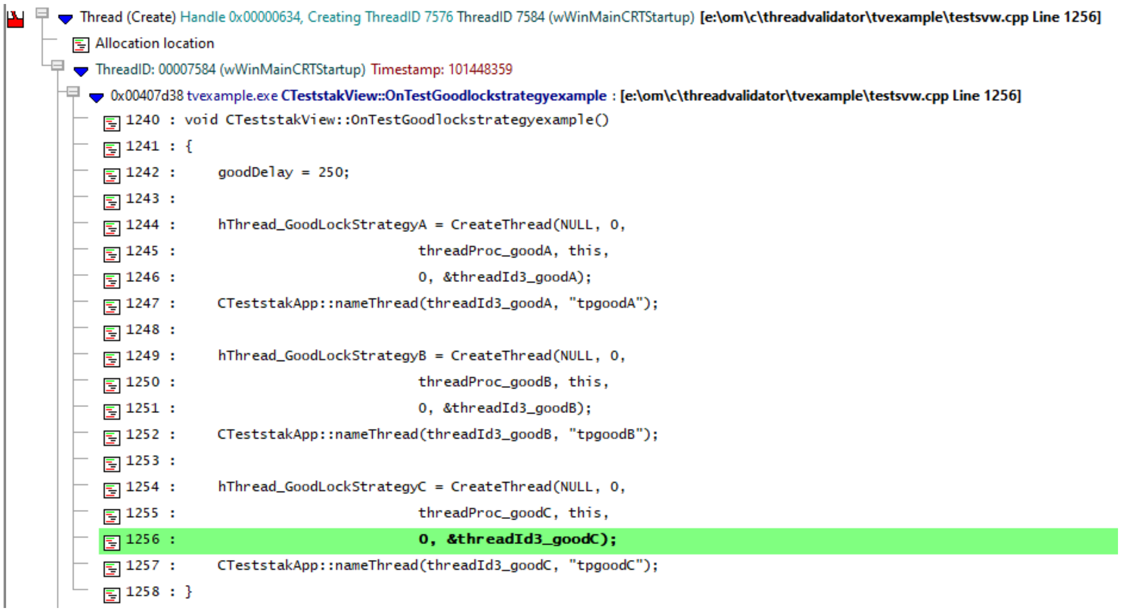
Source Code Example

```
[165] 0x40013466 : yourApplication.exe [sourceFile.cpp Line 234]
  if (index < 0)
    return;

  // get current expand/collapsed status

  BOOL fExpanded = tree->GetItemExpand(index);

  // if control key is used we'll expand all dependents
```

Source browsing

When viewing sections of source code, you can choose to see the whole function or a few lines either side of the line of interest.

- **Show entire function** ❯ shows the whole function source as below:



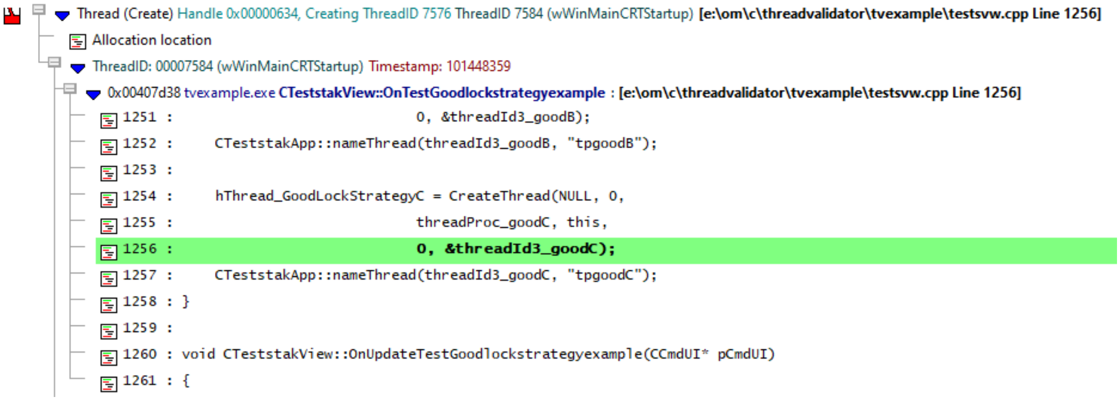
```
Thread (Create) Handle 0x00000634, Creating ThreadID 7576 ThreadID 7584 (wWinMainCRTStartup) [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1256]
  Allocation location
  ThreadID: 00007584 (wWinMainCRTStartup) Timestamp: 101448359
    0x00407d38 tvexample.exe CTeststakView::OnTestGoodlockstrategyexample : [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1256]
      1240 : void CTeststakView::OnTestGoodlockstrategyexample()
      1241 : {
      1242 :     goodDelay = 250;
      1243 :
      1244 :     hThread_GoodLockStrategyA = CreateThread(NULL, 0,
      1245 :                                 threadProc_goodA, this,
      1246 :                                 0, &threadId3_goodA);
      1247 :     CTeststakApp::nameThread(threadId3_goodA, "tpgoodA");
      1248 :
      1249 :     hThread_GoodLockStrategyB = CreateThread(NULL, 0,
      1250 :                                 threadProc_goodB, this,
      1251 :                                 0, &threadId3_goodB);
      1252 :     CTeststakApp::nameThread(threadId3_goodB, "tpgoodB");
      1253 :
      1254 :     hThread_GoodLockStrategyC = CreateThread(NULL, 0,
      1255 :                                 threadProc_goodC, this,
      1256 :                                 0, &threadId3_goodC);
      1257 :     CTeststakApp::nameThread(threadId3_goodC, "tpgoodC");
      1258 : }
```

- **Show lines** › shows a given number of lines before and after the point of interest:

  - **Lines before trace** › number of lines *before*, from 0 to 100

  - **Lines after trace** › number of lines *after*, from 0 to 100

  The default is to show 5 lines above and below, as in this example:



```
Thread (Create) Handle 0x00000634, Creating ThreadID 7576 ThreadID 7584 (wWinMainCRTStartup) [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1256]
  Allocation location
  ThreadID: 00007584 (wWinMainCRTStartup) Timestamp: 101448359
    0x00407d38 tvexample.exe CTeststakView::OnTestGoodlockstrategyexample : [e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1256]
      1251 :                                 0, &threadId3_goodB);
      1252 :     CTeststakApp::nameThread(threadId3_goodB, "tpgoodB");
      1253 :
      1254 :     hThread_GoodLockStrategyC = CreateThread(NULL, 0,
      1255 :                                 threadProc_goodC, this,
      1256 :                                 0, &threadId3_goodC);
      1257 :     CTeststakApp::nameThread(threadId3_goodC, "tpgoodC");
      1258 : }
      1259 :
      1260 : void CTeststakView::OnUpdateTestGoodlockstrategyexample(CCmdUI* pCmdUI)
      1261 : {
```

Source browsing - how much to show?

Showing the entire function is more likely to show the full context of the line of interest, but if you have particularly long functions it may become cumbersome to browse query data!

Because of the unpredictable lengths of showing entire functions, the entire function is *not* the default setting.

Showing a set number of lines reduces the amount of source displayed to something that is consistent and manageable.

You may see parts of neighbouring functions that are not relevant (as above), or you may not see enough of the preceding lines to determine the full context of the line. If this happens often, try changing the number of lines displayed.

Tab size formatting

When formatting the source code being displayed you can also control the tab size

- **Tab width** ❯ set the tab size between 1 and 16 characters

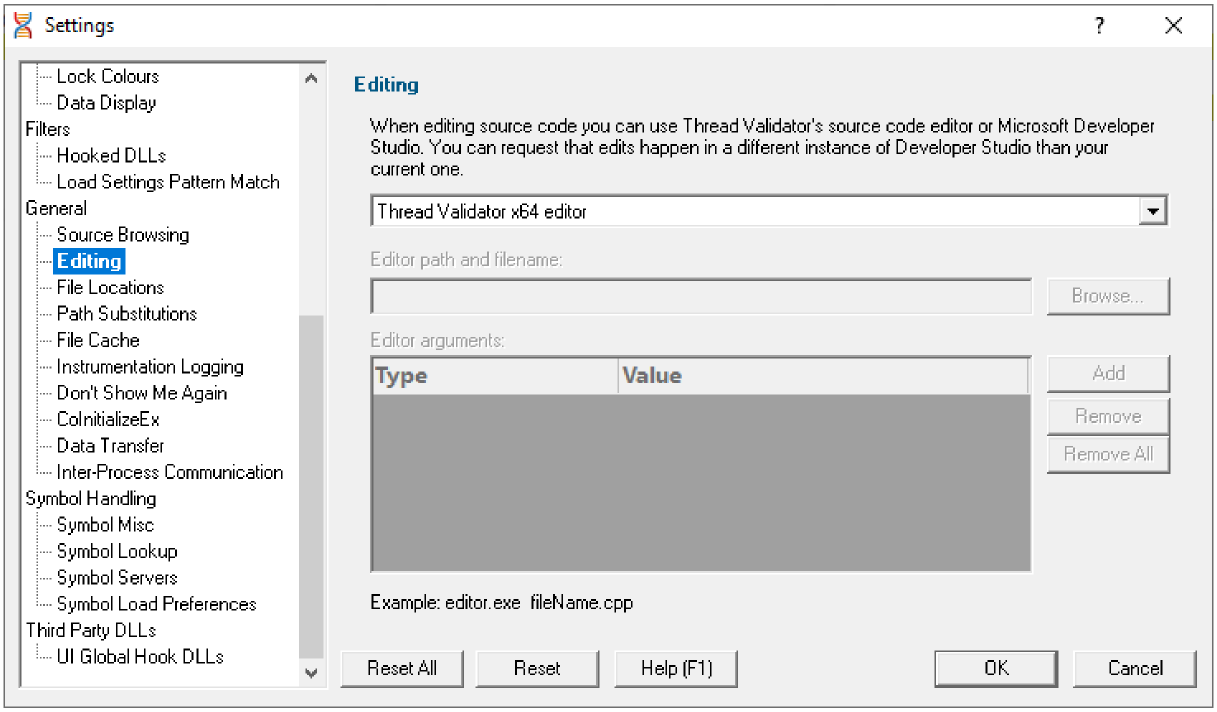
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.4.2 Editing

The **Editing** tab allows you to configure which editor Thread Validator will use for editing source code.

The default settings are shown below:



Editing

When editing source code you can use Thread Validator's source code editor or Microsoft Developer Studio. You can request that edits happen in a different instance of Developer Studio than your current one.

Thread Validator x64 editor

Editor path and filename:

Editor arguments:

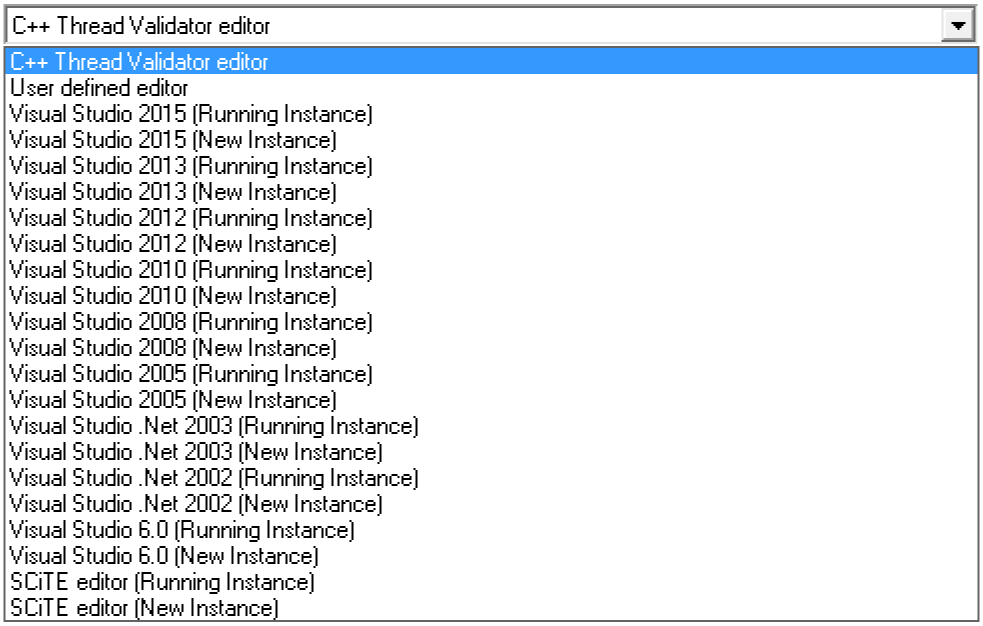
Type	Value

Example: editor.exe fileName.cpp

Editing source code

From the Tools menu, or any of the data views in the main tabs, you can right click to edit the source code.

By default, source code is opened in a provided source code editor using syntax colouring, but you can change where you edit code via the drop-down list:


C++ Thread Validator editor
C++ Thread Validator editor
User defined editor
Visual Studio 2015 (Running Instance)
Visual Studio 2015 (New Instance)
Visual Studio 2013 (Running Instance)
Visual Studio 2013 (New Instance)
Visual Studio 2012 (Running Instance)
Visual Studio 2012 (New Instance)
Visual Studio 2010 (Running Instance)
Visual Studio 2010 (New Instance)
Visual Studio 2008 (Running Instance)
Visual Studio 2008 (New Instance)
Visual Studio 2005 (Running Instance)
Visual Studio 2005 (New Instance)
Visual Studio .Net 2003 (Running Instance)
Visual Studio .Net 2003 (New Instance)
Visual Studio .Net 2002 (Running Instance)
Visual Studio .Net 2002 (New Instance)
Visual Studio 6.0 (Running Instance)
Visual Studio 6.0 (New Instance)
SCiTE editor (Running Instance)
SCiTE editor (New Instance)

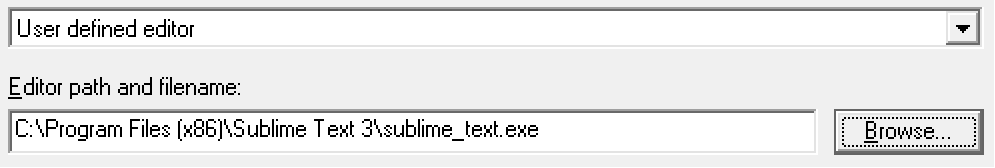
When choosing one of the editors listed, there are options for a currently open instance (e.g. the same one you're using to develop your application), or a new instance.

➜ SCiTE is included in the list of editors, but there are many text editors that can be used for source code on windows. Wikipedia has a comparison of editors including their programming feature support

Editing with your preferred editor

We've all got our favourite editors! To use yours:

- Select **User defined editor** from the list of options › enables the fields below
- Enter the **Editor path and filename** or just **Browse** › choose the executable for your preferred editor


User defined editor

Editor path and filename:

C:\Program Files (x86)\Sublime Text 3\sublime_text.exe

Now when you want to edit source code, that editor will be opened, but typically you'll need to specify some command line arguments with which to start the editor.

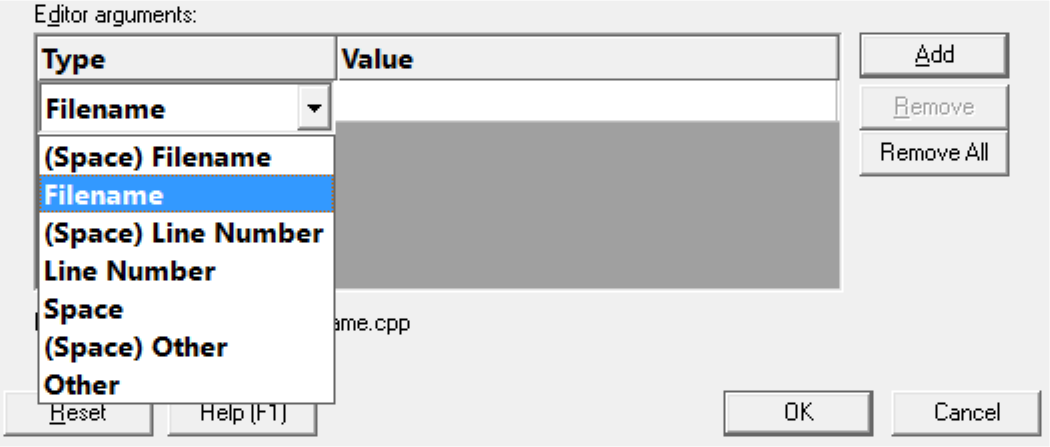
Starting your preferred editor with command line arguments

By default, just the file name is passed as a command line argument to the editor.

Depending on the editor, you may need to tailor the arguments, for example if you want the file scrolled to a particular line.

The arguments can be specified by adding them to the table provided, one at a time and in the order required

- **Add** › adds a row to the **Editor arguments** table › select an argument **Type** from the following options:



The possible arguments include:

- **(Space) Filename** › appends a space followed by the filename
- **Filename** › appends just the filename

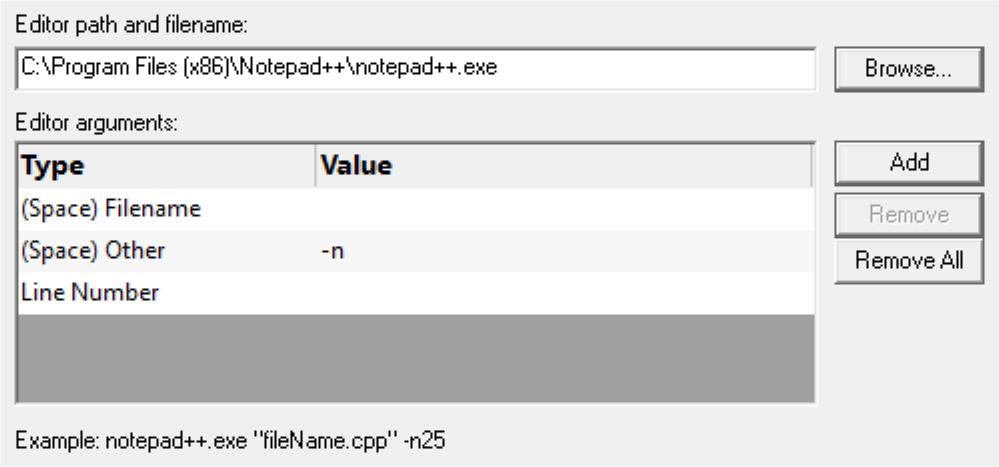
- **(Space) Line Number** › a space followed by the line number
- **Line Number** › just the line number

- **Space** › a space

- **(Space) Other** › a space followed by the text typed in the **Value** column of the list
- **Other** › just the text typed in the **Value** column of the list

Only the **Other** options need an entry in the **Value** column.
You will need to press **Return** after entering the value otherwise the entry won't get recognized.

The example below configures NotePad++ to edit a file at the required line using the -n switch



Editor path and filename:

C:\Program Files (x86)\Notepad++\notepad++.exe

Editor arguments:

Type	Value
(Space) Filename	
(Space) Other	-n
Line Number	

Example: notepad++.exe "fileName.cpp" -n25

As you modify the arguments an example command line is shown below the list.

Managing the command line arguments

Edit a **Type** or **Value** by double clicking the entry. The usual controls apply for removing list items:

- **Remove** › removes selected arguments in the list
- **Remove All** › removes all arguments, clearing the list

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

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

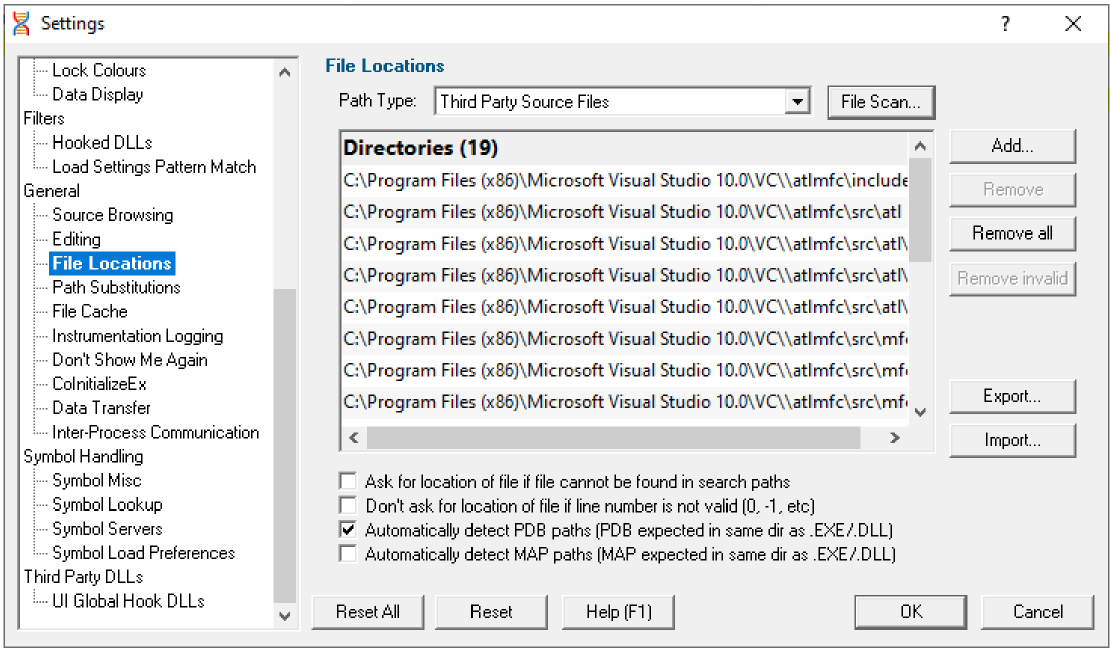
Reset - Resets the settings on the current page.

3.10.1.4.3 File Locations

The **File Locations** tab allows you to specify which directories Thread Validator should look in for source code files, whether that's your own or third party code.

The default settings are shown below:

Read on, or click on a setting in the picture below to find out more:



Path Type: Third Party Source Files

Directories (19)
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\\atlmfc\include
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\\atlmfc\src\atl
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\\atlmfc\src\atl\
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\\atlmfc\src\atl\
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\\atlmfc\src\atl\
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\\atlmfc\src\mf
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\\atlmfc\src\mf
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\\atlmfc\src\mf

☐ Ask for location of file if file cannot be found in search paths
☐ Don't ask for location of file if line number is not valid (0, -1, etc)
☑ Automatically detect PDB paths (PDB expected in same dir as .EXE/.DLL)
☐ Automatically detect MAP paths (MAP expected in same dir as .EXE/.DLL)

File locations

Sometimes the information Thread Validator has access to consists of the file name, but not the directory.

When this happens Thread Validator scans a set of directories that it knows about in order to find the file.

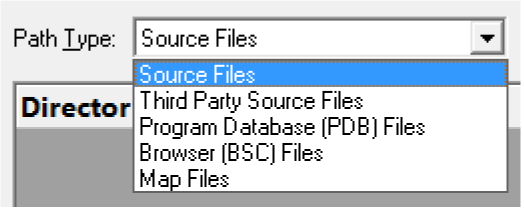
The options below allow you to specify those directories that should be searched for source files, PDB files and MAP files.

If a file can't be found, you'll get prompted for a location.

Setting directories for a path type

There are five path types, and a separate list of directories to scan for each one.

- **Path Type** › select the type of file with which you want to modify the list directory



You don't *have* to specify any directories, e.g. if you don't want to, or if you just don't have them. Nor do you have to give directories for *all* the path types.

Prompting for file locations

Whenever a file still cannot be found, then the default action is for a dialog to ask you where it is.

To avoid frequent user interruption, it is recommended that the directories for source code files (yours and third party) are specified, enabling Thread Validator to automatically load source code for browsing.

If however, you don't want to be prompted for locations, you can disable that too.

- **Ask for location of file...** › untick to stop prompting for file locations

Even when prompting is switched on, it can still happen that the line in question is invalid anyway, e.g. line number 0 or -1.

The default is not to prompt for invalid lines, but if you want to know when that happens, just switch that behaviour off.

- **Don't ask for location of file if line number is not valid...** › untick to be prompted for invalid lines anyway

PDB (program database) file paths

Normally PDB search paths are automatically generated, based on the same directories that .exe and .dll files are found in:

- **Automatically detect PDB paths** › automatically detect PDB locations (the default)

However, it is recommended that you specify paths for PDB (program database) files, especially if your build environment dictates that PDB files are kept in different directories to their binaries.

If you don't automatically generate PDB paths and you don't specify any paths for PDBs, the search path will be defined as the current directory plus any paths found in the following environment variables:

- _NT_SYMBOL_PATH
- _NT_ALTERNATE_SYMBOL_PATH
- SYSTEMROOT

MAP file paths

It's recommended that you specify paths for Map files if your build environment means they are kept in different directories to their binaries.

If you don't specify any paths for Map files, then search paths are automatically generated, based on the same directories that .exe and .dll files are found in.

Manually adding path type directories

Once you have chosen your path type 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

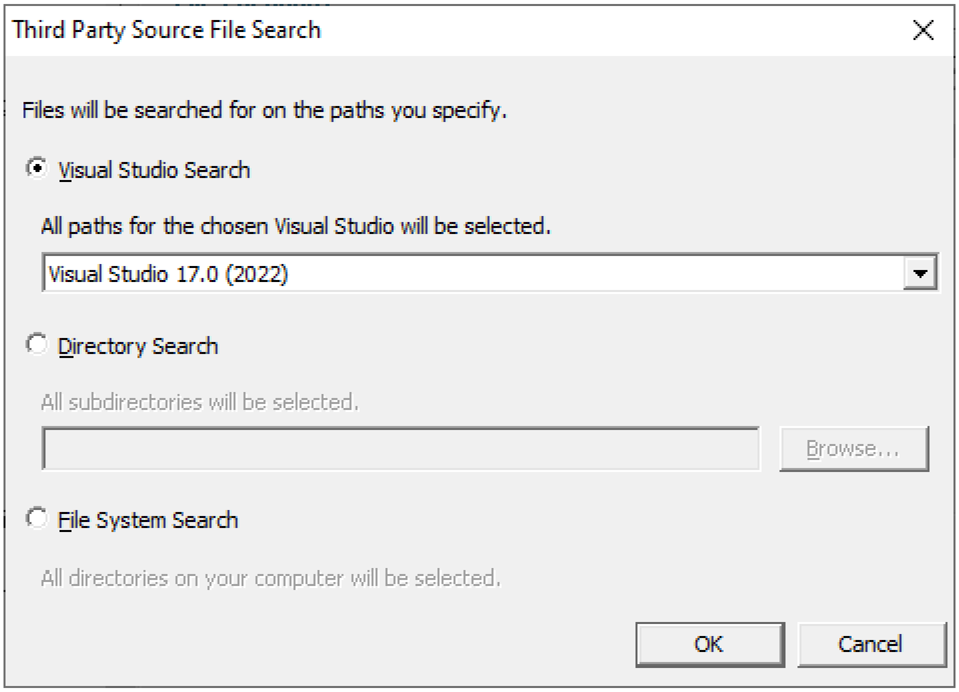
Edit a directory path by double clicking the entry. The usual controls apply for removing list items:

- **Remove** › removes selected items from the list
- **Remove all** › clears the list
- **Remove invalid** › removes all items that are not valid directories from the list

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

Scanning for directories to add

The **File Scan...** button displays the File Search dialog to provide three ways of specifying the files to scan.



Third Party Source File Search

Files will be searched for on the paths you specify.

● Visual Studio Search

All paths for the chosen Visual Studio will be selected.

Visual Studio 17.0 (2022)

○ Directory Search

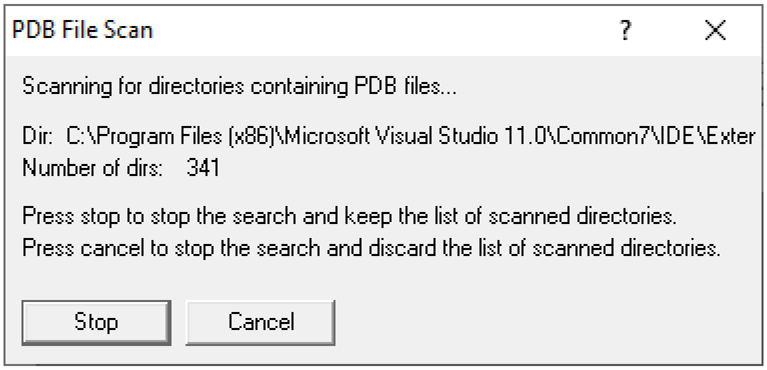
All subdirectories will be selected.

○ File System Search

All directories on your computer will be selected.

- **Visual Studio Search** › choose the version of Visual Studio › **OK** › starts a scan for directories related to that version of Visual Studio
- **Directory Search** › **Browse...** displays a directory browser › **navigate** to a location you want to scan within › **OK** › starts a scan for directories
- **File System Search** › **OK** › starts a scan of all drives for directories containing files

All options will bring up a **File Scan** dialog indicating number of relevant directories found, and giving you a chance to **Stop** or **Cancel** the scan at any time:

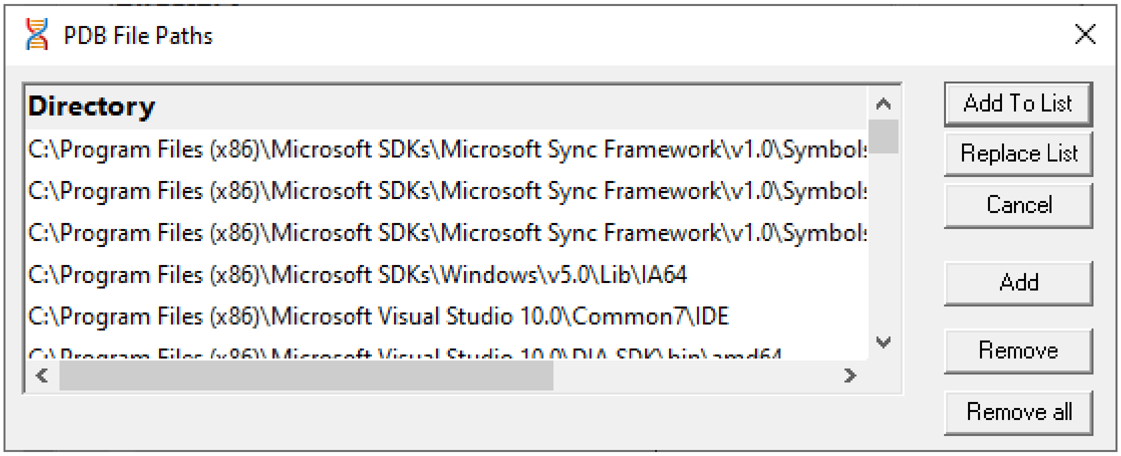


Scanning for directories containing PDB files...

Dir: C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\Exter
Number of dirs: 341

Press stop to stop the search and keep the list of scanned directories.
Press cancel to stop the search and discard the list of scanned directories.

Once the scan is complete you'll see the **File Paths** dialog showing you the scan results:



Directory
C:\Program Files (x86)\Microsoft SDKs\Microsoft Sync Framework\v1.0\Symbol:
C:\Program Files (x86)\Microsoft SDKs\Microsoft Sync Framework\v1.0\Symbol:
C:\Program Files (x86)\Microsoft SDKs\Microsoft Sync Framework\v1.0\Symbol:
C:\Program Files (x86)\Microsoft SDKs\Windows\v5.0\Lib\IA64
C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE
C:\Program Files (x86)\Microsoft Visual Studio 10.0\DIA SDK\bin\amd64

You can modify the list of resulting directories by adding, removing or editing, exactly as for the path type list above.

Once you're happy with the scan results, either append or replace the path type directories with the scan results.

- **Add To List** › adds the scan results list to the path type directories and closes the File Paths dialog
- **Replace List** › replaces the path type directories with the scan results
- **Cancel** › discard the scan results and close the dialog

Exporting and Importing

Since the list of path types and their file locations can be quite complicated to set up and optimise, you can export the settings to a file and import them again later. This is useful when switching between different target applications.

- **Export...** › **choose or enter** a filename › **Save** › outputs all the path types and their file locations to the file
- **Import...** › **navigate** to an existing *.tvxfl file › **Open** › loads the path types and their file locations

Export file format

The file format is plain text with one folder listed per line. Sections are denoted by a line containing `[Files]` (for source code files), `[Third]` (for third party source code files), `[PDB]` etc.

Example:

```
[Files]
c:\work\project1\
[Third]
d:\VisualStudio\VC98\Include
[PDB]
c:\work\project3\debug
c:\work\project3\release
[MAP]
c:\work\project3\debug
c:\work\project3\release
```

Checking directory scanning order

To see the order in which the *DbgHelp.dll* process checks directories to find symbols, see the diagnostic tab, showing *DbgHelp debug* in the drop-down.

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

Currently, the PDB path detection checkbox at the bottom of this page is **not** reset as part of the global settings.

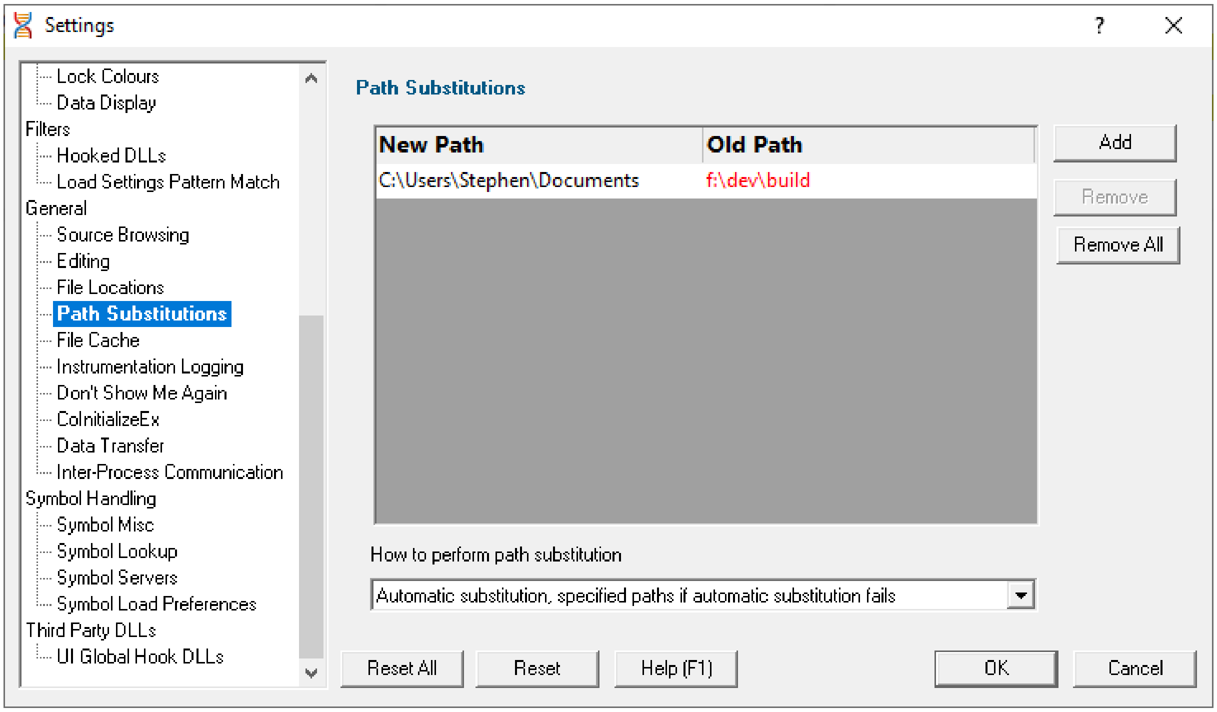
Reset - Resets the settings on the current page.

Currently, the PDB path detection checkbox at the bottom of this page is **not** reset as part of the global settings.

3.10.1.4.4 Path Substitutions

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

The default settings are shown below:



Path Substitutions

New Path	Old Path
C:\Users\Stephen\Documents	f:\dev\build

How to perform path substitution: Automatic substitution, specified paths if automatic substitution fails

File 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 Thread Validator 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**.

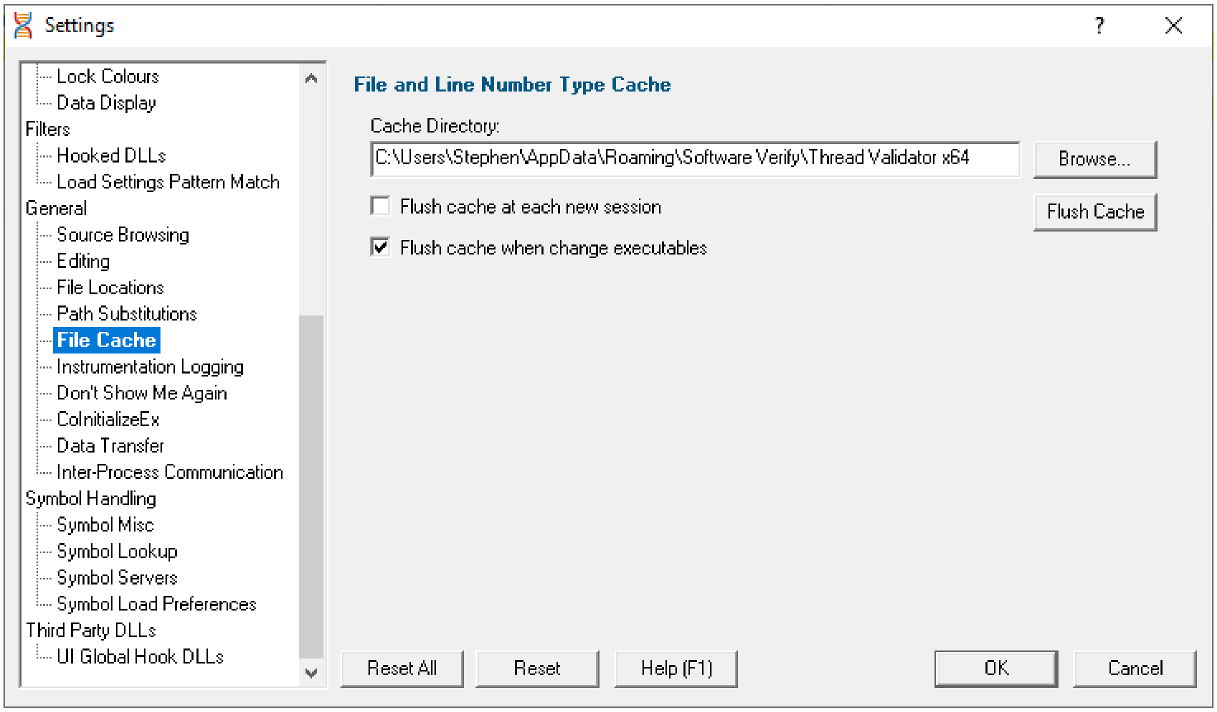
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.4.5 File Cache

The **File Cache** tab allows you to specify where cached information is stored and when it gets cleared.

The default settings are shown below:



File and Line Number Type Cache

Cache Directory: C:\Users\Stephen\AppData\Roaming\Software Verify\Thread Validator x64

☐ Flush cache at each new session

☑ Flush cache when change executables

Caching file locations

Thread Validator keeps a cache of known locations for files for which it needed to search, improving the speed at which files can be found.

- **Cache Directory** › type directly or **Browse** to find a directory for Thread Validator to cache its information

By default, the cache is only flushed when the executable changes between sessions.

- **Flush cache at each new session** › tick to flush the cache every session

  This slightly slows down relaunch of the same executable, as the cache needs rebuilding.

- **Flush cache when executable changes** › untick to prevent the cache being flushed at all

When not automatically flushing, you can manually flush the cache if necessary:

- **Flush Cache** › flush the cache now

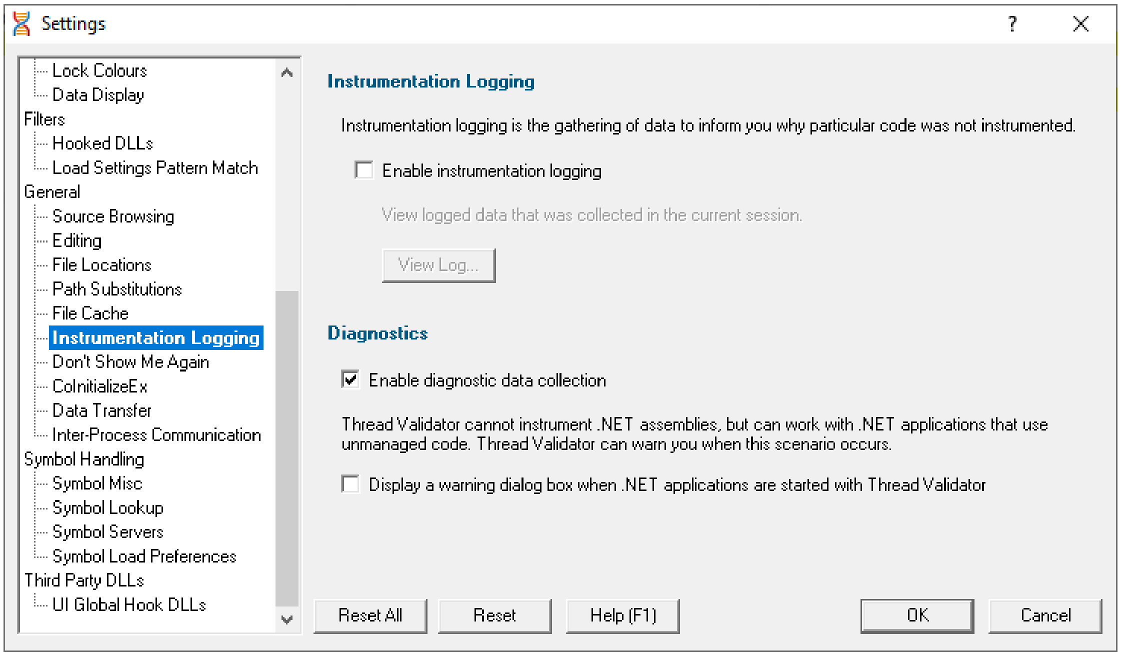
  This is only possible when no sessions are in the session manager. The button will be disabled if any sessions are loaded.

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

Reset - Resets the settings on the current page.

3.10.1.4.6 Instrumentation Logging

The **Instrumentation Logging** tab enables you to understand the reason why part of your code isn't getting the threading information you expect.



Instrumentation Logging

Instrumentation logging is the gathering of data to inform you why particular code was not instrumented.

☐ Enable instrumentation logging

View logged data that was collected in the current session.

Diagnostics

☑ Enable diagnostic data collection

Thread Validator cannot instrument .NET assemblies, but can work with .NET applications that use unmanaged code. Thread Validator can warn you when this scenario occurs.

☐ Display a warning dialog box when .NET applications are started with Thread Validator

Instrumentation logging

The logging of DLLs, source files, classes, methods and functions that are not instrumented can help you understand the reason why part of your code isn't getting the coverage information you expect.

Once enabled, and a session has started, you can view a list of items that have not been instrumented via the Tools menu.

- **Enable instrumentation logging** › check to enable logging once the next session starts

Diagnostics

A lot of diagnostic information is collected and displayed on the diagnostic tab when attaching to a target program.

Some of this information is *always* sent to Thread Validator, but you may not want to see it all.

- **Enable diagnostic data collection** › displays all diagnostic information in the diagnostic tab (on by default)

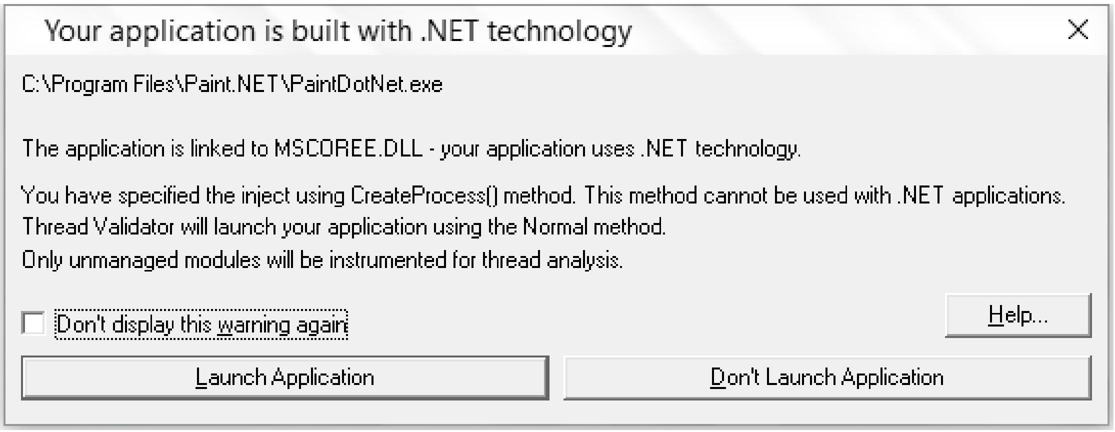
.NET warning

Thread Validator cannot instrument .NET assemblies and cannot monitor locks and thread synchronization primitives called from them (also known as "managed code").

However, locks and thread synchronization primitives *can* be monitored in any non-.NET DLLs even if part of a .NET application.

You can optionally be warned when trying to launch a .NET application:

- **Display a warning dialog box when .NET applications are started** › shows a warning dialog (example below) when using .NET applications (on by default)



Your application is built with .NET technology

C:\Program Files\Paint.NET\PaintDotNet.exe

The application is linked to MSCOREE.DLL - your application uses .NET technology.

You have specified the inject using CreateProcess() method. This method cannot be used with .NET applications.
Thread Validator will launch your application using the Normal method.
Only unmanaged modules will be instrumented for thread analysis.

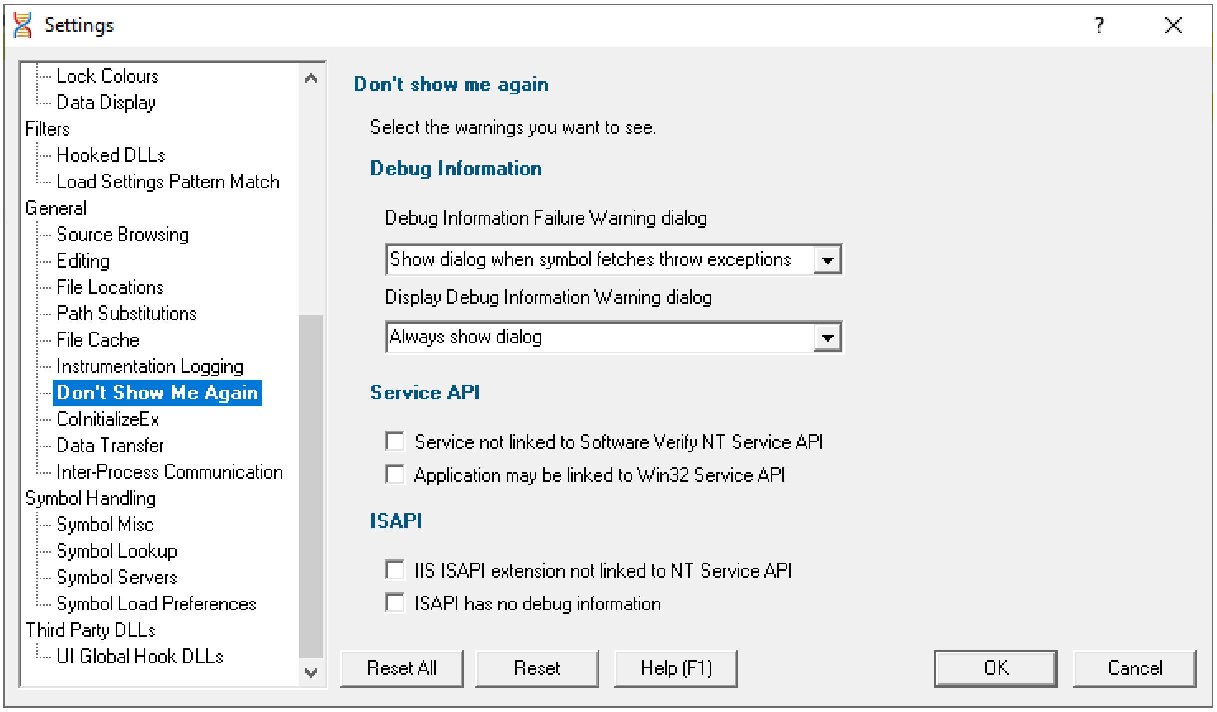
- **Help...** › will open the manual for User Permissions Warnings

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

Reset - Resets the settings on the current page.

3.10.1.4.7 Don't Show Me Again

The **Don't Show Me Again** page allows you to control warnings that Thread Validator displays.



Don't show me again

Select the warnings you want to see.

Debug Information

Debug Information Failure Warning dialog: Show dialog when symbol fetches throw exceptions

Display Debug Information Warning dialog: Always show dialog

Service API

□ Service not linked to Software Verify NT Service API

□ Application may be linked to Win32 Service API

ISAPI

□ IIS ISAPI extension not linked to NT Service API

□ ISAPI has no debug information

Debug Information

Debug Information Failure Warning

When there is a failure collecting debug symbols a warning can be displayed. The options are:

- Always show dialog
- Never show dialog
- Show dialog when symbol fetches throw exceptions

Display Debug Information Warning

When no debug information is available for at least one module a warning can be displayed. The options are:

- Always show dialog
- Never show dialog
- Show dialog when debug information is missing

Services API

- **Service not linked to Software Verify NT Service API** › warning will be shown if you try to monitor a service not linked to the Software Verify NT Service API. (on by default)

  When trying to monitor a service Thread Validator can detect if the service is not linked to the NT Service API and display a warning.

  It is possible to use the service API without linking to it (use GetProcAddress() to lookup the functions and call them) - in this case you would want to turn this warning off.

- **Application may be linked to Win32 Service API ›** warning will be shown if you try to start an application that appears to be a service - it uses Win32 Service APIs. (on by default)

ISAPI

NT Service API
When trying to monitor ISAPI extensions Thread Validator can detect if the ISAPI is not linked to the NT Service API and display a warning.

It is possible to use the service API without linking to it (use GetProcAddress() to lookup the functions and call them) - in this case you would want to turn this warning off.

Debug Information
Thread Validator can warn if the ISAPI has no debug information. There may be cases where you don't want to see this warning.

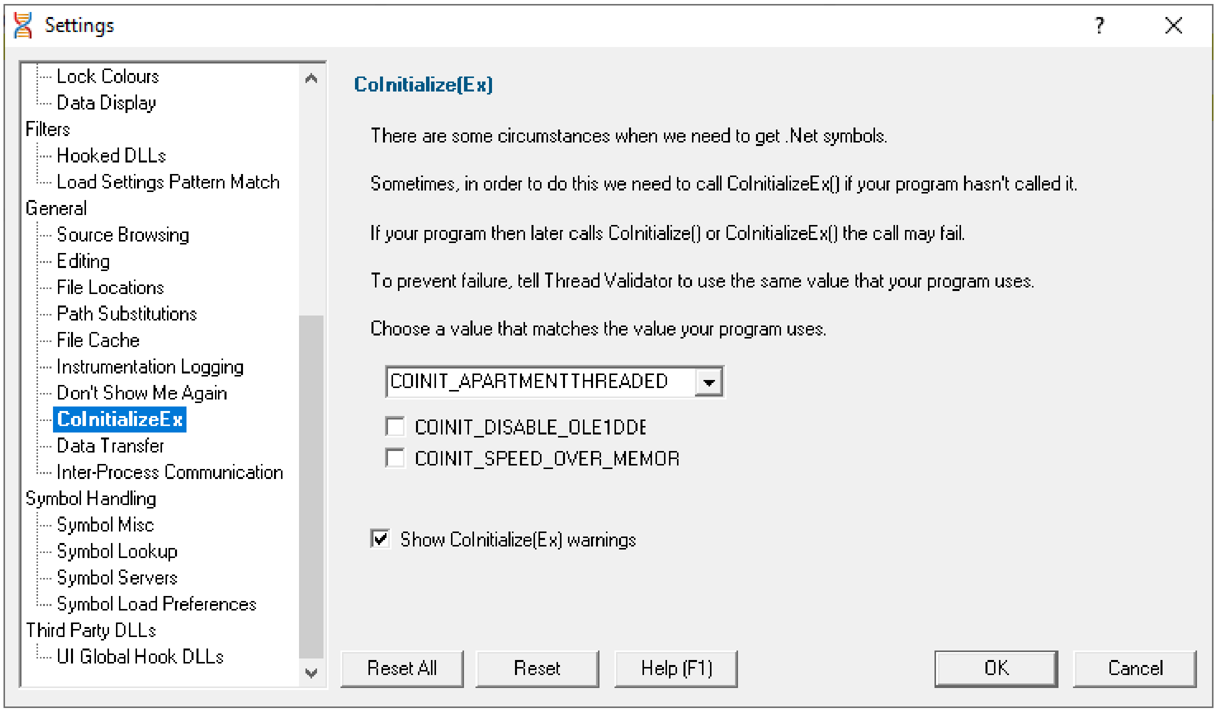
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.4.8 CoInitializeEx

The **CoInitializeEx** tab allows you to set the default behaviour used to initialize COM if Thread Validator needs to initialize COM to acquire symbols for .Net modules.

The default settings are shown below:



CoInitialize(Ex)

There are some circumstances when we need to get .Net symbols.

Sometimes, in order to do this we need to call CoInitializeEx() if your program hasn't called it.

If your program then later calls CoInitialize() or CoInitializeEx() the call may fail.

To prevent failure, tell Thread Validator to use the same value that your program uses.

Choose a value that matches the value your program uses.

COINIT_APARTMENTTHREADED

□ COINIT_DISABLE_OLE1DDE

□ COINIT_SPEED_OVER_MEMOR

☑ Show CoInitialize(Ex) warnings

CoInitializeEx

In some situations the Validator needs to get .Net symbols and to do that COM needs to be initialized. This normally isn't a problem, but if your program also performs COM initialization and the sequence of events results in your COM initialization coming after the Validator's COM initialisation rather than getting the expected `ERROR_SUCCESS` return code you'll get either `ERROR_INVALID_FUNCTION` or `RPC_E_CHANGED_MODE`.

If you get `ERROR_INVALID_FUNCTION` this is OK, this just means you've called CoInitialize() or CoInitializeEx() multiple times with the same flags. Your code needs to handle `ERROR_INVALID_FUNCTION` as **not an error**.

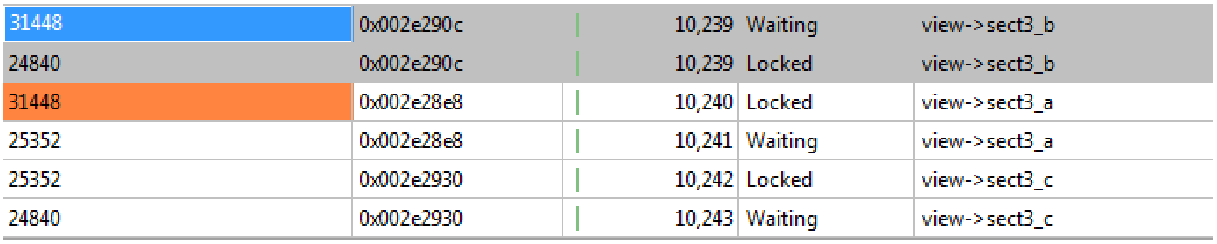
If you get `RPC_E_CHANGED_MODE` this means you need to change the Validator's default value to the same value your program is using. That's what this dialog allows you to do.

If you also wish to disable OLE DDE or favour speed rather than memory use we've provided appropriate options for you to select to add those flags to the threading mode.

See the Microsoft documentation for additional information on the behaviour of CoInitialize() and CoInitializeEx().

Runtime detection of CoInitializeEx conflict

When the above scenario happens, that the Validator has initialized COM before your code initializes COM and your call returns RPC_E_CHANGED_MODE, we display a dialog to warn you about this failure and provide you with the option of editing the default value for subsequent runs of your application.



31448	0x002e290c	10,239	Waiting	view->sect3_b
24840	0x002e290c	10,239	Locked	view->sect3_b
31448	0x002e28e8	10,240	Locked	view->sect3_a
25352	0x002e28e8	10,241	Waiting	view->sect3_a
25352	0x002e2930	10,242	Locked	view->sect3_c
24840	0x002e2930	10,243	Waiting	view->sect3_c

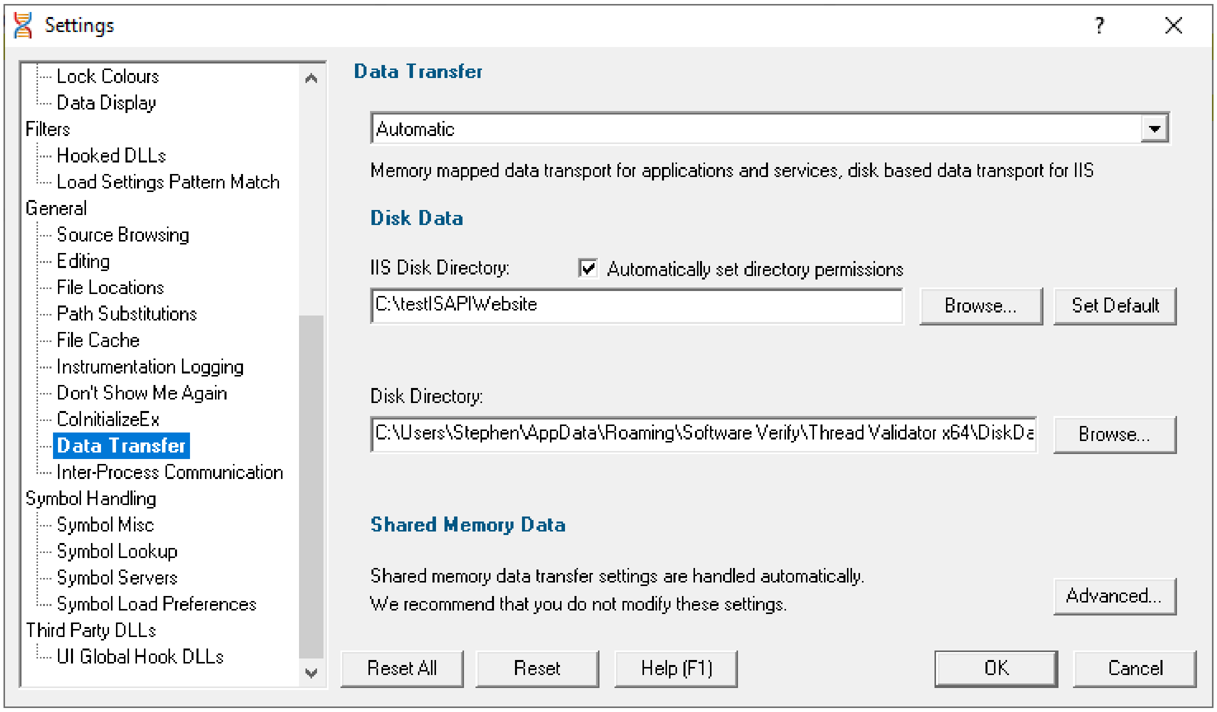
- **Edit Settings...** › opens the CoInitializeEx dialog shown above

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

Reset - Resets the settings on the current page.

3.10.1.4.9 Data Transfer

The **Data Transfer** tab allows you to specify the overall behaviour of data transfer between your application and Thread Validator.



Data Transfer

Automatic

Memory mapped data transport for applications and services, disk based data transport for IIS

Disk Data

IIS Disk Directory: ☑ Automatically set directory permissions

C:\testISAPIWebsite

Disk Directory:

C:\Users\Stephen\AppData\Roaming\Software Verify\Thread Validator x64\DiskDa

Shared Memory Data

Shared memory data transfer settings are handled automatically.
We recommend that you do not modify these settings.

Data Transport

Choose the type of data transport you wish to use.

- **Automatic**. Applications and services use shared memory to transfer data. IIS uses disk based data transfer.

- **Disk**. Applications, services and IIS use disk based data transfer.

- **High Volume**. Data transfer has no data throttling applied to it. This mode is for use with applications that generate very high volumes of data rapidly. They typically exceed the buffering capabilities of Thread Validator when working with shared memory. The High Volume setting uses a data transport that doesn't have a data-throttling requirement allowing the high volume application to continue without waiting.

Automatic

Under most circumstances data transfer between Thread Validator and the target program (desktop, service, etc) is via shared memory. This is handled automatically.

Disk Data

Some applications and services don't allow shared memory access. For these occasions we use a file based data transfer, where the files are stored in a directory of your choice.

We provide two options for this, one for most applications and services, and one for Internet Information Server, as this operates in a very restricted environment.

Both options are configured automatically, but you can override either by typing the path to a suitable directory or using the Microsoft directory browser.

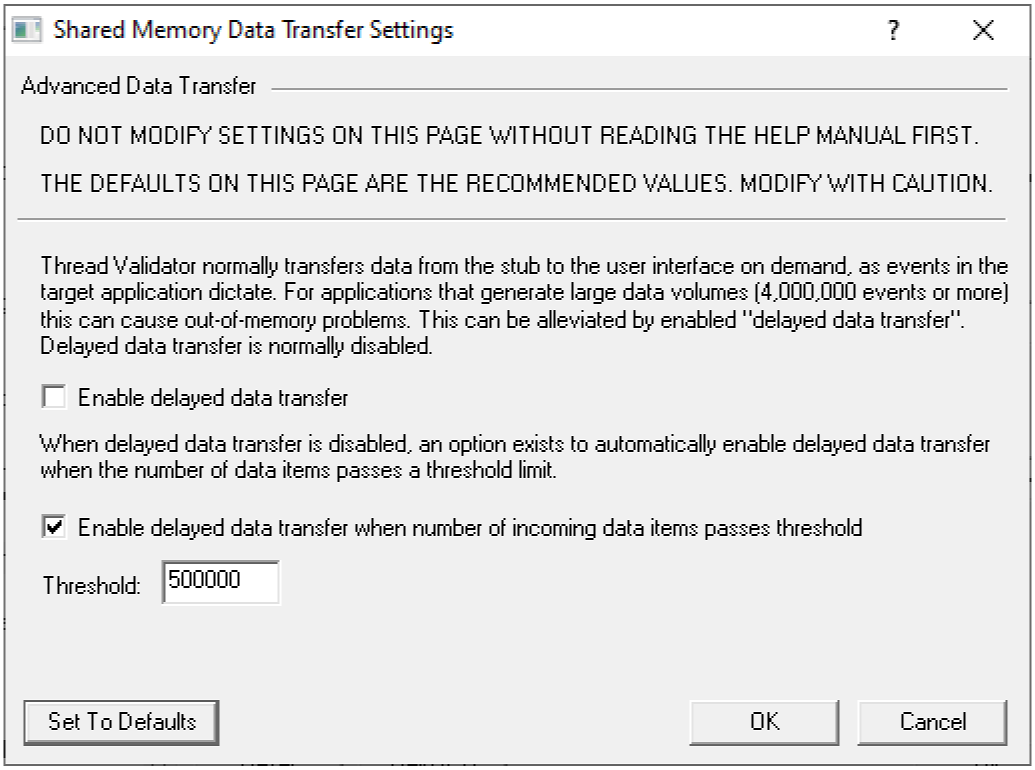
The ISS path you enter will be determined by the settings you have configured for IIS using the Internet Information Services Manager tool. We won't discuss that here because if you're using IIS we assume you already know how to configure IIS correctly.

Advanced

Shared memory data transfer can also be configured **but we strongly recommend that you leave these settings alone**.

The **Data Transfer Helper** is a separate application supplied in the installation directory.

- **Advanced...** › opens the data transfer settings dialog.



DO NOT MODIFY SETTINGS ON THIS PAGE WITHOUT READING THE HELP MANUAL FIRST.

THE DEFAULTS ON THIS PAGE ARE THE RECOMMENDED VALUES. MODIFY WITH CAUTION.

Thread Validator normally transfers data from the stub to the user interface on demand, as events in the target application dictate. For applications that generate large data volumes (4,000,000 events or more) this can cause out-of-memory problems. This can be alleviated by enabled "delayed data transfer". Delayed data transfer is normally disabled.

□ Enable delayed data transfer

When delayed data transfer is disabled, an option exists to automatically enable delayed data transfer when the number of data items passes a threshold limit.

☑ Enable delayed data transfer when number of incoming data items passes threshold

Threshold: 500000

Here be dragons!

Caution: Modifying the settings on this page and using the data transfer helper application can prevent Thread Validator from working correctly.

- **Set To Defaults** › if you have modified the settings, this resets them

  ➡ See also the Reset to default buttons on the data transfer helper application below

If in doubt, don't modify these settings. If you promise to be careful, read on!

Delayed data transfer

Delayed data transfer is the process of throttling data rates in the stub so that the slower user interface can keep up with processing the data received.

In the stub, as an event occurs, data is queued and then sent to the user interface.

In the user interface, data from the stub is received and queued again for processing.

Any delay is usually in the slower user interface, but still not a problem for most applications.

However, some data intensive applications can generate so much data that the user interface gets swamped and can't process it all before running out of memory.

Temporarily limiting the data rate in the *stub* allows the user interface to stabilize the data processing.

Managing data rates

We recommend the default settings as shown above:

- disable delay data transfer for most applications
- enable *automatic* delay data transfer at a threshold of between 100,000 and 1,000,000 data items

If delayed data transfer is enabled all the time, the automatic options don't apply.

If you have more than 1GB RAM, you can raise these thresholds.

Data transfer helper application

A separate data transfer helper application is supplied in the installation directory.

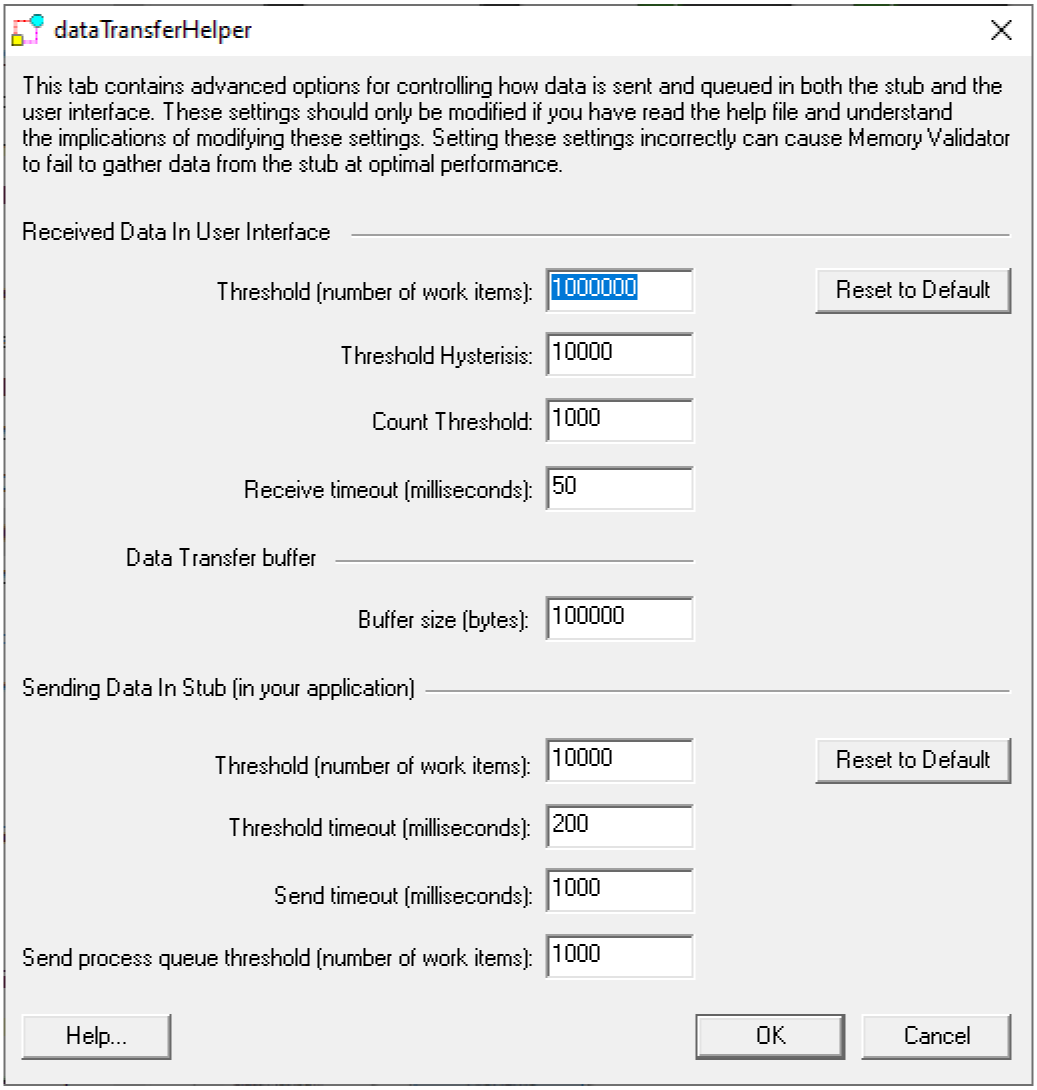
The helper application can be used to modify low level settings that apply when delay data transfer is activated as above.

The helper should be used with care. We already warned of dragons above, but here we are, warning you again!

An HTML help page for this application is available by clicking the **Help** button on the helper application.

You can also find the help page directly as `dataTransferHelp.html`.

Please do take a moment to read the help before use.



This tab contains advanced options for controlling how data is sent and queued in both the stub and the user interface. These settings should only be modified if you have read the help file and understand the implications of modifying these settings. Setting these settings incorrectly can cause Memory Validator to fail to gather data from the stub at optimal performance.

Received Data In User Interface

Threshold (number of work items): 1000000

Threshold Hysterisis: 10000

Count Threshold: 1000

Receive timeout (milliseconds): 50

Data Transfer buffer

Buffer size (bytes): 100000

Sending Data In Stub (in your application)

Threshold (number of work items): 10000

Threshold timeout (milliseconds): 200

Send timeout (milliseconds): 1000

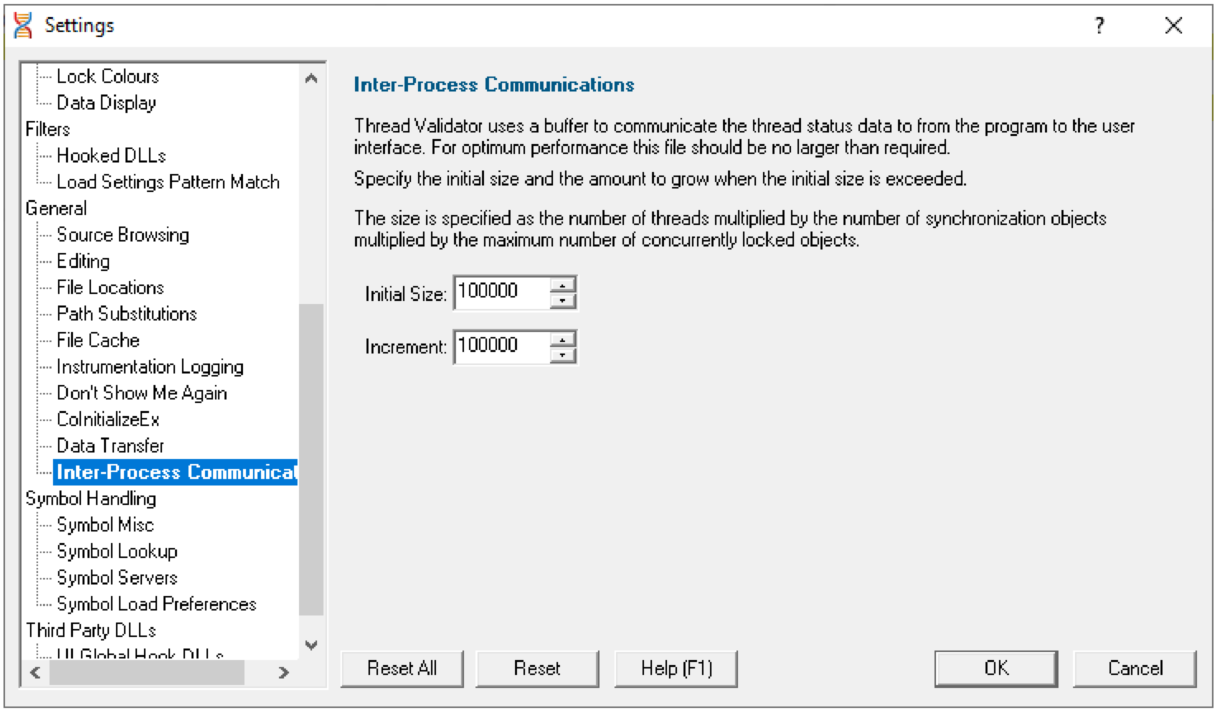
Send process queue threshold (number of work items): 1000

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

Reset - Resets the settings on the current page.

3.10.1.4.10 Inter-Process Communication

The **Inter-Process Communication** tab allows you to configure the size of the buffer used to communicate with the user interface.



Inter-Process Communications

Thread Validator uses a buffer to communicate the thread status data to from the program to the user interface. For optimum performance this file should be no larger than required.

Specify the initial size and the amount to grow when the initial size is exceeded.

The size is specified as the number of threads multiplied by the number of synchronization objects multiplied by the maximum number of concurrently locked objects.

Initial Size: 100000

Increment: 100000

Communication buffer size

Thread Validator communicates with the stub using a memory buffer to store data.

The size of the buffer is related to the number of synchronization objects that have been used.

This buffer size is increased as necessary, but this can sometimes cause a slight delay during runtime.

To avoid the buffer being expanded too often, we recommend setting the initial size of the buffer to be larger than the number of synchronization objects required.

- **Initial Size** › set the initial size of the buffer
- **Increment** › set the amount by which to increase the size of the buffer (if needed)

The size of the buffer can be calculated approximately as follows:

1 + `numThreads` × (1 + `numWaitForXXXObjects` + (`numCriticalSections` × 3))

Don't set values overly large as this may also adversely affects memory usage. The default is 10^5

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

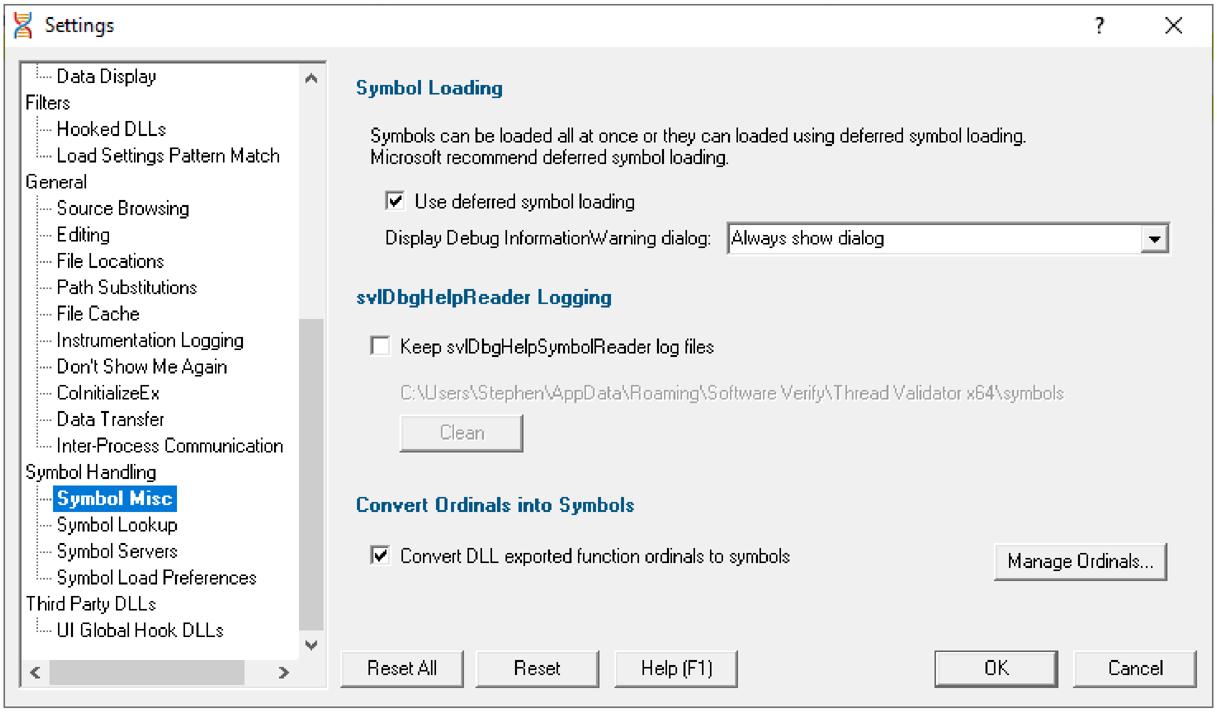
Reset - Resets the settings on the current page.

3.10.1.5 Symbol Handling

Enter topic text here.

3.10.1.5.1 Symbol Misc

The **Symbols and Warnings** tab allows you to set an assortment of symbol and diagnostic warning data.



Symbol Loading

Symbols can be loaded all at once or they can loaded using deferred symbol loading.
Microsoft recommend deferred symbol loading.

☑ Use deferred symbol loading

Display Debug InformationWarning dialog: Always show dialog

svlDbgHelpReader Logging

☐ Keep svlDbgHelpSymbolReader log files

C:\Users\Stephen\AppData\Roaming\Software Verify\Thread Validator x64\symbols

Convert Ordinals into Symbols

☑ Convert DLL exported function ordinals to symbols

Symbol loading - immediate or deferred

When converting program addresses to symbol names, you can choose immediate symbol loading, or defer loading until each symbol is needed.

- **Use deferred symbol loading** ❯ uses deferred symbol loading rather than 'all at once' (on by default)

  Microsoft® recommend deferred symbol loading, claiming it is the fastest option. We give you the choice.

Symbol Reader Logging

Symbols are fetched from symbol servers using a helper process svlDbgHelpSymbolReader.exe. We log the command line and behaviour of this helper tool. This is displayed on the diagnostic tab.

If you wish the log files can be kept for later analysis. By default this option is turned off.

- **Keep svlDbgHelpSymbolReader log files** › keep the log files after Thread Validator has finished processing them

The path to the directory containing the log files is shown.

- **Clean** › delete all svlDbgHelpSymbolReader log files

Convert Ordinals into symbols

- **Convert DLL exported function ordinals to symbols** › enable the ordinal to function name mapping

  You'll need to tick this to enable the use of mapped names defined in the list. If you don't, you won't see the names being used.

- **Manage Ordinals...** › shows the Ordinal Handling dialog to manage which .def files are associated with which DLLs

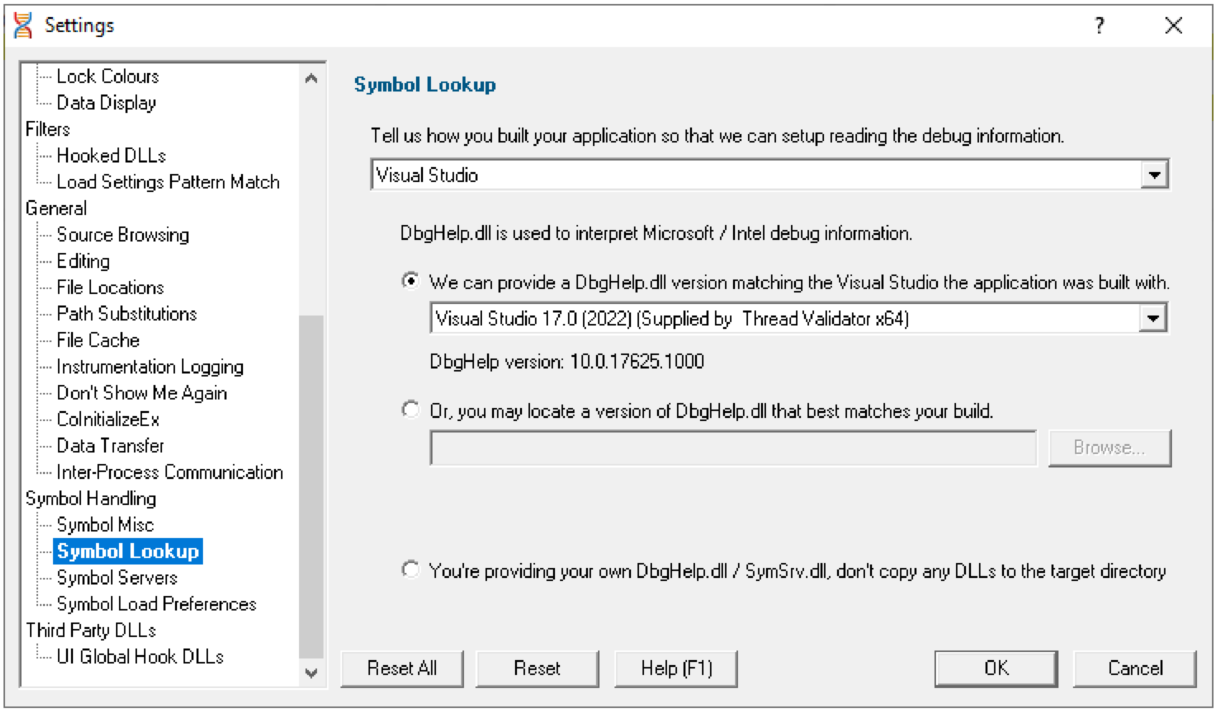
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.5.2 Symbol Lookup

The **Symbol Lookup** tab allows you to specify how and where symbolic information is retrieved for your application or service.

The default settings are shown below, although the Visual Studio version may vary.



Settings

- Lock Colours
- Data Display
- Filters
  - Hooked DLLs
  - Load Settings Pattern Match
- General
  - Source Browsing
  - Editing
  - File Locations
  - Path Substitutions
  - File Cache
  - Instrumentation Logging
  - Don't Show Me Again
  - CoInitializeEx
  - Data Transfer
  - Inter-Process Communication
- Symbol Handling
  - Symbol Misc
  - Symbol Lookup
  - Symbol Servers
  - Symbol Load Preferences
- Third Party DLLs
  - UI Global Hook DLLs

Symbol Lookup

Tell us how you built your application so that we can setup reading the debug information.

Visual Studio

DbgHelp.dll is used to interpret Microsoft / Intel debug information.

● We can provide a DbgHelp.dll version matching the Visual Studio the application was built with.

Visual Studio 17.0 (2022) (Supplied by Thread Validator x64)

DbgHelp version: 10.0.17625.1000

○ Or, you may locate a version of DbgHelp.dll that best matches your build.

○ You're providing your own DbgHelp.dll / SymSrv.dll, don't copy any DLLs to the target directory

Compiler / IDE Choice

Use the first combo box to choose which compiler / IDE you used to build your software.

Thread Validator will use the appropriate methods to read your symbols.

The choices are:

- Visual Studio
- Visual Basic 6
- Delphi or C++ Builder
- MingW
- Rust
- Dev C++
- Metrowerks CodeWarrior
- Salford Fortran 95
- Other

Symbol lookup for Microsoft / Intel compilers

- **We can provide a Dbghelp.dll** › choose one of Thread Validator's known good DbgHelp.dll's based on the version of Visual Studio you are using

  Thread Validator fetches symbols for your application using an appropriate symbol handler for the type of debugging information you have.

For Microsoft Visual Studio users each version of Visual Studio provides different debugging formats which are readable by the appropriate DbgHelp.dll supplied by Visual Studio. A given version of DbgHelp.dll is usually able to read earlier formats of Microsoft debugging information but is not able to read a future format. For example Visual Studio 2005 (version 8) can read Visual Studio 6 debug information but cannot read Visual Studio 2008 debug information.

Visual Studio 6.0 doesn't supply a DbgHelp.dll so we have provided one for use with Visual Studio 6.0.

Visual Studio 10 is unusual in that the DbgHelp.dll (6.12) supplied by Visual Studio cannot read the debug information created by Visual Studio. To solve this problem we have supplied DbgHelp.dll (6.11) as an alternative.

Thread Validator will choose the appropriate (most recent) version of Visual Studio automatically. You can override Thread Validator's choice by choosing the Visual Studio version from the **Visual Studio** combo box.

Specify your own DbgHelp.dll

- **Or, you may locate a version of DbgHelp.dll** › specify your own DbgHelp.dll to use with Thread Validator

If you wish to explicitly specify which DbgHelp.dll to use choose the **Or, you may locate a version of DbgHelp.dll** option enter the path in the **DbgHelp.dll** edit field or use the **Browse...** button to select the dbgHelp.dll.

Note that the directory that contains DbgHelp.dll ***should also contain symsrv.dll*** if you wish to use symbol servers with Thread Validator.

Don't update DbgHelp.dll

- **You're providing your own DbgHelp.dll** › use the DbgHelp.dll that ships with your application

If your application needs to use a specific version of DbgHelp.dll that you're already providing with your application you should choose the **You're providing your own DbgHelp.dll** option to prevent Thread Validator from overwriting your DbgHelp.dll.

Note that the directory that contains DbgHelp.dll ***should also contain symsrv.dll*** if you wish to use symbol servers with Thread Validator.

Visual Studio DbgHelp.dll version compatibility

For Microsoft Visual Studio users, each VS version provides different debugging formats which are readable by the appropriate DbgHelp.dll supplied with Visual Studio.

These handlers are usually backwards compatible, but not forwards compatible. For example Visual Studio 2005 (version 8) can read Visual Studio 6 debug information but cannot read Visual Studio 2008 debug information.

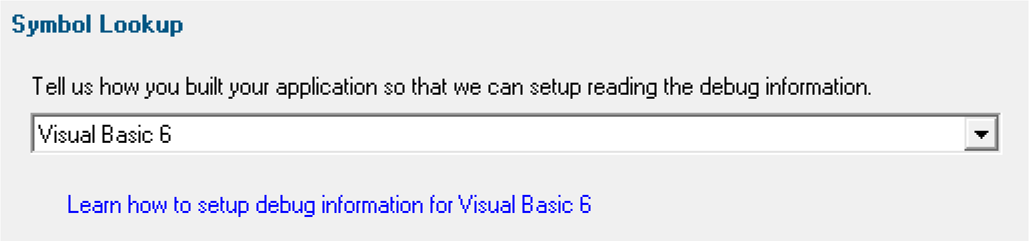
Visual Studio 6.0 doesn't supply a DbgHelp.dll so we have provided one for use with Visual Studio 6.0.

Visual Studio 10 is unusual in that the DbgHelp.dll (6.12) supplied by Visual Studio cannot read the debug information created by Visual Studio! To solve this problem we supply version 6.11 as an alternative.

To see the order in which the *DbgHelp.dll* process checks directories to find symbols, see the diagnostic tab with the filter set to *DbgHelp debug*.

Symbol lookup for other compilers

If you are using another compiler click the link to see information about configuring debug information for that compiler.



Symbol Lookup

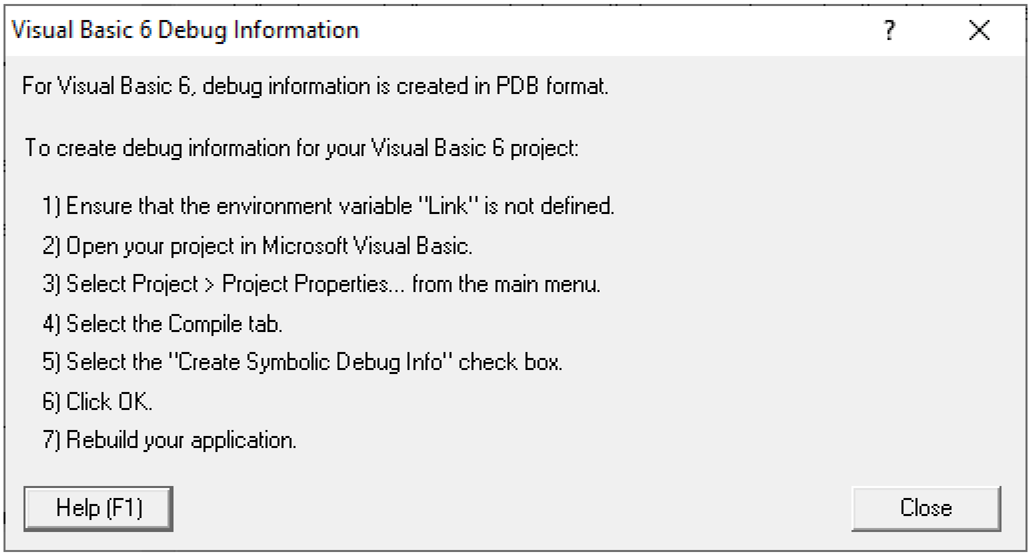
Tell us how you built your application so that we can setup reading the debug information.

Visual Basic 6

Learn how to setup debug information for Visual Basic 6

After selecting the compiler, clicking the link will show a dialog box containing information relevant to the selected compiler.

For example:



Visual Basic 6 Debug Information

For Visual Basic 6, debug information is created in PDB format.

To create debug information for your Visual Basic 6 project:

1) Ensure that the environment variable "Link" is not defined.
2) Open your project in Microsoft Visual Basic.
3) Select Project > Project Properties... from the main menu.
4) Select the Compile tab.
5) Select the "Create Symbolic Debug Info" check box.
6) Click OK.
7) Rebuild your application.

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

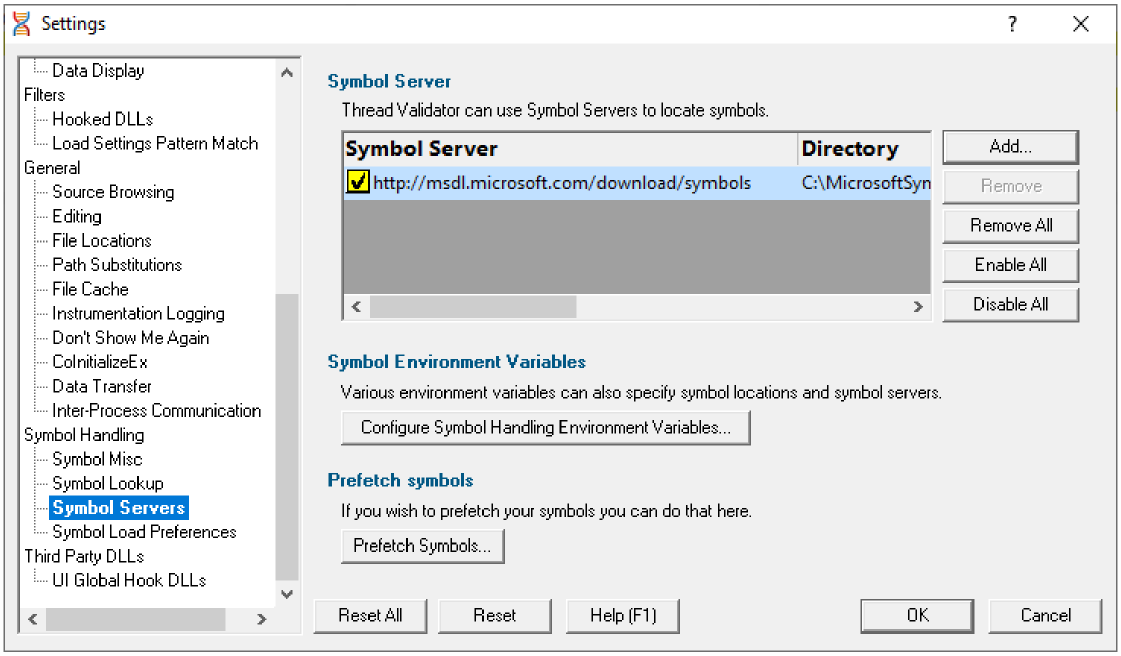
Reset - Resets the settings on the current page.

3.10.1.5.3 Symbol Servers

The **Symbol Servers** tab allows you to specify Symbol Servers to retrieve symbols used in your application.

You do not *need* to specify symbol servers if you do not wish to, and Thread Validator will work correctly without them.

Read on, or click a setting in the picture below to find out more.



Symbol Server

Thread Validator can use Symbol Servers to locate symbols.

Symbol Server	Directory
☑ http://msdl.microsoft.com/download/symbols	C:\MicrosoftSyn

Symbol Environment Variables

Various environment variables can also specify symbol locations and symbol servers.

Prefetch symbols

If you wish to prefetch your symbols you can do that here.

Symbol servers

Symbol servers are entirely optional, but are useful for obtaining symbols from a centralized company resource or for obtaining operating symbols from Microsoft.

The default symbol server is the Microsoft symbol server used for acquiring symbols about Microsoft's operating system DLLs. You may also wish to add some symbol servers for any software builds in your organisation.

A symbol server is defined by at least the following:

- the symbol server dll to be used to handle the symbol server interaction
- a directory location where symbol definitions are saved
- the server location - a url

Each symbol server can be enabled or disabled allowing you to keep multiple symbol server configurations available without constantly editing their definitions.

You can define up to four symbol servers and more than one can be enabled at a time.

Symbol Server Errors

Any symbol server entry shown in red indicates there is a problem with parts of the definition of that symbol server.

In the image shown above the symbol server at http://127.0.0.42:8000 cannot be reached. It is either offline or does not exist.

Managing symbol servers

- **Add...** › displays the symbol server dialog described below

- **Remove** › remove selected symbol server(s) in the list
- **Remove All** › remove all symbol servers

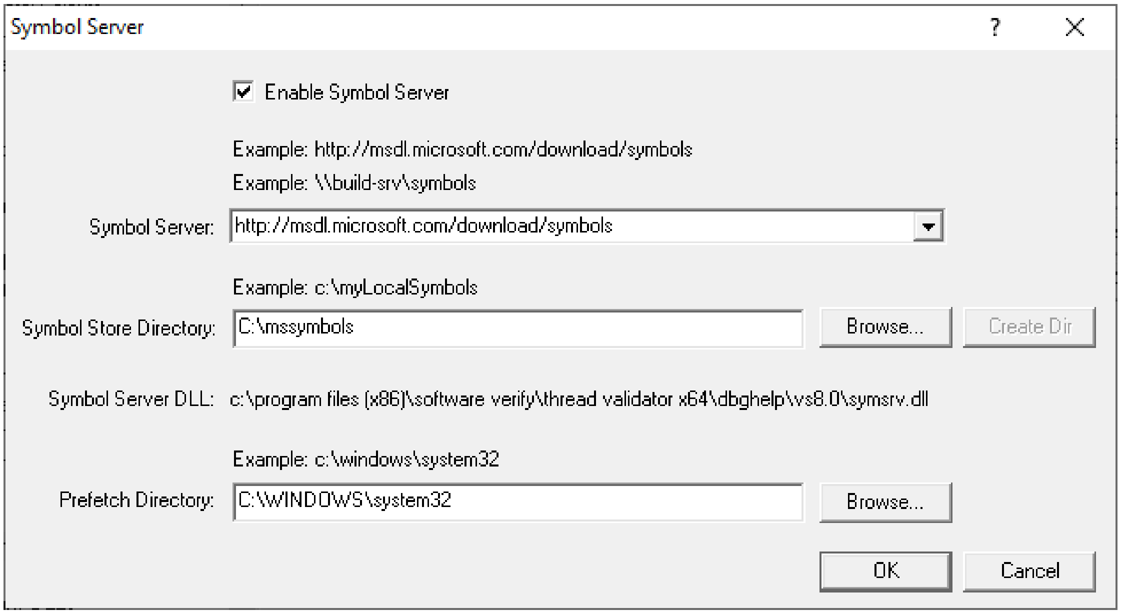
- **Enable All** › enables all symbol servers in the list
- **Disable All** › disables all symbol servers

> You can also enable or disable an item in the list via the yellow check box at the left of each row.

To edit the details for a symbol server, just double click the entry in the list to show the symbol server dialog again.

Symbol server dialog

The dialog initially appears pre-populated with some default values and allows you to set up or edit the definition of a symbol server. Some of the default values can be changed.



☑ Enable Symbol Server

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

Example: \\build-srv\symbols

Symbol Server: http://msdl.microsoft.com/download/symbols

Example: c:\myLocalSymbols

Symbol Store Directory: C:\mssymbols

Symbol Server DLL: c:\program files (x86)\software verify\thread validator x64\dbghelp\vs8.0\symsrv.dll

Example: c:\windows\system32

Prefetch Directory: C:\WINDOWS\system32

- **Enable Symbol Server** › enable or disable this server

The following three entries must be set to enable the **OK** button and define the symbol server.

> ⊟ OK button not enabled?The OK button will only be enabled when the following entries have a valid value: - Symbol Server DLL names a dll present in the Thread Validator install directory. - Symbol Store Directory has been specified and exists. - Symbol Server URL has been specified (this value will not be checked for correctness).

- **Symbol Server** › select a predefined public symbol server or enter the URL of the symbol server you wish to use - the Microsoft server is initially set as the default
- **Symbol Store Directory** › enter or **Browse** to set the directory that will contain local copies of the downloaded symbols
  - **Create Dir** › creates a directory if you entered a directory name that does not exist yet

The **Symbol Server DLL** is set based on the Symbol Lookup settings you have chosen.

You can optionally associate a directory to scan when you are prefetching symbols (below)

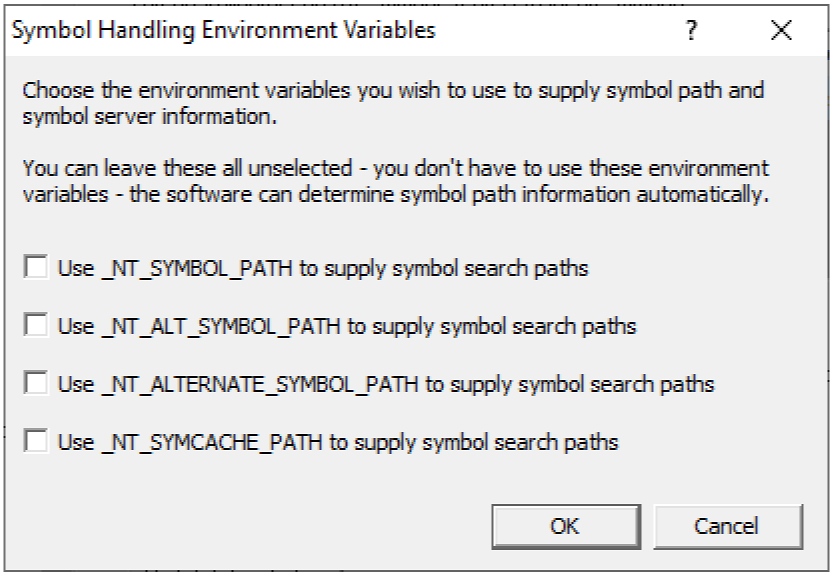
- **Prefetch Directory** › specify the directory to scan for symbols

Environment variables related to symbols

If you wish, you can set some environment variables to supply symbol paths.

- **Configure Symbol Handling Environment Variables** › opens the dialog below

Check the desired options - if any.



Symbol Handling Environment Variables

Choose the environment variables you wish to use to supply symbol path and symbol server information.

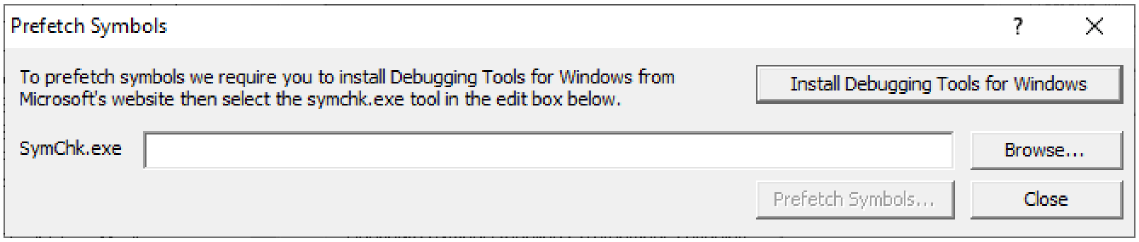
You can leave these all unselected - you don't have to use these environment variables - the software can determine symbol path information automatically.

- □ Use _NT_SYMBOL_PATH to supply symbol search paths
- □ Use _NT_ALT_SYMBOL_PATH to supply symbol search paths
- □ Use _NT_ALTERNATE_SYMBOL_PATH to supply symbol search paths
- □ Use _NT_SYMCACHE_PATH to supply symbol search paths

Pre-fetching symbols

To avoid delays when using symbol servers, you can trigger the retrieval of symbols (by running SymChk.exe) to collect symbols for all executable files specified in the exe/dll which you associated with each symbol server.

- **Prefetch Symbols...** › open the Prefetch Symbols dialog below to continue



Prefetch Symbols

To prefetch symbols we require you to install Debugging Tools for Windows from Microsoft's website then select the symchk.exe tool in the edit box below.

SymChk.exe

Prerequisites for pre-fetching symbols

The pre-fetching of symbols requires the installation of Microsoft's Debugging Tools.

You may already have Debugging Tools if you've previously installed the Windows Driver Kit (DDK or WDK) or the Windows SDK.

- **Install Debugging Tools for Windows** › opens a web page (as above) to download and install the x86 or x64 Debugging Tools for Windows

After installing the Debugging Tools, you must specify the location of SymChk.exe from the installed area.

- **SymChk.exe** › enter or **Browse** to SymChk.exe location

A typical path might be `C:\WinDDK\7600.16385.1\Debuggers\symchk.exe`

Getting the symbols

Note that prefetching symbols may consume a large amount of disk space and download bandwidth.

You should ensure that you have at least 2 or 3Gb of disk free space, because of the total size of the download packages.

- **Prefetch Symbols...** › runs SymChk.exe to get all the symbols

  The symbols for each symbol server are stored in the associated symbol store directory.

If no symbol servers are specified in the symbol server settings above, you'll see a warning dialog and no symbols will be fetched.

Command line pre-fetching of symbols with the SymChk utility

The section on Pre-fetching symbols above is a convenient alternative to manually using the SymChk,exe utility.

To avoid delays when using symbol servers, you can pre-fetch symbols using the SymChk.exe command line tool that is part of Microsoft's Debugging Tools.

You may want to add the folder of the Debugging Tools for Windows package to the PATH environment variable on your system so that you can access this tool easily from any command prompt.

Example:

To use SymChk.exe to download symbol files for all of the components in the `c:\windows\System32` folder, you might use the command:

```
symchk.exe /r c:\windows\system32 /s SRV*c:\symbols\*http://msdl.microsoft.com/download/sym
```

where

`/r c:\windows\system32` finds all symbols for files in that folder *and any sub-folders*

`/s SRV*c:\symbols*http://msdl.microsoft.com/download/symbols` specifies the symbol path to use for symbol resolution.

In this case, `c:\symbols` is the local folder where the symbols will be copied from the symbol server.

➡ To obtain more information about the command-line options for SymChk.exe, type `symchk /?` at a command prompt.

Other options include the ability to specify the name or the process ID (PID) of an executable file that is running.

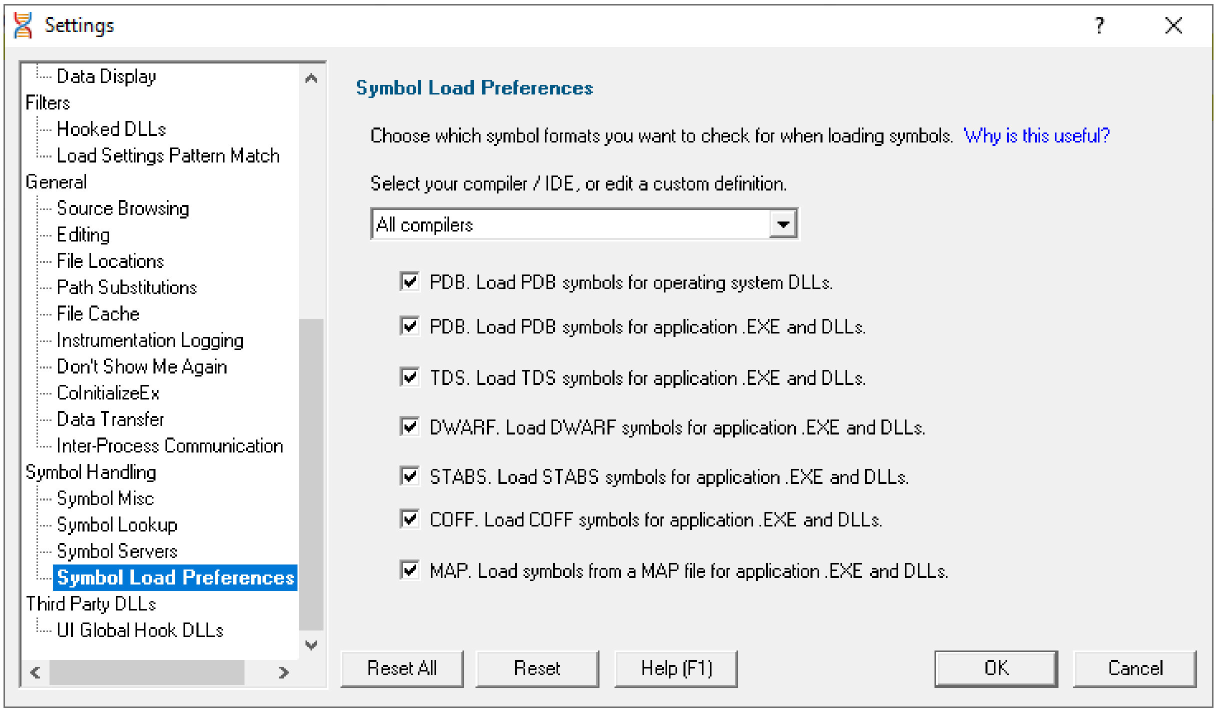
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.5.4 Symbol Load Preferences

The **Symbol Load Preferences** tab allows you to configure which debug information types are looked for and which are ignored.

This can save some time fetching symbols each time a DLL is loaded.



Symbol Load Preferences

Choose which symbol formats you want to check for when loading symbols. Why is this useful?

Select your compiler / IDE, or edit a custom definition.

All compilers

☑ PDB. Load PDB symbols for operating system DLLs.

☑ PDB. Load PDB symbols for application .EXE and DLLs.

☑ TDS. Load TDS symbols for application .EXE and DLLs.

☑ DWARF. Load DWARF symbols for application .EXE and DLLs.

☑ STABS. Load STABS symbols for application .EXE and DLLs.

☑ COFF. Load COFF symbols for application .EXE and DLLs.

☑ MAP. Load symbols from a MAP file for application .EXE and DLLs.

- **Select your compiler / IDE... ›** choose a preset definition for a compiler / IDE, or edit one of four custom symbol load preferences

The present definitions are:

- **I don't know which compilers ›** choose this if you don't know which compilers were used to build the software
- **All compilers ›** choose this to let Thread Validator fetch the symbols
- **Visual Studio ›** choose this if you're only using Visual Studio
- **Visual Basic 6 ›** choose this if you're only using Visual Basic 6
- **Delphi ›** choose this if you're only using Delphi
- **C++ Builder ›** choose this if you're using C++ Builder on 32 bit Windows
- **C++ Builder 32 bit ›** choose this if you're using C++ Builder to build 32 bit applications

- **C++ Builder 64 bit** › choose this if you're using C++ Builder to build 64 bit applications
- **MingW / gcc / g++ / QtCreator / Dev C++** › choose this if you're using MingW / gcc / g++
- **QtCreator / Dev C++** › choose this if you're using QtCreator
- **Dev C++** › choose this if you're using Dev C++
- **Rust** › choose this if you're using Rust
- **Salford Fortran 95** › choose this if you're using Salford Fortran 95
- **Custom 1** › choose this to edit a definition you can reuse
- **Custom 2** › choose this to edit a definition you can reuse
- **Custom 3** › choose this to edit a definition you can reuse
- **Custom 4** › choose this to edit a definition you can reuse

Editing a definition

Once a definition has been selected the appropriate check boxes next to each debug information type are populated.

You can edit these selections, for example to include or exclude PDB debug information for operating system DLLs, or allow Thread Validator to search for COFF debug information, whatever is optimal for the way you are working.

Custom definitions

Only the custom definitions will be remembered if they are edited.

The four custom definitions will be remembered, so the next time you choose them you'll get the definition you edited. If you choose one of the preset definitions and edit it, you'll use the edited definition, but if you then change to a different preset (or a custom definition) and then back to the original preset you'll get the preset definition, not your edited version of the preset definition.

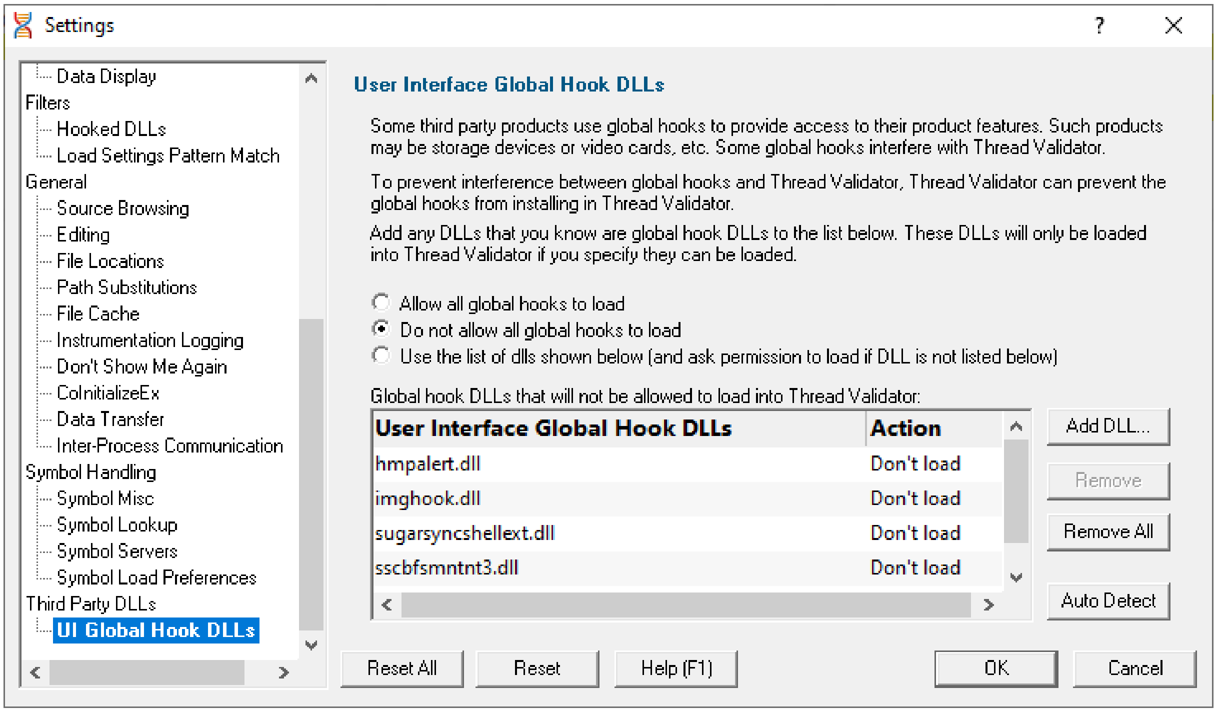
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.6 Third Party DLLs

3.10.1.6.1 UI Global Hook DLLs

The **User Interface Global Hook DLLs** page allows you to detect and specify global hook DLLs that may not be wanted in the Thread Validator user interface process.



User Interface Global Hook DLLs

Some third party products use global hooks to provide access to their product features. Such products may be storage devices or video cards, etc. Some global hooks interfere with Thread Validator.

To prevent interference between global hooks and Thread Validator, Thread Validator can prevent the global hooks from installing in Thread Validator.

Add any DLLs that you know are global hook DLLs to the list below. These DLLs will only be loaded into Thread Validator if you specify they can be loaded.

○ Allow all global hooks to load
● Do not allow all global hooks to load
○ Use the list of dlls shown below (and ask permission to load if DLL is not listed below)

Global hook DLLs that will not be allowed to load into Thread Validator:

User Interface Global Hook DLLs	Action
hmpalert.dll	Don't load
imghook.dll	Don't load
sugarsyncshellext.dll	Don't load
sscbfsmntnt3.dll	Don't load

About global hook DLLS

Some third party products such as storage devices and video cards are supplied with software to help integrate the hardware device into the computer desktop environment.

An example is the Iomega® Zip® drive. This uses a global hook via the IMGHOOK.DLL which allows the *browse for files* and *browse for folders* interfaces to correctly display all the storage devices on the computer, including the zip drive and any special options for the drive.

Some global (or *system*) hook DLLs can interfere with the correct operation of Thread Validator when it inserts hooks into the target program, (although the IMGHOOK.DLL mentioned above doesn't).

The settings below allow you to specify and/or detect DLLs that should be treated as global hook DLLs.

Any DLL listed will fail to load into the target program when loaded via `LoadLibrary()` or `LoadLibraryEx()`.

For situations where the hook DLL is already present in the target program, it can optionally be forcibly unloaded. This may happen if it was loaded before Thread Validator attached to the process.

The user interface hook DLL loading rule

The default behaviour is not to allow the global hooks to load, but you can change this if necessary:

- **Allow all global hooks to load** › allows all global hook DLLs to load into Thread Validator

- **Do not allow any global hooks to load** › prevent any global hook DLLs from loading (the default)

- **Use the list of dlls shown** › provide per-DLL control over which DLLs load or don't load via the **User Interface Global Hook DLLs** list

  Any global hook DLLs not listed will result in the user being asked for permission to load a DLL via the Global Hook Warning Dialog below.

Managing user interface global hook DLLs

- **Add DLL...** › browse and select one or more DLLs › **Open** › adds the chosen DLLs to the **Global Hook DLLs** list

  Having added a DLL to the list, you can change whether the DLL is allowed to load or not, by double clicking in the second column and changing the value: **Load** or **Don't load.**

- **Remove** › removes any selected DLL from the list

- **Remove All** › removes all DLLs from the list

Auto detecting global hook DLLs

Thread Validator can detect any DLLs in its own process that are not ones it uses itself. Such DLLs are likely to be global hook DLLs:

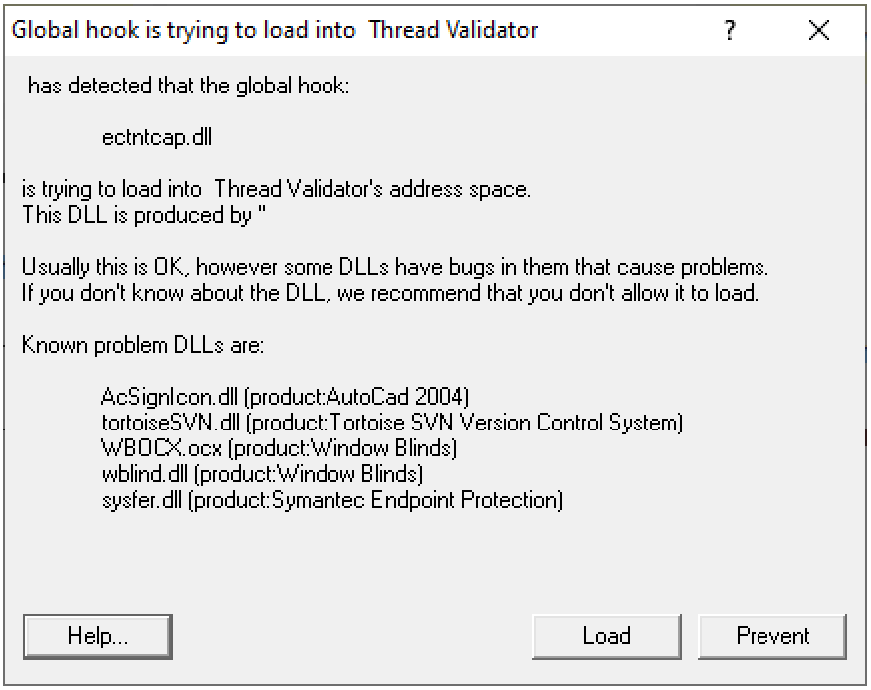
- **Auto Detect** › automatically detect DLLs which may be global hook DLLs, *adding* them to the **Global Hook DLLs** list

Global Hook Warning Dialog

When the global hook loading rule above is set to **Use the list of dlls shown**, the **Allow load** column controls whether the hook DLL is loaded.

When a global hook is loaded that is *not* on the list of known global hooks, the user is presented with a warning dialog like that shown below.

The user can then accept or block the global hook from loading. The dialog lists a couple of known problematic DLLs.



Global hook is trying to load into Thread Validator

has detected that the global hook:

ectntcap.dll

is trying to load into Thread Validator's address space.
This DLL is produced by "

Usually this is OK, however some DLLs have bugs in them that cause problems.
If you don't know about the DLL, we recommend that you don't allow it to load.

Known problem DLLs are:

AcSignIcon.dll (product:AutoCad 2004)
tortoiseSVN.dll (product:Tortoise SVN Version Control System)
WBOCX.ocx (product:Window Blinds)
wblind.dll (product:Window Blinds)
sysfer.dll (product:Symantec Endpoint Protection)

- **Help** › displays this help page
- **Yes** › lets the DLL load
- **No** › blocks the DLL

Your response is automatically recorded in the **Global Hook DLLs** list, so that you won't be asked again.

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

Reset - Resets the settings on the current page.

3.10.2 Loading and Saving Settings

Saving and loading settings files

Thread Validator settings can be saved to a file and restored at any time.

Settings menu › **Save Settings...** › save settings to a file

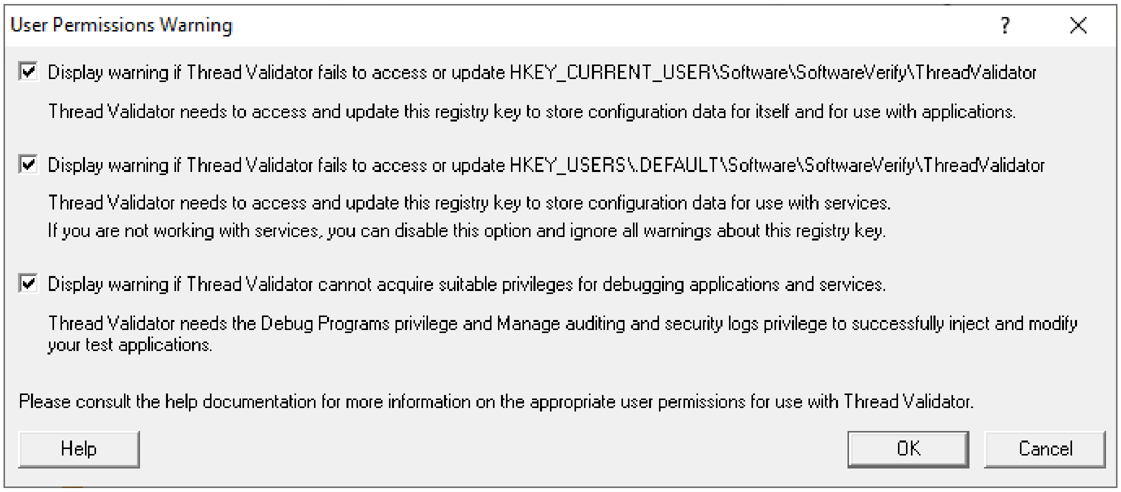
Settings menu › **Load Settings...** › load a previously saved settings file

3.10.3 User Permissions Warnings

You may see warning dialogs when Thread Validator receives an error accessing the registry or obtaining debugging privileges.

These warnings are enabled by default, but you can opt not to see them:

Settings menu › **User Permissions Warnings...** › shows the User Permissions Warnings dialog below


User Permissions Warning

☑ Display warning if Thread Validator fails to access or update HKEY_CURRENT_USER\Software\SoftwareVerify\ThreadValidator

Thread Validator needs to access and update this registry key to store configuration data for itself and for use with applications.

☑ Display warning if Thread Validator fails to access or update HKEY_USERS\.DEFAULT\Software\SoftwareVerify\ThreadValidator

Thread Validator needs to access and update this registry key to store configuration data for use with services.
If you are not working with services, you can disable this option and ignore all warnings about this registry key.

☑ Display warning if Thread Validator cannot acquire suitable privileges for debugging applications and services.

Thread Validator needs the Debug Programs privilege and Manage auditing and security logs privilege to successfully inject and modify your test applications.

Please consult the help documentation for more information on the appropriate user permissions for use with Thread Validator.

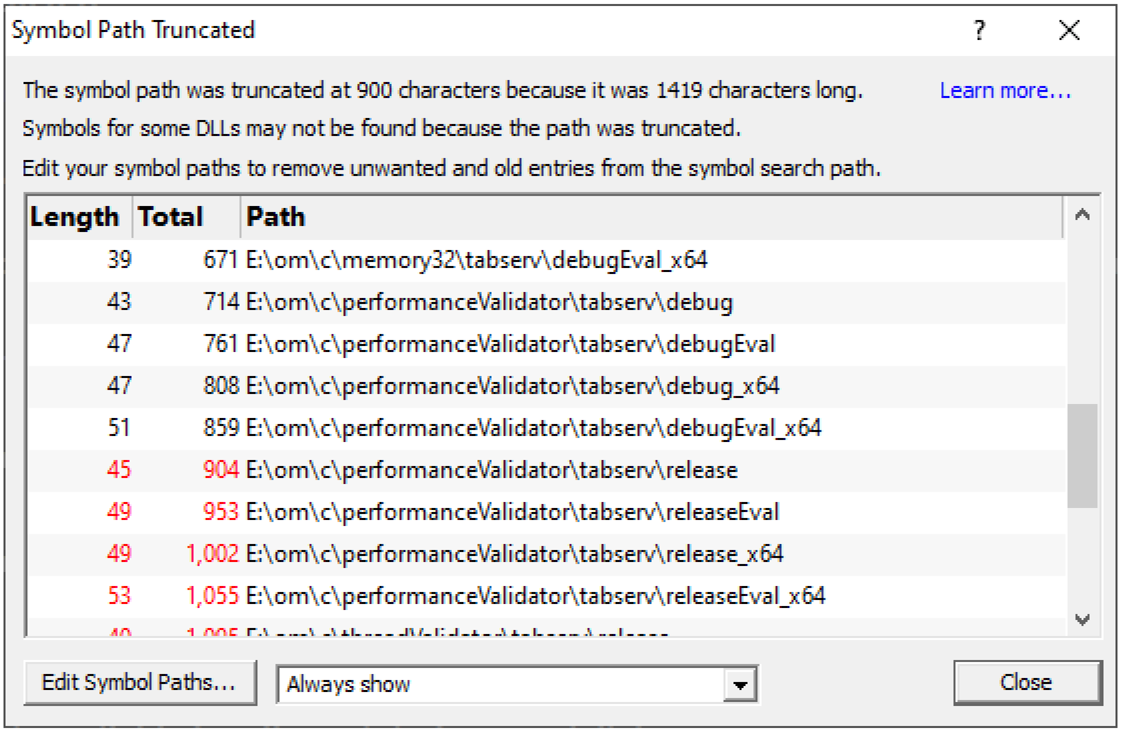
The Help button displays the User Permissions help topic.

➡ See also, the question about creating Power User accounts on Windows XP.

3.10.4 Symbol Path Truncated Warning

The Symbol Path Truncated warning dialog

The symbol path truncated warning dialog is displayed to warn you when the symbol path is too long.



Symbol Path Truncated

The symbol path was truncated at 900 characters because it was 1419 characters long.

Symbols for some DLLs may not be found because the path was truncated.

Edit your symbol paths to remove unwanted and old entries from the symbol search path.

Length	Total	Path
39	671	E:\om\c\memory32\tabserv\debugEval_x64
43	714	E:\om\c\performanceValidator\tabserv\debug
47	761	E:\om\c\performanceValidator\tabserv\debugEval
47	808	E:\om\c\performanceValidator\tabserv\debug_x64
51	859	E:\om\c\performanceValidator\tabserv\debugEval_x64
45	904	E:\om\c\performanceValidator\tabserv\release
49	953	E:\om\c\performanceValidator\tabserv\releaseEval
49	1,002	E:\om\c\performanceValidator\tabserv\release_x64
53	1,055	E:\om\c\performanceValidator\tabserv\releaseEval_x64

Always show

- **Edit Symbol Paths...** › shows the file locations dialog so that you can edit the paths used for subsequent runs of the program.

You can choose when this dialog is displayed.

- **Always show** › The dialog is always shown when the symbol path is too long.
- **Show when path changes** › The dialog is shown when the symbol path is too long, but only if the symbol path is different than last time this warning was shown.
- **Never show** › The dialog is never shown.

Whether this dialog is displayed or not there is always a warning message written to the diagnostic window when the symbol path is truncated.

The display lists each path with it's length (including the unshown ';' path separator) and the total length so far so that you can see which paths exceed the truncation point (length and total displayed in red).

Any paths that don't exist on this computer are displayed in red.

Why is this dialog displayed?

You may see a Symbol Path Truncated warning dialog in some rare circumstances.

This dialog is displayed when the symbol path that has been calculated to pass to DbgHelp.dll to load Microsoft debugging symbols (found in .PDB files) is too long.

If the symbol path has been truncated because it is too long it is possible this may mean that some symbol searches will fail, resulting in failure to load some symbols. We display this dialog so that you are aware that the symbol path is too long and would benefit from editing to make the symbol path shorter.

Passing a symbol path that is too long to DbgHelp.dll will cause the program being tested to end with an EXCEPTION_INVALID_CRUNTIME_PARAMETER C runtime error. This happens because internally DbgHelp.dll is using a fixed length array to format a string. To prevent this fatal termination of the test program we limit the length of the path passed to DbgHelp.dll.

Typically if a path that is long enough to cause this problem is passed to DbgHelp it's because the number of paths in the calculated path contain paths not relevant to finding symbols for the test program. We use the Symbol Path Truncated warning dialog to show you the calculated paths so that you can work out which paths to delete.

The calculated symbol paths come from several places:

- File locations PDB paths
- Symbol server symbol storage directories
- Symbol handling environment variables

Fixing the symbol path

For this example, we are testing the program **E:\om\c\3RD_SRC\cdplayer\Release\cdplayer.exe**

In the image shown above you can see that seven paths exceed the truncation limit, one of the 7 paths doesn't exist.

To work out what to do we need to do several actions:

1. Looking at the environment variable settings shows that none of the environment variables are being used. We do not need to consider the content of these environment variables.

2. Examining the symbol servers shows that **C:\Users\Admin** is a local symbol storage location. We should keep this path.

3. We should delete the path that doesn't exist: **E:\om\c\testApps\testStdinStdoutRedirectEx\Release**. We do this using the file locations dialog by clicking **Edit Symbol Paths...** then click **Delete invalid**.

4. Examining the paths in the file locations dialog we can identify any paths not relevant to the program we are testing. In this case the following paths are not relevant and can be deleted.

E:\om\c\testApps\testStdinStdoutRedirect\Release
E:\om\c\testApps\testAppTheReadsFromStdinAndWritesToStdout\Release
E:\om\c\testApps\testSimpleMemoryLeak\Release
e:\om\c\3rd_src\cppunit-1.12.1\examples\cppunittest\release
e:\om\c\3rd_src\cppunit-1.12.1\examples\cppunittest\releasedll

3.11 Ordinal Handling

Ordinal values and .def files

Some DLLs, including some from Microsoft, may export their functions by ordinal value, instead of by the usual readable name.

However, having access to the module's original .def file means those ordinal values can be used to look up the symbol names to display in Thread Validator.

The .def file will contain the function names and ordinal values, allowing a DLL's exported ordinal value to be mapped to the symbol name.

For example, here's a small section of mfc90.def showing the ordinal values 332 to 335 and a selection of decorated names:

```
??0CBrush@@QEAA@K@Z @ 332 NONAME
??0CBrush@@QEAA@PEAVCBitmap@@@Z @ 333 NONAME
??0CByteArray@@QEAA@XZ @ 334 NONAME
??0CChevronOwnerDrawMenu@@QEAA@XZ @ 335 NONAME
```

You can use the ordinal handler dialog (below) to associate a def file with a DLL. These associations will persist between sessions.

If you need different ordinal configurations for different DLL usage, you can export this mapping between DLL and .def to a file, to be used at a later date.

Switching usage of mapped names on and off

The option to switch the ordinal mapping on or off is in the global settings dialog:

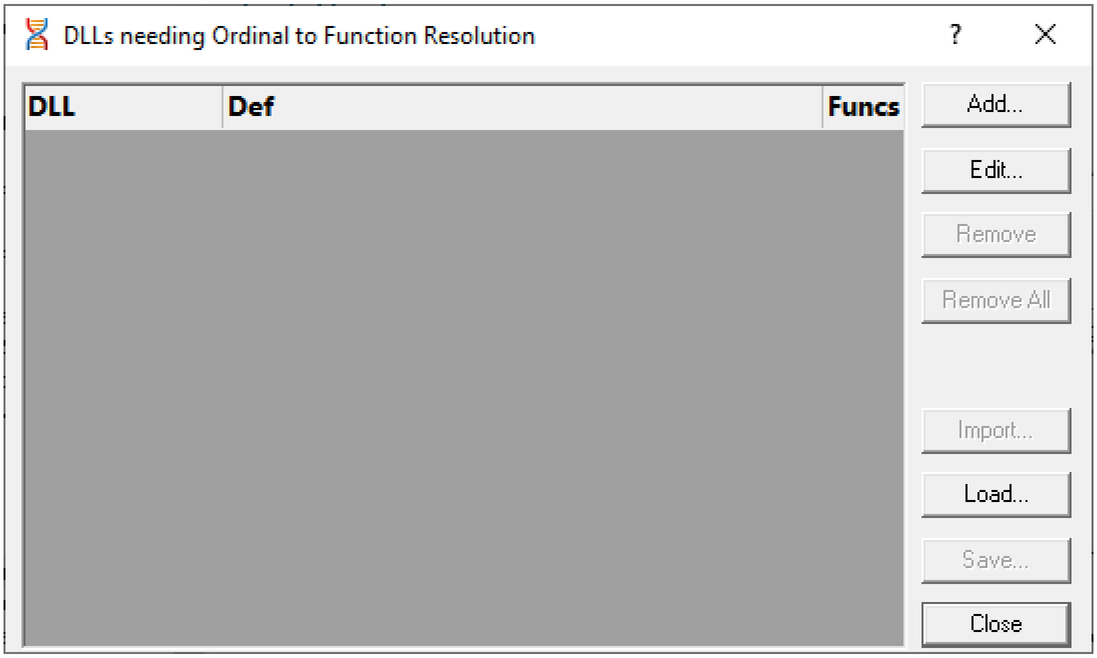
Settings menu › **Edit Settings...** › **General** › **Symbols and Warnings** page › **Convert DLL exported function ordinals to symbols** › enable the ordinal to function name mapping

> You'll need to tick this to enable the use of mapped names defined in the dialog below. If you don't, you won't see the names being used.

The ordinal handler dialog

The ordinal handler dialog lets you manage which .def files are associated with which DLLs.

Settings menu › **Edit Settings...** › **General** › **Symbols and Warnings** page › **Manage Ordinals...** › shows the ordinal handler dialog below



- **Add...** › shows the ordinal-to-function converter dialog, described below, so you can add a new mapping

- **Edit...** › opens a selected mapping in the ordinal-to-function converter dialog

  Double clicking an item in the list also does this.

- **Remove** › removes the selected associations from the list

- **Remove All** › clears all the associations in the list

- **Import...** › choose a previously saved set of associations to ***add*** to the list

- **Load...** › as for Import, but ***replaces*** the contents of the list

- **Save...** › saves the associations in the list to a .ord file of your choice

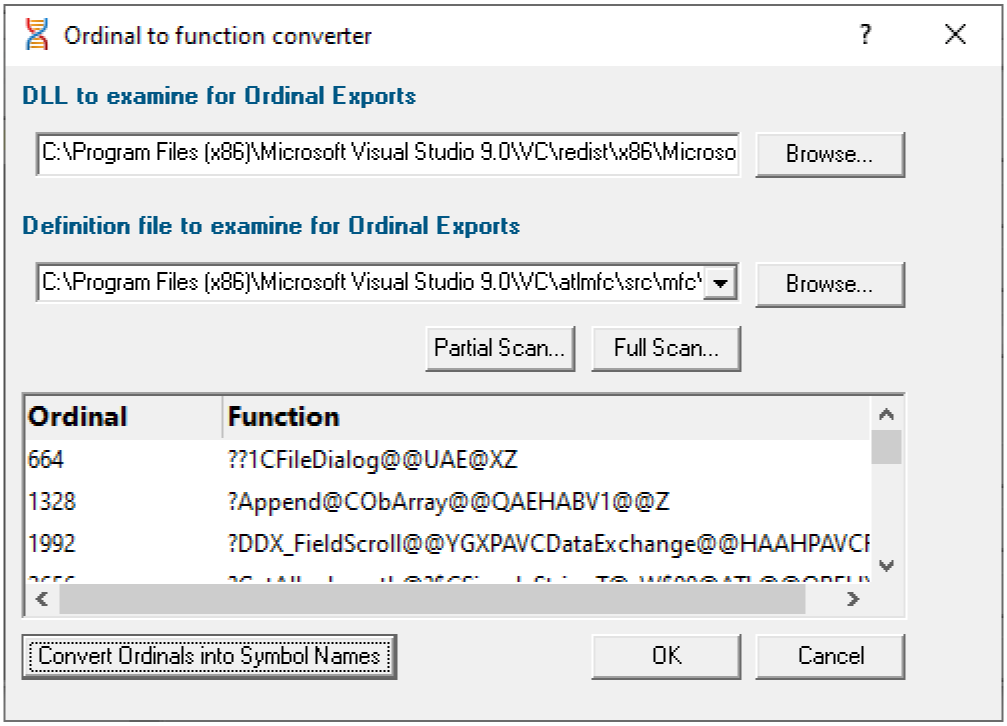
The ordinal-to-function converter dialog

After clicking Add... on the Ordinal Handler Dialog (above) you'll see the Ordinal to function converter dialog below.

The basic process for adding a new mapping is to:

- choose a DLL
- find its matching .def file
- convert the ordinal values to symbol names
- add the file association to the Ordinal Handler dialog

This example shows the Microsoft MFC90.dll associated with its .def file:


Ordinal to function converter
?
Browse...
Browse...
Partial Scan...
Full Scan...
Convert Ordinals into Symbol Names
OK
Cancel

DLL to examine for Ordinal Exports

C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\redist\x86\Microso

Definition file to examine for Ordinal Exports

C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\

Ordinal	Function
664	??1CFileDialog@@UAE@XZ
1328	?Append@CObArray@@QAEHABV1@@Z
1992	?DDX_FieldScroll@@YGXPAVCDataExchange@@HAAHPAVCF

Associating the DLL and .def files

In the Ordinal to function converter dialog:

- **DLL to examine...** › type or **Browse** to enter the DLL you want to map
- **Definition file to examine...** › type or **Browse** to enter the .def file you want to associate with the DLL, or choose one from the drop-down list (see scanning for files below)

  If a .def file matching the name of the .dll exists in the same directory as the DLL, this will be set automatically.

- **Convert Ordinals into Symbol Names** › convert the exported ordinals from the chosen DLL into symbol names using the specified .def file

  All the ordinals and symbol names found will be displayed in the list at the bottom.

- **OK** › closes the dialog and adds the association between the DLL and the .def to the ordinal handler dialog

Don't forget to select the **Convert DLL exported function ordinals to symbols** check box.

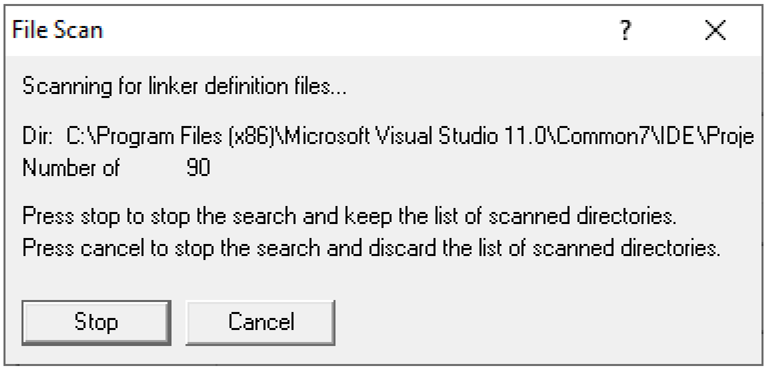
Scanning for .def files

The .def files aren't always in the same directory as the DLLs and may be hard to find, so a search option is available.

Any .def files found during a scan can be used to extend or replace the choices in the drop-down list of the **Definition file to examine...** option above.

- **Partial Scan...** › choose a folder and scan it for .def files
- **Full Scan...** › scan all drives for .def files

While scanning, a progress dialog shows the search location and the number of .def files found:



File Scan

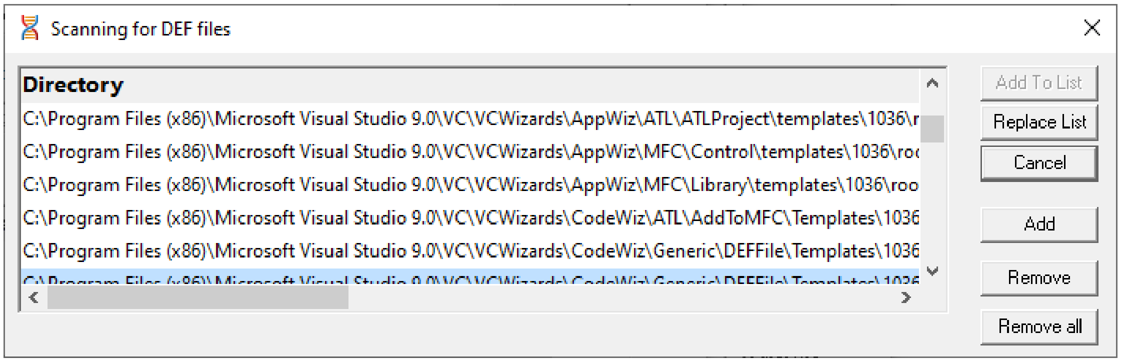
Scanning for linker definition files...

Dir: C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\Proje
Number of   90

Press stop to stop the search and keep the list of scanned directories.
Press cancel to stop the search and discard the list of scanned directories.

- **Stop** › stop the scan and show results found so far
- **Cancel** › stop the scan and discard any results

When the scan is complete a dialog shows any .def files found:



Scanning for DEF files

Directory
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\AppWiz\ATL\ATLProject\templates\1036\r
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\AppWiz\MFC\Control\templates\1036\ro
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\AppWiz\MFC\Library\templates\1036\roo
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\CodeWiz\ATL\AddToMFC\Templates\1036
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\CodeWiz\Generic\DEFFile\Templates\1036
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\CodeWiz\Generic\DEFFile\Templates\1036

- **Add To List** › extend the drop-down list of the **Definition file to examine...** option by adding all the search results in the list
- **Replace List** › as above but replaces the list rather than extending it
- **Add** › adds an entry to the list so you can manually enter a file path
- **Remove** › remove any selected items from the list

- **Remove All** › clears the list

- **Cancel** › closes the dialog, discarding the list of results

3.12 Managers

The Managers menu provides a handful of powerful tools to manage or inspect data collected by Thread Validator.

The tools include:

- session management

- thread filters

- watermarks and bookmarks

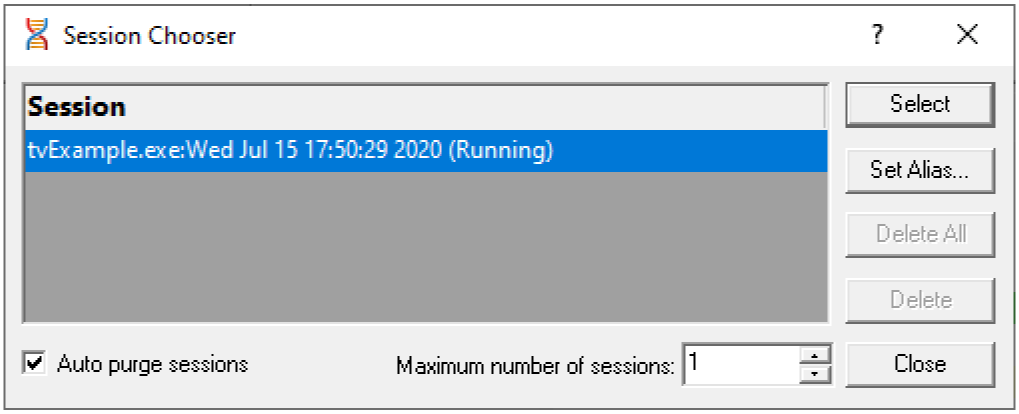
3.12.1 Session Manager

Managing multiple sessions

Thread Validator can manage multiple sessions at once.

As well as the actively running session, open sessions may include those run since Thread Validator started, or reloaded sessions that had been saved earlier.

Managers menu › **Session Manager...** › shows the **Session Chooser** dialog below, highlighting the current session



Session
tvExample.exe:Wed Jul 15 17:50:29 2020 (Running)

☑ Auto purge sessions

Maximum number of sessions: 1

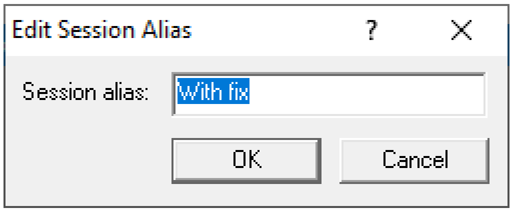
Each time a session is started or loaded it is added to this list, using the name of the executable program and the date and time the session started.

Managing the sessions

- **Select** › makes the selected entry the *current* session, i.e. the one for which data will be displayed

  Some tab views may update immediately, others may need a manual refresh.

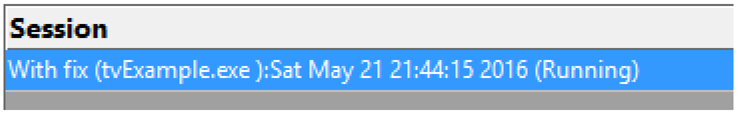
- **Set Alias...** › opens the **Edit Session Alias** dialog so you can give the session a more useful name

  Edit Session Alias

  Session alias: With fix

  The alias is added before the session details:

  Session

  With fix (tvExample.exe):Sat May 21 21:44:15 2016 (Running)

- **Delete** › removes the selected session

  You can't delete a session that is actively collecting data.

- **Delete All** › removes all the loaded sessions

  If one of the session is actively collecting data, this will be disabled.

- **Close** › closes the dialog (as opposed to closing any selected sessions!)

Limiting the number of sessions

You can choose to limit the maximum number of sessions open at once.

Once this limit is reached, then each time a new session is added, the oldest session may automatically be removed:

- **Auto purge sessions** › ensures that the number of loaded sessions is limited to the maximum (below)

- **Maximum number of sessions** › sets the maximum number of sessions allowed if auto-purge is on

3.12.2 Thread Filter

Thread filtering affects data displayed on All Locks, Locks Per Thread, Active Locks, Thread Locks, Active Objects and Analysis tabs

Thread filters

Thread filtering simply allows you to exclude data according to the thread id.

Filtering by thread id is only going to be useful for multi-threaded applications.

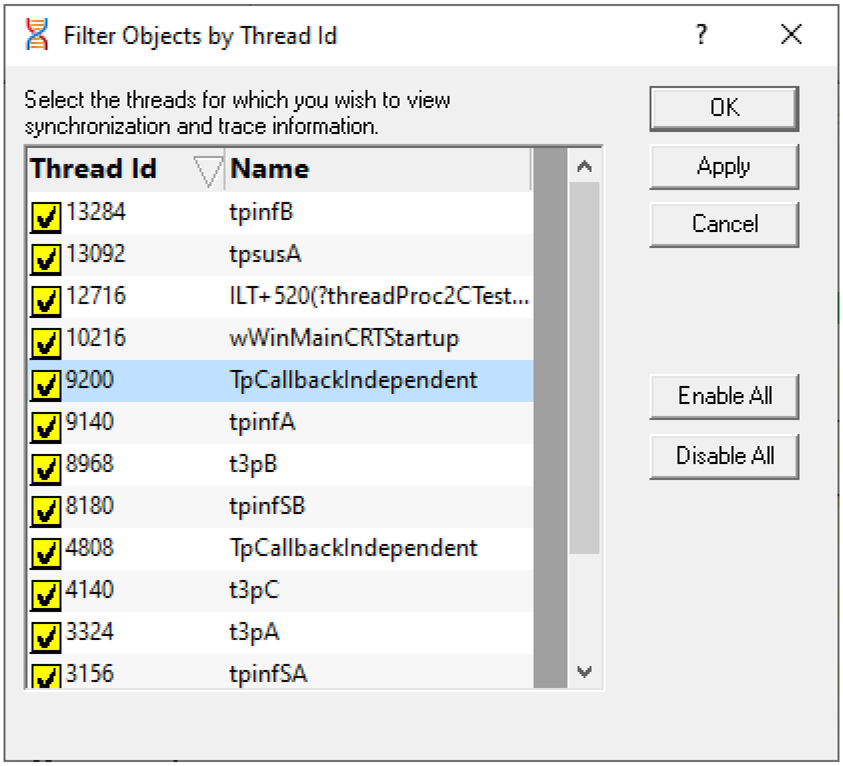
Many applications have threads that you may want to exclude from your data as being beyond your control.

The thread filter dialog below also serves the dual purpose of manually naming threads. The names are used elsewhere in the display of data and selection of threads.

The thread filter dialog

Managers menu › **Thread Filter...** › shows the Filter Objects by Thread Id dialog

The dialog has a list of all the threads in the target program, with check boxes to enable and disable the display of data from each thread id.



Select the threads for which you wish to view synchronization and trace information.

Thread Id	Name
☑ 13284	tpinfB
☑ 13092	tpsusA
☑ 12716	ILT+520(?threadProc2CTest...
☑ 10216	wWinMainCRTStartup
☑ 9200	TpCallbackIndependent
☑ 9140	tpinfA
☑ 8968	t3pB
☑ 8180	tpinfSB
☑ 4808	TpCallbackIndependent
☑ 4140	t3pC
☑ 3324	t3pA
☑ 3156	tpinfSA

- **Enable All** › checks all threads, meaning that no thread filtering will occur
- **Disable All** › unchecks all threads, making it easier to enable just one or two of many threads

To sort the list on id or name, click one of the headers.

To edit a thread name, double click on the name.

Thread names

If a thread has been named using the Win32 `RaiseException` method, the Win32 `SetThreadDescription()`, or using `tvSetThreadName()` its name is shown in the list above. See the link below for more details.

For unnamed threads, you can give it a name here by double clicking on the name column and entering a name for the thread. Click outside the box or press return to complete the entry.

Names of threads will be used where relevant in other parts of the application such as the **Thread** or **Owning Module** columns in tabular data or in callstack information.

➡ How can I give a name to a thread from my code?

3.12.3 Watermarks

Watermarks

In Thread Validator, Watermarks are event *markers* in the allocation history at which you can say other events occurred either before or after the watermark.

Watermarks are used in the Active Objects and Analysis tabs as a kind of filter to constrain displayed data to that which happened between two watermarks.

Adding watermarks

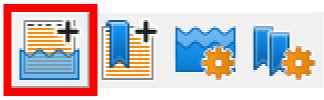
You can add new watermarks directly from allocation items displayed in the Active Objects and Analysis tabs, using the popup menu options.

Watermarks added this way are initially named using the .exe or DLL and the function name as in the picture below, but you can change this if you want.

Alternatively, you can add a watermark at the most recent recorded allocation event:

☰ **Managers** menu › choose **Add watermark at most recent trace** › **enter** a name › click **OK**

Or use the option on the Watermarks Toolbar:



The most recent trace event may not actually be visible in any of the displays as it could be filtered or hidden for other reasons.

First and last watermarks

There are two permanent special watermarks, not directly associated with particular events:

- The **first watermark** is the point *before* every other event
- The **last watermark** is the point *after* every other event

These two watermarks are the default settings, meaning that no data is filtered due to watermarks.

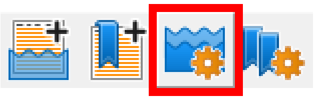
You cannot remove or rename the first and last watermarks.

The watermarks dialog

When a session is active, you can show the watermark manager to see a list of watermarks, change their names, or apply selected watermarks to the data views:

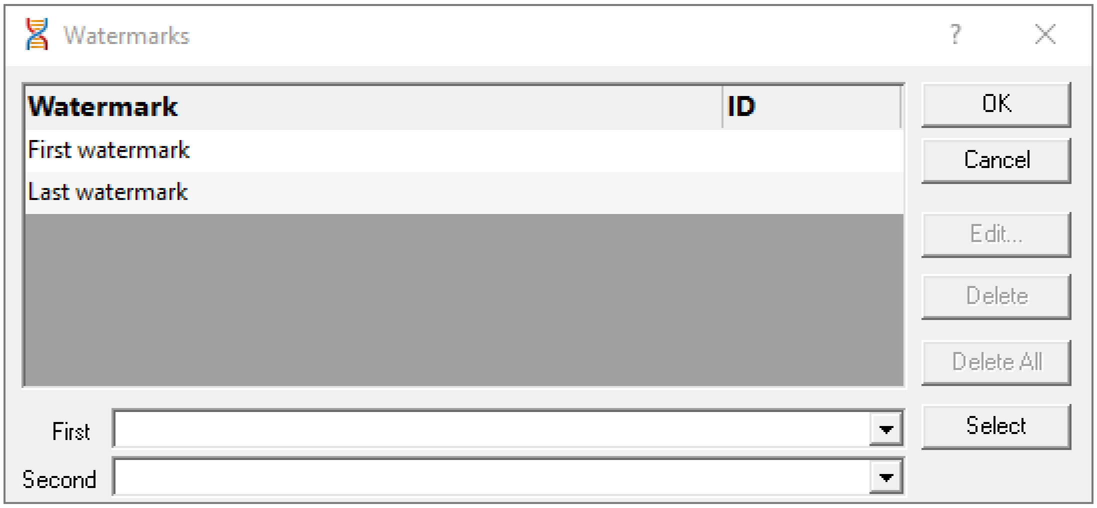
Managers menu › **Watermark Manager...** › shows the Watermarks dialog

Or use the Watermarks Toolbar option:



Watermarks are shown in order of their associated event allocation history.

If you haven't added any watermarks yet, this will just be the special first and last watermarks.



Watermark	ID
First watermark	
Last watermark	

The watermarks dialog can only be shown when the Active Objects or Analysis tab is open.

Managing the watermarks

There's a few options for renaming and removing watermarks:

- **Edit...** › rename the selected watermark

  Double clicking on the watermark also works.

  You can't edit the first and last watermarks.

- **Delete** › delete the selected watermark

  You can't remove the first and last watermarks.

- **Delete All** › delete all the watermarks except the first and last

You can't change watermark locations. If you want to do that, delete the watermarks you don't want and add new ones.

Applying watermarks to the data displays

You can override the local watermark settings in the Active Objects and Analysis tabs.

At the bottom of the watermarks dialog choose the watermarks:

- **First** › set the earlier of the watermark range

- **Second** › set the later of the watermark range

  You can't choose a second watermark which is earlier than (or the same as) the first one.

3.12.4 Bookmarks

Bookmarks

Bookmarks are event *markers* in the allocation history. You can use the Bookmarks dialog to jump back to a bookmarked location any time.

Bookmarks are only used in the Active Objects and Analysis tabs.

Adding bookmarks

Adding bookmarks is very similar to adding watermarks.

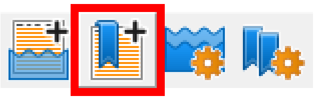
You can add new bookmarks directly from allocation items displayed in the Active Objects and Analysis tabs, using the popup menu options.

Bookmarks added this way are initially named using the .exe or DLL and the function name as in the picture below, but you can change this if you want.

Alternatively, you can add a bookmark at the most recent recorded allocation event:

Managers menu › choose **Add bookmark at most recent trace** › **enter** a name › click **OK**

Or use the option on the Watermarks Toolbar:



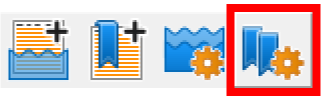
The most recent trace event may not actually be visible in any of the displays as it could be filtered or hidden for other reasons.

The bookmarks dialog

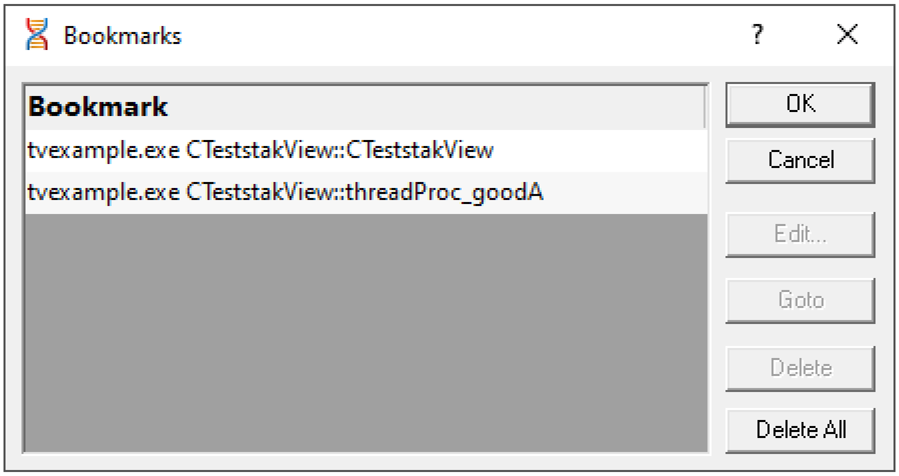
When a session is active, you can show the bookmark manager to see a list of bookmarks, change their names, or jump to a bookmark location:

Managers menu › **Bookmark Manager...** › shows the Bookmarks dialog

Or use the Watermarks Toolbar option:



Unlike watermarks, bookmarks are shown in the order you add them.



Bookmarks

Bookmark
tvexample.exe CTeststakView::CTeststakView
tvexample.exe CTeststakView::threadProc_goodA

The bookmarks dialog can only be shown when the Active Objects or Analysis tab is open.

Jumping to bookmark locations

The most useful option in this dialog is the Goto:

- **Goto** › scrolls the open tab to the selected bookmark and selects it (if it exists in the current tab)

  Double clicking on the bookmark also jumps to its location.

Managing bookmarks

There's also a few options for renaming and removing bookmarks:

- **Edit...** › rename the selected bookmark
- **Delete** › delete the selected bookmark
- **Delete All** › delete all the bookmarks in the list

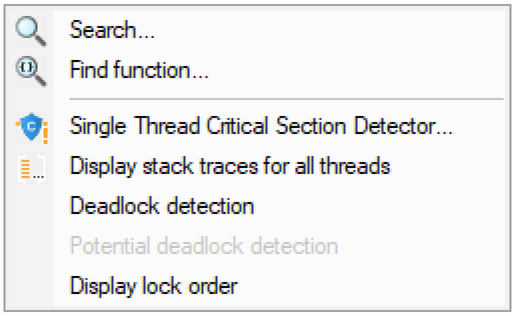
You can't change bookmark locations. If you want to do that, delete the bookmarks you don't want and add new ones.

3.13 Query and Search

Tools to search for allocations

The following tools help you find memory and handle allocations using different criteria and are all found on the Query Menu.

Click on an item in the picture or in the list below to find out more in the following topics.



Search...
Find function...
Single Thread Critical Section Detector...
Display stack traces for all threads
Deadlock detection
Potential deadlock detection
Display lock order

- **Search** › use the Find Sychronization Object dialog to search the data in some of the tab views for synchronization events

- **Find Function** › use the Find Functions dialog to search for synchronization enter/exits and synchronization object allocations/deallocation occurring in certain functions

- **Single Thread Critical Section Detector** › use the Single Thread Critical Section Detector dialog detect potentially unnecessary synchronization objects that have only been used in one thread

- **Display stack traces for all threads** › use the Deadlocked Thread PostMortem dialog to analyse your application that has already deadlocked

- **Deadlock detection** › Perform deadlock detection in your application

- **Potential deadlock detection** › Perform potential deadlock detection in your application

- **Display lock order** › use the Locks and Waits in Sequence Order dialog to display the order in which locks and waits were entered

3.13.1 Finding synchronization objects

Searching for synchronization objects

Using the Find Synchronization Object dialog below, you can search for objects in the Active Objects Tab.

Searches can be:

- based on allocation location such as in a function, file or module

- based on the allocated object such as type or address

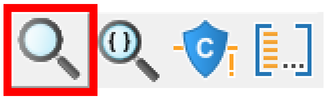
The Analysis Tab has its own Find function which does the same search within data collected in its own view.

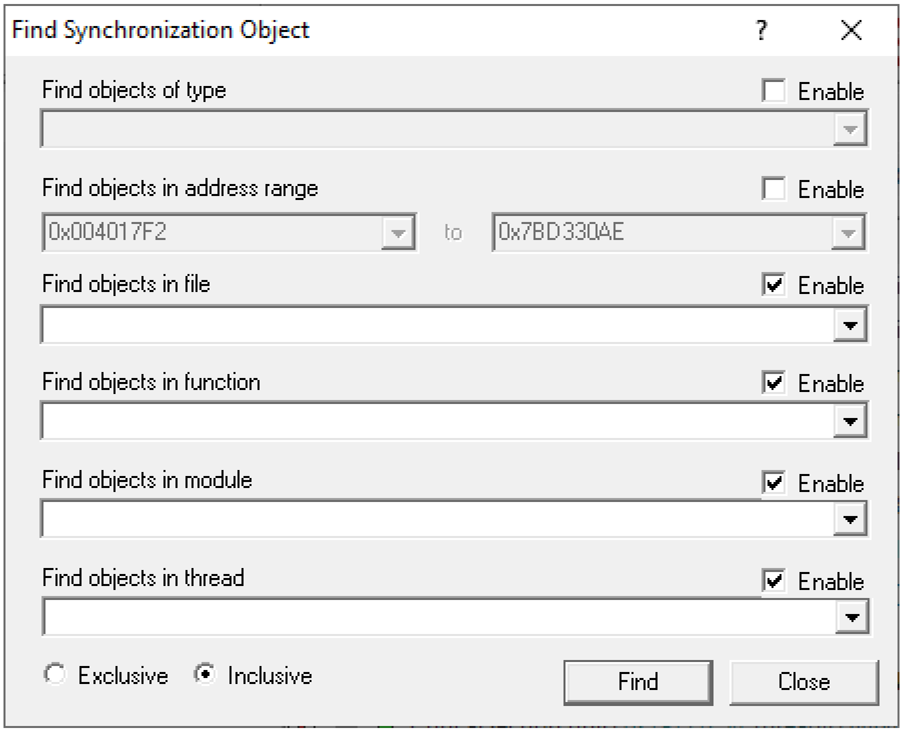
The Find Synchronization Object dialog

To show the Find Synchronization Object dialog, choose the menu option below:

Query menu › choose **Search...** › displays the dialog below

Or use the following icon on the Query Toolbar.





Find Synchronization Object

Find objects of type □ Enable

Find objects in address range □ Enable

0x004017F2 to 0x7BD330AE

Find objects in file ☑ Enable

Find objects in function ☑ Enable

Find objects in module ☑ Enable

Find objects in thread ☑ Enable

○ Exclusive ● Inclusive

Note that this dialog can remain open while you inspect the results.

Search Criteria

Each of the desired search options needs to be enabled and criteria specified:

To find objects...	enable the option and choose...
• **of type**	a data type from the drop down list
• **in function**	a function from the list
• **allocated in file**	a file from the list
• **in module**	an .exe, DLL or other module

- **in thread**	a thread or "Any" to accept any thread
Objects between...	
- **addresses**	type in the values or choose the current minimum or maximum from the drop down list

For options where there is a set choice of values, the lists show all the options that Thread Validator knows about at the time the dialog is shown.

Where a range is specified, you can enter the same value for the beginning and end of the range.

Exclusive or inclusive searches

There are two ways of searching:

- **Exclusive** › search results satisfy ***all*** the enabled search criteria

- **Inclusive** › search results satisfy ***at least one*** of the enabled search criteria

As a general rule, exclusive searches return a tightly focused set of results, while inclusive searches return a broader set of results.

For those who prefer logic notation, exclusive searches use AND logic, while inclusive use OR logic.

Performing the search

Once search criteria are enabled, and search mode selected, results can be found:

- **Find...** › matching results in the corresponding tab are highlighted

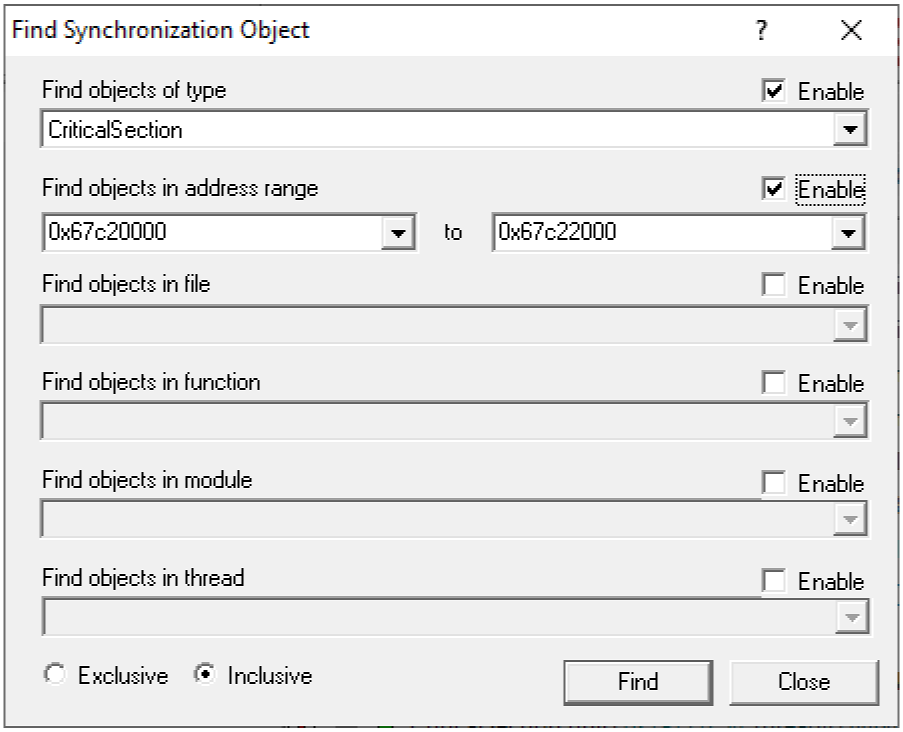
  The colour of selected objects is set on the colours tab of the Global Settings dialog.

Example search

Here are two example searches performed on the Example Application, one inclusive, one exclusive.

Inclusive

Search for type `CriticalSection`, in the range `0x67c20000` to `0x67c22000`, inclusively. All other search criteria are disabled.



Find Synchronization Object

Find objects of type ☑ Enable
CriticalSection

Find objects in address range ☑ Enable
0x67c20000 to 0x67c22000

Find objects in file □ Enable

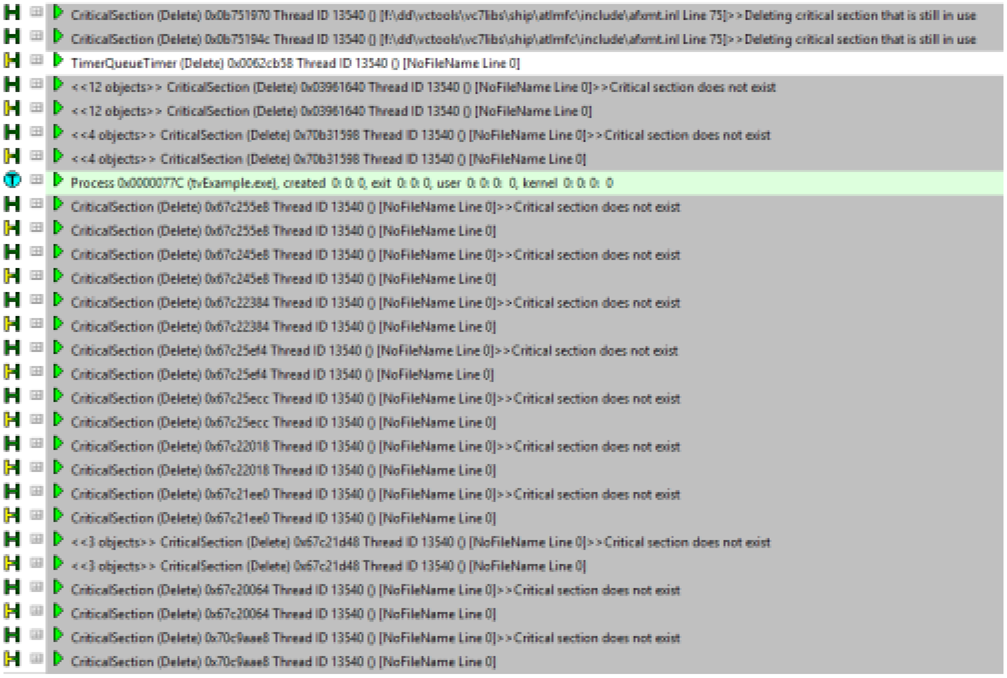
Find objects in function □ Enable

Find objects in module □ Enable

Find objects in thread □ Enable

○ Exclusive ● Inclusive

The results are shown below - all `CriticalSection` objects are selected.

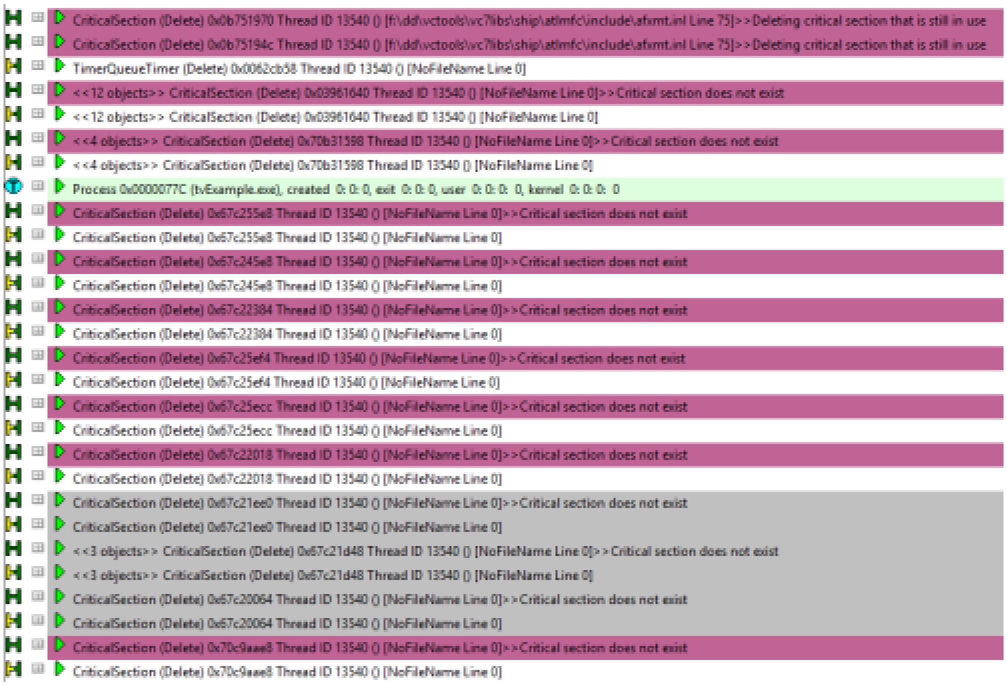


```
CriticalSection (Delete) 0x0b751970 Thread ID 13540 () [f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 75]>>Deleting critical section that is still in use
CriticalSection (Delete) 0x0b75194c Thread ID 13540 () [f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 75]>>Deleting critical section that is still in use
TimerQueueTimer (Delete) 0x0062cb58 Thread ID 13540 () [NoFileName Line 0]
<<12 objects>> CriticalSection (Delete) 0x03961640 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
<<12 objects>> CriticalSection (Delete) 0x03961640 Thread ID 13540 () [NoFileName Line 0]
<<4 objects>> CriticalSection (Delete) 0x70b31598 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
<<4 objects>> CriticalSection (Delete) 0x70b31598 Thread ID 13540 () [NoFileName Line 0]
Process 0x0000077C (tvExample.exe), created  0: 0: 0, exit  0: 0: 0, user  0: 0: 0:  0, kernel  0: 0: 0:  0
CriticalSection (Delete) 0x67c255e8 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c255e8 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c245e8 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c245e8 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c22384 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c22384 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c25ef4 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c25ef4 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c25ecc Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c25ecc Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c22018 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c22018 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c21ee0 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c21ee0 Thread ID 13540 () [NoFileName Line 0]
<<3 objects>> CriticalSection (Delete) 0x67c21d48 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
<<3 objects>> CriticalSection (Delete) 0x67c21d48 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c20064 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c20064 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x70c9aae8 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x70c9aae8 Thread ID 13540 () [NoFileName Line 0]
```

Exclusive

The search is repeated using same search criteria as above, but with the exclusive option selected.

The results are shown below and only `CriticalSection` objects in the specified address range are now selected.



```
CriticalSection (Delete) 0x0b751970 Thread ID 13540 () [f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 75]>>Deleting critical section that is still in use
CriticalSection (Delete) 0x0b75194c Thread ID 13540 () [f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 75]>>Deleting critical section that is still in use
TimerQueueTimer (Delete) 0x0062cb58 Thread ID 13540 () [NoFileName Line 0]
<<12 objects>> CriticalSection (Delete) 0x03961640 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
<<12 objects>> CriticalSection (Delete) 0x03961640 Thread ID 13540 () [NoFileName Line 0]
<<4 objects>> CriticalSection (Delete) 0x70b31598 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
<<4 objects>> CriticalSection (Delete) 0x70b31598 Thread ID 13540 () [NoFileName Line 0]
Process 0x0000077C (tvExample.exe), created  0: 0: 0, exit  0: 0: 0, user  0: 0: 0:  0, kernel  0: 0: 0:  0
CriticalSection (Delete) 0x67c255e8 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c255e8 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c245e8 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c245e8 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c22384 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c22384 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c25ef4 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c25ef4 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c25ecc Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c25ecc Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c22018 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c22018 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c21ee0 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c21ee0 Thread ID 13540 () [NoFileName Line 0]
<<3 objects>> CriticalSection (Delete) 0x67c21d48 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
<<3 objects>> CriticalSection (Delete) 0x67c21d48 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x67c20064 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x67c20064 Thread ID 13540 () [NoFileName Line 0]
CriticalSection (Delete) 0x70c9aae8 Thread ID 13540 () [NoFileName Line 0]>>Critical section does not exist
CriticalSection (Delete) 0x70c9aae8 Thread ID 13540 () [NoFileName Line 0]
```

3.13.2 Finding functions

Searching for object allocations in functions

Using the Find Function dialog below, you can search for objects occurring in certain functions.

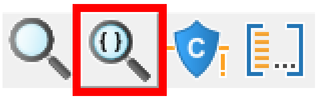
The functions can be the allocation/creation function or anywhere in the callstack

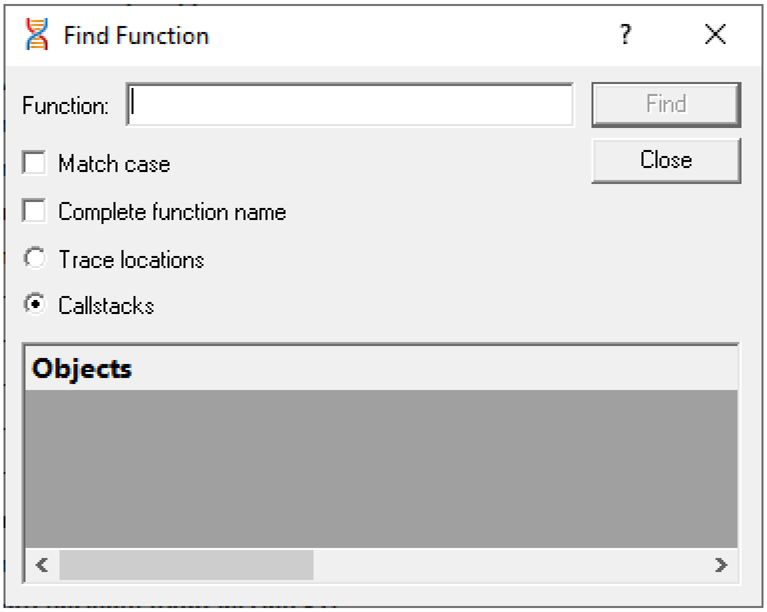
The find function dialog

To show the Find Function dialog, choose the menu option below:

Query menu › choose **Find Function...** › displays the Find Function dialog

Or use the following icon on the Query Toolbar.





Search criteria

Enter a function name, and optionally any other search characteristics to find objects allocated in matching functions

- **Function** › enter full or partial function name
- **Match case** › tick to do a case-sensitive match
- **Complete function name** › tick to only match the whole name

  For C++ methods, complete names must be of the form `classname::methodname`.

- **Trace locations** › matches only the function containing the object allocation
- **Callstacks** › matches any function in the allocation callstack

Finding results

- **Find** › performs the search displaying results in the list

  Results *replace* any previous search.

You can expand the search results, and double click the data items to edit source code in your preferred editor.

Examples of finding allocations in functions

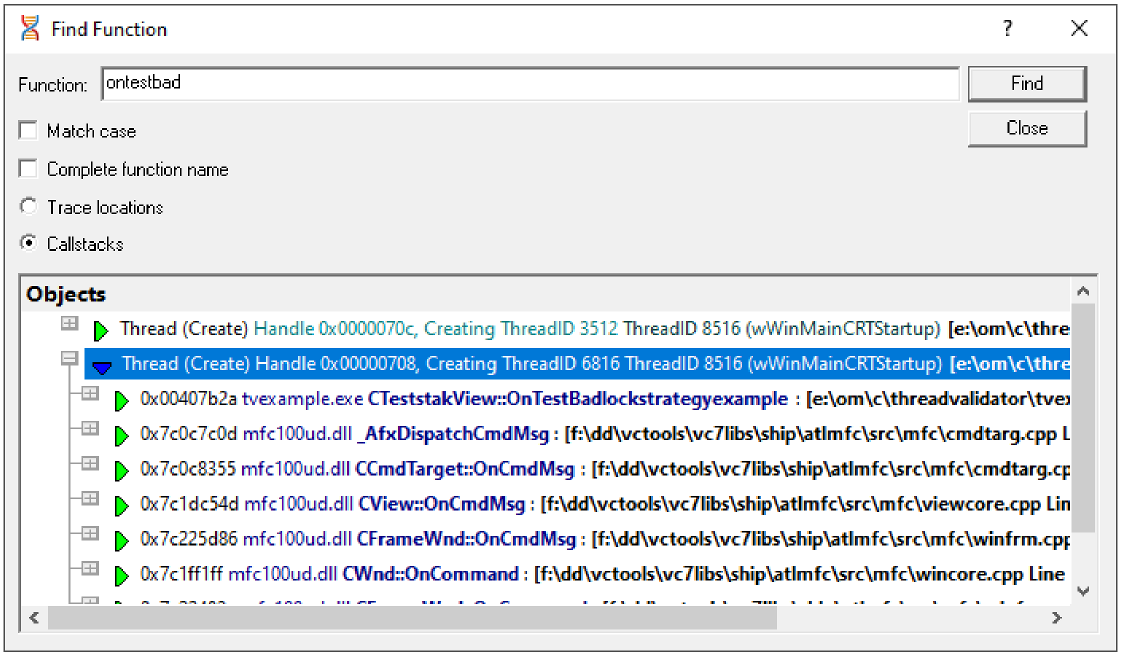
Thread Validator has an example program you can use to safely explore features.

In the example program, the **Test** menu has options for a **Bad** and **Good lock strategy example**.

After doing both of these, the example searches can be made below.

Searching only within matching functions

Searching for critical section usage only in functions that have `ontestbad` as part of their name finds these results in `CTeststakView::`**`OnTestBadlockstrategyexample`**



Find Function

Function: ontestbad

□ Match case

□ Complete function name

○ Trace locations

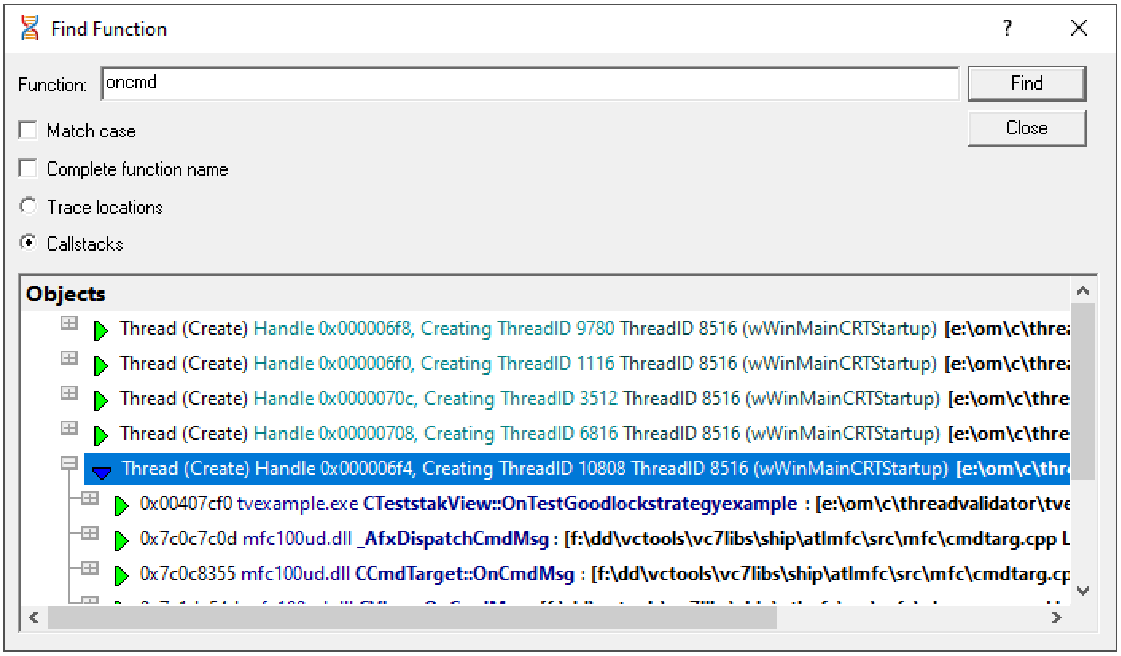
● Callstacks

Objects

```
Thread (Create) Handle 0x0000070c, Creating ThreadID 3512 ThreadID 8516 (wWinMainCRTStartup) [e:\om\c\thre
Thread (Create) Handle 0x00000708, Creating ThreadID 6816 ThreadID 8516 (wWinMainCRTStartup) [e:\om\c\thre
  0x00407b2a tvexample.exe CTeststakView::OnTestBadlockstrategyexample : [e:\om\c\threadvalidator\tvex
  0x7c0c7c0d mfc100ud.dll _AfxDispatchCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp L
  0x7c0c8355 mfc100ud.dll CCmdTarget::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cp
  0x7c1dc54d mfc100ud.dll CView::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\viewcore.cpp Lin
  0x7c225d86 mfc100ud.dll CFrameWnd::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp
  0x7c1ff1ff mfc100ud.dll CWnd::OnCommand : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\wincore.cpp Line
```

Searching anywhere in the callstack

Changing the function name to `oncmd` and now searching in *any* part of the callstack finds both the 'Good' and 'Bad' lock strategy functions called from `CCmdTarget::`**`OnCmdMsg`**



Function: oncmd

□ Match case

□ Complete function name

○ Trace locations

● Callstacks

Objects

```
⊞ ▷ Thread (Create) Handle 0x000006f8, Creating ThreadID 9780 ThreadID 8516 (wWinMainCRTStartup) [e:\om\c\threa
⊞ ▷ Thread (Create) Handle 0x000006f0, Creating ThreadID 1116 ThreadID 8516 (wWinMainCRTStartup) [e:\om\c\threa
⊞ ▷ Thread (Create) Handle 0x0000070c, Creating ThreadID 3512 ThreadID 8516 (wWinMainCRTStartup) [e:\om\c\thre
⊞ ▷ Thread (Create) Handle 0x00000708, Creating ThreadID 6816 ThreadID 8516 (wWinMainCRTStartup) [e:\om\c\thre
⊟ ▽ Thread (Create) Handle 0x000006f4, Creating ThreadID 10808 ThreadID 8516 (wWinMainCRTStartup) [e:\om\c\thr
    ⊞ ▷ 0x00407cf0 tvexample.exe CTeststakView::OnTestGoodlockstrategyexample : [e:\om\c\threadvalidator\tve
    ⊞ ▷ 0x7c0c7c0d mfc100ud.dll _AfxDispatchCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp L
    ⊞ ▷ 0x7c0c8355 mfc100ud.dll CCmdTarget::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cp
```

3.13.3 Finding critical sections only used in one thread

Searching for critical sections only used in one thread

Using the Single Thread Critical Section Detector dialog below, you can search for potentially unnecessary synchronization objects that have only been used in one thread.

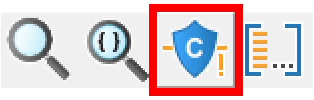
If further examination within your program confirms the critical sections are not needed, you may be able to improve performance in your program by removing them.

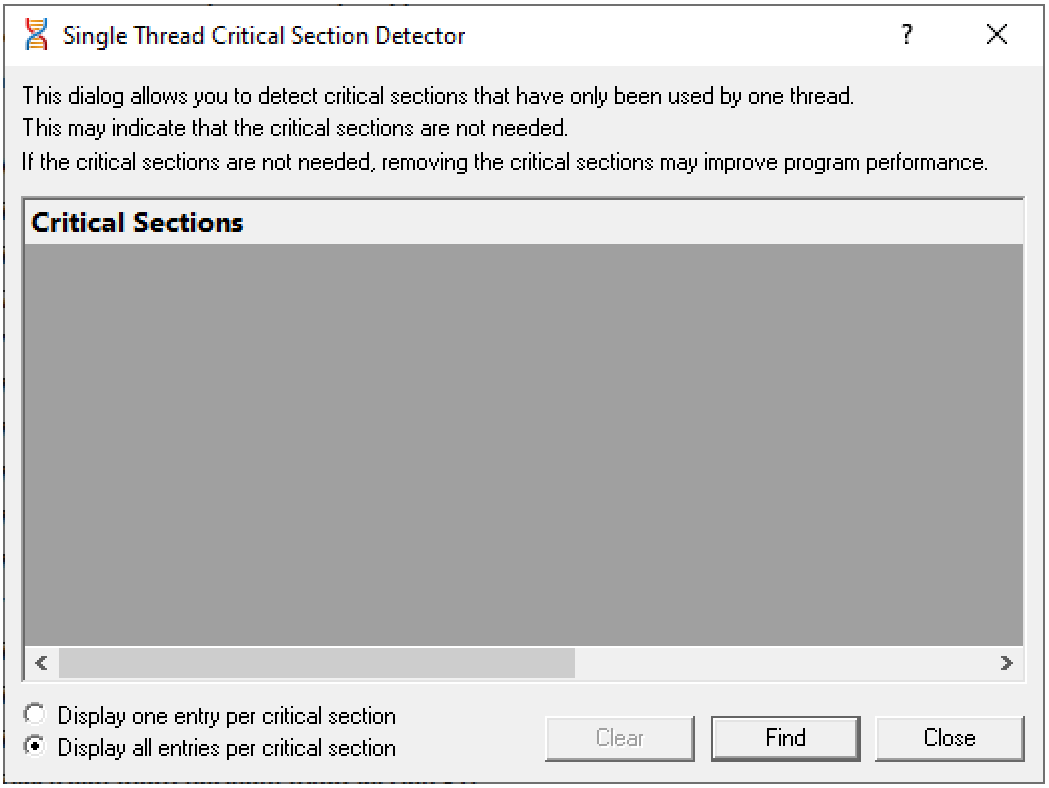
The Single Thread Critical Section Detector dialog

To show this dialog, choose the menu option below:

☰ **Query** menu › choose **Single Thread Critical Section Detector...** › displays the dialog below

Or use the following icon on the Query Toolbar.





Single Thread Critical Section Detector
?
X

This dialog allows you to detect critical sections that have only been used by one thread.
This may indicate that the critical sections are not needed.
If the critical sections are not needed, removing the critical sections may improve program performance.

Critical Sections
○ Display one entry per critical section
● Display all entries per critical section
Clear
Find
Close

Display options

- **Display one/all entry per critical section** › show just one use of a given critical section by one thread, or show all uses
- **Clear** › remove any existing results from the display

Finding results

- **Find** › performs the search, displaying results in the list

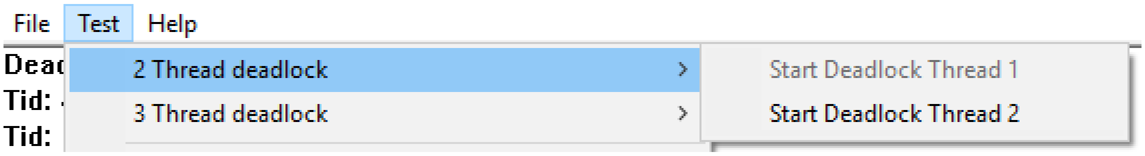
Results are *appended* to any previous search results.

You can expand the search results, and double click the data items to edit source code in your preferred editor.

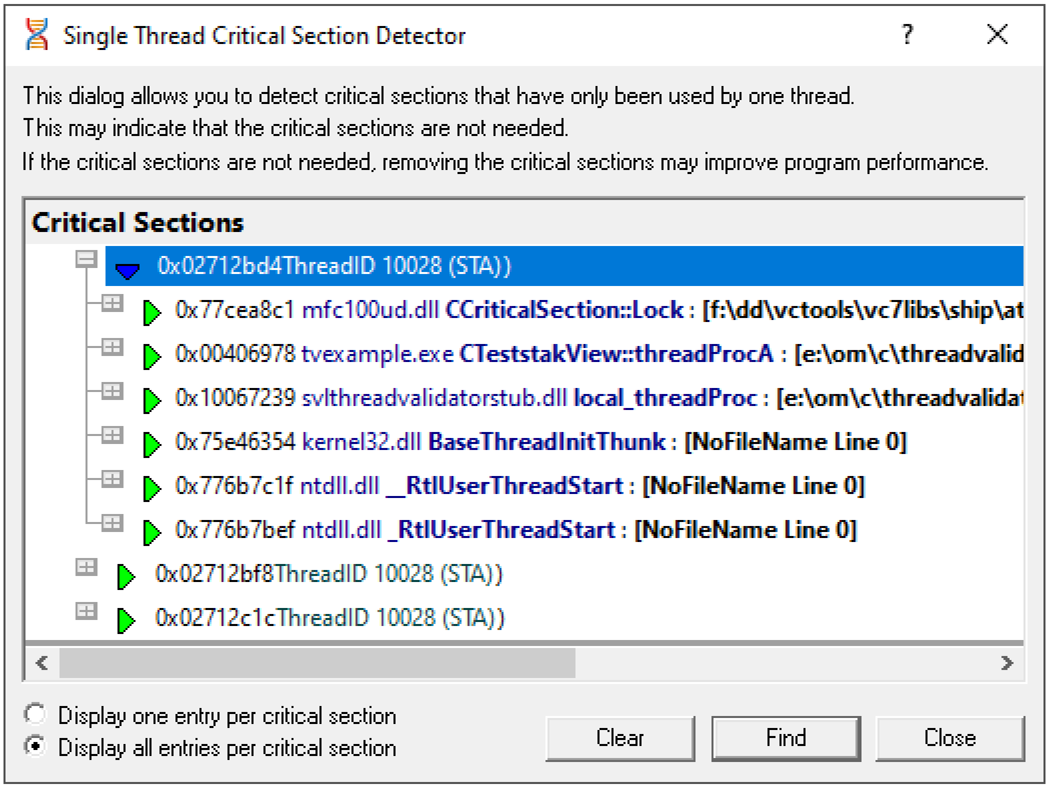
Examples of finding objects used in one thread

Thread Validator has an example program you can use to safely explore features.

In the example program, the **Test** menu has options for starting deadlock threads:



After starting deadlock Thread 1, that menu option is disabled and clicking **Find** on the Single Thread dialog shows two results:



This dialog allows you to detect critical sections that have only been used by one thread.
This may indicate that the critical sections are not needed.
If the critical sections are not needed, removing the critical sections may improve program performance.

```
0x02712bd4ThreadID 10028 (STA))
    0x77cea8c1 mfc100ud.dll CCriticalSection::Lock : [f:\dd\vctools\vc7libs\ship\at
    0x00406978 tvexample.exe CTeststakView::threadProcA : [e:\om\c\threadvalid
    0x10067239 svlthreadvalidatorstub.dll local_threadProc : [e:\om\c\threadvalidat
    0x75e46354 kernel32.dll BaseThreadInitThunk : [NoFileName Line 0]
    0x776b7c1f ntdll.dll __RtlUserThreadStart : [NoFileName Line 0]
    0x776b7bef ntdll.dll _RtlUserThreadStart : [NoFileName Line 0]
0x02712bf8ThreadID 10028 (STA))
0x02712c1cThreadID 10028 (STA))
```

○ Display one entry per critical section
● Display all entries per critical section

3.13.4 Fetching callstacks for all threads

Fetching callstacks for all threads

Using the Deadlocked Thread PostMortem dialog below, you can retrieve all thread callstacks in your application.

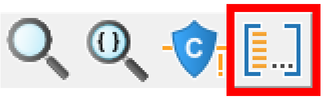
This may be useful in many ways, but in particular, if you know a deadlock has occurred in your program, then examining the callstacks of each thread will help determine where the deadlock occurred.

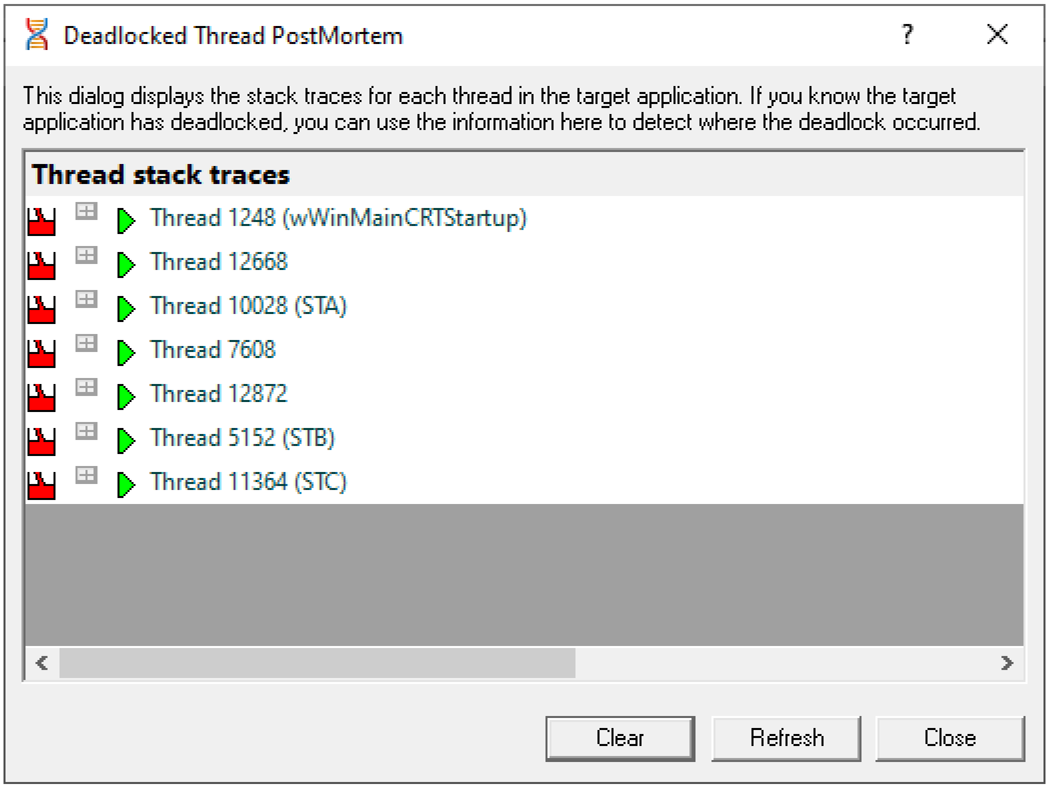
The Deadlocked Thread PostMortem dialog

To show this dialog, choose the menu option below:

☰ **Query** menu › choose **Display stack traces for all threads...** › displays the dialog below

Or use the following icon on the Query Toolbar.





This dialog displays the stack traces for each thread in the target application. If you know the target application has deadlocked, you can use the information here to detect where the deadlock occurred.

Thread stack traces

- Thread 1248 (wWinMainCRTStartup)
- Thread 12668
- Thread 10028 (STA)
- Thread 7608
- Thread 12872
- Thread 5152 (STB)
- Thread 11364 (STC)

The dialog appears already populated with the callstack information

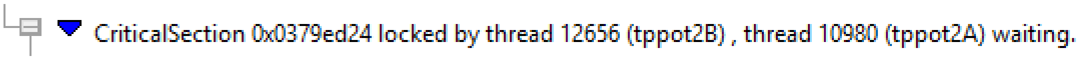
- **Clear** › remove any existing results
- **Refresh** › update the list of callstacks shown in the display

Viewing the callstacks

The image below shows the results of a search that has been expanded to show a callstack.

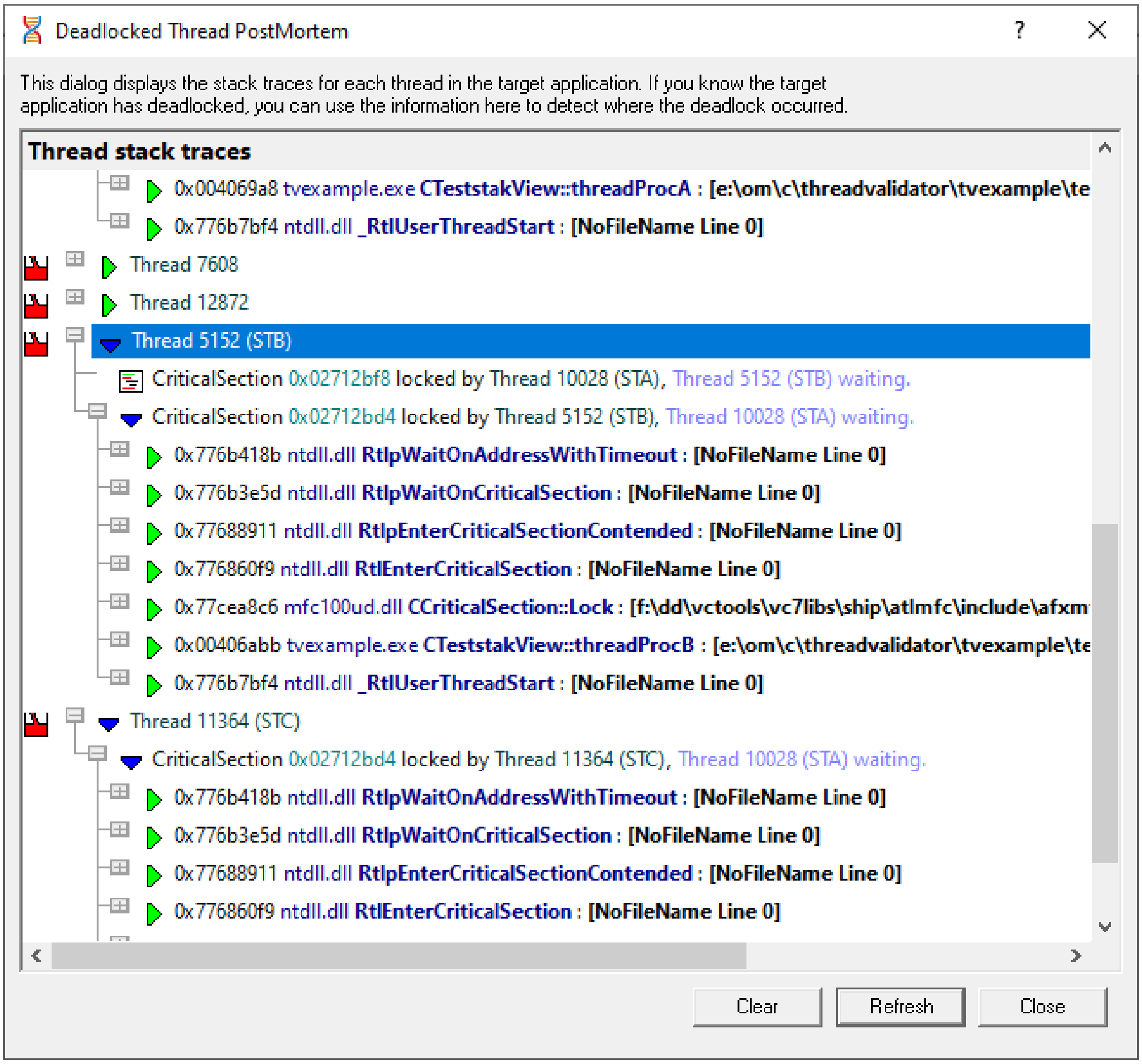
Note that above each callstack, the thread id (and name if the thread is named) is displayed followed by a list of critical sections which this thread has locked or is waiting upon.

For example:



CriticalSection 0x0379ed24 locked by thread 12656 (tppot2B) , thread 10980 (tppot2A) waiting.

If the thread is *not* waiting upon any critical sections and is not involved in a wait, a message **No critical sections or waits for this thread** is displayed.



Deadlocked Thread PostMortem

This dialog displays the stack traces for each thread in the target application. If you know the target application has deadlocked, you can use the information here to detect where the deadlock occurred.

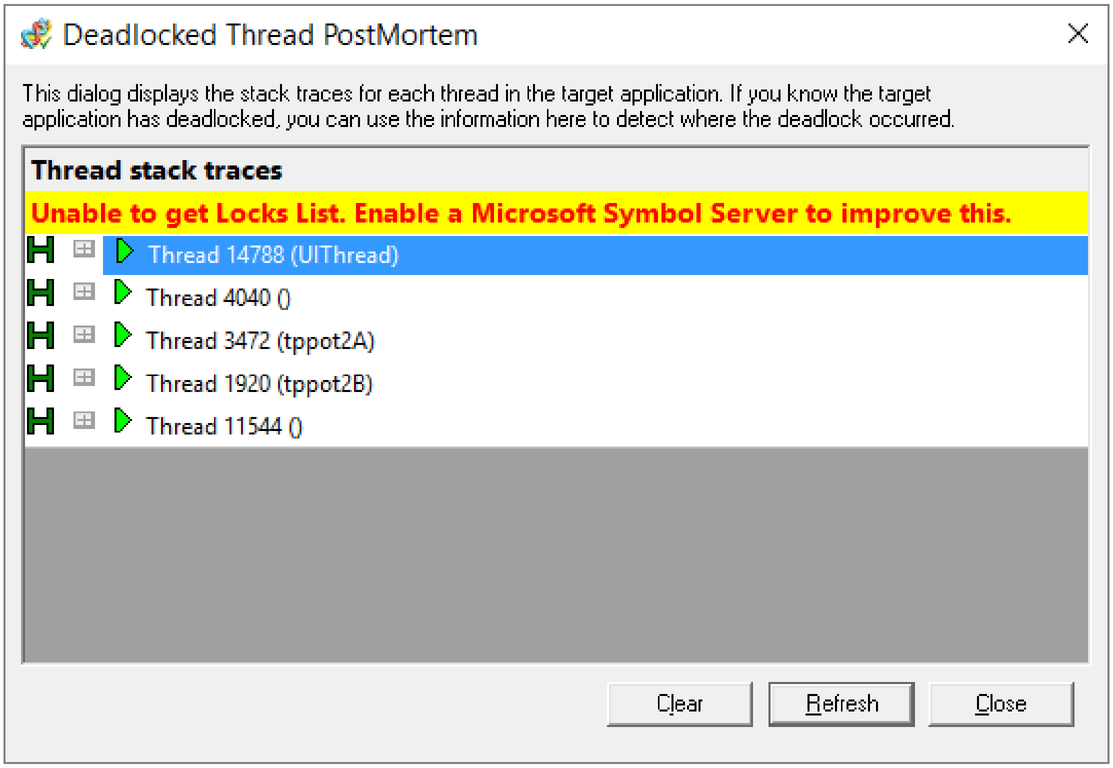
Thread stack traces

0x004069a8 tvexample.exe **CTeststakView::threadProcA : [e:\om\c\threadvalidator\tvexample\te**
0x776b7bf4 ntdll.dll **_RtlUserThreadStart : [NoFileName Line 0]**
Thread 7608
Thread 12872
Thread 5152 (STB)
CriticalSection 0x02712bf8 locked by Thread 10028 (STA), Thread 5152 (STB) waiting.
CriticalSection 0x02712bd4 locked by Thread 5152 (STB), Thread 10028 (STA) waiting.
0x776b418b ntdll.dll **RtlpWaitOnAddressWithTimeout : [NoFileName Line 0]**
0x776b3e5d ntdll.dll **RtlpWaitOnCriticalSection : [NoFileName Line 0]**
0x77688911 ntdll.dll **RtlpEnterCriticalSectionContended : [NoFileName Line 0]**
0x776860f9 ntdll.dll **RtlEnterCriticalSection : [NoFileName Line 0]**
0x77cea8c6 mfc100ud.dll **CCriticalSection::Lock : [f:\dd\vctools\vc7libs\ship\atlmfc\include\afxm**
0x00406abb tvexample.exe **CTeststakView::threadProcB : [e:\om\c\threadvalidator\tvexample\te**
0x776b7bf4 ntdll.dll **_RtlUserThreadStart : [NoFileName Line 0]**
Thread 11364 (STC)
CriticalSection 0x02712bd4 locked by Thread 11364 (STC), Thread 10028 (STA) waiting.
0x776b418b ntdll.dll **RtlpWaitOnAddressWithTimeout : [NoFileName Line 0]**
0x776b3e5d ntdll.dll **RtlpWaitOnCriticalSection : [NoFileName Line 0]**
0x77688911 ntdll.dll **RtlpEnterCriticalSectionContended : [NoFileName Line 0]**
0x776860f9 ntdll.dll **RtlEnterCriticalSection : [NoFileName Line 0]**

'Unable to get locks list' warning

If Thread Validator fails to find the process locks list it will provide a warning and information for what to do to enable the list to be found.

In this example, the suggestion is to enable a Microsoft Symbol Server.



Deadlocked Thread PostMortem

This dialog displays the stack traces for each thread in the target application. If you know the target application has deadlocked, you can use the information here to detect where the deadlock occurred.

Thread stack traces

Unable to get Locks List. Enable a Microsoft Symbol Server to improve this.

Thread 14788 (UIThread)

Thread 4040 ()

Thread 3472 (tppot2A)

Thread 1920 (tppot2B)

Thread 11544 ()

3.13.5 Deadlock detection

The deadlock and potential deadlock settings allow Thread Validator to regularly scan for various error conditions using the same data collected by the options on the Collect settings tab.

If your application has many threads, you may prefer to disable these options to prevent Thread Validator from using too much processor time.

However, at run time you can force Thread Validator to check for deadlocks or potential deadlocks in the application under test.

Forcing a deadlock check

If deadlock detection is off by default, you can manually trigger a deadlock check:

Query menu › choose **Deadlock detection** › runs a deadlock check updating displays accordingly

This menu option will be disabled if the **Deadlocks** option on the Detect settings tab is enabled.

Example

The example application can be used to demonstrate the deadlock checking:

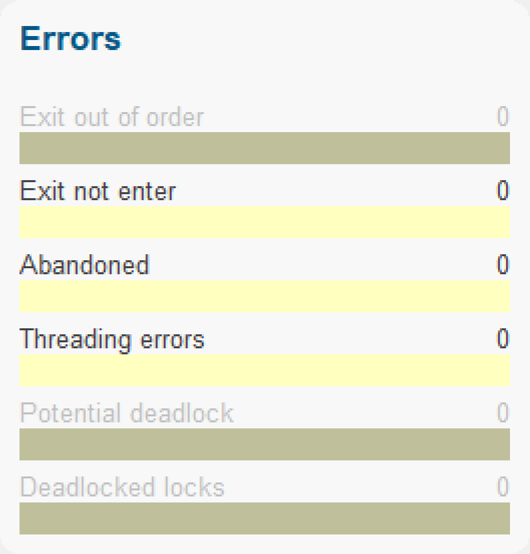
1. Turn *off* the **Deadlocks** option on the Detect settings tab and

2. Launch the example application.

3. Use some of the built in tests to create a deadlock, for example the 3 thread deadlock.

4. Notice that Thread Validator does not register the deadlocks including for example, in the following three displays.

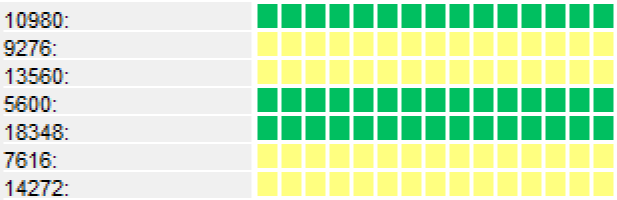
In the Errors panel of the Summary tab:



Errors

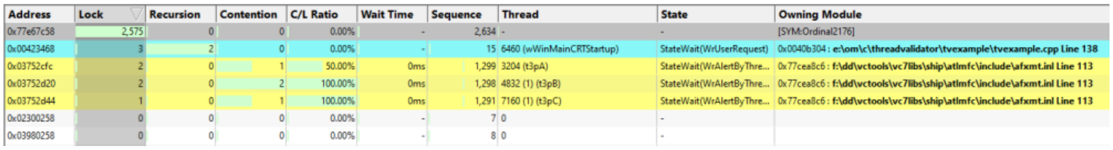
Exit out of order	0
Exit not enter	0
Abandoned	0
Threading errors	0
Potential deadlock	0
Deadlocked locks	0

In the snapshots view of the Threads tab



10980:	🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩
9276:	🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨
13560:	🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨
5600:	🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩
18348:	🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩
7616:	🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨
14272:	🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨🟨

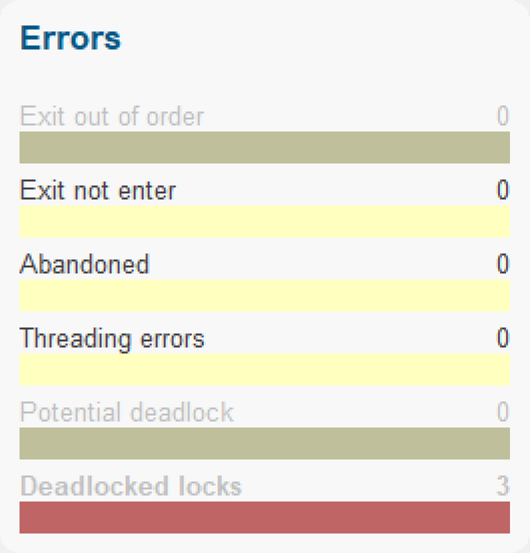
Or in the All Locks tab:



Address	Lock	Recursion	Contention	C/L Ratio	Wait Time	Sequence	Thread	State	Owning Module
0x77e67c58	2,575	0	0	0.00%	-	2,634	-	-	[SYM:Ordinal2176]
0x00423468	3	2	0	0.00%	-	15	6460 (wWinMainCRTStartup)	StateWait(WrUserRequest)	0x0040b304 : **e:\om\c\threadvalidator\tvexample\tvexample.cpp Line 138**
0x03752cfc	2	0	1	50.00%	0ms	1,299	3204 (t3pA)	StateWait(WrAlertByThre...	0x77cea8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x03752d20	2	0	2	100.00%	0ms	1,298	4832 (1) (t3pB)	StateWait(WrAlertByThre...	0x77cea8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x03752d44	1	0	1	100.00%	0ms	1,291	7160 (1) (t3pC)	StateWait(WrAlertByThre...	0x77cea8c6 : **f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 113**
0x02300258	0	0	0	0.00%	-	7	0	-	
0x03980258	0	0	0	0.00%	-	8	0	-	

5. Force a deadlock check as described above, and each of these displays will update to show the deadlocks found

In the Errors panel of the Summary tab, where the red bar indicates 3 deadlocked locks:



Errors

Exit out of order 0

Exit not enter 0

Abandoned 0

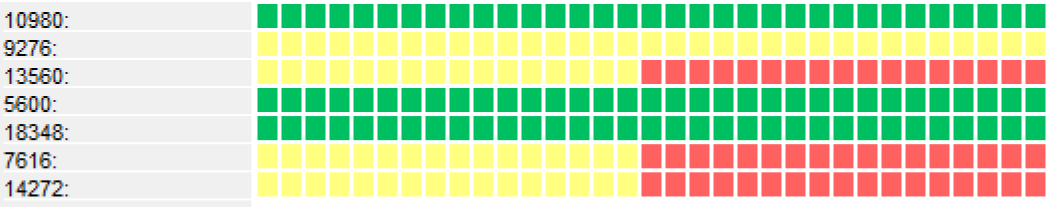
Threading errors 0

Potential deadlock 0

Deadlocked locks 3

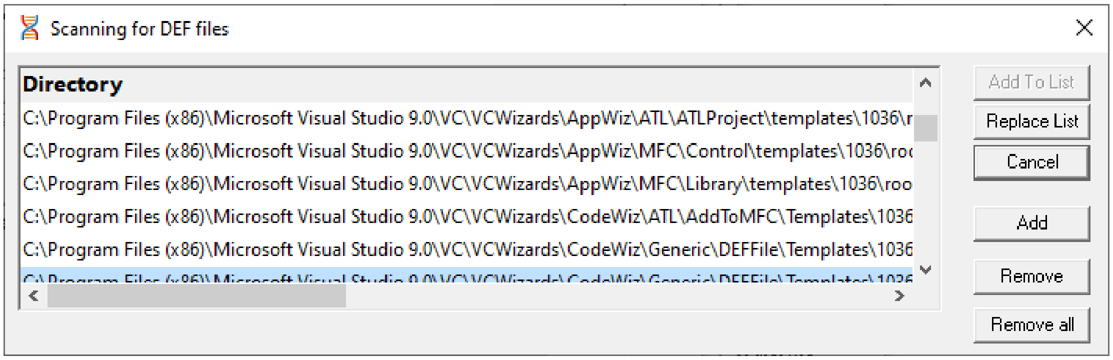
Note these options are disabled, reflecting the disabled general deadlock detection setting which prevents interactive exploration of the deadlock data. Enable the setting to explore further.

In the snapshots view of the Threads tab, where the thread snapshots now starts to register the threads as locked (red)



10980:
9276:
13560:
5600:
18348:
7616:
14272:

Or in the All Locks tab, where the locked threads are also now highlighted in red:



Scanning for DEF files

Directory
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\AppWiz\ATL\ATLProject\templates\1036\r
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\AppWiz\MFC\Control\templates\1036\roo
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\AppWiz\MFC\Library\templates\1036\roo
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\CodeWiz\ATL\AddToMFC\Templates\1036
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCWizards\CodeWiz\Generic\DEFFile\Templates\1036

3.13.6 Potential deadlock detection

The deadlock and potential deadlock settings allow Thread Validator to regularly scan for various error conditions using the same data collected by the options on the Collect tab.

If your application has many threads, you may prefer to disable these options to prevent Thread Validator from using too much processor time.

However, at run time you can force Thread Validator to check for deadlocks or potential deadlocks in the application under test.

Forcing a potential deadlock check

If potential deadlock detection is on but the detection interval is set to a long time or to *Never*, you may wish to manually trigger a potential deadlock check:

☰ **Query** menu › choose **Potential deadlock detection** › runs a potential deadlock check, updating displays accordingly

> This menu option will only be enabled if the **Potential Deadlocks** option on the Detect settings tab is enabled.

Example

The example application can be used to demonstrate the deadlock checking:

1. Turn *on* the **Potential Deadlocks** option on the Detect settings tab

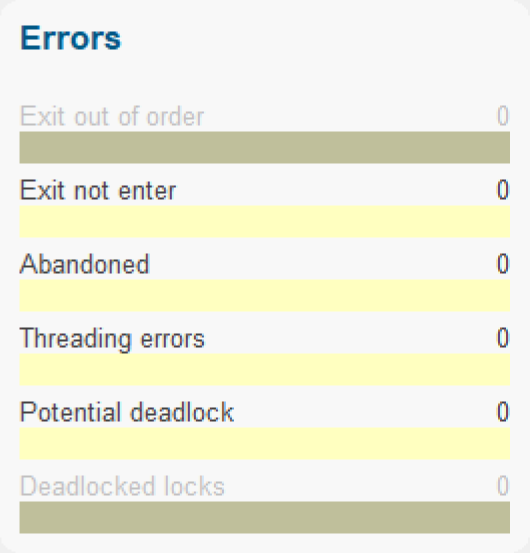
2. Set the Deadlock detect interval to *Never* (on the same setting page)

3. Launch the example application.

4. Use some of the built in tests to create a potential deadlock, for example the Potential deadlocks 2 thread.

5. Notice that Thread Validator does not register the potential deadlocks including for example, in the following three displays.

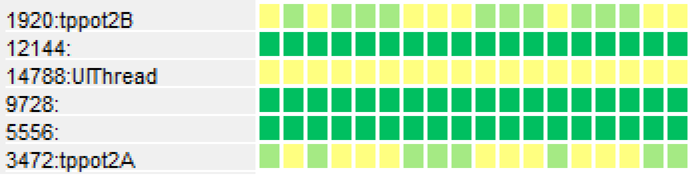
In the Errors panel of the Summary tab:



Errors

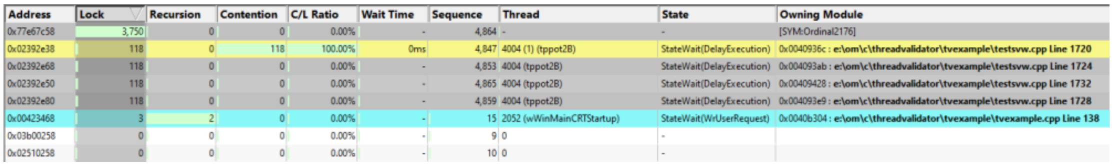
Exit out of order	0
Exit not enter	0
Abandoned	0
Threading errors	0
Potential deadlock	0
Deadlocked locks	0

In the snapshots view of the Threads tab



1920:tppot2B
12144:
14788:UIThread
9728:
5556:
3472:tppot2A

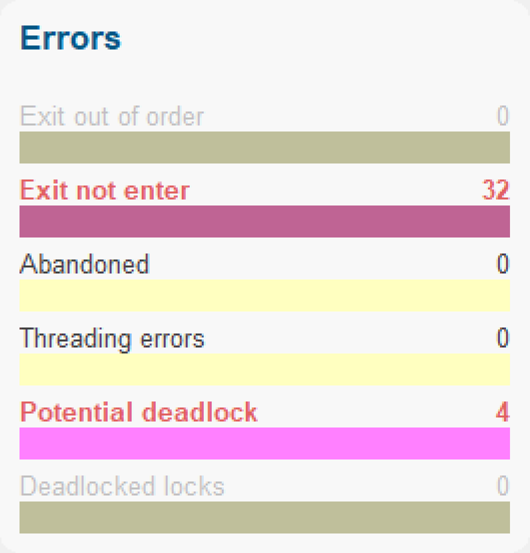
Or in the All Locks tab:



Address	Lock	Recursion	Contention	C/L Ratio	Wait Time	Sequence	Thread	State	Owning Module
0x77e67c58	3,750	0	0	0.00%	-	4,864	-	-	[SYM:Ordinal2176]
0x02392e38	118	0	118	100.00%	0ms	4,847	4004 (1) (tppot2B)	StateWait(DelayExecution)	0x0040936c : e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1720
0x02392e68	118	0	0	0.00%	-	4,853	4004 (tppot2B)	StateWait(DelayExecution)	0x004093ab : e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1724
0x02392e50	118	0	0	0.00%	-	4,865	4004 (tppot2B)	StateWait(DelayExecution)	0x00409428 : e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1732
0x02392e80	118	0	0	0.00%	-	4,859	4004 (tppot2B)	StateWait(DelayExecution)	0x004093e9 : e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1728
0x00423468	3	2	0	0.00%	-	15	2052 (wWinMainCRTStartup)	StateWait(WrUserRequest)	0x0040b304 : e:\om\c\threadvalidator\tvexample\tvexample.cpp Line 138
0x03b00258	0	0	0	0.00%	-	9	0	-	
0x02510258	0	0	0	0.00%	-	10	0	-	

6. Force a potential deadlock check as described above, and each of these displays will update to show the potential deadlocks found

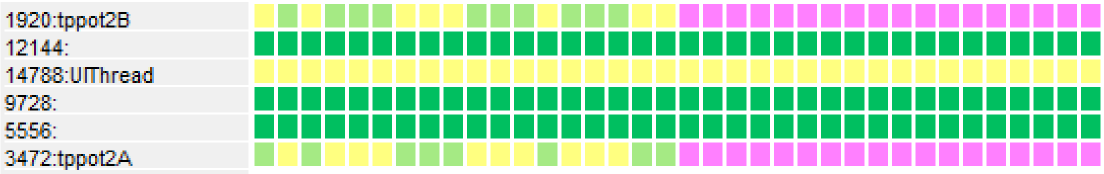
In the Errors panel of the Summary tab, where the pink 🟪 bar indicates 2 potential deadlocks:



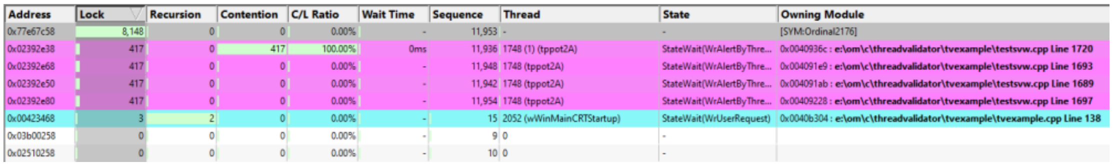
Errors

Exit out of order	0
Exit not enter	32
Abandoned	0
Threading errors	0
Potential deadlock	4
Deadlocked locks	0

In the snapshots view of the Threads tab, where the thread snapshots now starts to register the threads as having potential deadlocks (pink 🟪)



Or in the All Locks tab, where the locked threads are also now highlighted in pink:



Address	Lock	Recursion	Contention	C/L Ratio	Wait Time	Sequence	Thread	State	Owning Module
0x77e67c58	8,148	0	0	0.00%	-	11,953	-	-	[SYM:Ordinal2176]
0x02392e38	417	0	417	100.00%	0ms	11,936	1748 (1) (tppot2A)	StateWait(WrAlertByThre...	0x0040936c : **e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1720**
0x02392e68	417	0	0	0.00%	-	11,948	1748 (tppot2A)	StateWait(WrAlertByThre...	0x004091e9 : **e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1693**
0x02392e50	417	0	0	0.00%	-	11,942	1748 (tppot2A)	StateWait(WrAlertByThre...	0x004091ab : **e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1689**
0x02392e80	417	0	0	0.00%	-	11,954	1748 (tppot2A)	StateWait(WrAlertByThre...	0x00409228 : **e:\om\c\threadvalidator\tvexample\testsvw.cpp Line 1697**
0x00423468	3	2	0	0.00%	-	15	2052 (wWinMainCRTStartup)	StateWait(WrUserRequest)	0x0040b304 : **e:\om\c\threadvalidator\tvexample\tvexample.cpp Line 138**
0x03b00258	0	0	0	0.00%	-	9	0	-	
0x02510258	0	0	0	0.00%	-	10	0	-	

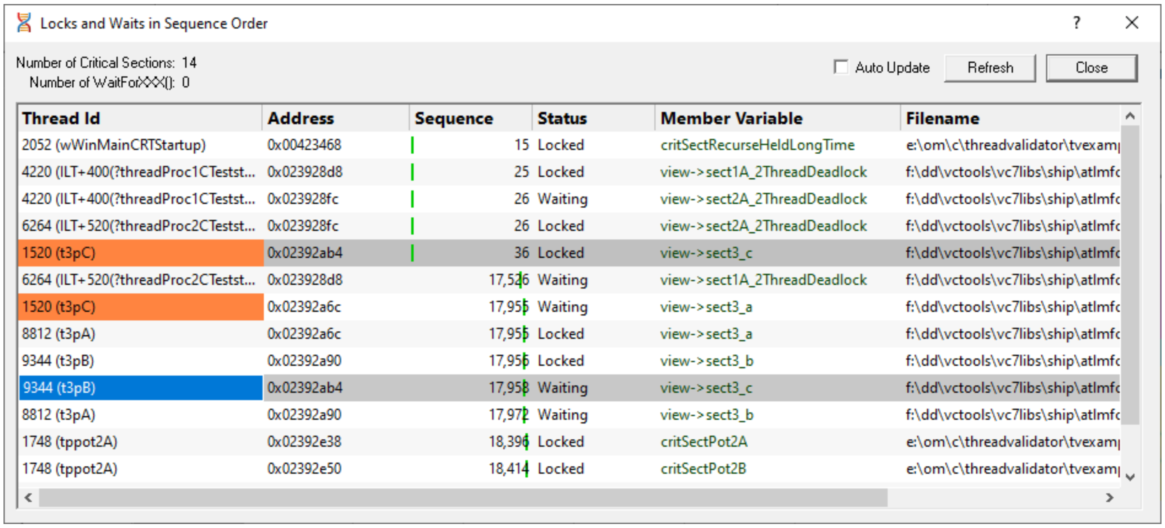
3.13.7 Display lock order

Lock order dialog

Thread Validator can display the order in which locks and waits were entered.

The lock order dialog (i.e. Locks and Waits in Sequence Order dialog) can be started in two different ways:

- **All Locks tab** > **popup** menu > **Lock Acquisition Order...**
- **Query** menu > **Display lock order**



Number of Critical Sections: 14
Number of WaitForXXX(): 0

Thread Id	Address	Sequence	Status	Member Variable	Filename
2052 (wWinMainCRTStartup)	0x00423468	15	Locked	critSectRecurseHeldLongTime	e:\om\c\threadvalidator\tvexam
4220 (ILT+400(?threadProc1CTestst...	0x023928d8	25	Locked	view->sect1A_2ThreadDeadlock	f:\dd\vctools\vc7libs\ship\atlmfc
4220 (ILT+400(?threadProc1CTestst...	0x023928fc	26	Waiting	view->sect2A_2ThreadDeadlock	f:\dd\vctools\vc7libs\ship\atlmfc
6264 (ILT+520(?threadProc2CTestst...	0x023928fc	26	Locked	view->sect2A_2ThreadDeadlock	f:\dd\vctools\vc7libs\ship\atlmfc
1520 (t3pC)	0x02392ab4	36	Locked	view->sect3_c	f:\dd\vctools\vc7libs\ship\atlmfc
6264 (ILT+520(?threadProc2CTestst...	0x023928d8	17,526	Waiting	view->sect1A_2ThreadDeadlock	f:\dd\vctools\vc7libs\ship\atlmfc
1520 (t3pC)	0x02392a6c	17,955	Waiting	view->sect3_a	f:\dd\vctools\vc7libs\ship\atlmfc
8812 (t3pA)	0x02392a6c	17,955	Locked	view->sect3_a	f:\dd\vctools\vc7libs\ship\atlmfc
9344 (t3pB)	0x02392a90	17,956	Locked	view->sect3_b	f:\dd\vctools\vc7libs\ship\atlmfc
9344 (t3pB)	0x02392ab4	17,958	Waiting	view->sect3_c	f:\dd\vctools\vc7libs\ship\atlmfc
8812 (t3pA)	0x02392a90	17,972	Waiting	view->sect3_b	f:\dd\vctools\vc7libs\ship\atlmfc
1748 (tppot2A)	0x02392e38	18,396	Locked	critSectPot2A	e:\om\c\threadvalidator\tvexam
1748 (tppot2A)	0x02392e50	18,414	Locked	critSectPot2B	e:\om\c\threadvalidator\tvexam

The dialog displays the order...

- in which critical sections are locked and waited upon
- that waits are entered into
- in which threads sleep and are suspended

For this, you need to enable the **Functions working with waitable handles** option on the Hook Insertion settings page.

You don't need to enable the individual **Sleep** option.

Data displayed in the table

- **Thread Id** › displays the thread id and a thread name if one is available
- **Address** › the critical section address that is locked or being waited upon

For `WaitForSingleObject` calls, this is the *handle* that is being waited upon.

For `WaitForMultipleObjects` calls, the *number* of handles being waited upon is shown.

- **Sequence** › displays the sequence id of the lock or wait

Green bars in this column indicate the sequence id relative to the known range of all ids.

- **Status** › displays the word Locked or Waiting for critical sections

For `WaitForSingleObject` or `WaitForMultipleObjects` calls the appropriate function is displayed instead.

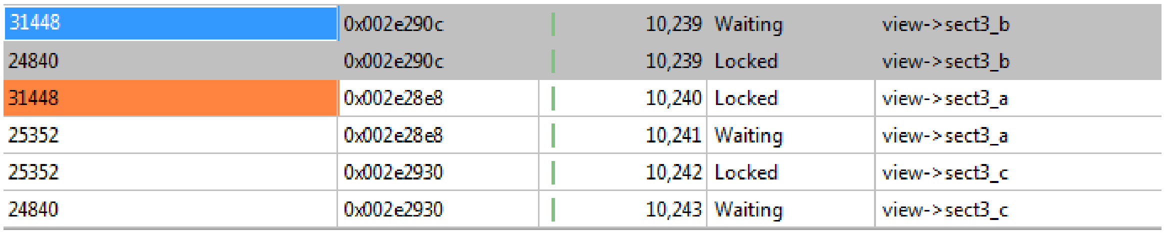
- **Member variable** › shows information about the storage variable used when entering the critical section

- **Filename** › displays the filename of the function where the critical section was entered

Understanding the data

The image below shows that three threads have deadlocked.

Each thread has acquired one lock and is waiting upon another, so there's two entries in this view for each thread.



31448	0x002e290c	10,239	Waiting	view->sect3_b
24840	0x002e290c	10,239	Locked	view->sect3_b
31448	0x002e28e8	10,240	Locked	view->sect3_a
25352	0x002e28e8	10,241	Waiting	view->sect3_a
25352	0x002e2930	10,242	Locked	view->sect3_c
24840	0x002e2930	10,243	Waiting	view->sect3_c

The order the threads acquired and is waiting upon locks is the same as the order of the data in the list, which is in turn the same as the sequence ids in the Sequence column.

This situation was caused by the bad lock strategy used in the example program.

In this example three threads are created - lets call them A, B and C - and each *repeatedly* attempts to lock two of the critical sections a,b and c as follows:

- Thread **A** locks sections **a** and **b** and then unlocks b and a

- Thread **B** locks sections **b** and **c** and then unlocks c and b

- Thread **C** locks sections **c** and **a** and then unlocks a and c

It should be easy to see that a three lock circular deadlock will occur quickly occur as each thread waits for a lock that another thread has, and indeed this is what the image above shows us has happened.

Sleeping threads

When a sleeping thread is listed:

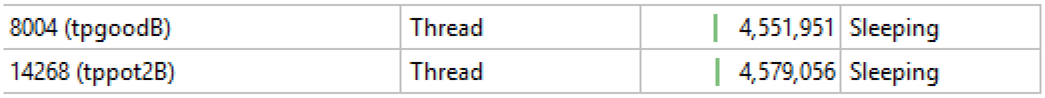
- the **Address** column displays *Thread*

If the Hook Insertion settings has **Functions working with waitable handles** enabled:

- the **Sequence** column indicates the sequence id for the start of the thread sleeping

  The horizontal position of the green bar in this column changes to show relative size of the sequence id compared to the most recently allocated id.

- the **Status** column displays *Sleeping*.



8004 (tpgoodB)	Thread	4,551,951	Sleeping
14268 (tppot2B)	Thread	4,579,056	Sleeping

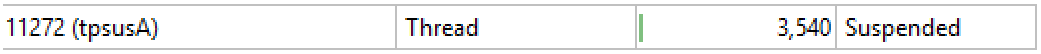
Suspended threads

When a suspended thread is listed:

- the **Address** column displays *Thread*

If the Hook Insertion settings has **Functions working with waitable handles** enabled:

- the **Sequence** column indicates the sequence id for the start of the thread suspension
- the **Status** column displays *Suspended*

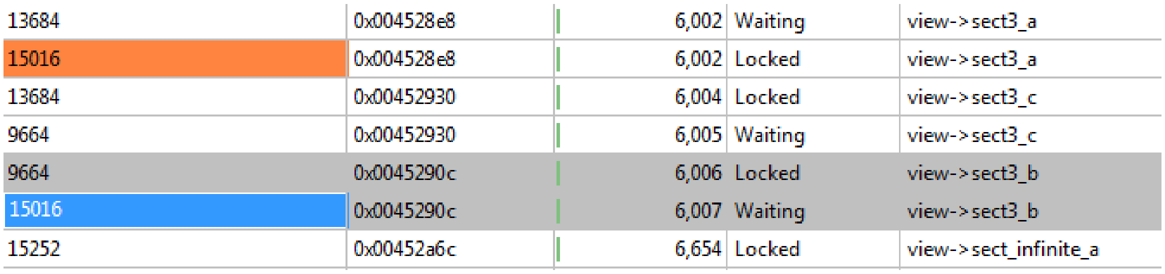


11272 (tpsusA)	Thread	3,540	Suspended

Related threads

Selecting a critical section or wait handle in the list highlights (in grey) any *other* threads using the same critical section or handle (same address).

Any other occurrences of the *same* thread waiting on a lock or holding a lock will also be highlighted with the thread information colour (orange 🟧 by default).



13684	0x004528e8	6,002	Waiting	view->sect3_a
15016	0x004528e8	6,002	Locked	view->sect3_a
13684	0x00452930	6,004	Locked	view->sect3_c
9664	0x00452930	6,005	Waiting	view->sect3_c
9664	0x0045290c	6,006	Locked	view->sect3_b
15016	0x0045290c	6,007	Waiting	view->sect3_b
15252	0x00452a6c	6,654	Locked	view->sect_infinite_a

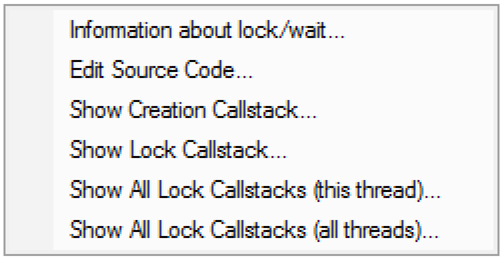
Updating the display

- **Auto Update** › refresh the contents of the list once per second
- **Refresh** › refresh the list manually when you need to

The green bars in the **Sequence** column are automatically updated periodically.

Menu Options

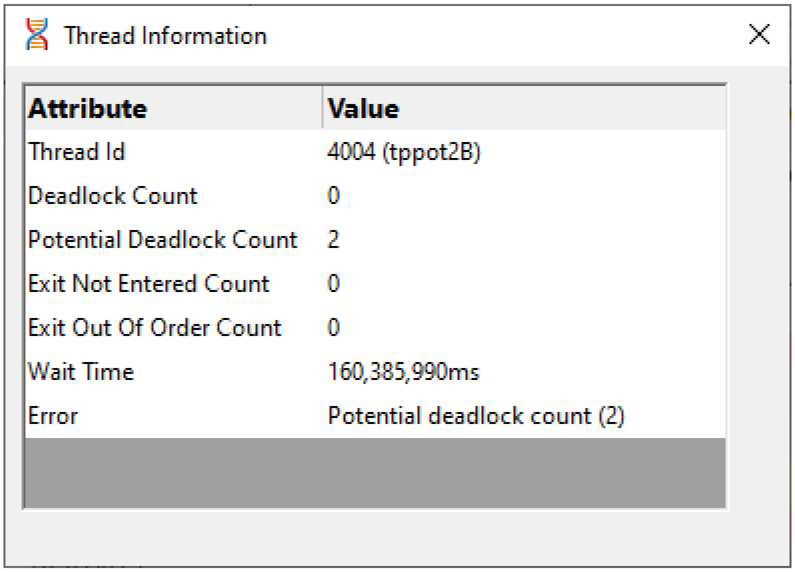
A popup menu is available by right clicking on any item in the list:


Information about lock/wait...
Edit Source Code...
Show Creation Callstack...
Show Lock Callstack...
Show All Lock Callstacks (this thread)...
Show All Lock Callstacks (all threads)...

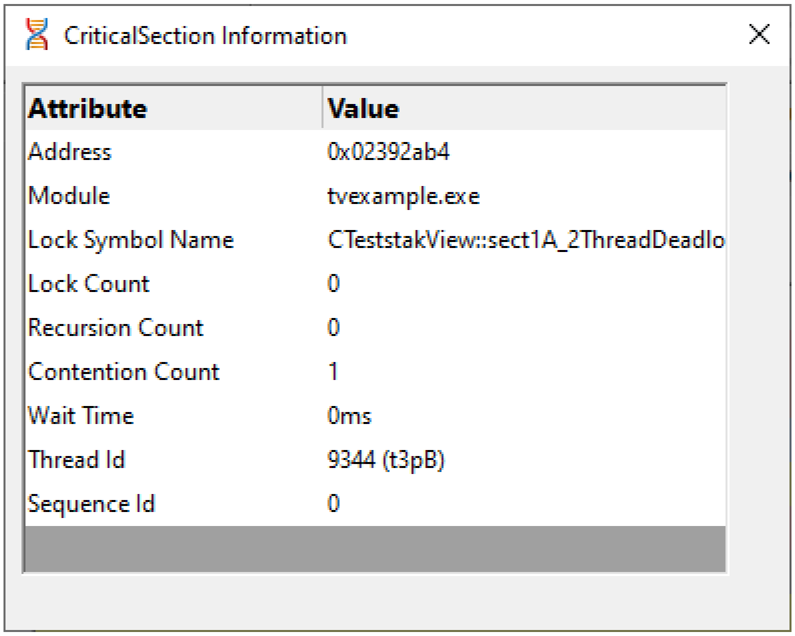
Menu option: Information about lock or wait

- **Information about lock/wait...** › shows the relevant information dialog from those below, depending on the type of item selected

  These information dialogs do not block the application so you can show as many as you need, either from this tab or others, and leave them open to compare or investigate later.

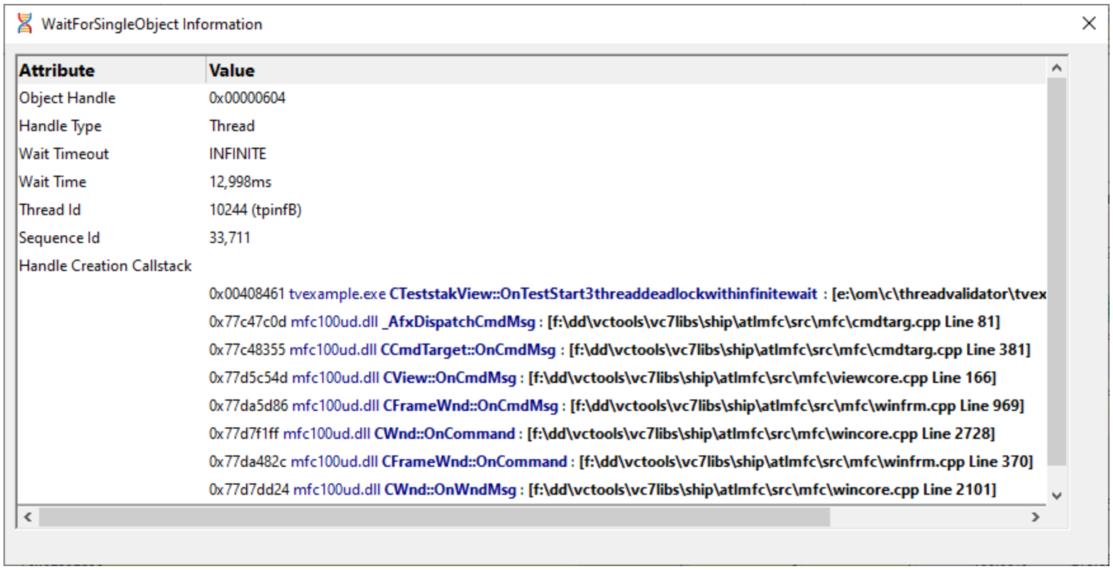


Attribute	Value
Thread Id	4004 (tppot2B)
Deadlock Count	0
Potential Deadlock Count	2
Exit Not Entered Count	0
Exit Out Of Order Count	0
Wait Time	160,385,990ms
Error	Potential deadlock count (2)



CriticalSection Information

Attribute	Value
Address	0x02392ab4
Module	tvexample.exe
Lock Symbol Name	CTeststakView::sect1A_2ThreadDeadlo
Lock Count	0
Recursion Count	0
Contention Count	1
Wait Time	0ms
Thread Id	9344 (t3pB)
Sequence Id	0



WaitForSingleObject Information

Attribute	Value
Object Handle	0x00000604
Handle Type	Thread
Wait Timeout	INFINITE
Wait Time	12,998ms
Thread Id	10244 (tpinfB)
Sequence Id	33,711
Handle Creation Callstack	
	0x00408461 tvexample.exe CTeststakView::OnTestStart3threaddeadlockwithinfinitewait : [e:\om\c\threadvalidator\tvex
	0x77c47c0d mfc100ud.dll _AfxDispatchCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 81]
	0x77c48355 mfc100ud.dll CCmdTarget::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\cmdtarg.cpp Line 381]
	0x77d5c54d mfc100ud.dll CView::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\viewcore.cpp Line 166]
	0x77da5d86 mfc100ud.dll CFrameWnd::OnCmdMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 969]
	0x77d7f1ff mfc100ud.dll CWnd::OnCommand : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\wincore.cpp Line 2728]
	0x77da482c mfc100ud.dll CFrameWnd::OnCommand : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 370]
	0x77d7dd24 mfc100ud.dll CWnd::OnWndMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\wincore.cpp Line 2101]

Menu option: editing source code

- **Edit Source Code...** › opens the default or preferred editor to edit the source code

Menu options: show ... callstacks

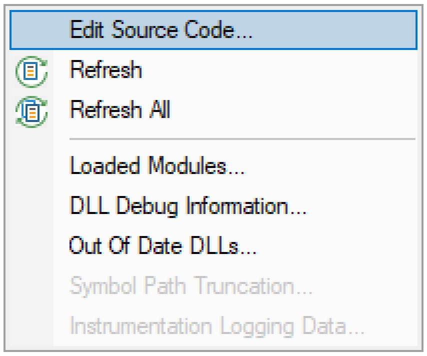
- **Show Creation Callstack...** › shows the callstack for the creation of this item, i.e. the locked or waiting critical section and thread

- **Show Lock Callstack...** › shows the callstack for the locked or waiting critical section and thread

- **Show All Lock Callstacks (this thread)...** › shows all callstacks for this critical section and *only* this thread
- **Show All Lock Callstacks (all threads)...** › as above but for *any* thread

3.14 Tools

Click on an item in the picture of the Tools Menu below to jump to the relevant topic:



3.14.1 Colour coded source code editor

Source code editing

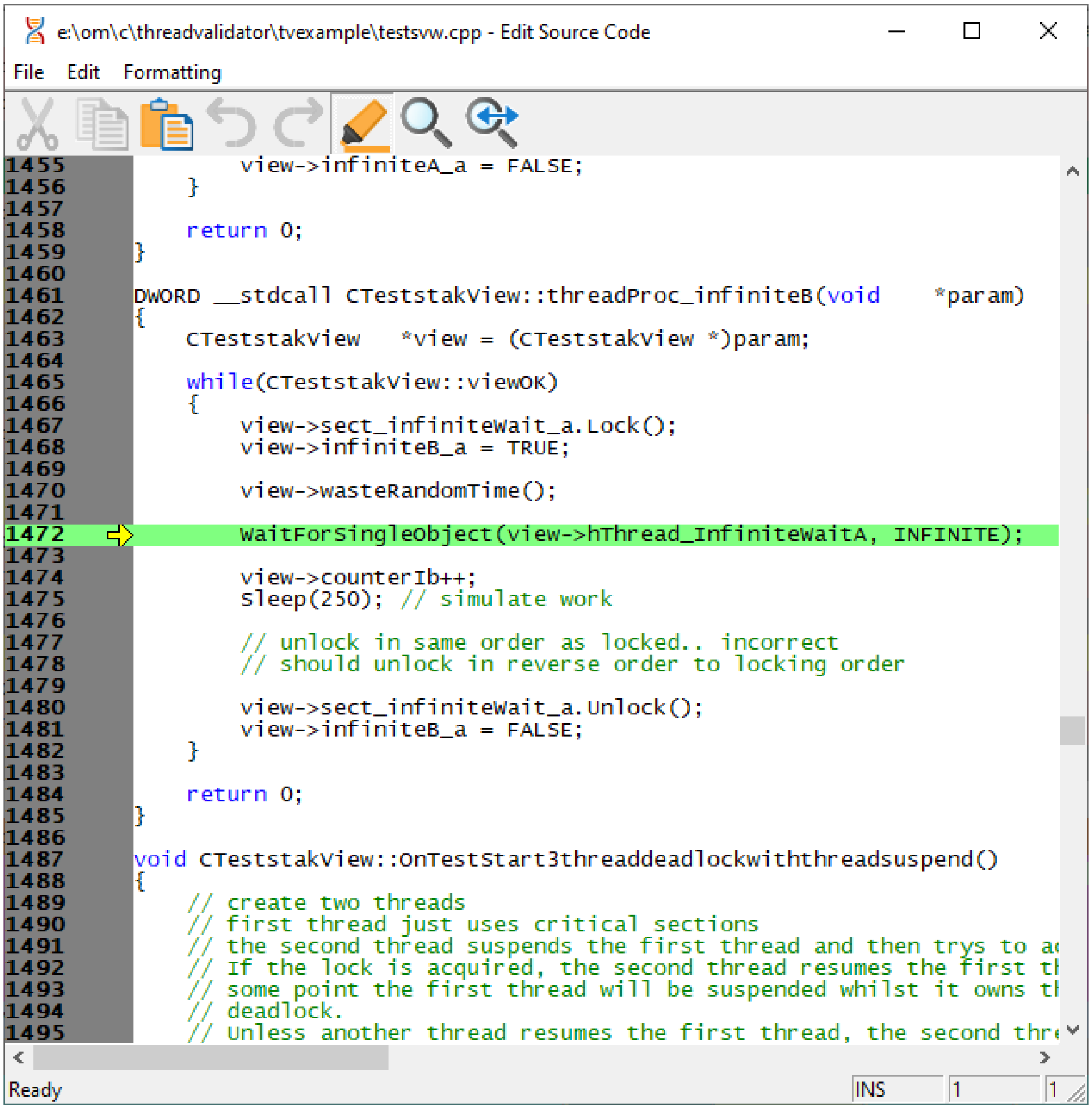
The editing settings let you set an editor of your choice to view or edit source code. Thread Validator's built-in editor is one of those options.

The built-in editor can be started in several ways:

- double click on a source code fragment in one of the views
- **popup** menu › **Edit Source Code...**
- **Tools** menu › **Edit Source Code...**

Using the built-in editor

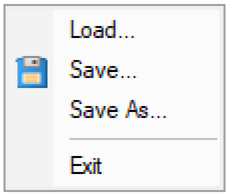
The built-in editor supports the basic operations expected for editing source code:


e:\om\c\threadvalidator\tvexample\testsvw.cpp - Edit Source Code
File
Edit
Formatting
Ready
INS
1
1

```
        view->infiniteA_a = FALSE;
    }

    return 0;
}

DWORD __stdcall CTeststakView::threadProc_infiniteB(void    *param)
{
    CTeststakView   *view = (CTeststakView *)param;

    while(CTeststakView::viewOK)
    {
        view->sect_infiniteWait_a.Lock();
        view->infiniteB_a = TRUE;

        view->wasteRandomTime();

        WaitForSingleObject(view->hThread_InfiniteWaitA, INFINITE);

        view->counterIb++;
        Sleep(250); // simulate work

        // unlock in same order as locked.. incorrect
        // should unlock in reverse order to locking order

        view->sect_infiniteWait_a.Unlock();
        view->infiniteB_a = FALSE;
    }

    return 0;
}

void CTeststakView::OnTestStart3threaddeadlockwiththreadsuspend()
{
    // create two threads
    // first thread just uses critical sections
    // the second thread suspends the first thread and then trys to ac
    // If the lock is acquired, the second thread resumes the first th
    // some point the first thread will be suspended whilst it owns th
    // deadlock.
    // Unless another thread resumes the first thread, the second thre
```

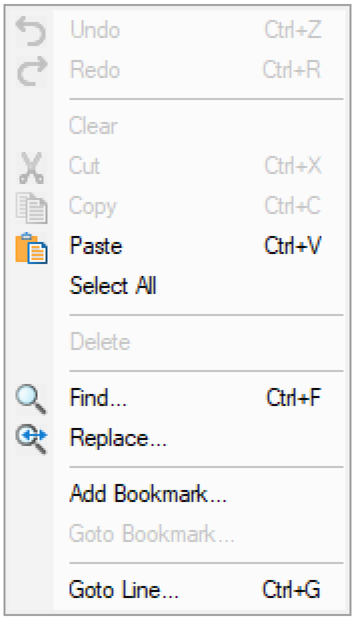
File menu

The file options need no explanation:



Edit menu

All the following edit options should also be familiar:

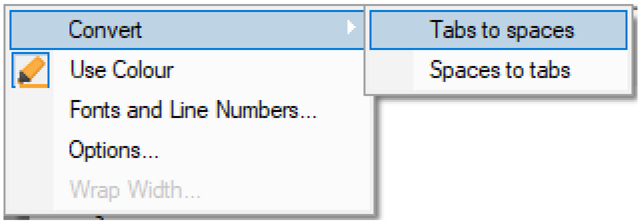


Undo/Redo is unlimited by default, but this can be changed in the options below.

Editing bookmarks has nothing to do with Thread Validator's own bookmarks.

Formatting menu

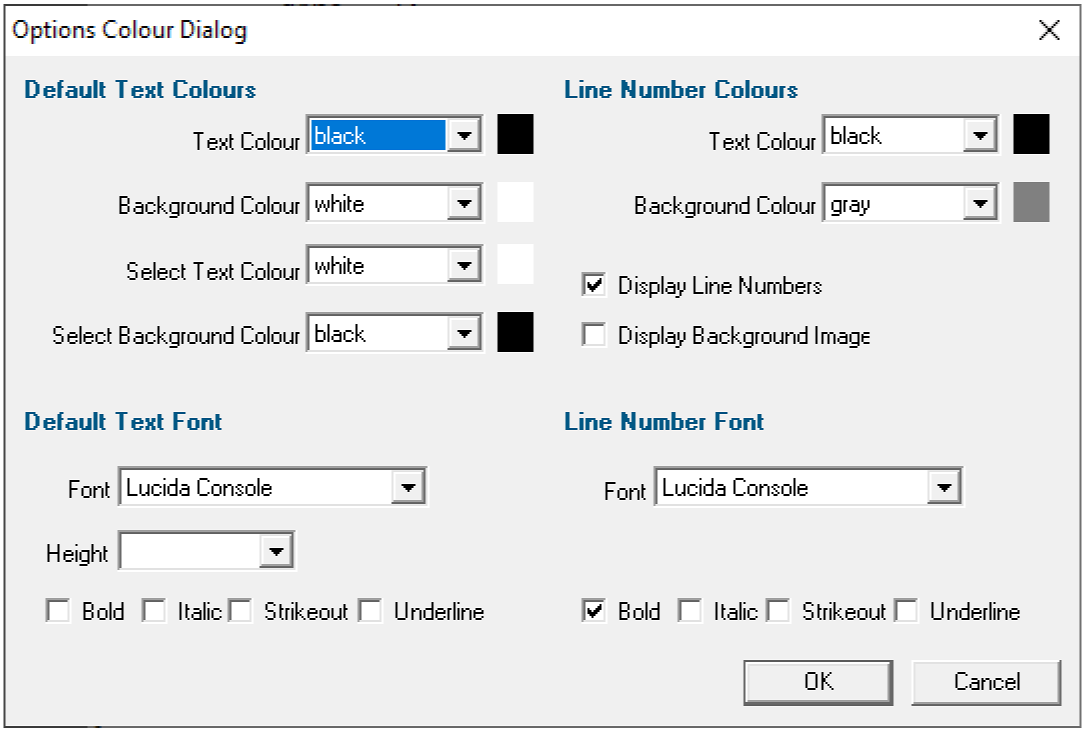
The formatting menu has general display and editing options



- **Convert** > **Tabs to spaces** > turns all tabs into spaces
- **Convert** > **Spaces to tabs** > turns all spaces into tabs

- **Use Colour** > toggles the colour coded display

- **Fonts and Line Numbers...** > change text colours, fonts and line numbers



Options Colour Dialog

Default Text Colours

Text Colour: black

Background Colour: white

Select Text Colour: white

Select Background Colour: black

Line Number Colours

Text Colour: black

Background Colour: gray

☑ Display Line Numbers

☐ Display Background Image

Default Text Font

Font: Lucida Console

Height:

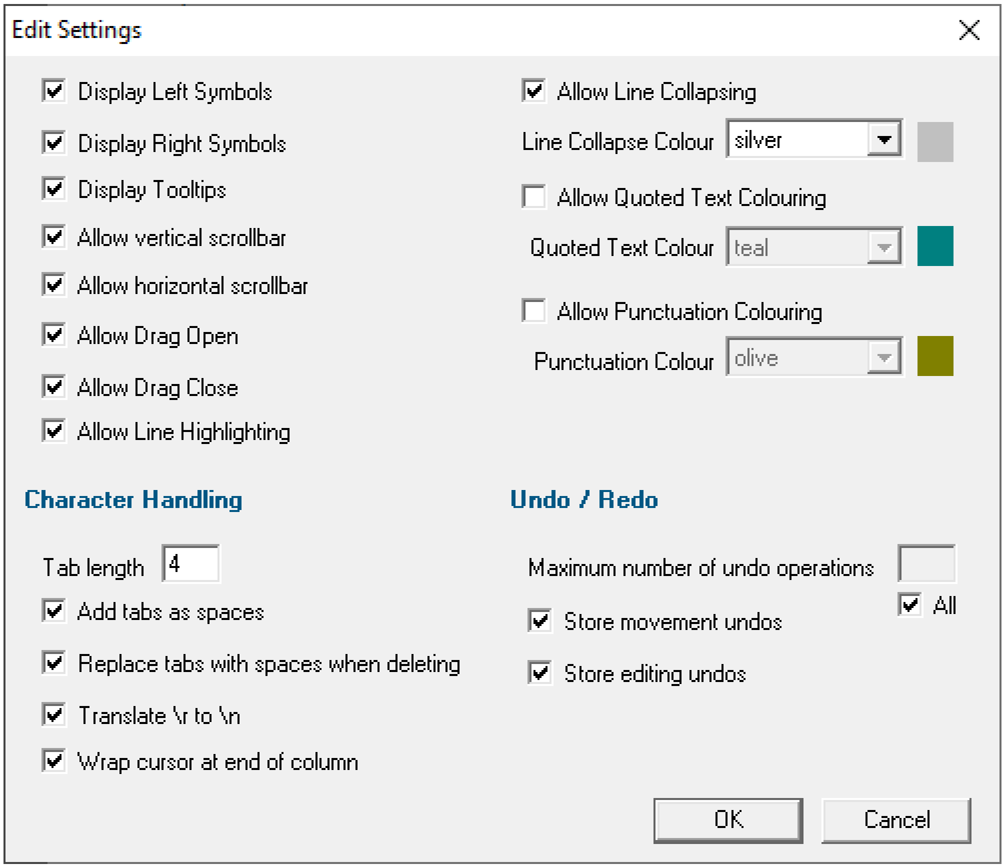
☐ Bold ☐ Italic ☐ Strikeout ☐ Underline

Line Number Font

Font: Lucida Console

☑ Bold ☐ Italic ☐ Strikeout ☐ Underline

- **Options...** › set tab length and other options



Edit Settings

☑ Display Left Symbols

☑ Display Right Symbols

☑ Display Tooltips

☑ Allow vertical scrollbar

☑ Allow horizontal scrollbar

☑ Allow Drag Open

☑ Allow Drag Close

☑ Allow Line Highlighting

☑ Allow Line Collapsing

Line Collapse Colour: silver

☐ Allow Quoted Text Colouring

Quoted Text Colour: teal

☐ Allow Punctuation Colouring

Punctuation Colour: olive

Character Handling

Tab length: 4

☑ Add tabs as spaces

☑ Replace tabs with spaces when deleting

☑ Translate \r to \n

☑ Wrap cursor at end of column

Undo / Redo

Maximum number of undo operations: ☑ All

☑ Store movement undos

☑ Store editing undos

- **Wrap Width...** > changes the column width at which lines will wrap in the display

Status bar

The status bar shows help text at the bottom as you hover over menu and toolbar options.

To the right of the status bar are insert mode, column number and line number.

Line collapsing

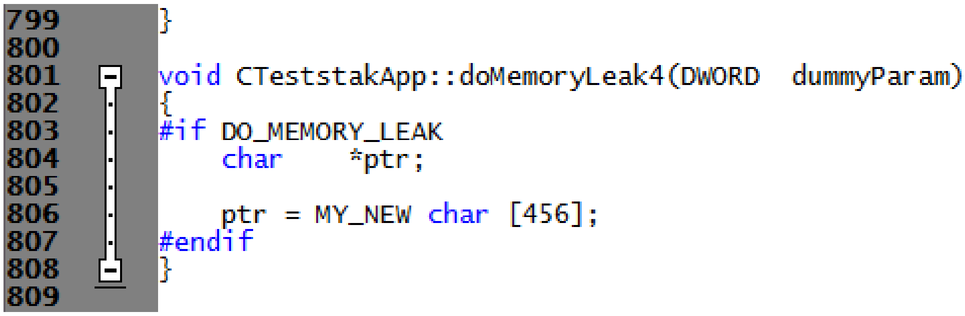
You can temporarily collapse sections of code as follows:

- **Left click** in the margin to start the section > **Drag** to define the length > **Release** to set the end of the section

  Click anywhere on the resulting indicator to collapse, and on the + to expand a section.

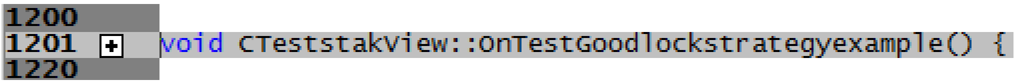
  You'll need to drag near the middle of the empty grey area, as not all of the margin is active.

Expanded:



```
}

void CTeststakApp::doMemoryLeak4(DWORD  dummyParam)
{
#if DO_MEMORY_LEAK
    char     *ptr;

    ptr = MY_NEW char [456];
#endif
}
```

Collapsed:



```
void CTeststakView::OnTestGoodlockstrategyexample() {
```

Line collapsing is *temporary* and not remembered between edit sessions.

3.14.2 Refresh and Refresh All

Refreshing data

You have the option in some views to automatically update in the view at an interval of your choice.

Sometimes you need to refresh the data when *you* want to, especially while inspecting the data.

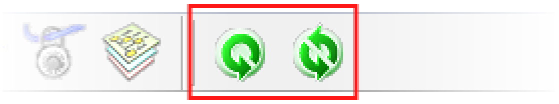
Most views have a local Refresh button, which updates the data.

The same function is found in the Tools menu, as well as an option to update all views at once.

Tools menu › **Refresh** › refresh the data displayed only on the current tabbed view

Tools menu › **Refresh All** › refresh the data on *all* the tabbed views

or use the **Refresh** and **Refresh All** icons on the Tools toolbar.



3.14.3 Loaded Modules

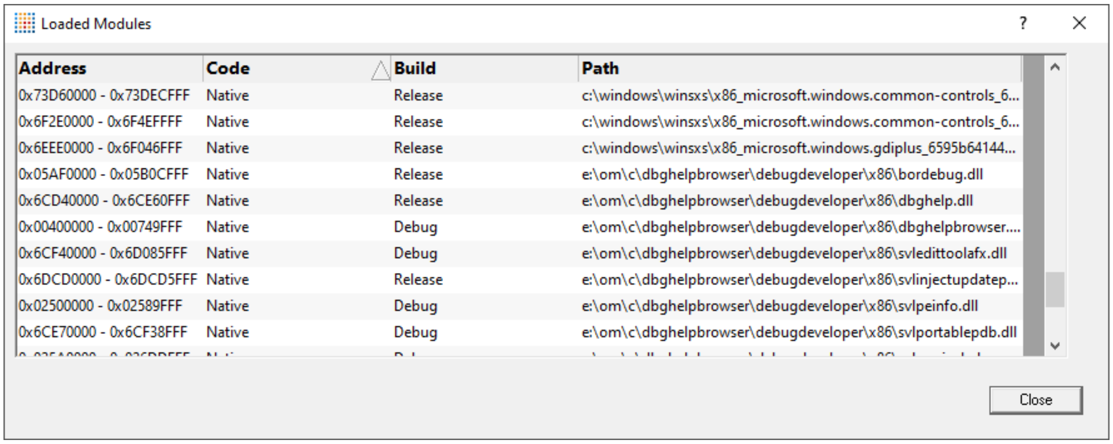
Viewing the loaded modules

You can view a list of the modules which are loaded by your target application.

Tools menu › **Loaded Modules...** › shows the Loaded Modules dialog below

The dialog shows:

- the **Address** space occupied by the module (DLL or EXE)
- the type of **Code** in the module (native, managed, mixed mode or resources only
- the type of **Build** - is the code debug or release?
- the **Path** the module was loaded from



Loaded Modules

Address	Code	Build	Path
0x73D60000 - 0x73DECFFF	Native	Release	c:\windows\winsxs\x86_microsoft.windows.common-controls_6...
0x6F2E0000 - 0x6F4EFFFF	Native	Release	c:\windows\winsxs\x86_microsoft.windows.common-controls_6...
0x6EEE0000 - 0x6F046FFF	Native	Release	c:\windows\winsxs\x86_microsoft.windows.gdiplus_6595b64144...
0x05AF0000 - 0x05B0CFFF	Native	Release	e:\om\c\dbghelpbrowser\debugdeveloper\x86\bordebug.dll
0x6CD40000 - 0x6CE60FFF	Native	Release	e:\om\c\dbghelpbrowser\debugdeveloper\x86\dbghelp.dll
0x00400000 - 0x00749FFF	Native	Debug	e:\om\c\dbghelpbrowser\debugdeveloper\x86\dbghelpbrowser....
0x6CF40000 - 0x6D085FFF	Native	Debug	e:\om\c\dbghelpbrowser\debugdeveloper\x86\svledittoolafx.dll
0x6DCD0000 - 0x6DCD5FFF	Native	Release	e:\om\c\dbghelpbrowser\debugdeveloper\x86\svlinjectupdatep...
0x02500000 - 0x02589FFF	Native	Debug	e:\om\c\dbghelpbrowser\debugdeveloper\x86\svlpeinfo.dll
0x6CE70000 - 0x6CF38FFF	Native	Debug	e:\om\c\dbghelpbrowser\debugdeveloper\x86\svlportablepdb.dll

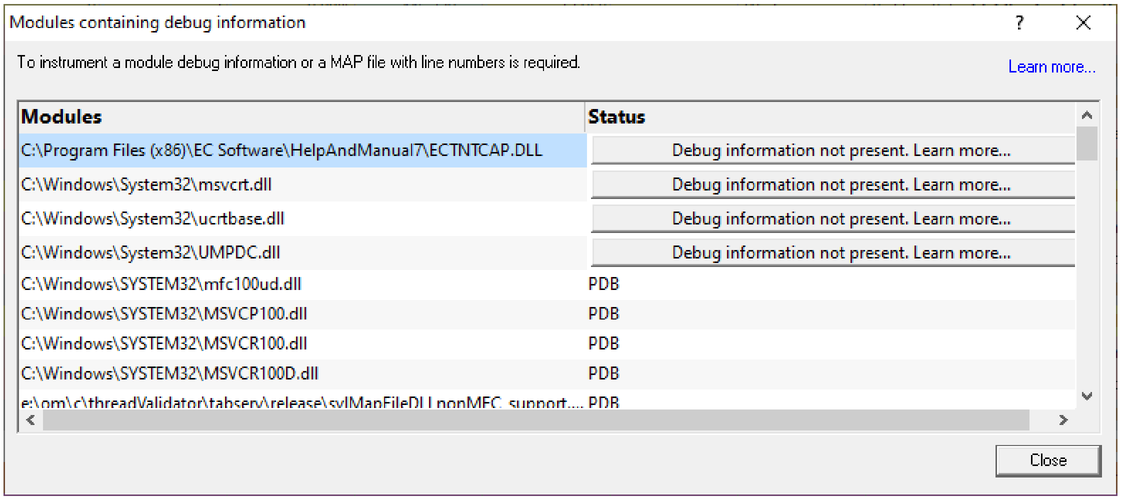
3.14.4 DLL Debug Information

Viewing the DLL debug information

If you are having problems collecting thread data for a particular EXE/DLL the problem may be that the debug information that is required to perform the instrumentation of the software cannot be found.

You can view a list of the debug information status of modules loaded by your target application.

Tools menu › **DLL Debug Information...** › shows the DLL Debug Information dialog below



Modules containing debug information

To instrument a module debug information or a MAP file with line numbers is required.

Modules	Status
C:\Program Files (x86)\EC Software\HelpAndManual7\ECTNTCAP.DLL	Debug information not present. Learn more...
C:\Windows\System32\msvcrt.dll	Debug information not present. Learn more...
C:\Windows\System32\ucrtbase.dll	Debug information not present. Learn more...
C:\Windows\System32\UMPDC.dll	Debug information not present. Learn more...
C:\Windows\SYSTEM32\mfc100ud.dll	PDB
C:\Windows\SYSTEM32\MSVCP100.dll	PDB
C:\Windows\SYSTEM32\MSVCR100.dll	PDB
C:\Windows\SYSTEM32\MSVCR100D.dll	PDB
e:\om\c\threadValidator\tabserv\release\svlMapFileDLLnonMFC_support....	PDB

The dialog shows:

- the path from which **Modules** (DLL or EXE) were loaded
- the debug **Status** (below)
- if any symbol server is not reachable (offline or doesn't exist) a message will be shown in red at the bottom of the dialog. You can edit the symbol server definitions here.

Debug status

There are various reasons why a module may not have its debug information read.

The dialog shows a comment or reason in the status column. Examples might be:

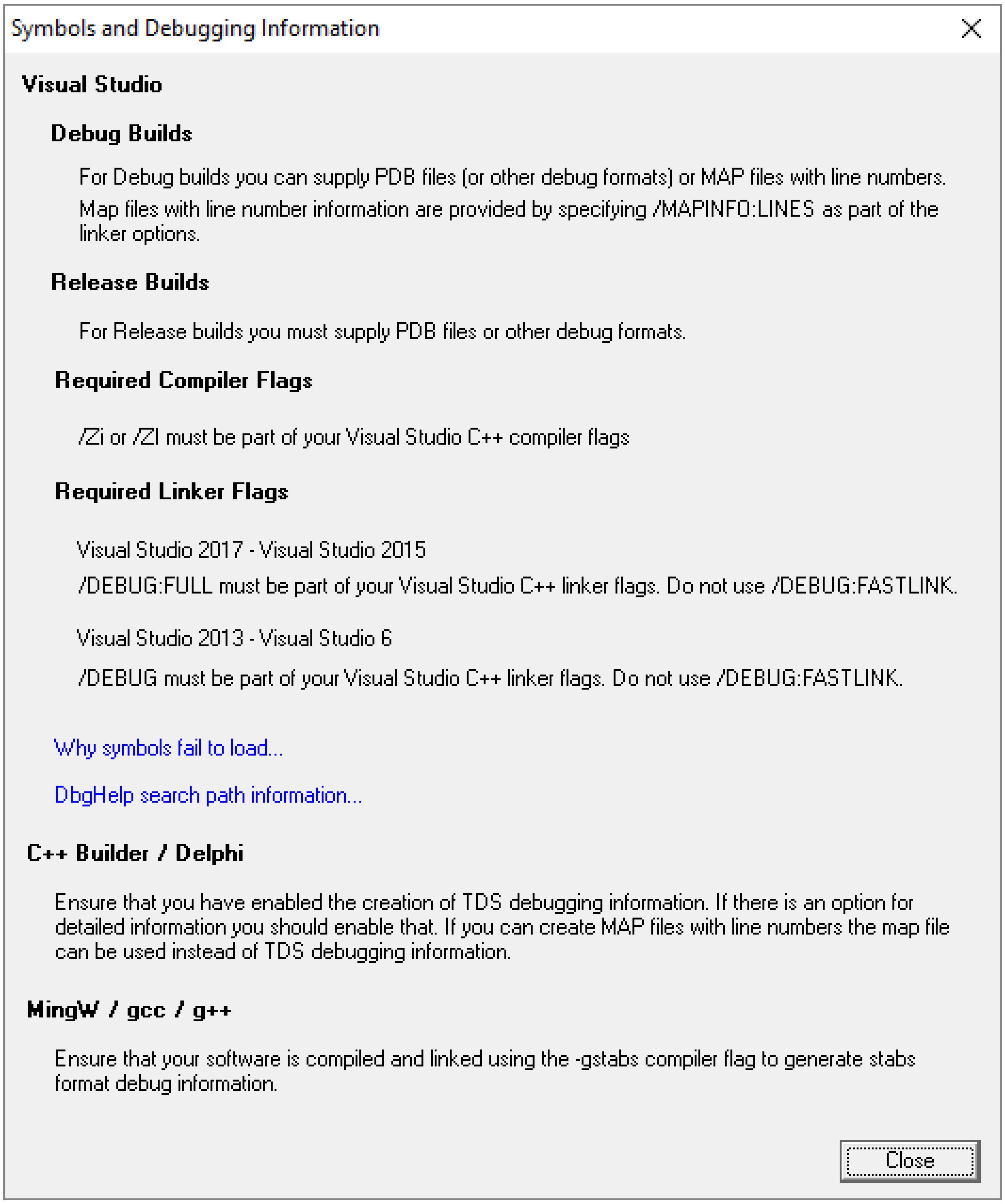
- PDB or MAP if the debug information *was* found and used
- Debug information not present
- A reason for being ignored

- Module is a part of the C Run-time Library (CRT) or Standard Template Library (STL)
- Location is a system directory
- Ignored due to Hooked DLLs advanced settings
- File is a Software Verify own module
- Module has been specified as a 3rd party
- No executable code is contained
- The module only has GUI resources

More information about PDB and MAP files

Clicking on the **Learn more...** link at the top right of the dialog shows some more details with additional links to topics in this help.

Click the links below to read more in our frequently asked questions.



Symbols and Debugging Information

Visual Studio

Debug Builds

For Debug builds you can supply PDB files (or other debug formats) or MAP files with line numbers.

Map files with line number information are provided by specifying /MAPINFO:LINES as part of the linker options.

Release Builds

For Release builds you must supply PDB files or other debug formats.

Required Compiler Flags

/Zi or /ZI must be part of your Visual Studio C++ compiler flags

Required Linker Flags

Visual Studio 2017 - Visual Studio 2015

/DEBUG:FULL must be part of your Visual Studio C++ linker flags. Do not use /DEBUG:FASTLINK.

Visual Studio 2013 - Visual Studio 6

/DEBUG must be part of your Visual Studio C++ linker flags. Do not use /DEBUG:FASTLINK.

Why symbols fail to load...

DbgHelp search path information...

C++ Builder / Delphi

Ensure that you have enabled the creation of TDS debugging information. If there is an option for detailed information you should enable that. If you can create MAP files with line numbers the map file can be used instead of TDS debugging information.

MingW / gcc / g++

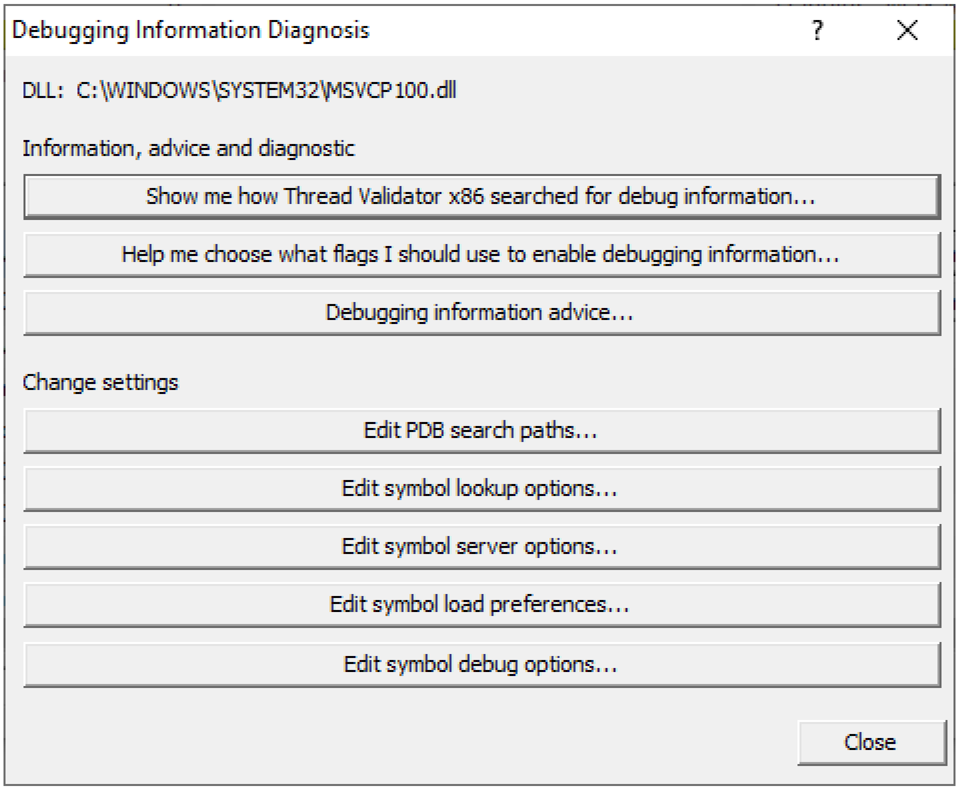
Ensure that your software is compiled and linked using the -gstabs compiler flag to generate stabs format debug information.

Finding out more using the Debugging Information Diagnosis Dialog

When debug information is not present for a given module the DLL Debug Information dialog (above) may display a button in the Status column to show the Debugging Information Diagnosis dialog.

The dialog shows:

- Information, advice, and diagnostic help
- Quick links to change settings



Debugging Information Diagnosis

DLL: C:\WINDOWS\SYSTEM32\MSVCP100.dll

Information, advice and diagnostic

? ×
Show me how Thread Validator x86 searched for debug information...
Help me choose what flags I should use to enable debugging information...
Debugging information advice...

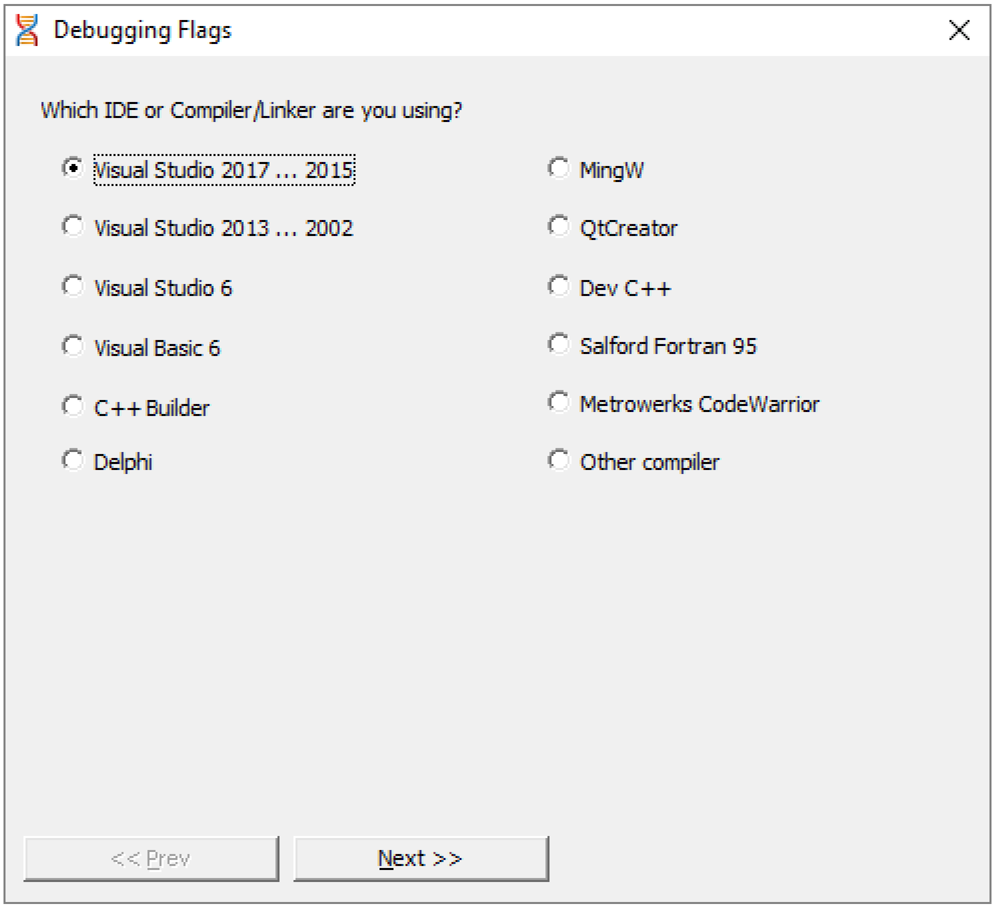
Change settings

Edit PDB search paths...
Edit symbol lookup options...
Edit symbol server options...
Edit symbol load preferences...
Edit symbol debug options...
Close

The information options include:

- **Debugging information advice...** › shows the Symbols and Debugging Information dialog above.
- **Help me choose what flags...** › shows the Debugging Flags wizard

  Use the wizard to first select the compiler or linker you're using



Which IDE or Compiler/Linker are you using?

● Visual Studio 2017 ... 2015

○ Visual Studio 2013 ... 2002

○ Visual Studio 6

○ Visual Basic 6

○ C++ Builder

○ Delphi

○ MingW

○ QtCreator

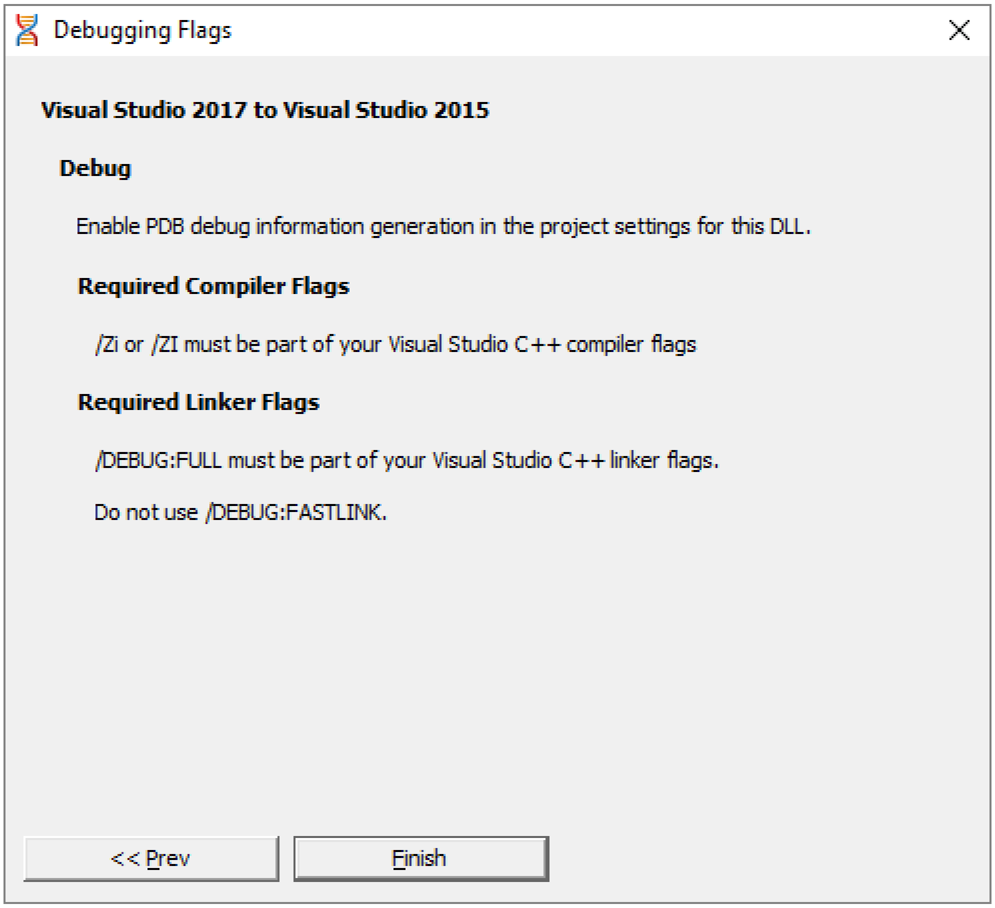
○ Dev C++

○ Salford Fortran 95

○ Metrowerks CodeWarrior

○ Other compiler

Next >> › Provides the relevant debug compiler and linker flags. An example for Visual Studio 2017 to 2015 is below:



Visual Studio 2017 to Visual Studio 2015

Debug

Enable PDB debug information generation in the project settings for this DLL.

Required Compiler Flags

/Zi or /ZI must be part of your Visual Studio C++ compiler flags

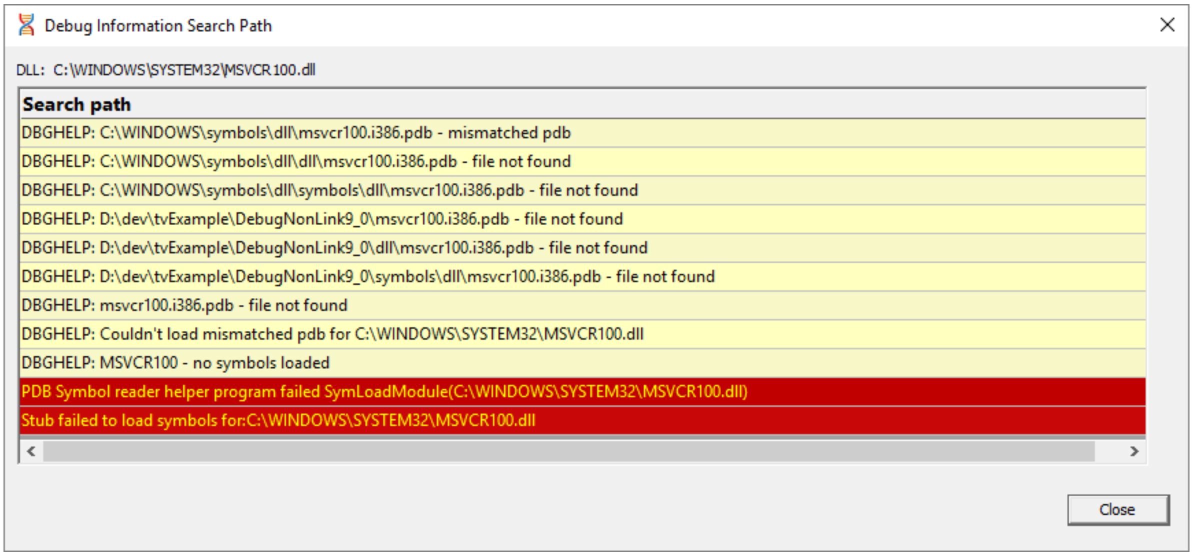
Required Linker Flags

/DEBUG:FULL must be part of your Visual Studio C++ linker flags.

Do not use /DEBUG:FASTLINK.

- **Show me how debug information was searched for...** › shows the Debug Information Search Path dialog

This information is extracted from the Diagnostic tab and shows only the relevant information for the module selected in the DLL Debug Information dialog.



DLL: C:\WINDOWS\SYSTEM32\MSVCR100.dll

Search path
DBGHELP: C:\WINDOWS\symbols\dll\msvcr100.i386.pdb - mismatched pdb
DBGHELP: C:\WINDOWS\symbols\dll\dll\msvcr100.i386.pdb - file not found
DBGHELP: C:\WINDOWS\symbols\dll\symbols\dll\msvcr100.i386.pdb - file not found
DBGHELP: D:\dev\tvExample\DebugNonLink9_0\msvcr100.i386.pdb - file not found
DBGHELP: D:\dev\tvExample\DebugNonLink9_0\dll\msvcr100.i386.pdb - file not found
DBGHELP: D:\dev\tvExample\DebugNonLink9_0\symbols\dll\msvcr100.i386.pdb - file not found
DBGHELP: msvcr100.i386.pdb - file not found
DBGHELP: Couldn't load mismatched pdb for C:\WINDOWS\SYSTEM32\MSVCR100.dll
DBGHELP: MSVCR100 - no symbols loaded
PDB Symbol reader helper program failed SymLoadModule(C:\WINDOWS\SYSTEM32\MSVCR100.dll)
Stub failed to load symbols for:C:\WINDOWS\SYSTEM32\MSVCR100.dll

The options for changing settings include quick links to the following pages from the Global Settings Dialog

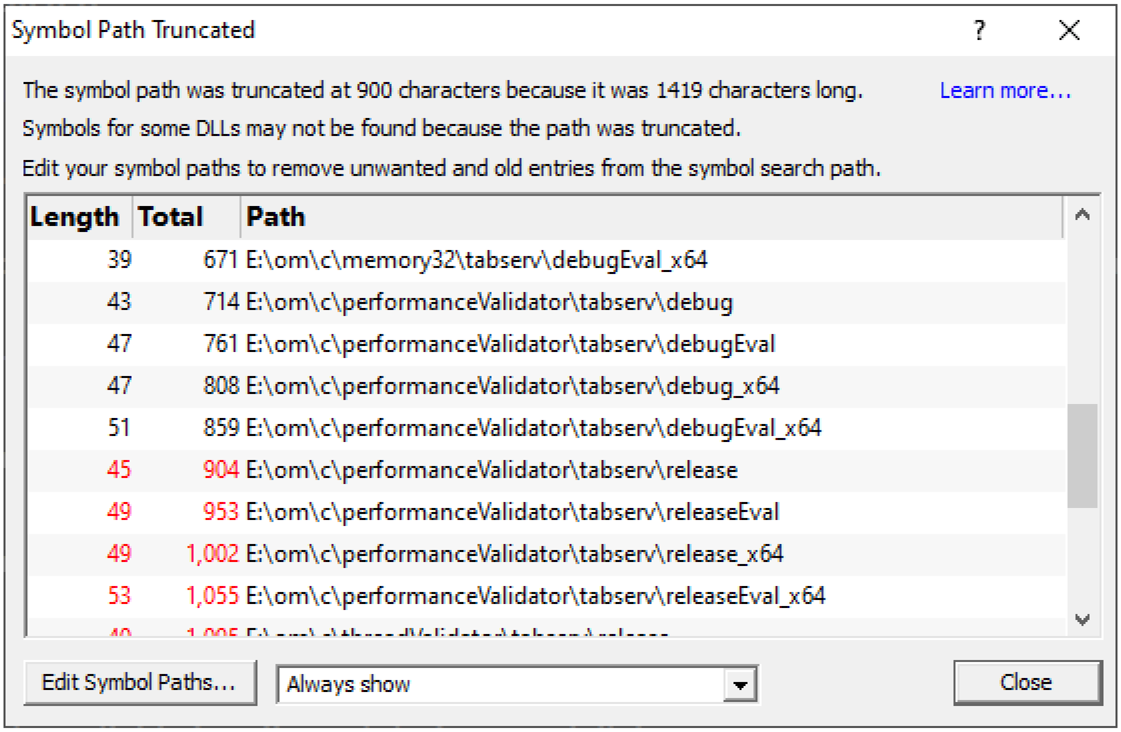
- **Edit PDB search paths...** › shows the File Location settings page for PDB files.
- **Edit symbol lookup options...** › shows the Symbol Lookup settings page
- **Edit symbol server options...** › shows the Symbol Servers settings page
- **Edit symbol load preferences...** › shows the Symbol Load Preferences settings page
- **Edit symbol debug options...** › shows the Symbol Misc settings page

3.14.5 Symbol Path Truncation

Tools menu › **Symbol Path Truncation...** › shows the Symbol Path Truncation dialog

The Symbol Path Truncated warning dialog

The symbol path truncated warning dialog is displayed to warn you when the symbol path is too long.



The symbol path was truncated at 900 characters because it was 1419 characters long.
Symbols for some DLLs may not be found because the path was truncated.
Edit your symbol paths to remove unwanted and old entries from the symbol search path.

Length	Total	Path
39	671	E:\om\c\memory32\tabserv\debugEval_x64
43	714	E:\om\c\performanceValidator\tabserv\debug
47	761	E:\om\c\performanceValidator\tabserv\debugEval
47	808	E:\om\c\performanceValidator\tabserv\debug_x64
51	859	E:\om\c\performanceValidator\tabserv\debugEval_x64
45	904	E:\om\c\performanceValidator\tabserv\release
49	953	E:\om\c\performanceValidator\tabserv\releaseEval
49	1,002	E:\om\c\performanceValidator\tabserv\release_x64
53	1,055	E:\om\c\performanceValidator\tabserv\releaseEval_x64

Always show ▾

- **Edit Symbol Paths...** › shows the file locations dialog so that you can edit the paths used for subsequent runs of the program.

You can choose when this dialog is displayed.

- **Always show** › The dialog is always shown when the symbol path is too long.
- **Show when path changes** › The dialog is shown when the symbol path is too long, but only if the symbol path is different than last time this warning was shown.
- **Never show** › The dialog is never shown.

Whether this dialog is displayed or not there is always a warning message written to the diagnostic window when the symbol path is truncated.

The display lists each path with it's length (including the unshown ';' path separator) and the total length so far so that you can see which paths exceed the truncation point (length and total displayed in red).

Any paths that don't exist on this computer are displayed in red.

Why is this dialog displayed?

You may see a Symbol Path Truncated warning dialog in some rare circumstances.

This dialog is displayed when the symbol path that has been calculated to pass to DbgHelp.dll to load Microsoft debugging symbols (found in .PDB files) is too long.

If the symbol path has been truncated because it is too long it is possible this may mean that some symbol searches will fail, resulting in failure to load some symbols. We display this dialog so that you are aware that the symbol path is too long and would benefit from editing to make the symbol path shorter.

Passing a symbol path that is too long to DbgHelp.dll will cause the program being tested to end with an EXCEPTION_INVALID_CRUNTIME_PARAMETER C runtime error. This happens because internally DbgHelp.dll is using a fixed length array to format a string. To prevent this fatal termination of the test program we limit the length of the path passed to DbgHelp.dll.

Typically if a path that is long enough to cause this problem is passed to DbgHelp it's because the number of paths in the calculated path contain paths not relevant to finding symbols for the test program. We use the Symbol Path Truncated warning dialog to show you the calculated paths so that you can work out which paths to delete.

The calculated symbol paths come from several places:

- File locations PDB paths
- Symbol server symbol storage directories
- Symbol handling environment variables

Fixing the symbol path

For this example, we are testing the program **E:\om\c\3RD_SRC\cdplayer\Release\cdplayer.exe**

In the image shown above you can see that seven paths exceed the truncation limit, one of the 7 paths doesn't exist.

To work out what to do we need to do several actions:

1. Looking at the environment variable settings shows that none of the environment variables are being used. We do not need to consider the content of these environment variables.

2. Examining the symbol servers shows that **C:\Users\Admin** is a local symbol storage location. We should keep this path.

3. We should delete the path that doesn't exist: **E:\om\c\testApps\testStdinStdoutRedirectEx\Release**. We do this using the file locations dialog by clicking **Edit Symbol Paths...** then click **Delete invalid**.

4. Examining the paths in the file locations dialog we can identify any paths not relevant to the program we are testing. In this case the following paths are not relevant and can be deleted.

E:\om\c\testApps\testStdinStdoutRedirect\Release
E:\om\c\testApps\testAppTheReadsFromStdinAndWritesToStdout\Release
E:\om\c\testApps\testSimpleMemoryLeak\Release
e:\om\c\3rd_src\cppunit-1.12.1\examples\cppunittest\release
e:\om\c\3rd_src\cppunit-1.12.1\examples\cppunittest\releasedll

3.14.6 Out Of Date DLLs

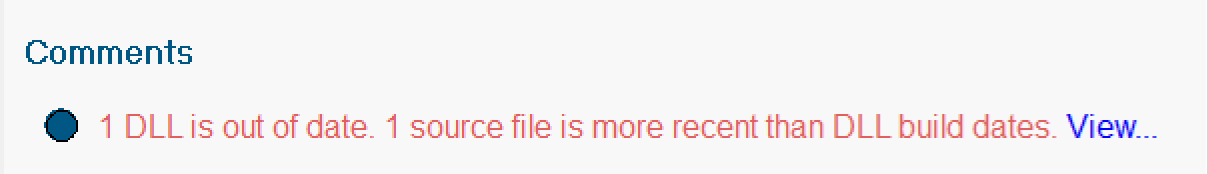
It may happen that if you forget to build a DLL, or if a build error occurs that you perform deadlock detection on DLLs that are not built with the most recent version of your source code.

We refer to these DLLs are out of date DLLs because they are out of date compared to the source code that is compiled to create the DLLs.

Thread Validator can detect this, and warn you about it.

Summary tab

When out of date DLLs are found a warning is displayed on the summary tab, in the lower section of the display.



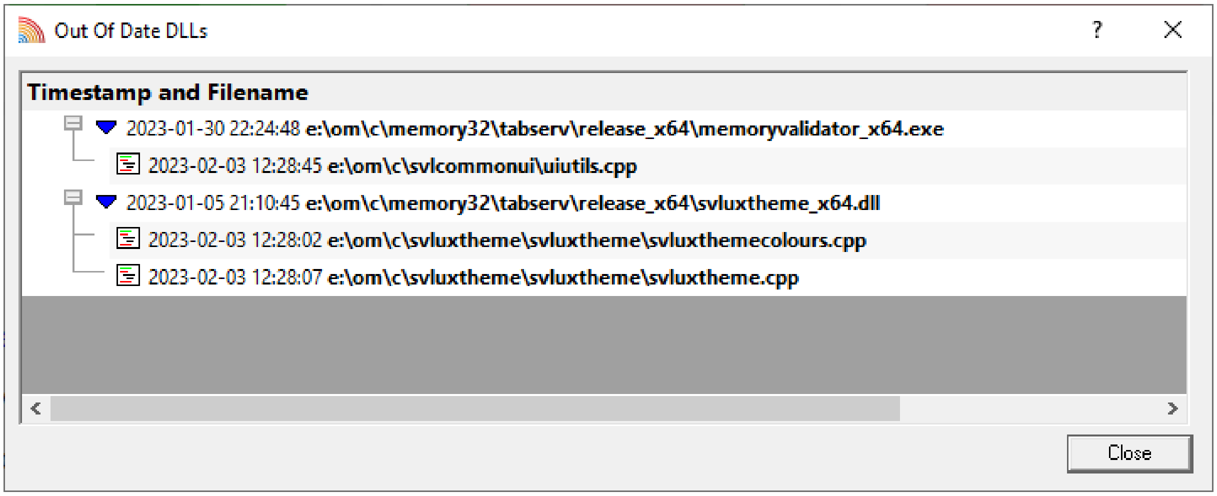
Comments

● 1 DLL is out of date. 1 source file is more recent than DLL build dates. View...

Out Of Date DLLs Dialog

The **View...** link will display the Out Of Date DLLs dialog.

There is also an option to display the Out Of Date DLLs dialog on the Tools menu.



Out Of Date DLLs

Timestamp and Filename

```
⊟ ▼ 2023-01-30 22:24:48 e:\om\c\memory32\tabserv\release_x64\memoryvalidator_x64.exe
      2023-02-03 12:28:45 e:\om\c\svlcommonui\uiutils.cpp
⊟ ▼ 2023-01-05 21:10:45 e:\om\c\memory32\tabserv\release_x64\svluxtheme_x64.dll
      2023-02-03 12:28:02 e:\om\c\svluxtheme\svluxtheme\svluxthemecolours.cpp
      2023-02-03 12:28:07 e:\om\c\svluxtheme\svluxtheme\svluxtheme.cpp
```

The Out of date DLLs dialog shows the DLLs that are out of date, and the source files for each DLLs. The dates of both the DLLs and the source files are displayed.

The above image shows 1 DLL that is out of date, with 1 file being more recently edited than the build timestamp for the respective DLL.

3.14.7 Instrumentation Logging Data

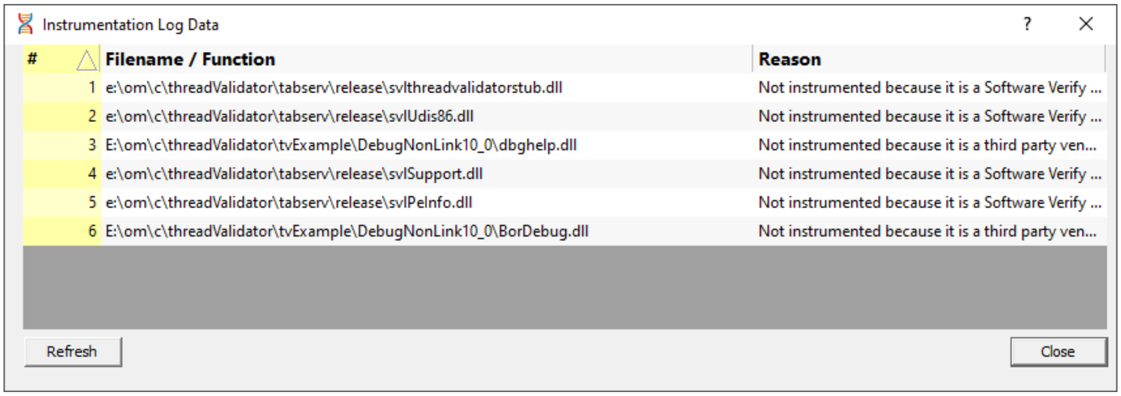
Instrumentation Log Data

It can be very useful to know why your code hasn't been instrumented. For example, a file or part of a file you were expecting to receive coverage information may have been excluded by one or more of the DLL filters in the Global Settings dialog.

To log details of why dlls are not instrumented, first switch on the instrumentation logging settings.

Once enabled, and a session has started, you can view a list of the files that have not been instrumented via the Tools menu.

Tools menu › **Instrumentation Logging Data...** › shows the Instrumentation Log Data dialog



#	Filename / Function	Reason
1	e:\om\c\threadValidator\tabserv\release\svlthreadvalidatorstub.dll	Not instrumented because it is a Software Verify ...
2	e:\om\c\threadValidator\tabserv\release\svlUdis86.dll	Not instrumented because it is a Software Verify ...
3	E:\om\c\threadValidator\tvExample\DebugNonLink10_0\dbghelp.dll	Not instrumented because it is a third party ven...
4	e:\om\c\threadValidator\tabserv\release\svlSupport.dll	Not instrumented because it is a Software Verify ...
5	e:\om\c\threadValidator\tabserv\release\svlPeInfo.dll	Not instrumented because it is a Software Verify ...
6	E:\om\c\threadValidator\tvExample\DebugNonLink10_0\BorDebug.dll	Not instrumented because it is a third party ven...

The dialog shows:

- log order
- the name of the item that hasn't been instrumented
- the reason why each item wasn't instrumented

Example reasons why an item might not be instrumented include the following:

- MFC file ignored
- Microsoft C Runtime file ignored
- Microsoft DLL ignored
- CRT DLL ignored

- Software Verify's own DLL ignored

3.15 Software Updates

This topic covers the items on the Software Updates menu:

- checking for software updates
- configuring your update schedule
- renewing your software maintenance
- setting your software update credentials
- setting the software update directory

Software updates

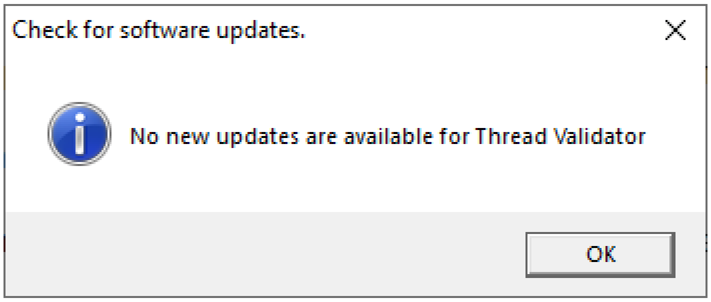
If you've been notified of a new software release to Thread Validator or just want to see if there's a new version, this feature makes it easy to update.

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



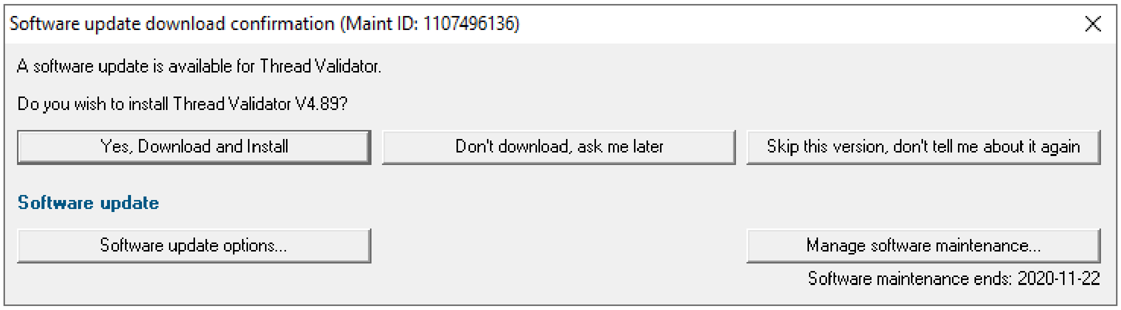
Check for software updates.

No new updates are available for Thread Validator

Note that evaluation versions cannot be updated.

Software Update dialog

If a software update is available for Thread Validator you'll see the software update dialog, unless your maintenance has expired.



Software update download confirmation (Maint ID: 1107496136)

A software update is available for Thread Validator.

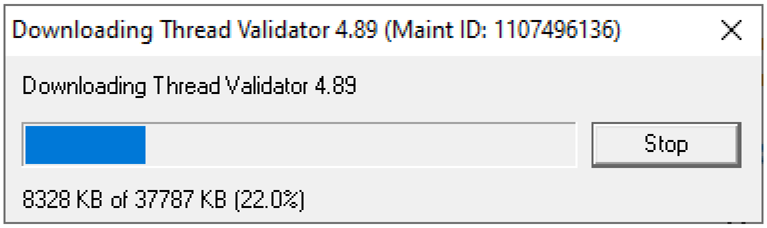
Do you wish to install Thread Validator V4.89?

Yes, Download and Install
Don't download, ask me later
Skip this version, don't tell me about it again
Software update
Software update options...
Manage software maintenance...

Software maintenance ends: 2020-11-22

- **Download and install** › prompts you for login details if not known, and then downloads the update, showing progress

  If you haven't set login credentials, you may be asked for your user email and password, which you'll have received when you purchased Thread Validator or subsequent update email.

  Downloading Thread Validator 4.89 (Maint ID: 1107496136)

  Downloading Thread Validator 4.89

  8328 KB of 37787 KB (22.0%)

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

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

- **Don't download** › Doesn't download, but you'll be prompted for it again next time you start Thread Validator

- **Skip this version** › Doesn't download the update and doesn't bother you again until there's an even newer update

- **Software update options...** › edit the software update schedule

- **Manage software maintenance...** › opens your browser ready for maintenance renewal

Problems downloading or installing?

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

- Log in to https://www.softwareverify.com/authdownload.php with the details provided when you purchased Thread Validator

- Download the latest installer manually, via one of the .exe, .xyz or .zip files that are available

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

- Ensure any open sessions are completed
- Ensure any target programs started by Thread Validator are closed
- Ensure Thread Validator and its help manual is also closed
- Ensure any error dialogs from the previous installation are closed

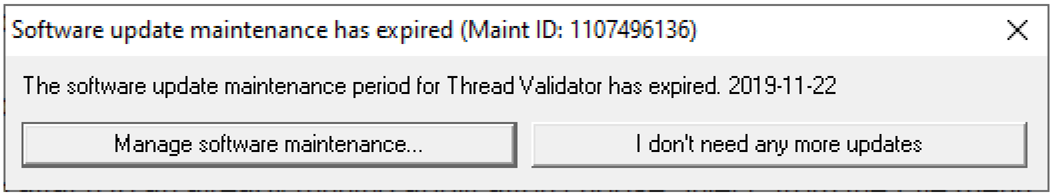
Have your license details handy as you may need to copy information into the license dialog

You should now be ready to run the new version.

Software maintenance expiry

If the software maintenance period has expired you won't be able to automatically update Thread Validator as above.

Instead, you'll see the software update maintenance expiry dialog:



Software update maintenance has expired (Maint ID: 1107496136)

The software update maintenance period for Thread Validator has expired. 2019-11-22

You can manage your software maintenance or choose to stop receiving any more software updates.

Software update schedule

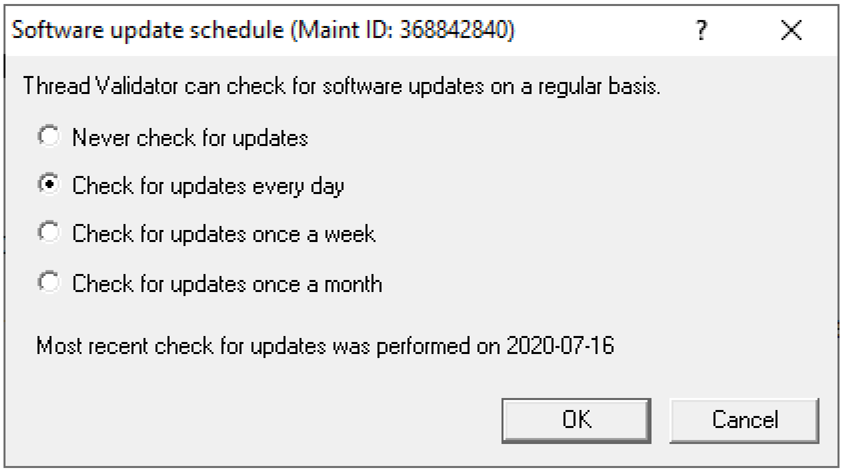
Thread Validator can automatically check to see if a new version of Thread Validator is available for downloading.

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

The update options are:

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

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



Software update schedule (Maint ID: 368842840)

Thread Validator can check for software updates on a regular basis.

○ Never check for updates

● Check for updates every day

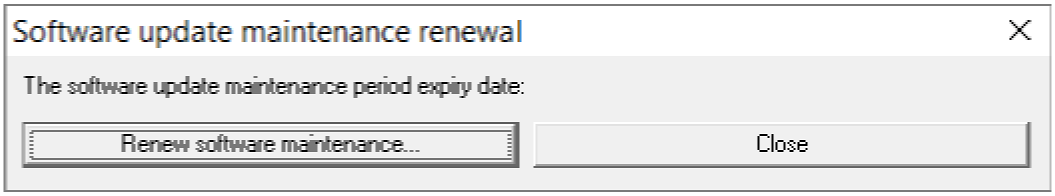
○ Check for updates once a week

○ Check for updates once a month

Most recent check for updates was performed on 2020-07-16

Managing software maintenance

☰ **Software Updates** menu › **Renew software updates** › shows the Software update maintenance renewal dialog



Software update maintenance renewal

The software update maintenance period expiry date:

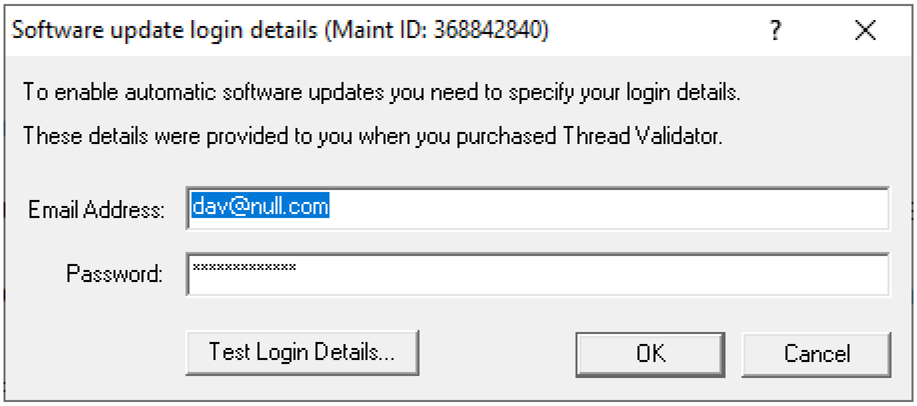
- **Renew software maintenance** › Opens your browser, logging you in to our website from which you can purchase maintenance

Your maintenance expiry date is shown. If you don't need to do anything just **Close** the dialog.

Managing software update credentials

You can configure your software update credentials within the application.

☰ **Software Updates** menu › **Set software update credentials** › shows the Software update login details dialog



Software update login details (Maint ID: 368842840)

To enable automatic software updates you need to specify your login details.

These details were provided to you when you purchased Thread Validator.

Email Address: dav@null.com

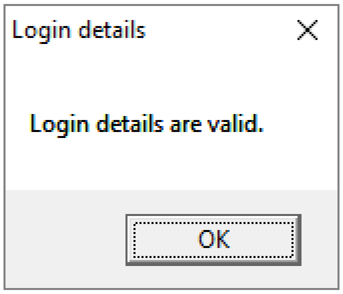
Password: *************

The text will be shown in red if the email address looks incorrectly formatted.

Testing the login details checks they're valid:

- **Test login details ›** check your entered details are valid (requires an internet connection)

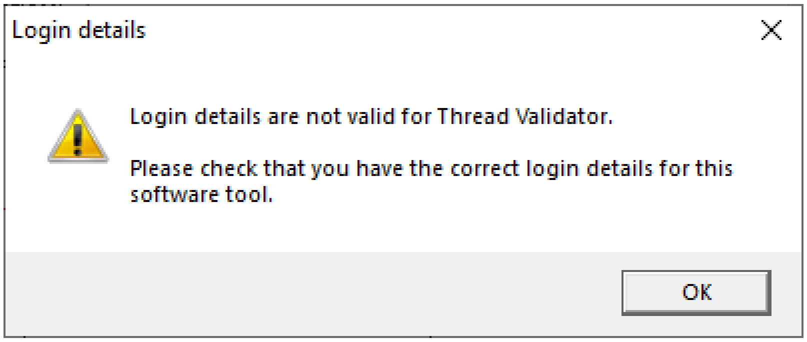
Valid details will be confirmed:



Login details

Login details are valid.

Invalid details may mean you entered credentials for another application in the Validator suite, or they could have been entered incorrectly.



Login details

Login details are not valid for Thread Validator.

Please check that you have the correct login details for this software tool.

You should have received the correct credentials when you purchased Thread Validator, or with any software update emails.

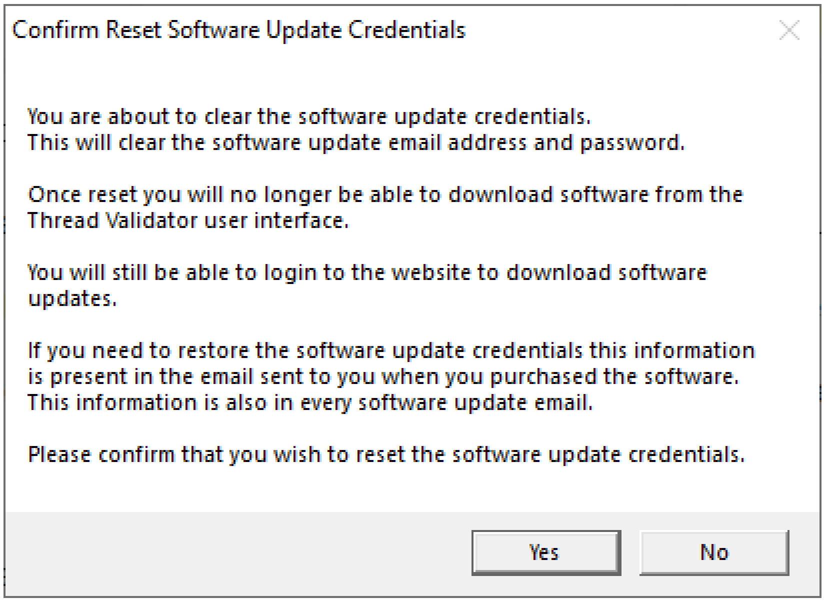
If you experience problems, check with your system administrator or contact Software Verify.

If you need to clear the update credentials, you can do this directly from the menu.

Software Updates menu › **Reset software update credentials** › clears the email and password details stored in the application

You will be asked to confirm the reset. After resetting the credentials, no software updates will occur.

If you later need to restore your credentials, you should have received that information when you purchased Thread Validator, or with any software update emails.



Confirm Reset Software Update Credentials

You are about to clear the software update credentials.
This will clear the software update email address and password.

Once reset you will no longer be able to download software from the Thread Validator user interface.

You will still be able to login to the website to download software updates.

If you need to restore the software update credentials this information is present in the email sent to you when you purchased the software.
This information is also in every software update email.

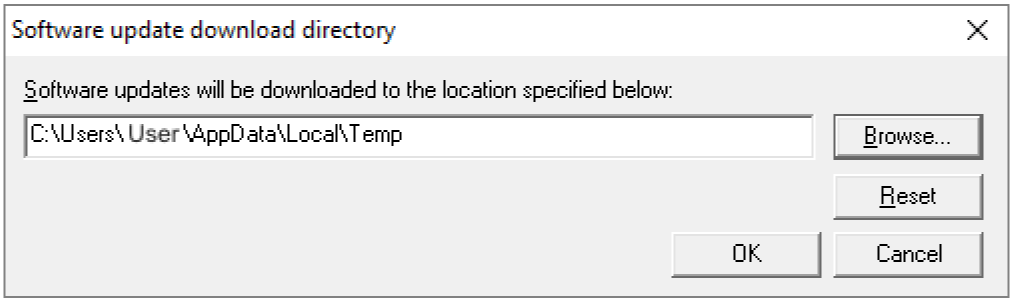
Please confirm that you wish to reset the software update credentials.

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



Software update download directory

Software updates will be downloaded to the location specified below:

C:\Users\ User \AppData\Local\Temp

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`

3.16 Loading, Saving, Exporting, Closing

Thread Validator allows sessions to be saved so that you can send the session to a colleague or examine the session at a later date. To complement the save capability sessions can be loaded.

Sessions can be exported in HTML and XML formats.

When you have finished working with a session, a session can be closed. The session can be reopened later if so desired.

Working with sessions

Sessions with Thread Validator can be saved to and loaded from a file so that you can:

- share the session with a colleague

- examine the session at a later date

Sessions can be even exported in HTML and XML formats.

You can have multiple sessions open at once.

Closing a session

When you've finished working with a session, it can be closed.

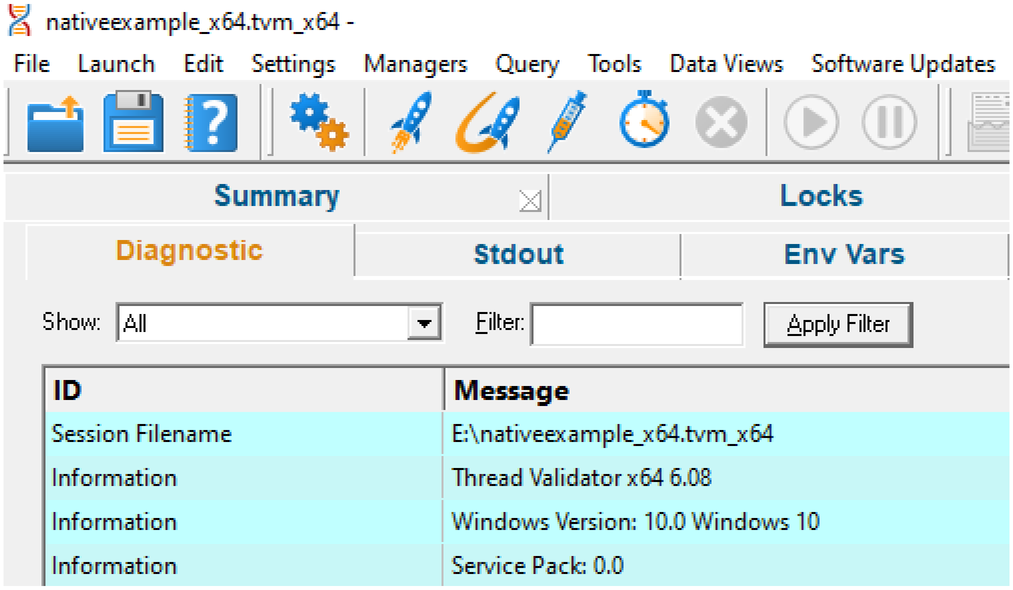
File menu › **Close Session...** › closes the session, clearing the displays

Closing a session may happen automatically if you start a new session and the session count limit is 1.

If the maximum session count allows, closed sessions still appear in the Session Manager, where they can be reopened or deleted.

Session Filename

The session filename is displayed as the first line of the diagnostic data on the Diagnostic tab.



ID	Message
Session Filename	E:\nativeexample_x64.tvm_x64
Information	Thread Validator x64 6.08
Information	Windows Version: 10.0 Windows 10
Information	Service Pack: 0.0

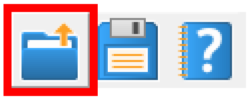
3.16.1 Loading & Saving Sessions

Loading sessions

Load a session using any of the following options.

File menu > Open Session... > open a previously saved session from file (*.tvm)

Or click on the Open Session icon on the standard toolbar.



Or use the shortcut:

Ctrl + O Open session

If you have a limit of one session to be open at a time, any open session will be closed first, otherwise you can open multiple sessions at a time.

Saving sessions

Save a session using any of the following options.

File menu > **Save Session...** > saves all the session data to a file (*.tvm), prompting for a file name if necessary

File menu > **Save As...** > saves the session to a new file

Or click on the Save Session icon on the standard toolbar.

Or use the shortcut:

Ctrl + S Save session

Unlike exports, there are no options here, as all the session data is saved.

3.16.2 Exporting Sessions

Exporting to HTML or XML

Exporting sessions allows you to use external tools to analyse or view session data for whatever reasons you might need.

You can export to HTML or XML format:

File menu > **Export Session...** > Choose **HTML Report** or **XML Report** > shows the Export Session dialog below

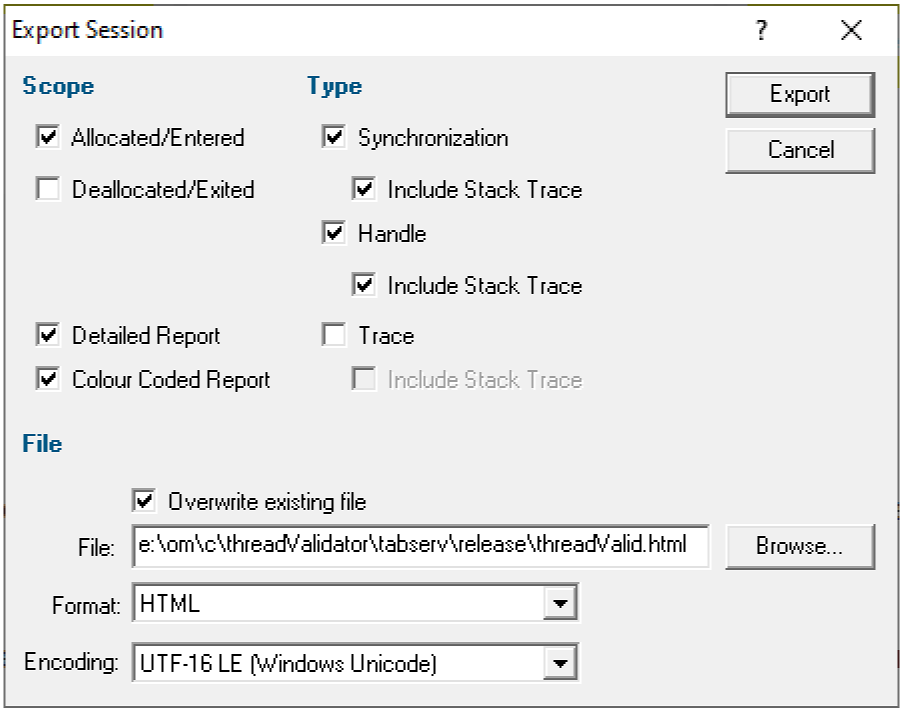
Exporting is not saving

You can't import session data.

Use save and load if you want to save session data for loading back into Thread Validator at a later date.

The Export Session dialog

The Export Session dialog looks very similar to the Save Session dialog, except there are more options enabled.



Scope

☑ Allocated/Entered

□ Deallocated/Exited

☑ Detailed Report

☑ Colour Coded Report

Type

☑ Synchronization

☑ Include Stack Trace

☑ Handle

☑ Include Stack Trace

□ Trace

□ Include Stack Trace

File

☑ Overwrite existing file

File: e:\om\c\threadValidator\tabserv\release\threadValid.html

Format: HTML

Encoding: UTF-16 LE (Windows Unicode)

Scope section

Critical sections and waits will be exported automatically. The following additional information is optional

- **Allocated/Entered** › object types specified below that have been allocated or entered
- **Deallocated/Exited** › object types specified below that have been deallocated or exited from

Type section

Choose what type of data you want to include

- **Synchronization** › export synchronization objects
- **Handle** › export handles (e.g. notifications)
- **Trace** › `TRACE()` and `OutputDebugString()` messages

For each type of data

- **Include Stack Trace** › includes the relevant stack trace information in the export

Extra Information section

- **Detailed Report** › adds Thread ID and timestamp information to the report

- **Colour Coded Report** › for HTML reports, exports a coloured HTML table layout

  The colour scheme is not configurable.

  If you want a custom style, export a detailed XML report and process that to generate the HTML report.

File section

Specify the output destination and format:

- **File** › type the filename or **Browse** to a location
- **Format** › set whether exporting HTML or XML

  Defaults to the menu option selected, but included here to more easily export one format and then the other.

- **Encoding** › set whether UTF-16 LE, UTF-8 or ASCII encoding. By default the exported file is saved in the Windows Unicode format UTF-16 little endian. You can also save in UTF-8 and ASCII. ASCII has no byte order mark at the start of the file.
- **OK** › exports the session data

  Check the **overwrite existing file** option if you want to be warned about overwrites.

3.16.2.1 XML Export Tags

This section describes the XML tags used to export session data from Thread Validator.

Application and program details

An exported XML file starts with a few details about Thread Validator and the target program:

```
<XML>
   <VALIDATORINFO>Thread Validator information online</VALIDATORINFO>
   <VALIDATOR>Thread Validator name</VALIDATOR>
   <VALIDATORVERSION>Version</VALIDATORVERSION>
   <VALIDATORDATE>Build date</VALIDATORDATE>
   <VALIDATORTIME>Build time</VALIDATORTIME>
   <TITLE>Target program name</TITLE>
   <EXITCODE>Program exit status code and description - if collected</EXITCODE>
```

Thread and Lock information

Thread and lock information is listing the following tag pairs:

```
<LOCK_DATA>...</LOCK_DATA>
```

Example lock data might be as follows:

```
<LOCK>
 <CRITICAL_SECTION>0x0b22194c</CRITICAL_SECTION>
 <ENTERED>388</ENTERED>
 <RECURSE>0</RECURSE>
 <CONTENDED>0</CONTENDED>
 <CONTENDRATIO>0.00%</CONTENDRATIO>
 <SEQUENCE>7,746,320</SEQUENCE>
 <THREAD>9632</THREAD>
 <STATUS>StateWait(DelayExecution)</STATUS>
 <CALLSTACK>
  <ENTRY>0x620c4667 mfc90ud.dll  CCriticalSection::Lock : [f:\dd\vctools\vc7libs\ship\atlmfc
  <ENTRY>0x00405551 nativeExample.exe  CTeststakView::threadProc1 : [c:\program files (x86)\
  <ENTRY>0x1004bb09 svlthreadvalidatorstub.dll  svl_setIgnoreGetProcAddress</ENTRY>
  <ENTRY>0x757c38ef KERNEL32.DLL  BaseThreadInitThunk</ENTRY>
  <ENTRY>0x77275dde ntdll.dll  RtlUnicodeStringToInteger</ENTRY>
  <ENTRY>0x77275dde ntdll.dll  RtlUnicodeStringToInteger</ENTRY>
 </CALLSTACK>
 <LOCATION>0x620c466c : f:\dd\vctools\vc7libs\ship\atlmfc\include\afxmt.inl Line 81</LOCATIO
</LOCK>
```

Lock data may contain the following:

- `<OBJECT>Handle of waited object</OBJECT>`
- `<TIMEOUT>Infinite</TIMEOUT>`
- `<TIMEOUT>Timeout in milliseconds</TIMEOUT>`
- `<WAITING>Number of waiting processor cycles</WAITING>`
- `<THREAD>Thread id</THREAD>`
- `<STATUS>Thread status</STATUS>`
- `<LOCATION>Source code line where object was allocated, locked or waiting</LOCATION>`
- `<CALLSTACK>List of <ENTRY></ENTRY> tag pairs</CALLSTACK>`
- `<ENTRY>Callstack entry</ENTRY>`
- `<NUM_HANDLES>Number of handles waited for a multiple wait</NUM_HANDLES>`
- `<OBJECTS>List of <OBJ></OBJ> tag pairs</OBJECTS>`
- `<OBJ>Object handle</OBJ>`
- `<CRITICAL_SECTION>Critical section address</CRITICAL_SECTION>`
- `<ENTERED>Entered count</ENTERED>`
- `<RECURSE>Recursion count</RECURSE>`
- `<CONTENDED>Contended count</CONTENDED>`
- `<CONTENDRATIO>Contention ratio (contention vs entered)</CONTENDRATIO>`
- `<THREAD_WAITING>Number of waiting threads</THREAD_WAITING>`

Not all of these tags will appear for a given data item in a session. Some of them only appear when certain data items are monitored using Thread Validator.

Depending on how you use Thread Validator you may in fact never see some of these tags.

All hexadecimal numbers will have leading zeros, e.g. `0x620c4667`.

Allocation types

Allocated/entered, deallocated/exited objects are listed in one of two containers

```
<ALLOCATED>...</ALLOCATED>
<FREE>...</FREE>
```

Allocation events

The next level of tags are shown in optional <EVENT> tags.

```
<EVENT>...</EVENT>
```

Here's an example for a detailed report. The non-detailed report excludes thread and timestamp entries:

```
<EVENT>
 <ID>82</ID>
 <THREAD>7068</THREAD>
 <TIME>18508531</TIME>
 <File>c:\program files (x86)\software verification\c++ thread validator\examples\nativeexam
 <Line>226</Line>
 <Synchronization>CriticalSection</Synchronization>
 <STACKTRACE>
  <SYMBOL>0x00402d1d nativeExample.exe  CTeststakView::CTeststakView : [c:\program files (x8
  <SYMBOL>0x004025a7 nativeExample.exe  CTeststakView::CreateObject : [c:\program files (x86
  <SYMBOL>0x6242b6a7 mfc90ud.dll  CRuntimeClass::CreateObject : [f:\dd\vctools\vc7libs\ship\
  <SYMBOL>0x620d3ece mfc90ud.dll  CFrameWnd::CreateView : [f:\dd\vctools\vc7libs\ship\atlmfc
  <SYMBOL>0x620d4003 mfc90ud.dll  CFrameWnd::OnCreateClient : [f:\dd\vctools\vc7libs\ship\at
  <SYMBOL>0x620d40c4 mfc90ud.dll  CFrameWnd::OnCreateHelper : [f:\dd\vctools\vc7libs\ship\at
  <SYMBOL>0x620d4073 mfc90ud.dll  CFrameWnd::OnCreate : [f:\dd\vctools\vc7libs\ship\atlmfc\s
  <SYMBOL>0x620c9337 mfc90ud.dll  CWnd::OnWndMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mf
 </STACKTRACE>
</EVENT>
```

For allocated handles rather than synchronization objects you may see handle information instead

```
<Handle>0x6FCB0000</Handle>
<HandleType>Notification</HandleType>
```

Event data may have the following, in addition to the <STACKTRACE> tags in the next section:

- <ID>the sequence number of the event in the recorded history of all events <ID>
- <File>the source file location of the allocation event <File>
- <Line>the source line number in the file <Line>
- <Type>a string indicating the datatype of the allocated object, if known <Type>
- <Handle>the value of the allocated handle <Handle>
- <HandleType>the type of handle allocated, if known <HandleType>
- <Synchronization>the type of synchronization object <Synchronization>
- <THREAD>the id of the thread in which the allocation was made (shown as a decimal value) <TH

- `<TIME>the timestamp of the allocation <TIME>`
  This is a relative 'ticker' time rather than an absolute time, and is not measured in hours/mins/secs.
- `<ReportType>type of trace message <ReportType>`
- `<Message>a message from a TRACE() macro or OutputDebugString() <Message>`

Not all of these event tags will appear for a given allocation in a session. Some of them only appear when certain data items are monitored using Thread Validator.

As with the lock data you may never see some of these tags, depending on how you use Thread Validator .

Any hexadecimal numbers will have a leading `0x....`

Stacktrace tags

The stacktrace for the event is defined in the tags

```
<STACKTRACE>...</STACKTRACE>
```

In the stacktrace are a number of symbols

```
<SYMBOL>symbol data</SYMBOL>
```

The symbol data includes:

- hexadecimal address
- dll/exe name terminated by a semi-colon
- function name
- filename and line number in square brackets, if known

Example (from the XML fragment above):

```
<SYMBOL>0x00402d1d nativeExample.exe  CTeststakView::CTeststakView : [c:\program files (x8
```

Error conditions

The presence of any of the following tags indicate an error condition:

- `<DEADLOCKED/>`
- `<POTENTIAL_DEADLOCK/>`
- `<EXIT_OUT_OF_ORDER/>`
- `<NOT_ENTERED/>`
- `<MISC_ERROR/>`

3.17 Starting your target program

Starting options

There are seven ways to start a target program and have Thread Validator collect data from it.

- Launch your program in a specified directory, with as many command line arguments as you want
- Inject Thread Validator into an already running program
- Wait until a specific program starts to run before attaching to it - e.g. for programs started as an OLE server
- Monitor a service
- Monitor IIS and ISAPI
- Use the Native API to start Thread Validator from code that you control
- Start Thread Validator from the command line, allowing you to automate your use of Thread Validator

Modules without PDB files and without MAP files

For your application to be processed for thread data, each module to be monitored must have a PDB file with debug data, or a MAP file with line number data.

Use the Debug DLLs dialog to see whether debug information was not found for any modules, and check the Diagnostics tab for failure messages.

3.17.1 Launching the program

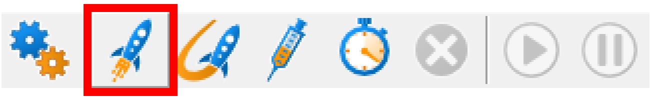
Launching the application

Having Thread Validator launch your program is the most common way to start up

When you're ready to start running a target program:

File menu › **Start Application...** › Shows the launch program wizard or dialog below

Or click on the launch icon on the session toolbar.



Or use the shortcut:

F4 Start application

➡ You can easily re-launch the most recently run program.

User interface mode

There are two interface modes used while starting a program

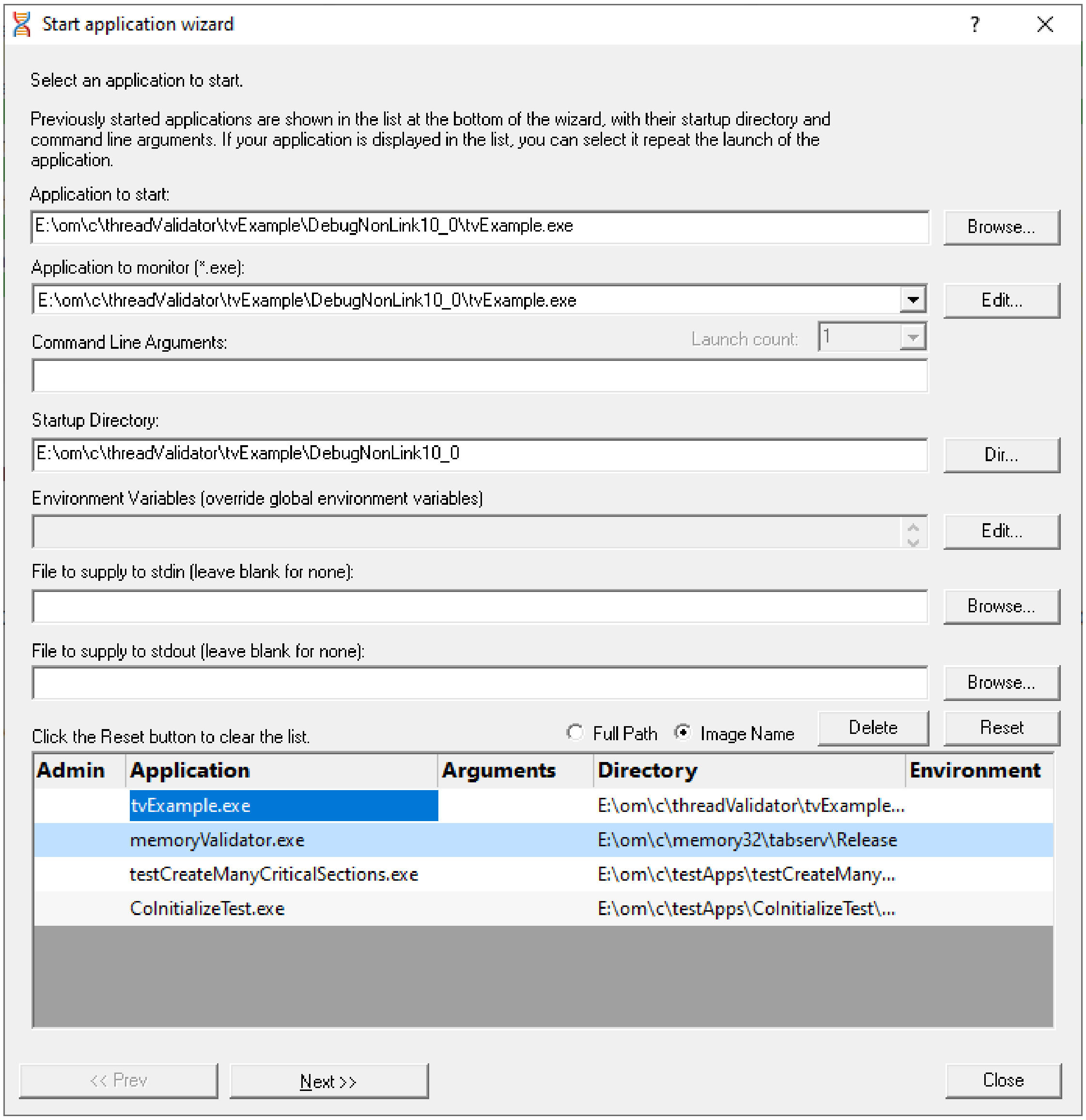
- **Wizard** mode guides you through the tasks in a linear fashion
- **Dialog** mode has all options contained in a single dialog

All the options are the same - just in different places.

In this section we'll cover the Wizard mode first and the Dialog mode later.

The start application wizard

On first use, the wizard appears with fields cleared, but here's an example with a few fields set:



Start application wizard

Select an application to start.

Previously started applications are shown in the list at the bottom of the wizard, with their startup directory and command line arguments. If your application is displayed in the list, you can select it repeat the launch of the application.

Application to start:
E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe

Application to monitor (*.exe):
E:\om\c\threadValidator\tvExample\DebugNonLink10_0\tvExample.exe

Command Line Arguments:

Launch count: 1

Startup Directory:
E:\om\c\threadValidator\tvExample\DebugNonLink10_0

Environment Variables (override global environment variables)

File to supply to stdin (leave blank for none):

File to supply to stdout (leave blank for none):

Click the Reset button to clear the list. ○ Full Path ● Image Name

Admin	Application	Arguments	Directory	Environment
	tvExample.exe		E:\om\c\threadValidator\tvExample...	
	memoryValidator.exe		E:\om\c\memory32\tabserv\Release	
	testCreateManyCriticalSections.exe		E:\om\c\testApps\testCreateMany...	
	CoInitializeTest.exe		E:\om\c\testApps\CoInitializeTest\...	

Enter the details for your program, or if you want to run a previous program select it from the application list to repopulate the details.

After entering the details click **Next >>** for the next page of the wizard.

Administrator privileges when launching your program

The following applies only if you did *not* start Thread Validator in administrator mode.

Anywhere you see the 🛡 icon indicates that administrator privileges will be required to proceed.

If you started Thread Validator in administrator mode, you won't see any of these warnings, and everything will behave as normal.

Page 1: Entering details

- **Application to start** › type or **Browse** to set the program name to launch

  You can also choose a batch file and the first executable started in the batch file will be launched.

  You can also choose a powershell script and the first executable started in the powershell script will be launched.

  Manually typing a path will show red text until a valid path is entered, after which the text becomes black.

- **Application to monitor** › choose the application that actually gets monitored

  This will typically just be the program that you set to start - unless otherwise specified.

  Alternatively you can monitor another application which will get launched by the start program.

  If the start application has already been added to the Applications to Monitor settings with a default application then that default will be entered here automatically.

  Otherwise, if nothing has been set up yet, you can do it from here:

  - **Edit...** › set the child applications that can be monitored for the start program

    This uses the Applications to Monitor dialog - which is exactly equivalent to using the Applications to Monitor settings page.

  A fallback option is to start monitoring `<<Any application that is launched>>`.

  If in doubt, just use the same as the start application.

  ➡ See also: Application to Monitor settings

- **Launch Count** › when monitoring a *child* application, set its nth invocation as the one to monitor

  If the application to start is the same as the application to monitor then this is set to 1 and cannot be changed.

  This will be reset to 1 every time the Application to Monitor field selection changes.

  If in doubt, leave it set to 1.

  ➡ See also: Launch Count.

- **Command Line Arguments** › enter program arguments exactly as passed to the target program

- **Startup Directory** › enter or click **Dir...** to set the directory for the program to start in

  When setting your target program, this will default to the location of the executable

Manually typing a directory path will show red text until a valid path is entered, after which the text becomes black.

- **Environment Variables** › click **Edit...** to set any additional environment variables before your program starts

  These are managed in the Environment Variables Dialog.

- **File to supply to stdin** › optionally enter or **Browse** to set a file to be read and piped to the standard input of the application

- **File to supply to stdout** › optionally enter or **Browse** to set a file to be written with data piped from the standard output of the application

Page 1: Using details from a previous run

The list at the bottom of the wizard shows previously run programs.

Selecting an item in the list populates all the details above as used on the last run for that program.

You can still edit those details before starting.

- **Full path** › shows the full path to the executable in the list

- **Image Name** › shows the short program name without path

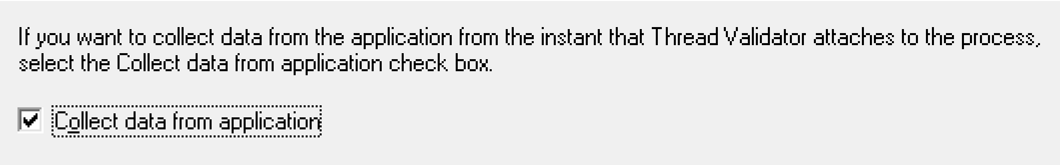
- **Delete** › removes a selected program from the list

- **Reset** › clears all details in the wizard - including the list of previously run applications below

The **Admin** column in the list of previous runs may show a 🛡 symbol to indicate a requirement for administrator privileges in order to run the program. This is automatically detected from the program's manifest.

Page 2: Data collection

- **Collect data from application** › if it's the startup threads you want to monitor, then obviously start collecting data from launch



If you want to collect data from the application from the instant that Thread Validator attaches to the process, select the Collect data from application check box.

☑ Collect data from application

➡ See the section on controlling data collection for how to turn collection on and off after launch.

- **Redirect standard output** › Controls redirection of stdout and stderr

Use this option if you want to collect the output of stdout and stderr for later analysis.

Be aware that if the output of the program under test generates a lot of data via stdout or stderr that this data will need to be stored in memory and could exhaust Thread Validator's memory.

- **Display command prompt** › Shows or hides the launched application.

  If you are collecting stdout and stderr you may not be interested in viewing the application (or the command prompt if it is a console application). This provides you the option to hide the application when it is running.

  Be aware that if you hide a command prompt you will not be able to type anything into the application.

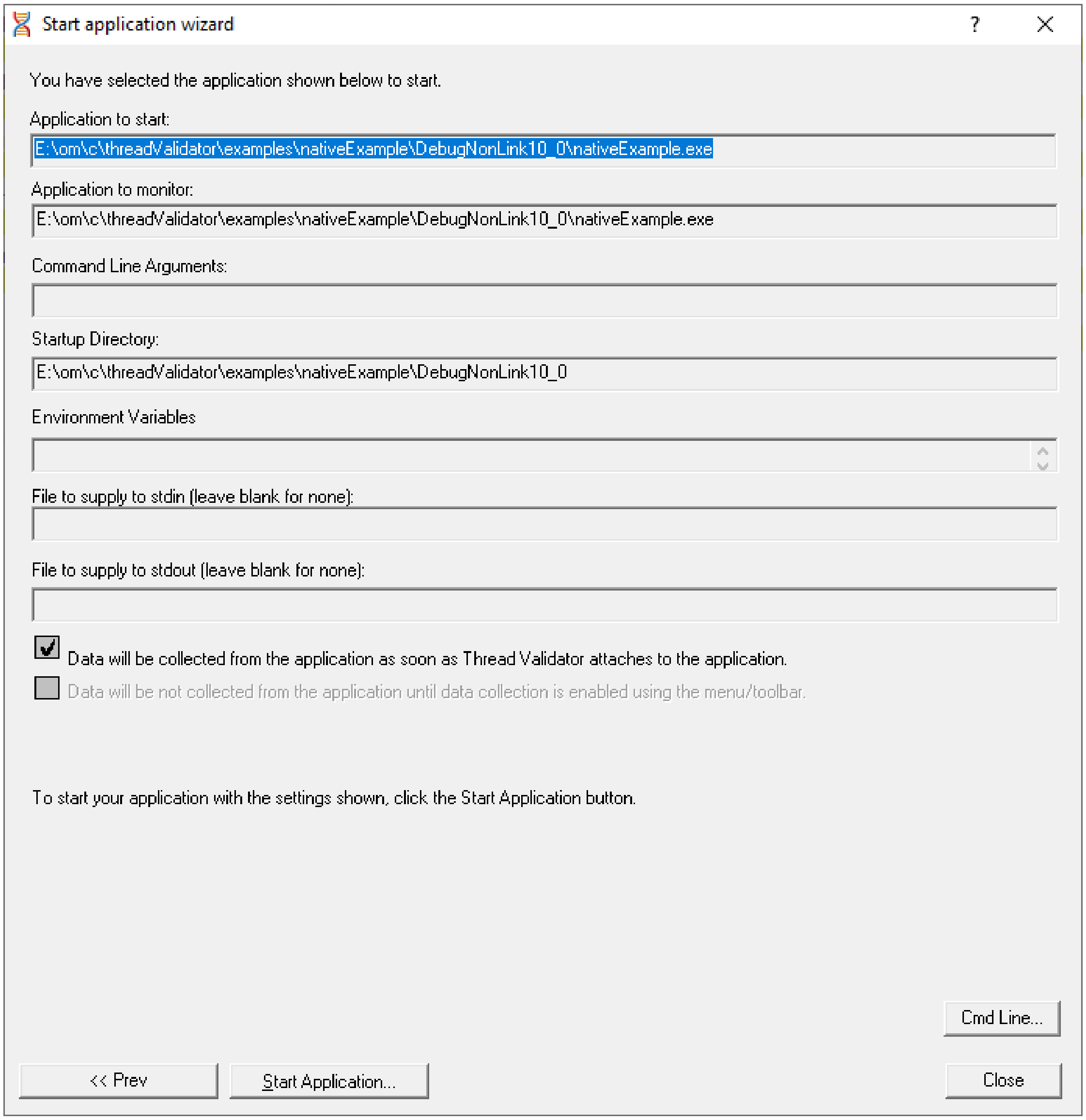
Page 3: Summary and starting your program

The last page is just a summary of the options you have chosen.

Something missing? The choice of launch method is no longer necessary and has been removed.

If you're happy with the settings, go ahead:

- **Start Application...** › start your program and attach Thread Validator to it
- **Cmd Line...** › display the command line builder



Start application wizard

You have selected the application shown below to start.

Application to start:
E:\om\c\threadValidator\examples\nativeExample\DebugNonLink10_0\nativeExample.exe

Application to monitor:
E:\om\c\threadValidator\examples\nativeExample\DebugNonLink10_0\nativeExample.exe

Command Line Arguments:

Startup Directory:
E:\om\c\threadValidator\examples\nativeExample\DebugNonLink10_0

Environment Variables

File to supply to stdin (leave blank for none):

File to supply to stdout (leave blank for none):

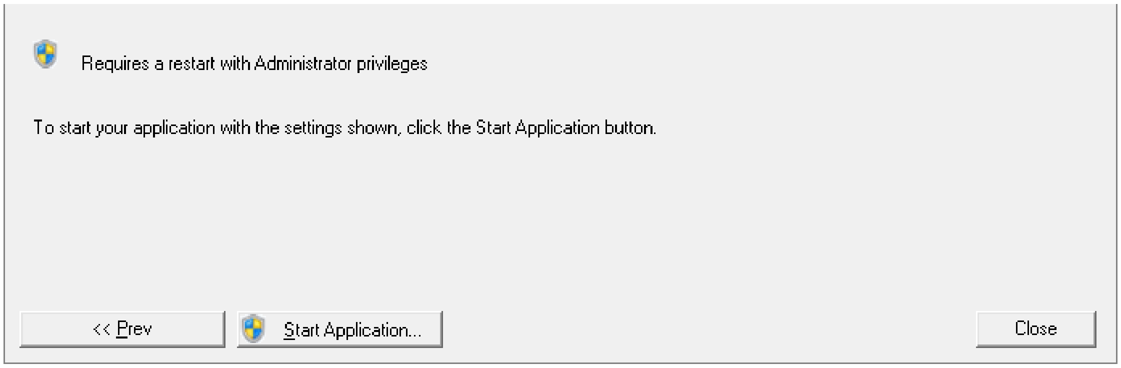
☑ Data will be collected from the application as soon as Thread Validator attaches to the application.

□ Data will be not collected from the application until data collection is enabled using the menu/toolbar.

To start your application with the settings shown, click the Start Application button.

Administrator privileges in wizard mode

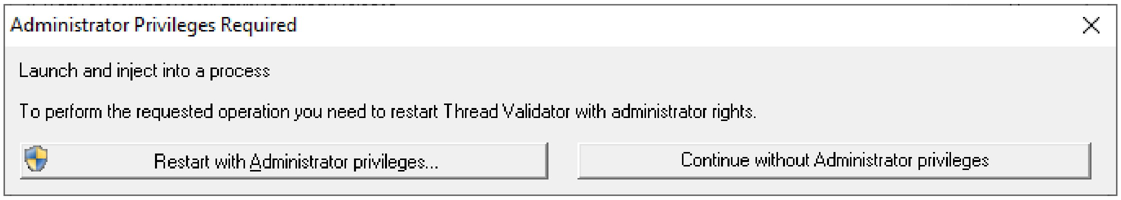
If administrator privileges are required you'll be reminded of the need to restart here:



Requires a restart with Administrator privileges

To start your application with the settings shown, click the Start Application button.

- **Start Application...** › shows the Administrator Privileges Required confirmation dialog before restarting



Administrator Privileges Required

Launch and inject into a process

To perform the requested operation you need to restart Thread Validator with administrator rights.

Restart with Administrator privileges...
Continue without Administrator privileges

Dialog mode

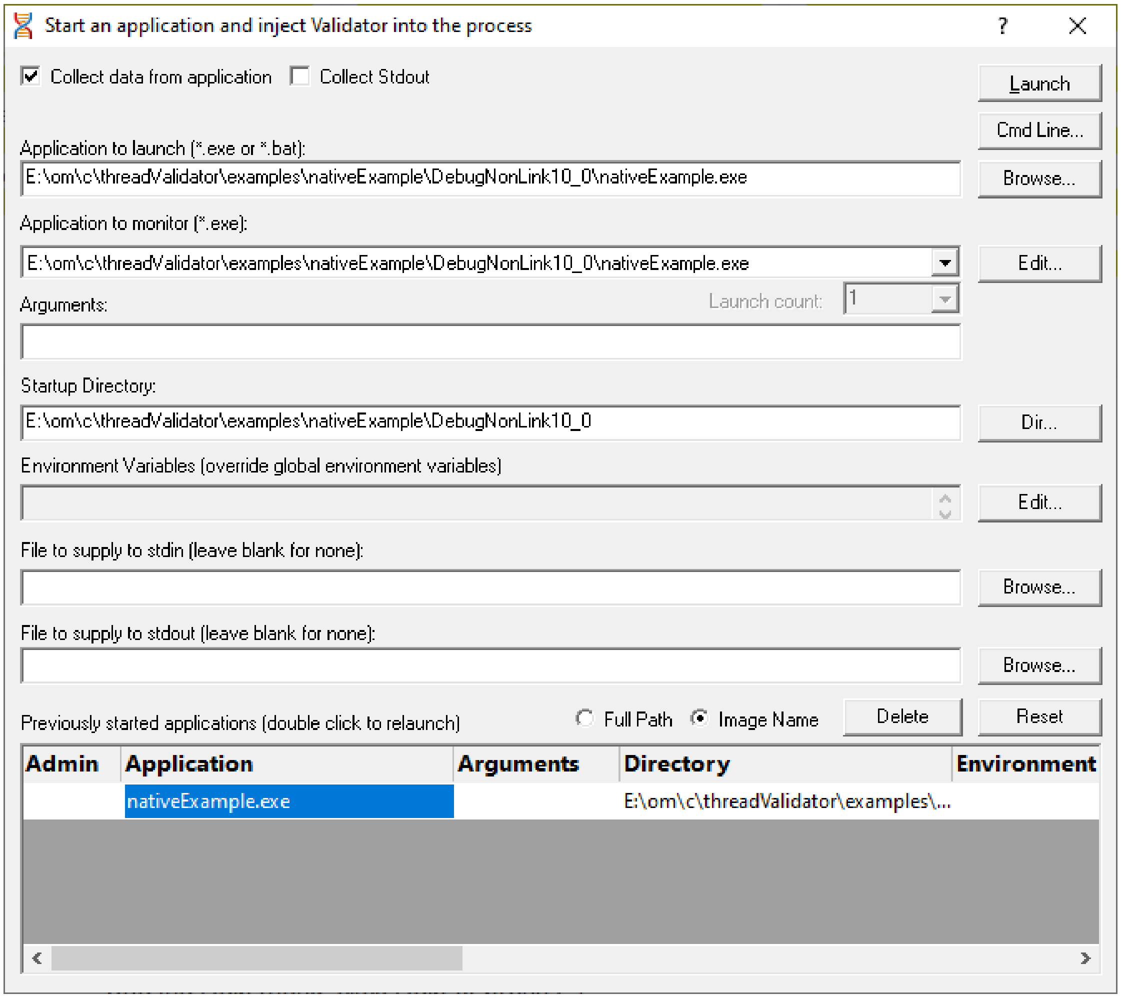
In Dialog mode, all the settings are in one dialog which looks very much like the first page of the launch wizard above.

At the top are the options to collect line times and to start collecting data immediately.

- **Launch** › start your program and attach Thread Validator to it

- **Cmd Line...** › display the command line builder

Double clicking a program in the list will also start it immediately.


Start an application and inject Validator into the process

☑ Collect data from application ☐ Collect Stdout

Application to launch (*.exe or *.bat):
E:\om\c\threadValidator\examples\nativeExample\DebugNonLink10_0\nativeExample.exe

Application to monitor (*.exe):
E:\om\c\threadValidator\examples\nativeExample\DebugNonLink10_0\nativeExample.exe

Arguments:

Launch count: 1

Startup Directory:
E:\om\c\threadValidator\examples\nativeExample\DebugNonLink10_0

Environment Variables (override global environment variables)

File to supply to stdin (leave blank for none):

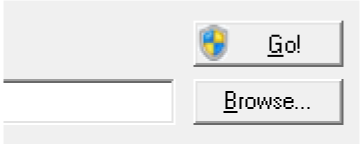
File to supply to stdout (leave blank for none):

Previously started applications (double click to relaunch) ○ Full Path ● Image Name

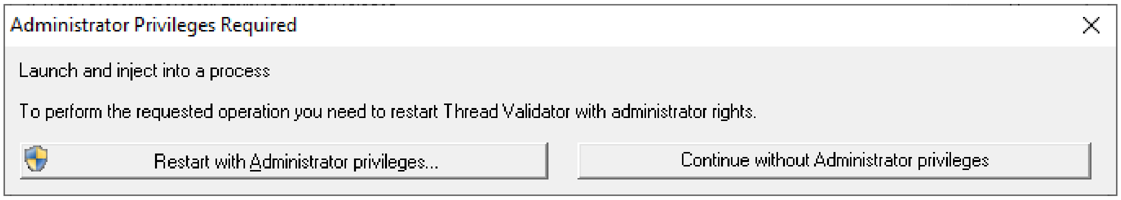
Admin	Application	Arguments	Directory	Environment
	nativeExample.exe		E:\om\c\threadValidator\examples\...	

Administrator privileges in dialog mode

If administrator privileges are required, the Launch button will show the privileges icon reminding you of the need to restart.



- **Launch** ❯ shows the Administrator Privileges Required confirmation dialog before restarting.


Administrator Privileges Required

Launch and inject into a process

To perform the requested operation you need to restart Thread Validator with administrator rights.

Restart with Administrator privileges...
Continue without Administrator privileges

If you started Thread Validator in administrator mode, you won't see any of these warnings, and everything will behave as normal.

How do I use Application to Monitor and Launch Count?

The three fields **Application to Start**, **Application to Monitor** and **Launch Count** work together to control which application actually gets monitored by Thread Validator.

By example, let's say we have a program **P**.

In the simplest case, set options as follows:

- *start* **P**
- *monitor* **P**
- the Launch Count defaults to **1** and cannot be changed.

If **P** launches an application and you just want to monitor whatever that is:

- *start* **P**
- *monitor* `<<Any application that is launched>>`
- leave the Launch Count at **1**

If **P** launches an application **A** and maybe others as well, and you specifically want to monitor only **A** as it's launched:

- use the Application to Monitor settings to add a definition for P and child applications **A**
- *start* **P**
- *monitor* **A**
- leave the Launch Count at **1**

If **P** launches an application **A** many times and you specifically want to monitor the third invocation:

- use the Application to Monitor settings to add a definition for P and child applications **A**
- *start* **P**
- *monitor* **A**
- set the Launch Count to **3**

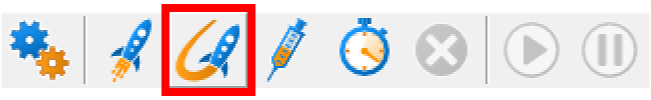
3.17.2 Re-Launching the program

Re-launching the application

It's very easy to start another session using the most recently run program and settings:

☰ **File** menu › **Re-Start Application...** › starts the most recently launched program

Or click on the re-launch icon on the session toolbar.



Or use the shortcut:

F5 Re-start application

No wizards or dialogs appear, so be ready for the application to start right away.

➡ For troubleshooting launch problems, see *Why might Inject or Launch fail?* in the general questions .

There is no difference between wizard and dialog interface mode when re-launching.

3.17.3 Injecting into a running program

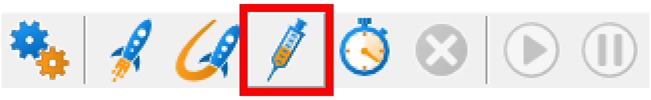
Injecting into a running program

Thread Validator attaches to a running process by injecting the stub into the process so it can start collecting data.

Choose one of these methods of starting the injection:

File menu › **Inject...** › shows the Attach to Running Process wizard or dialog below

Or click on the Inject icon on the session toolbar.



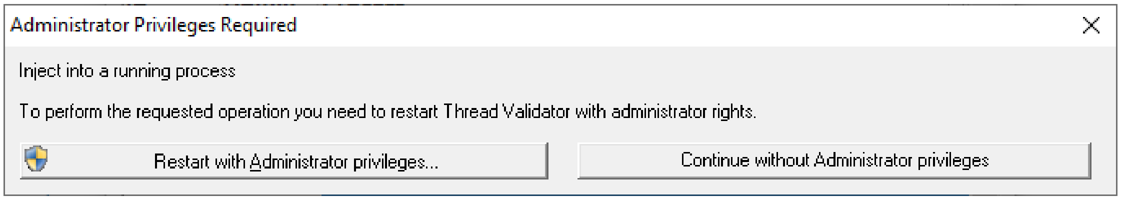
Or use the shortcut

F3 Inject into running application

Administrator privileges

The following applies only if you did *not* start Thread Validator in administrator mode.

When choosing the inject method described in this topic, a restart of Thread Validator with administrator privileges will be required to proceed.


Administrator Privileges Required

Inject into a running process

To perform the requested operation you need to restart Thread Validator with administrator rights.

Restart with Administrator privileges...
Continue without Administrator privileges

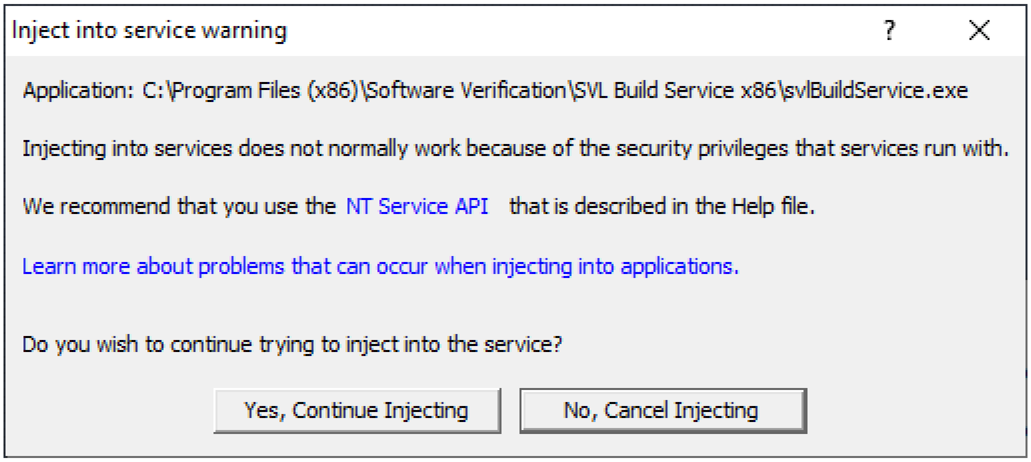
Injecting into a service?

If your process is a service, Thread Validator won't be able to attach to it.

Services can't have process handles opened by third party applications, even with Administrator privileges.

In order to work with services, you can use the NT service API and monitor the service

You may see this warning dialog when trying to inject into a service:


Inject into service warning

Application: C:\Program Files (x86)\Software Verification\SVL Build Service x86\svlBuildService.exe

Injecting into services does not normally work because of the security privileges that services run with.

We recommend that you use the NT Service API that is described in the Help file.

Learn more about problems that can occur when injecting into applications.

Do you wish to continue trying to inject into the service?

User interface mode

There are two interface modes used while starting a program

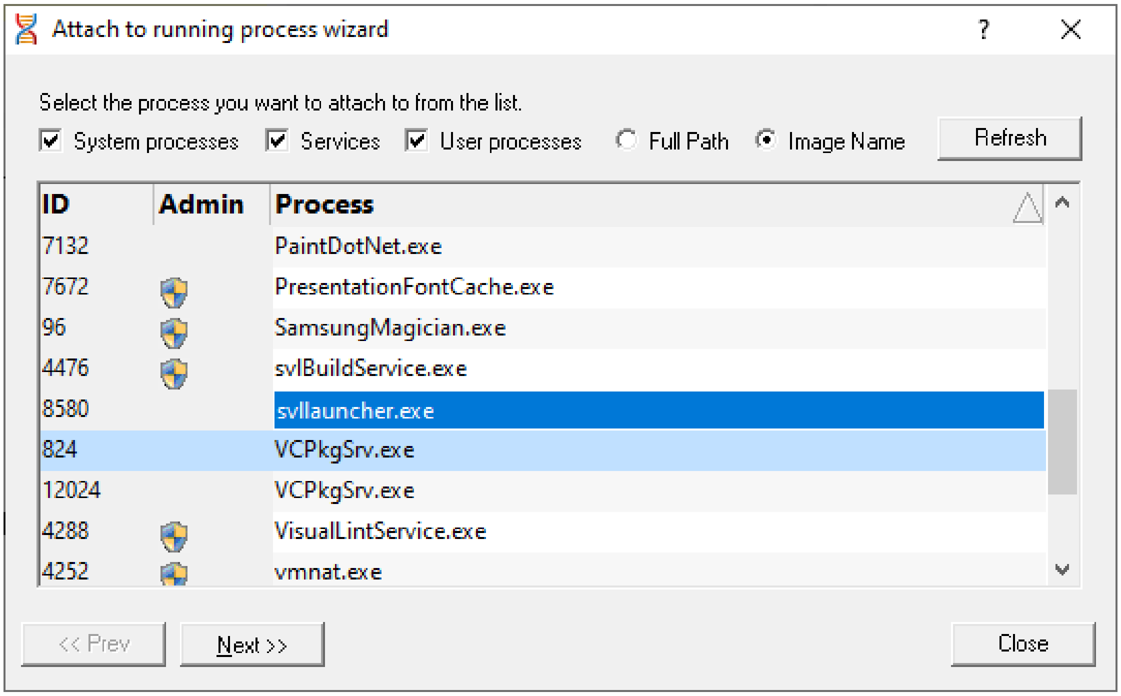
- **Wizard** mode guides you through the tasks in a linear fashion
- **Dialog** mode has all options contained in a single dialog

All the options are the same - just in slightly different places

In this section we'll cover the Wizard mode first and the Dialog mode later.

The attach to running process wizard

The first page of the wizard shows a list of running system and user processes.



Select the process you want to attach to from the list.

☑ System processes ☑ Services ☑ User processes ○ Full Path ● Image Name

ID	Admin	Process
7132		PaintDotNet.exe
7672	🛡	PresentationFontCache.exe
96	🛡	SamsungMagician.exe
4476	🛡	svlBuildService.exe
8580		svllauncher.exe
824		VCPkgSrv.exe
12024		VCPkgSrv.exe
4288	🛡	VisualLintService.exe
4252	🛡	vmnat.exe

The list shows the following information:

- **ID** › The process ID
- **Admin** › may show a 🛡 symbol to indicate a requirement for administrator privileges in order to run the program.

  This requirement is automatically detected from the manifest for the process.
- **Process** › The process executable name

Choose a process before continuing:

- **Next >>** › move to the next page of the wizard

  The button will show the 🛡 symbol if you have selected a process which requires elevated privileges to run.

Page 1: Choosing the process

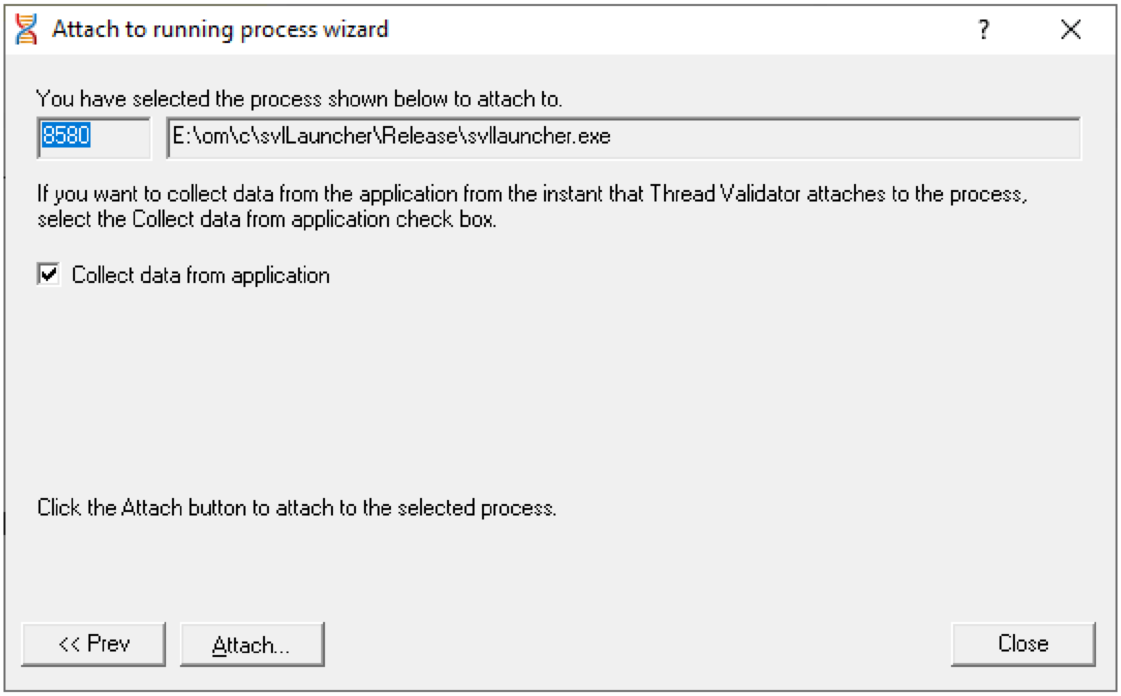
- **System processes / Services / User processes** › show any or all of services and system or user processes in the list
- **Full path** › shows the full path to the process executable in the list
- **Image Name** › shows the short program name without path

- **Refresh ❯** update the list with currently running processes

Clicking on the headers of the list will sort them by ID or by name using the full name or short name, depending on what's displayed.

Page 2: Data collection

The second page controls when to start collecting information.



You have selected the process shown below to attach to.

8580 E:\om\c\svlLauncher\Release\svllauncher.exe

If you want to collect data from the application from the instant that Thread Validator attaches to the process, select the Collect data from application check box.

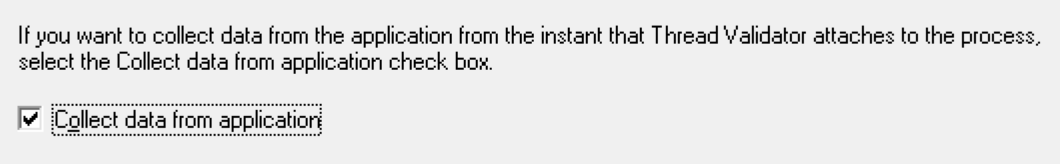
☑ Collect data from application

Click the Attach button to attach to the selected process.

Depending on your application, and what you want to test, you may want to start collecting data as soon as injection happens, or do it later.

If your program has a complex start-up procedure, initialising lots of data, it may be much faster *not* to collect data until the program has launched.

- **Collect data from application ❯** if it's the startup Thread you want to monitor, then obviously start collecting data from launch



If you want to collect data from the application from the instant that Thread Validator attaches to the process, select the Collect data from application check box.

☑ Collect data from application

➡ See the section on controlling data collection for how to turn collection on and off after launch.

Summary and starting your program

The second page also confirms at the top which process you have selected to inject into, and prompts you to attach:

- **Attach...** › injects Thread Validator into the specified process, showing progress status

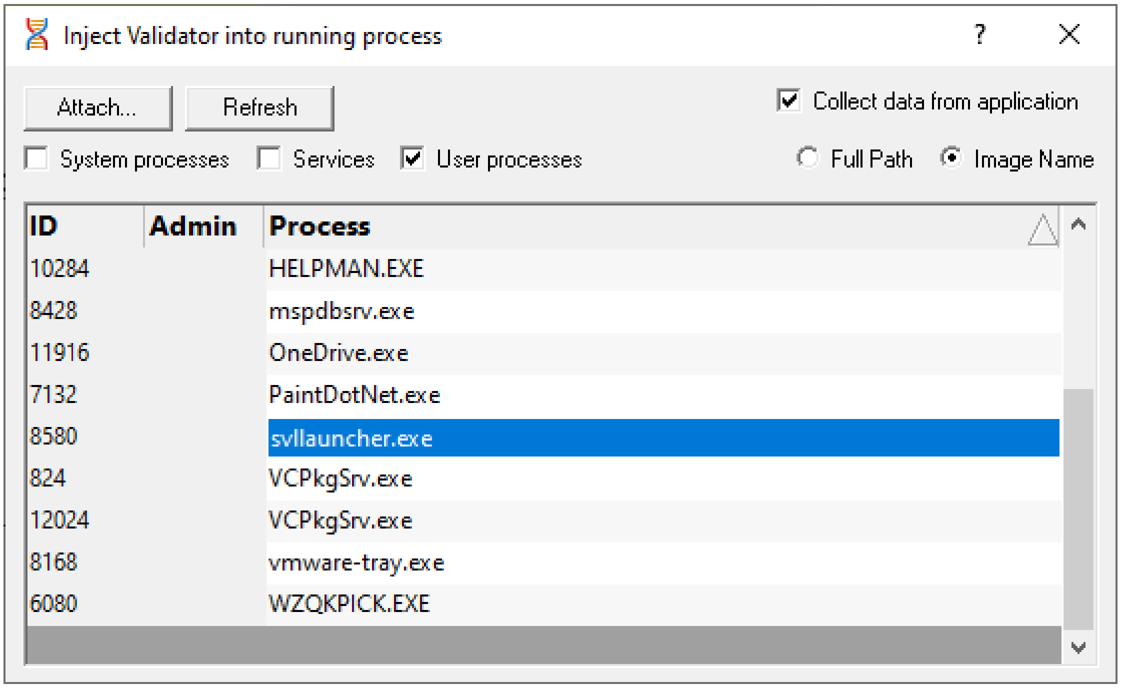
  The button will show the 🛡 symbol if you have selected a process which requires elevated privileges to run.

➡ In the general questions see *Why might Inject or Launch fail?* for troubleshooting launch problems.

Dialog mode

In Dialog mode, all the settings are in one dialog but which still looks similar to the first page of the wizard above.

The option of when to start collecting data, and whether to collect function and line times is at the top, as is the **Attach...** button itself.



☑ Collect data from application

□ System processes □ Services ☑ User processes ○ Full Path ● Image Name

ID	Admin	Process
10284		HELPMAN.EXE
8428		mspdbsrv.exe
11916		OneDrive.exe
7132		PaintDotNet.exe
8580		svllauncher.exe
824		VCPkgSrv.exe
12024		VCPkgSrv.exe
8168		vmware-tray.exe
6080		WZQKPICK.EXE

3.17.4 Waiting for a program

Waiting for a program

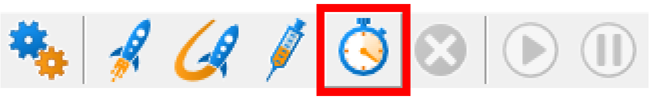
Waiting for a program is essentially the same as injection except that instead of injecting into a running program, Thread Validator watches for the process starting up and *then* injects.

If the process is a service, Thread Validator won't be able to attach to it as services can't have process handles opened by third party applications, even with Administrator privileges.

Wait for Application can be accessed by one of these methods:

File menu › **Wait for Application...** › shows the Wait for application wizard or dialog below

Or click on the Wait (timer) icon on the session toolbar.



Or use the shortcut:

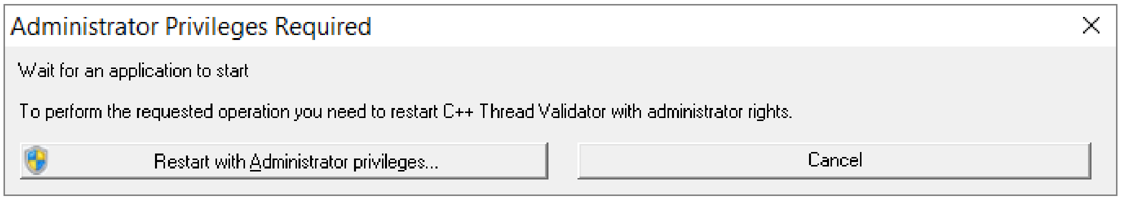
F2 Wait for application

Administrator privileges

The following applies only if you did *not* start Thread Validator in administrator mode.

If the application you want to wait for is running with Administrator privileges, Thread Validator will also need to run with Administrator privileges.

When choosing the 'wait for program' method described in this topic, a restart of Thread Validator with administrator privileges will be required to proceed.



Administrator Privileges Required

Wait for an application to start

To perform the requested operation you need to restart C++ Thread Validator with administrator rights.

Waiting for a service?

If your process is a service, Thread Validator won't be able to attach to it.

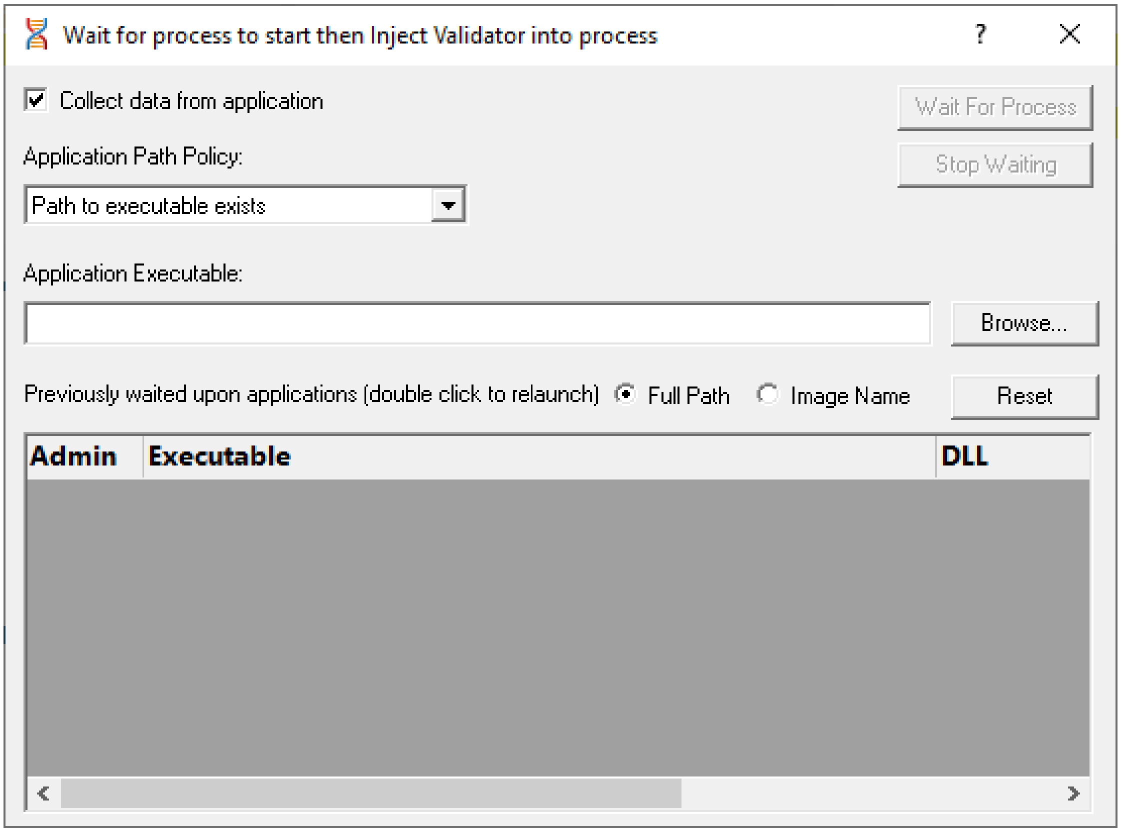
Services can't have process handles opened by third party applications, even with Administrator privileges.

In order to work with services, you can use the NT service API and monitor the service

The wait for application dialog

The wait for application dialog lets you specify the application or choose one that you've waited for previously.

If you choose a previously waited for application the Application Path Policy will be set to the same value as used with that application.


Wait for process to start then Inject Validator into process
?
☑ Collect data from application
Wait For Process
Stop Waiting
Application Path Policy:
Path to executable exists
Application Executable:
Browse...
Previously waited upon applications (double click to relaunch)
● Full Path
○ Image Name
Reset
Admin
Executable
DLL

Data collection

- **Collect data from application ›** do want to collect data from the instant you attach to the application?

Depending on your application, and what you want to validate, you may want to start collecting data as soon as injection has happened, or do it later.

If your program has a complex start-up procedure, initialising lots of data, it may be much faster *not* to collect data until the program has launched.

If it's the startup procedure you want to validate, obviously start collecting data from launch.

➡ See the section on controlling data collection for how to turn collection on and off after launch.

Specifying the application

- **Application Path Policy ›** specify how the specified executable is treated

- ○ Path to executable exists › the executable will be checked that it exists and is appropriate for Thread Validator to work with
- ○ Path to executable is created dynamically › most pre-wait checks are not performed - use this if the path the executable is on does not exist at the time you start waiting for the process to start

- **Application to wait for** › type or **Browse** to set the application name to launch

Alternatively, select a previously waited for application from the list.

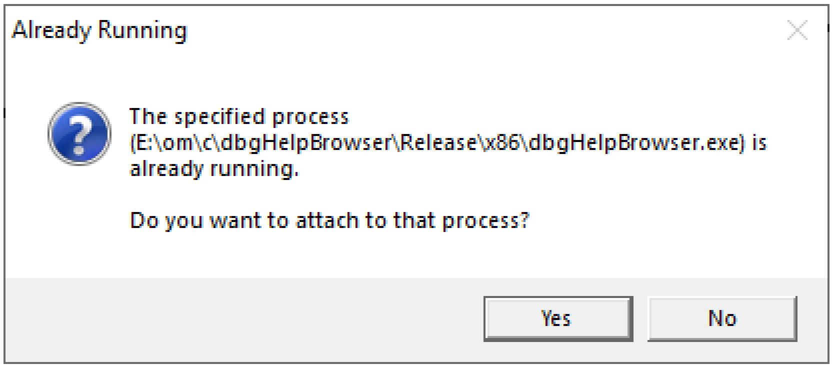
- **Full path** › shows the full path to the process executable in the list
- **Image Name** › shows the short program name without path
- **Reset** › clears the list

Waiting for an application

- **Wait For Process** › start waiting for the specified executable to start
- **Stop Waiting** › stop waiting for the specified executable to start

What could go wrong?

The program you're waiting for might already be running, in which case you'll be given the option to cancel or attach to the existing process:



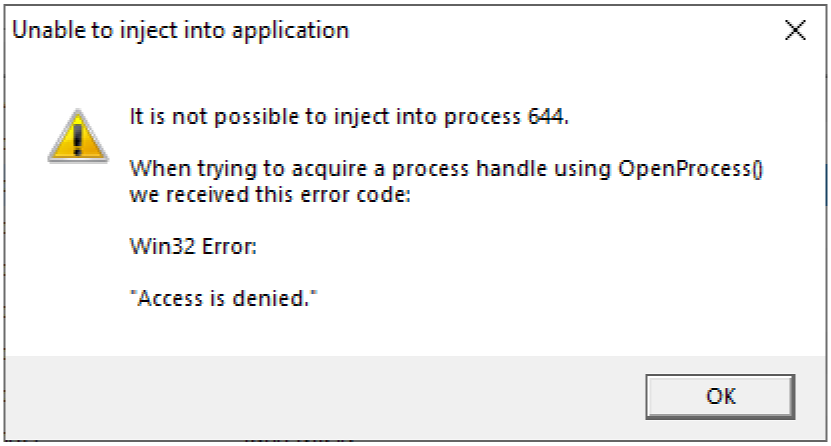
Already Running

The specified process (E:\om\c\dbgHelpBrowser\Release\x86\dbgHelpBrowser.exe) is already running.

Do you want to attach to that process?

Timing issues are inherit with native injecting into a program as it starts up.

This could cause the injection to fail in unpredictable ways and you may see dialogs like that below:



Unable to inject into application

It is not possible to inject into process 644.

When trying to acquire a process handle using OpenProcess() we received this error code:

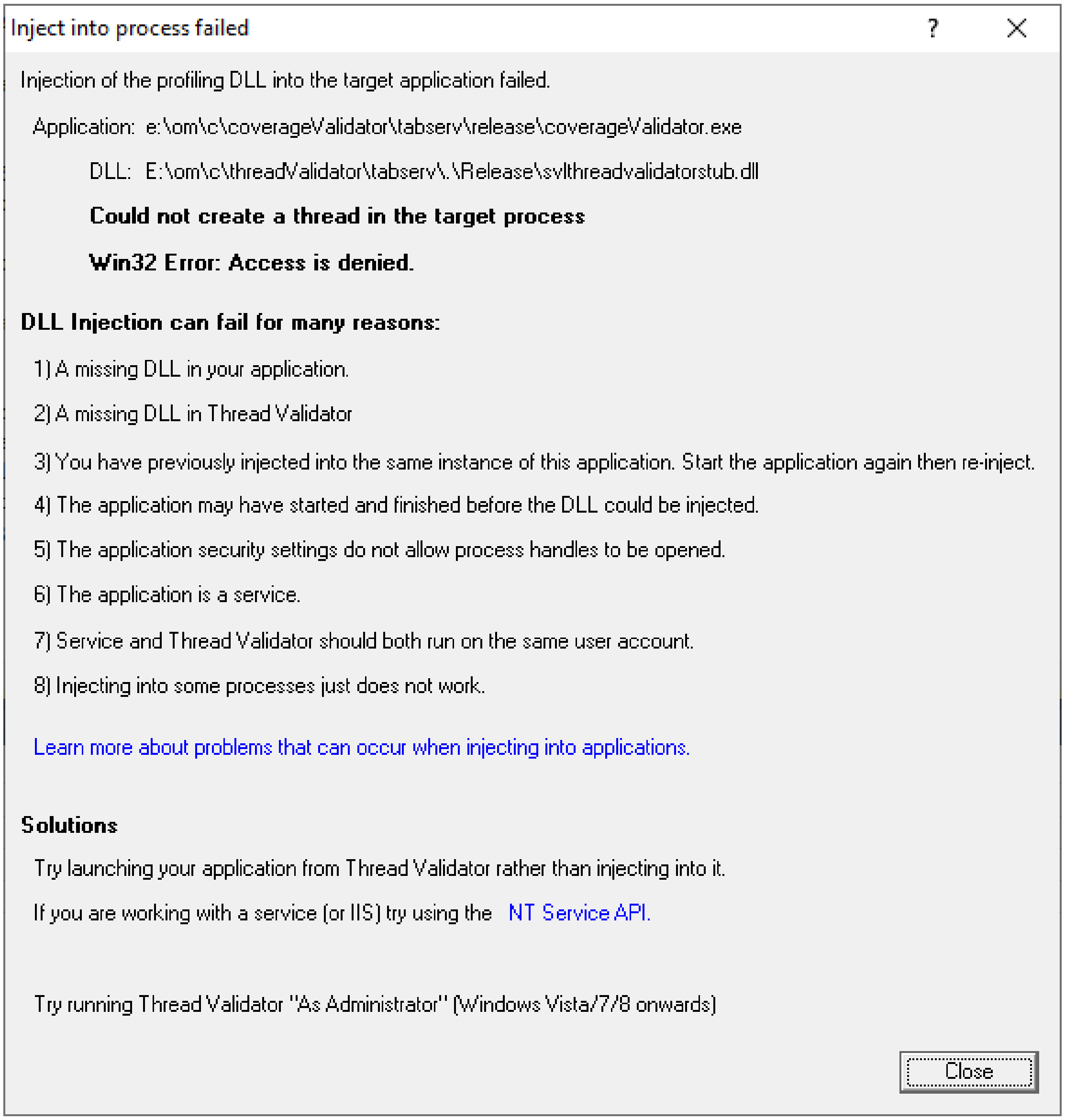
Win32 Error:

"Access is denied."

One case when this dialog can occur is if the program needs to run at an elevated privilege and is waiting for the user to give permission via the UAC dialog.

Injection may fail for different reasons and you might see the following information dialog showing:

- messages relating to the specific failure
- a selection of reasons why failure might be occurring
- some possible solutions to the problem



Injection of the profiling DLL into the target application failed.

Application: e:\om\c\coverageValidator\tabserv\release\coverageValidator.exe

DLL: E:\om\c\threadValidator\tabserv\.\Release\svlthreadvalidatorstub.dll

Could not create a thread in the target process

Win32 Error: Access is denied.

DLL Injection can fail for many reasons:

1) A missing DLL in your application.

2) A missing DLL in Thread Validator

3) You have previously injected into the same instance of this application. Start the application again then re-inject.

4) The application may have started and finished before the DLL could be injected.

5) The application security settings do not allow process handles to be opened.

6) The application is a service.

7) Service and Thread Validator should both run on the same user account.

8) Injecting into some processes just does not work.

Learn more about problems that can occur when injecting into applications.

Solutions

Try launching your application from Thread Validator rather than injecting into it.

If you are working with a service (or IIS) try using the NT Service API.

Try running Thread Validator "As Administrator" (Windows Vista/7/8 onwards)

Sometimes retrying a few times might catch a better moment for attaching to the process.

➡ In the general questions see *Why might Inject or Launch fail?* for troubleshooting launch problems.

3.17.5 Monitor a service

Monitoring a service

Monitoring a service works for:

- Native services
- .Net services
- Mixed mode services.

Native Services

If you're working with native services you must use the NT Service API in your service *as well as* using the Monitor a service method below.

.Net Services

Thread Validator won't attach until some .Net code is executed.

If there is native code being called prior to the .Net code, Thread Validator won't monitor that code, only the native code called after the first .Net code that is called.

To monitor any native code called *prior* to your .Net code, use the NT Service API.

Mixed Mode .Net Services

You don't need to use the NT Services API.

If you are working with a .Net service that loads native DLLs, or a mixed mode service, Thread Validator will recognize the service when the .Net runtime starts executing the .Net service main.

When working with Thread Validator and services, you still start the service the way you normally do - e.g. with the service control manager.

The code that you have embedded into your service then contacts Thread Validator, which you should have running before starting the service.

To start monitoring a service:

File menu › **Monitor a service...** › shows the Monitor a service dialog below

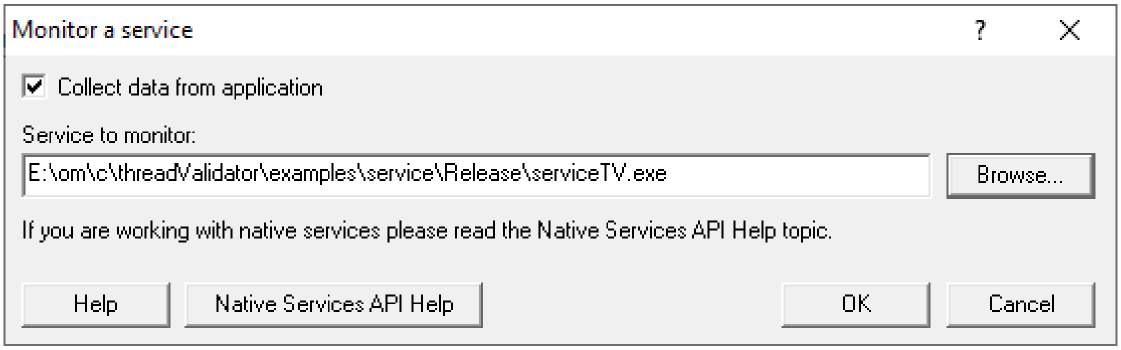
or use the shortcut

F6 Monitor a service

The monitor a service dialog

First ensure the service is installed, but not running.

Set the service to monitor, choose whether to start collecting data right away, and click OK.


Monitor a service

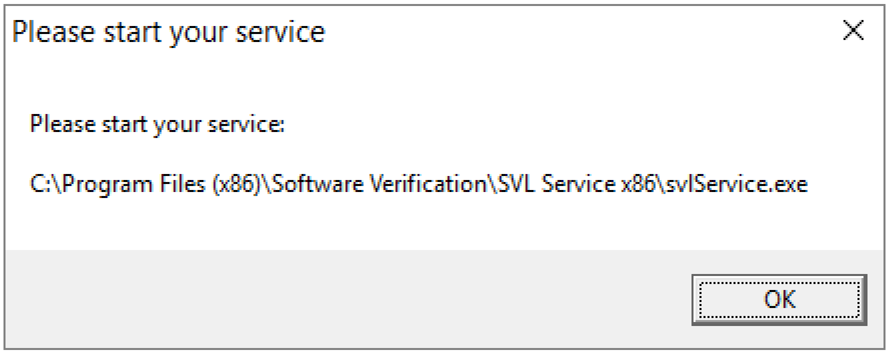
☑ Collect data from application

Service to monitor:

E:\om\c\threadValidator\examples\service\Release\serviceTV.exe

If you are working with native services please read the Native Services API Help topic.

- **Service to monitor** › type or **Browse** to set the service name to monitor
- **OK** › performs some brief setup work and then prompts you to start the service


Please start your service

Please start your service:

C:\Program Files (x86)\Software Verification\SVL Service x86\svlService.exe

Click OK to close the dialog and then start your native service or .Net service.

Start the service in the normal manner, e.g. from the control panel, the command line or programmatically.

Data collection

Depending on your application, and what you want to validate, you may want to start collecting data as soon as injection has happened, or do it later.

If your program has a complex start-up procedure, initialising lots of data, it may be much faster *not* to collect data until the program has launched.

If it's the startup procedure you want to validate, obviously start collecting data immediately.

➡ See the section on controlling data collection for how to turn collection on and off after launch.

Examples

Example demonstrating how to monitor a service.

Example demonstrating how to monitor an application launched from a service (how to monitor any application running on a service account).

3.17.6 IIS

Enter topic text here.

3.17.6.1 Monitoring IIS and ISAPI

Monitoring ISAPI

Monitoring ISAPI works for:

- Native ISAPI extensions.

Native ISAPI

> If you are working with native ISAPI you must use the NT Service API in your service *as well as* using the Monitor ISAPI method below.

To start monitoring ISAPI:

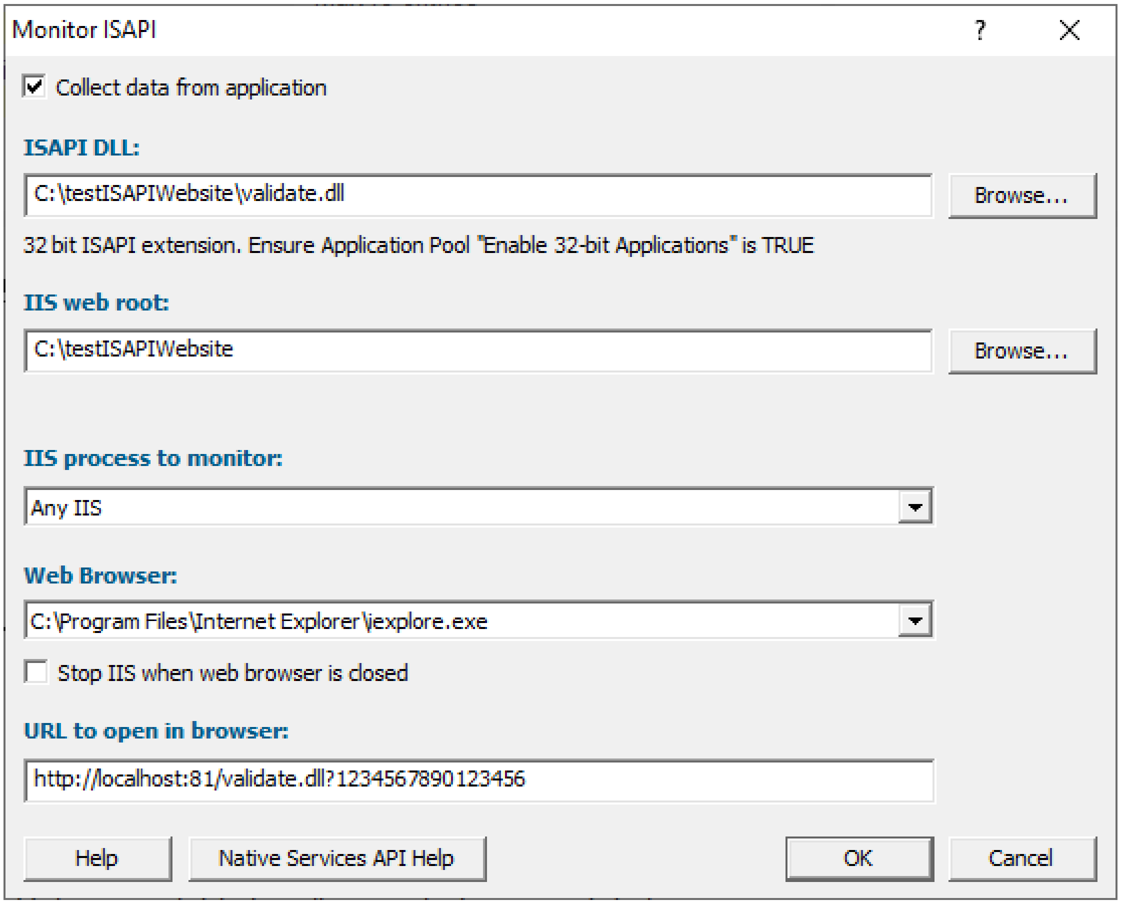
☰ **Launch** menu › **IIS** menu › **Monitor IIS and ISAPI...** › shows the Monitor ISAPI dialog below

or use the shortcut

F7 Monitor IIS and ISAPI

The monitor ISAPI dialog

Set the dll to monitor, the web root, the IIS process, an optional web browser to use and an optional url to launch, and click OK.



☑ Collect data from application

ISAPI DLL: C:\testISAPIWebsite\validate.dll

32 bit ISAPI extension. Ensure Application Pool "Enable 32-bit Applications" is TRUE

IIS web root: C:\testISAPIWebsite

IIS process to monitor: Any IIS

Web Browser: C:\Program Files\Internet Explorer\iexplore.exe

□ Stop IIS when web browser is closed

URL to open in browser: http://localhost:81/validate.dll?1234567890123456

- **ISAPI DLL** > type or **Browse** to set the ISAPI DLL that we're monitoring
- **IIS web** > type or **Browse** to set the web root for the IIS website we're working with
- **IIS process to monitor** > select the IIS process we're working with
- **Web Browser** > select the web browser that you're going to use to load the web page
- **URL to open in browser** > type the web page and arguments you want to load to cause the ISAPI to be loaded in IIS
- **OK** >resets IIS, setups all the variables, copies DLLs and settings into the web root and starts the web browser to load the specified web page

IIS is a protected process and can only execute, read and write files in specific directories. That's why Thread Validator copies data to the web root so that it can be read, written or executed.

Data collection

Depending on your application, and what you want to validate, you may want to start collecting data as soon as injection has happened, or do it later.

If your program has a complex start-up procedure, initialising lots of data, it may be much faster *not* to collect data until the program has launched.

If it's the startup procedure you want to validate, obviously start collecting data immediately.

➡ See the section on controlling data collection for how to turn collection on and off after launch.

3.17.6.2 Reset & Stop IIS

Reseting IIS

☰ **Launch** menu › **IIS** menu › **Reset IIS** › resets Internet Information Server (stops it and starts it again).

The session is not discarded when IIS is reset.

Stopping IIS

☰ **Launch** menu › **IIS** menu › **Stop IIS** › stops Internet Information Server.

The session is not discarded when IIS is stopped.

3.17.7 Linking to a program

Why link Thread Validator into your program?

There are cases when you might need to link Thread Validator directly into your program.

Sometimes the normal methods of launching and injecting aren't enough to get the data needed for a particular debugging task.

For example:

- maybe the data to be monitored has already been allocated before the stub was successfully injected
- maybe there is conflict with DLLs or a timing problem stopping the injection process from working as well as normal

These situations are rare, but given the variety of different applications, can happen.

Linking to your program

The library that you need to link to is:

- `svlThreadValidatorStubLib.lib` for 32 bit
- `svlThreadValidatorStubLib_x64.lib` for 64 bit

When linked and started, your program will automatically start Thread Validator.

The libraries should be linked to your program's **`.exe`**, not to a DLL that is loaded into your program.

3.17.8 Environment Variables

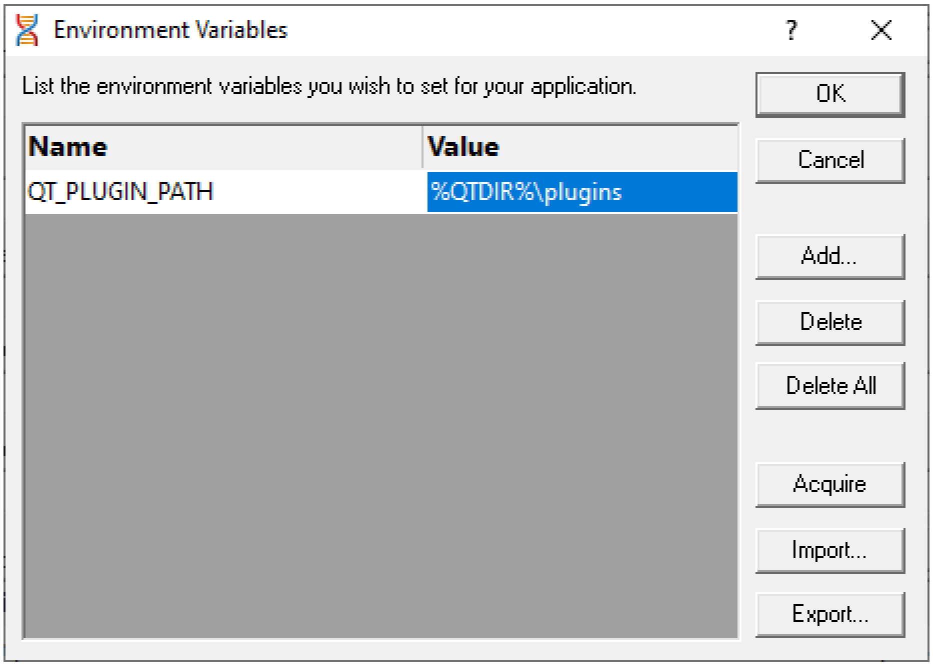
When launching an application, you might want to pass in some environment variables to your program.

The Environment Variables dialog lets you manage name/value pairs, including importing and exporting for use between programs or sessions.

The Environment Variables dialog

The dialog initially has no entries.

The example below shows the equivalent of `set QT_PLUGIN_PATH=%QTDIR%\plugins`



List the environment variables you wish to set for your application.

Name	Value
QT_PLUGIN_PATH	%QTDIR%\plugins

- **Add...** › adds a new item to the list › enter name in the first column, value in the second
- **Delete** › deletes a selected item in the list
- **Delete All** › clears the list
- **Acquire** › fetches all system environment variables, *adding* them to the list

- **Import...** › loads variables from a previously exported file, *adding* them to the list
- **Export...** › saves all entries in the list to a file of your choice

  The exported file is a simple ascii file with one entry per line of the form `name=value`
- **OK** › accepts all changes
- **Cancel** › ignores changes

3.18 Stopping your target program

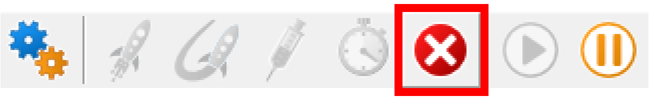
Stopping the application

You can stop or kill your program at any time using the task manager, or debugger.

You can also stop your program from within Thread Validator.

☰ **File** menu › **Abandon Application...** › stop the target program

or click on the red cross icon on the session toolbar.



The target program is ended using `ExitProcess()` from inside the stub.

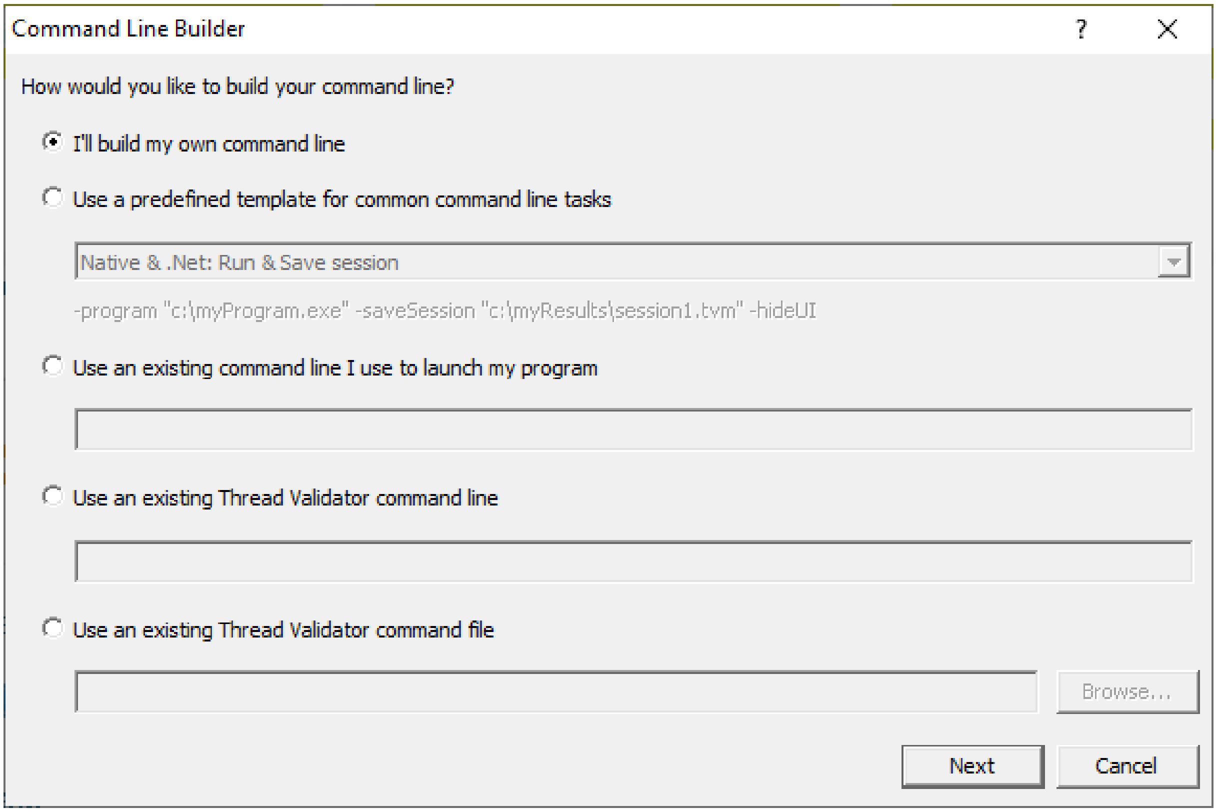
Since the session is discarded, using Thread Validator to stop the target program is usually quicker and more convenient than external stop methods.

➡ You can easily re-launch the program again using the same settings as before.

3.19 Command Line Builder

The command line builder helps you create command lines with valid options.

The command line builder is a two stage process, the first stage helping your choose how you want to build the command line, and the second stage actually building the command line based on the choices in the first stage.



Command Line Builder

How would you like to build your command line?

● I'll build my own command line

○ Use a predefined template for common command line tasks

Native & .Net: Run & Save session

-program "c:\myProgram.exe" -saveSession "c:\myResults\session1.tvm" -hideUI

○ Use an existing command line I use to launch my program

○ Use an existing Thread Validator command line

○ Use an existing Thread Validator command file

There are five options for building your command line:

- **I'll build my own** › you'll build your command line from scratch, with no predefined options

- **Use a predefined template** › choose from a list of predefined command lines that you can customize

  The predefined templates cover a range of tasks you may want to perform from the command line. These include running sessions, saving sessions, exporting to HTML and XML.

  Examples are provided for both Native and .Net applications, and .Net Core applications.

- **Use an existing command line** › use the command line you use to start the tool you want to collect profiling data for

  Example: `e:\dev\paintpot\release\paintpot.exe e:\testimages\venn.png -invert -mirror -phaseChange`

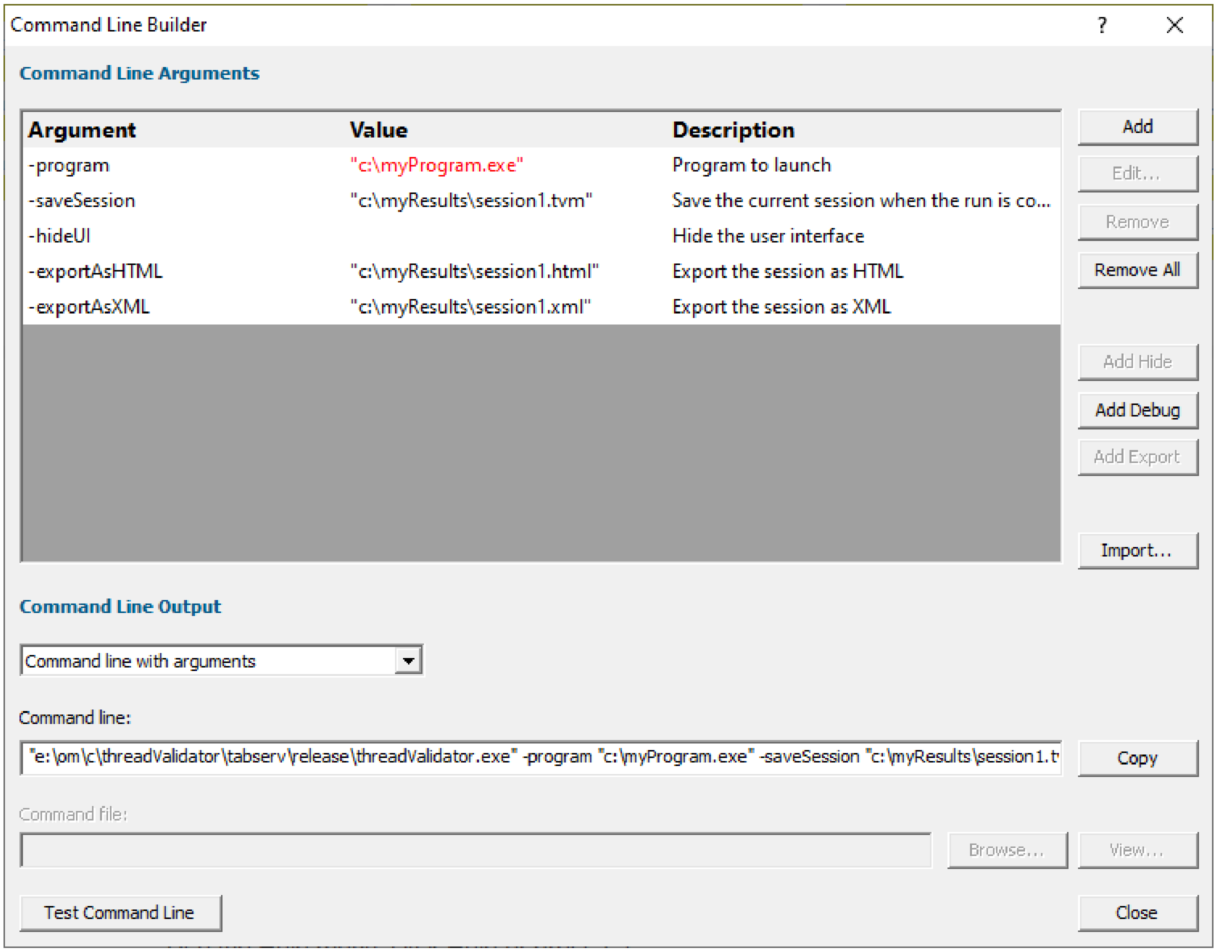
- **Use an existing Thread Validator command line** › use an existing Thread Validator command line and customize that

  Example: `-program e:\dev\paintpot\release\paintpot.exe -hideUI -exportAsHTML e:\testResults\gannt.html -allArgs e:\testimages\gantt.png -inflate:3`

- **Use an existing Thread Validator command file** › use an existing Thread Validator command file and customize that

  Example: `-commandFile e:\commandFiles\paintpot_gantt.cf`

When you have made your choice the **Next** button moves you to the customization part of the command line builder.



Command Line Builder

Command Line Arguments

Argument	Value	Description
-program	"c:\myProgram.exe"	Program to launch
-saveSession	"c:\myResults\session1.tvm"	Save the current session when the run is co...
-hideUI		Hide the user interface
-exportAsHTML	"c:\myResults\session1.html"	Export the session as HTML
-exportAsXML	"c:\myResults\session1.xml"	Export the session as XML

Command Line Output

Command line with arguments

Command line:

```
"e:\om\c\threadValidator\tabserv\release\threadValidator.exe" -program "c:\myProgram.exe" -saveSession "c:\myResults\session1.t
```

Command file:

The image above shows the command line builder populated with one of the predefined template choices. You can see a few entries refer to directories and files that do not exist on disk (they are red).

These are items you will need to customize to match the program you are testing.

Any entries that will only exist after they have been created by the test will also be shown in red.

Editing

To edit an argument, double click the argument. A combo box will display a list of valid arguments you can choose.

To edit a value, double the value. If the argument type has a list of known values a combo box will be provided, directories will display a directory chooser, files will display a file chooser, numbers will only allow numeric editing. All other values will be edited as text.

- **Add** › add a new argument to the grid
- **Remove** › remove the selected item
- **Remove All** › removes all items in the grid
- **Add Hide** › adds a -hideUI argument which will cause Thread Validator to hidden when running. Thread Validator will close after the target program finishes running
- **Add Debug** › adds various arguments which will cause Thread Validator to display error messages if there are problems with the command line.
- **Add Export** › adds export options that will cause Thread Validator to export html and/or xml reports after the target program finishes running
- **Import...** › you can import a command file, the contents of which will replace all the items in the grid

Command Line Output

There are two command line output styles.

- **Command line with arguments** › generates a command line containing all arguments and values shown in the grid
- **Command line with command file** › generates a command file containing all arguments and values shown in the grid, and a command line that references the command file

  When this option is chosen the command file edit field and the **Browse...** and **View...** buttons are enabled, allowing you to specify a command file name, and to view it's contents.

  If a command file has not been specified when this option is selected you will be prompted to select a name for the command file.

  When the command file name is selected the command file will be created with the arguments and values shown in the grid.

- **Copy** › copies the command line to the clipboard so that you can paste the command line in cmd prompts, batch files and automated scripts (Jenkins etc)
- **Browse...** › opens a Windows file dialog to allow you to specify the command file location
- **View...** › opens the command file using the Windows shell, this allows you to view the command file in your favourite editor

Testing

If you wish to test the command line, you have two options:

- **Manual test** › use the **Copy** button to copy the command line, then paste it into a cmd prompt and press return.
- **Test Command Line** › a new instance Thread Validator is started with the specified command line.

3.20 Data Collection

Collecting data

Once you've launched or injected into a program, you can stop and start data collection whilst the program is running.

This is a high level switch that controls *all* data collection, regardless of any other settings.

With data collection off, the target program runs at close to normal speed.

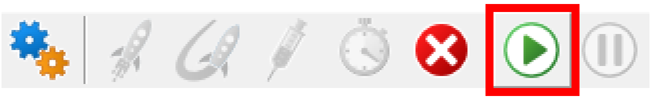
Temporarily turning off collection can be a good idea if you need to take actions to get the program into the right state for validation.

You can also turn data off from the start and only turn it on when you need it.

Starting and stopping data collection

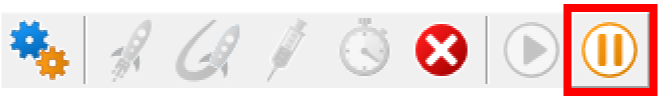
☰ **File** menu › **Start collecting data...** › starts collecting data immediately

or click on the green icon on the session toolbar to start collecting.



☰ **File** menu › **Stop collecting data...** › stops collecting

or click on the red icon on the session toolbar to stop.

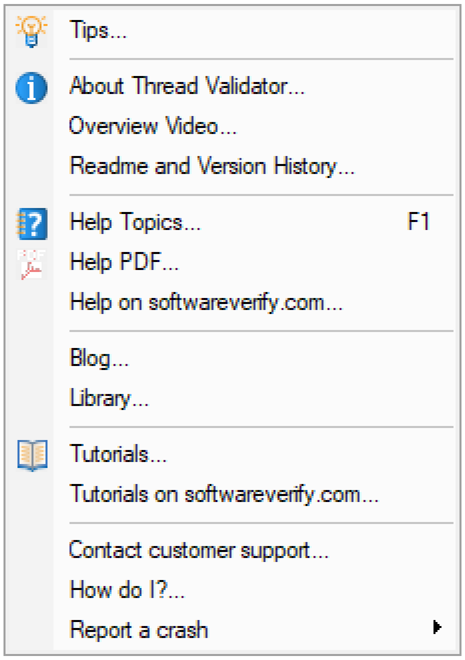


3.21 Help

The help menu

The help menu provides access to useful help, tips and tutorials.

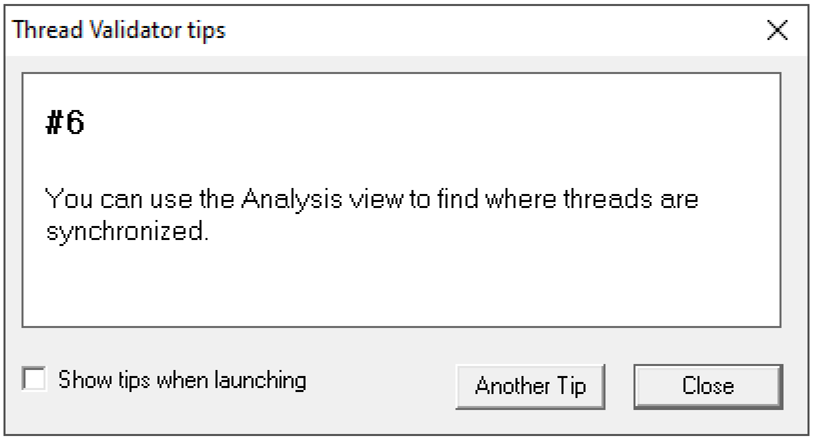
Each item is covered briefly below, in menu order.


Tips...
About Thread Validator...
Overview Video...
Readme and Version History...
Help Topics... F1
Help PDF...
Help on softwareverify.com...
Blog...
Library...
Tutorials...
Tutorials on softwareverify.com...
Contact customer support...
How do I?...
Report a crash

Tips

Help menu › **Tips...** › shows the tip dialog where you can browse tips in random order

Here you can also choose whether to display the tips dialog while launching programs.


Thread Validator tips

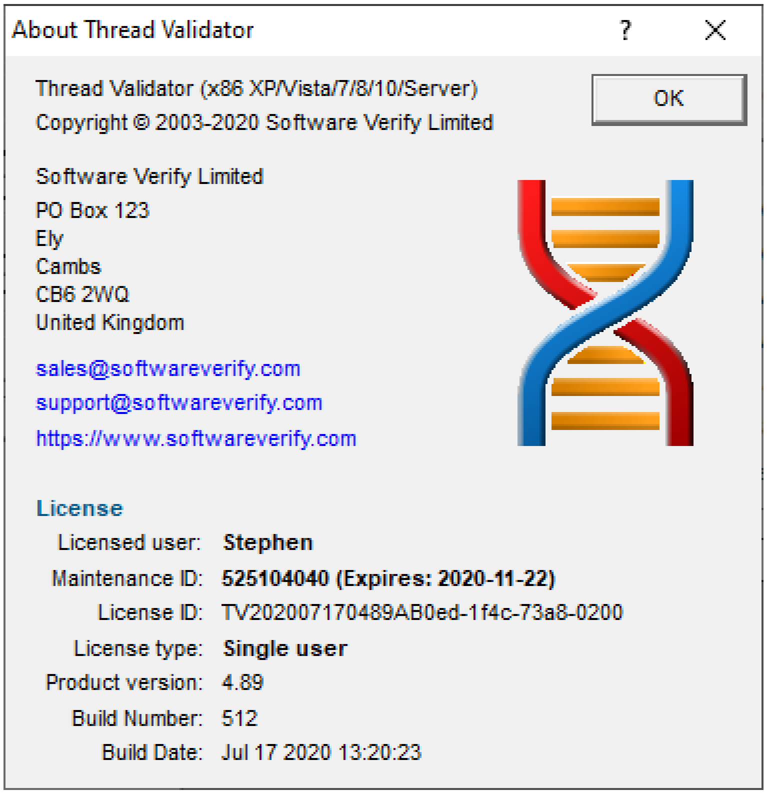
#6

You can use the Analysis view to find where threads are synchronized.

□ Show tips when launching

About box

Help menu › **About Thread Validator...** › shows contact and copyright information, as well as details of your license



Thread Validator (x86 XP/Vista/7/8/10/Server)
Copyright © 2003-2020 Software Verify Limited

Software Verify Limited
PO Box 123
Ely
Cambs
CB6 2WQ
United Kingdom

sales@softwareverify.com
support@softwareverify.com
https://www.softwareverify.com

License

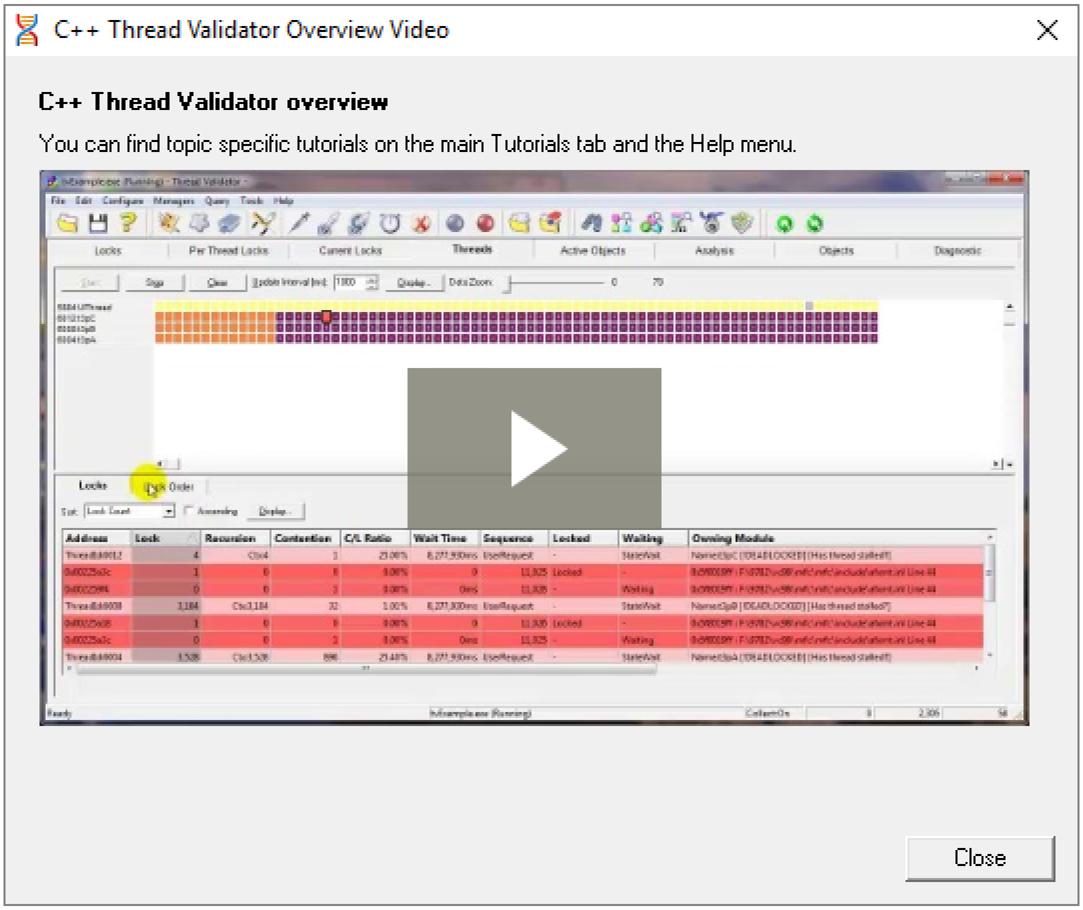
Licensed user:	**Stephen**
Maintenance ID:	**525104040 (Expires: 2020-11-22)**
License ID:	TV202007170489AB0ed-1f4c-73a8-0200
License type:	**Single user**
Product version:	4.89
Build Number:	512
Build Date:	Jul 17 2020 13:20:23

Overview Video

Help menu › **Overview Video...** › opens a new dialog showing a short video

The video has sound and does not play automatically.

The video is also available on the product website. Visit https://www.softwareverify.com/products.php and find the product link for Thread Validator.



C++ Thread Validator Overview Video

C++ Thread Validator overview

You can find topic specific tutorials on the main Tutorials tab and the Help menu.

Read-me and version history

Help menu › **Readme...** › opens the readme.html (from your installation) in your browser.

The readme file contains all the latest information about Thread Validator including:

- basic information about getting started and where to go for support
- known issues
- version history

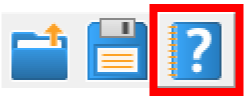
To see what's changed since the version you have installed see the latest version history.

Help HTML

Help menu › **Help topics...** › shows the HTML help dialog

You might be reading this right now!.

Or click on the question mark icon on the standard toolbar:



The F1 key also shows the help, but has the added bonus of jumping directly to the page relevant for the current view or dialog.

We occasionally get reports of customers seeing exception errors while viewing the HTML help. Unfortunately, we don't have a solution for this yet, but we'd appreciate knowing which pages are affecting you!

Help PDF

Help menu › **Help PDF** › shows the PDF version of this help

You will need a suitable PDF reader such as Adobe Acrobat Reader, but do beware of unwanted add-on installs.

PDF help for *all* our products are online.

Help on softwareverify.com

Help menu › **Help on softwareverify.com** › shows the online version of this help

Blog

Help menu › **Blog** › shows the Software Verify Blog.

Library

Help menu › **Library** › shows the Software Verify Library - all our best articles organised into related topics for easy access.

Tutorials

The tutorials are intended to guide you through learning how to use aspects of Thread Validator.

Latest tutorials are available online in the form of short videos and examples covering popular topics.

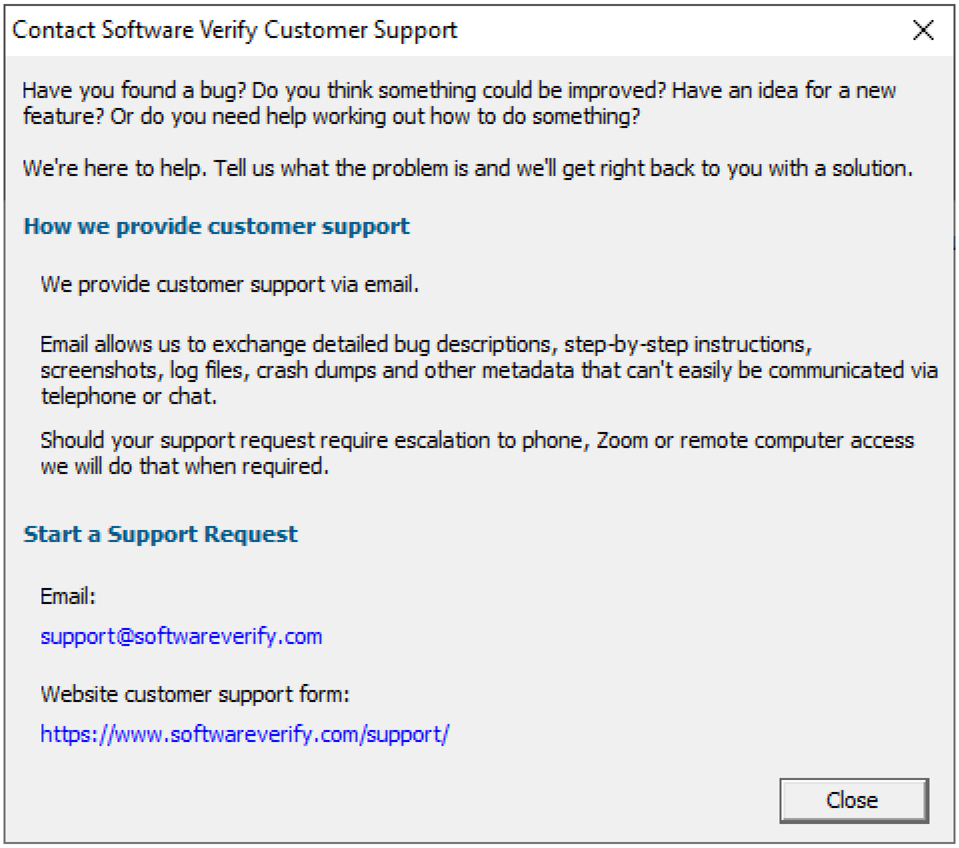
Help menu › **Tutorial...** › simply selects the Tutorial tab to show a list of the tutorials

Double click on the row of a tutorial in the list to open it in a browser.

Help menu › **Tutorials on softwareverify.com...** › opens the online tutorials in a browser

Contact customer support

☰ **Help** menu › **Contact customer support** › shows the Contact customer support dialog.



Contact Software Verify Customer Support

Have you found a bug? Do you think something could be improved? Have an idea for a new feature? Or do you need help working out how to do something?

We're here to help. Tell us what the problem is and we'll get right back to you with a solution.

How we provide customer support

We provide customer support via email.

Email allows us to exchange detailed bug descriptions, step-by-step instructions, screenshots, log files, crash dumps and other metadata that can't easily be communicated via telephone or chat.

Should your support request require escalation to phone, Zoom or remote computer access we will do that when required.

Start a Support Request

Email:

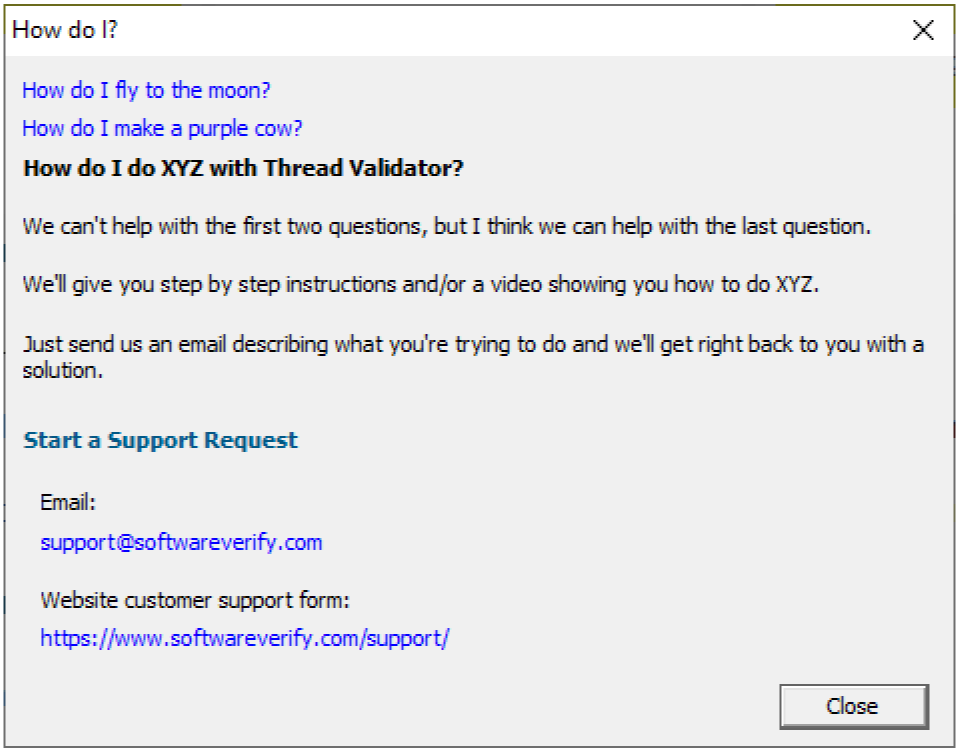
support@softwareverify.com

Website customer support form:

https://www.softwareverify.com/support/

How do I?

☰ **Help** menu › **How do I?** › shows the How do I? dialog.



How do I fly to the moon?

How do I make a purple cow?

How do I do XYZ with Thread Validator?

We can't help with the first two questions, but I think we can help with the last question.

We'll give you step by step instructions and/or a video showing you how to do XYZ.

Just send us an email describing what you're trying to do and we'll get right back to you with a solution.

Start a Support Request

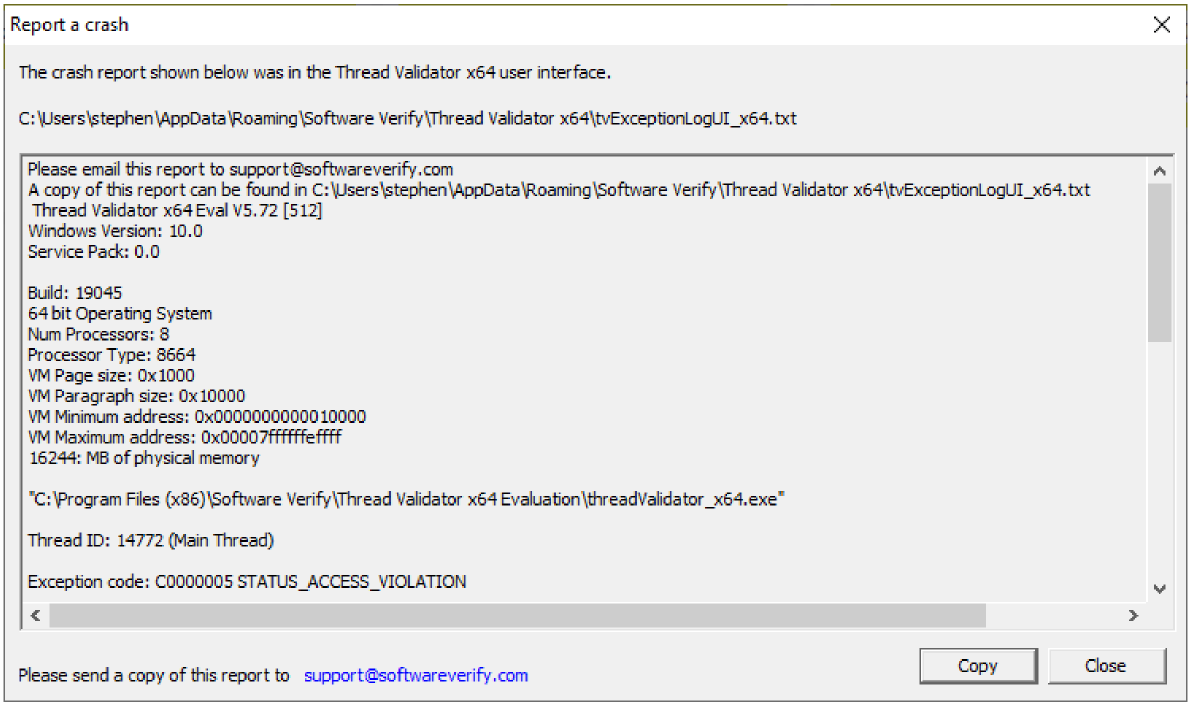
Email:
support@softwareverify.com

Website customer support form:
https://www.softwareverify.com/support/

Report a crash

Help menu › **Report a crash** › displays the options for reporting a crash.

If an exception report for the Thread Validator user interface, or an exception report for an application that Thread Validator was monitoring is available it will be displayed with options to copy it to the clipboard and contact customer support at Software Verify.



The crash report shown below was in the Thread Validator x64 user interface.

C:\Users\stephen\AppData\Roaming\Software Verify\Thread Validator x64\tvExceptionLogUI_x64.txt

```
Please email this report to support@softwareverify.com
A copy of this report can be found in C:\Users\stephen\AppData\Roaming\Software Verify\Thread Validator x64\tvExceptionLogUI_x64.txt
 Thread Validator x64 Eval V5.72 [512]
Windows Version: 10.0
Service Pack: 0.0

Build: 19045
64 bit Operating System
Num Processors: 8
Processor Type: 8664
VM Page size: 0x1000
VM Paragraph size: 0x10000
VM Minimum address: 0x0000000000010000
VM Maximum address: 0x00007ffffffeffff
16244: MB of physical memory

"C:\Program Files (x86)\Software Verify\Thread Validator x64 Evaluation\threadValidator_x64.exe"

Thread ID: 14772 (Main Thread)

Exception code: C0000005 STATUS_ACCESS_VIOLATION
```

Please send a copy of this report to support@softwareverify.com

Part IV

4 Command Line Interface

Thread Validator provides a command line interface to allow you to perform automated critical section data collection.

To run 32 bit thread validator run **`C:\Program Files (x86)\Software Verify\Thread Validator x86\threadValidator.exe`**

To run 64 bit thread validator run **`C:\Program Files (x86)\Software Verify\Thread Validator x64\threadValidator_x64.exe`**

Automated critical section data collection

Potential uses for automated critical section analysis are:

- In the regression test suite to ensure critical section behaviour
- In unit testing to ensure critical section performance of a certain level
- Quality assurance

Results from thread data collection sessions can be merged to form an aggregate result.

Typically, command line options allow Thread Validator to run by specifying:

- the target program to run
- arguments to pass to the target program
- the working directory to run in
- whether to run with or without the user interface
- a baseline session to compare with
- where and how to save results
- what to include or exclude from hooking
- how to merge results

Usually Thread Validator would exit between automated tests, but it can be made to stay running if necessary.

➡ See the command line reference for an alphabetical listing all the available commands.

Command line argument usage

There are a few basic rules to remember when using the command line arguments:

- separate arguments by spaces
- quote arguments if they contain spaces
- some arguments are only useful in conjunction with others
- some arguments are incompatible with others

If your command line is very long, consider using **-commandFile** to specify a command file for your arguments.

Unrecognised arguments

Any unrecognised arguments found on the command line are simply ignored, whether or not they are prefixed with a hyphen.

Arguments intended for your program will not conflict with the Thread Validator arguments in this manual as you should use -arg (or -allArgs) to redirect them to your program.

Need some help building the command line?

If you find creating command lines from nothing to be a bit daunting we've created a Command Line Builder tool to help you build command lines.

You'll still need to complete some details, but the builder will help prevent you making some mistakes.

Command Lines and Floating Licences

Thread Validator works from the GUI and from the command line with all types of software licence (floating licences and non-floating licences).

When floating licences are being used Thread Validator will wait to acquire a floating licence before proceeding to process the command line options.

There are no options to:

- Checkout a floating licence
- Release a floating licence
- Query for available licences

These options are managed automatically by Thread Validator, there is no need for such options to be manually controlled from the command line.

4.1 Example Command Lines

Typical command line examples

The following examples demonstrate a few different scenarios in which you might want to use Thread Validator via the command line.

To run 32 bit **`thread`** validator run **`C:\Program Files (x86)\Software Verify\`**Thread **`Validator x86\threadValidator.exe`**

To run 64 bit **`thread`** validator run **`C:\Program Files (x86)\Software Verify\`**Thread **`Validator x64\threadValidator_x64.exe`**

Example 1 - running a session and saving the results

This example starts the application, showing no progress dialog whilst attaching to the process.

On completion, the resulting session is saved, and some tabs are refreshed.

The last tab refreshed is displayed, resulting in the ActiveObjects tab being the current tab.

> threadValidator_x64.exe **-program** "c:\myProgram.exe" **-saveSession** "c:\myResults\session1.tvm_x64" **-displayUI** **-refreshSummary** **-refreshLocks** **-refreshActiveObjects**

A brief explanation of each argument:

Option	Argument	Description
-program	"c:\myProgram.exe"	The target program to launch
-saveSession	"c:\myResults\session1.tvm_x64"	After the application finishes, the session should be saved in this file
-displayUI		Show the user interface during the thread test
-refreshSummary		These tabs should be refreshed when the test completes
-refreshLocks		
-refreshActiveObjects		

Example 2 - running a session and saving the results and exiting

This example starts the application, showing no progress dialog whilst attaching to the process.

On completion, the resulting session is saved, and Thread Validator exits.

> threadValidator_x64.exe **-program** "c:\myProgram.exe" **-saveSession** "c:\myResults\session2.tvm_x64" **-hideUI**

A brief explanation of each argument:

Option	Argument	Description
-program	"c:\myProgram.exe"	The target program to launch
-saveSession	"c:\myResults\session2.tvm	After the application finishes, the session should be saved in this file

	_x64"	
-hideUI		Hide the user interface during the thread test and close the user interface after the test application finishes.

Example 3 - running a session and saving and exporting the results

This example starts the application, showing no progress dialog whilst attaching to the process.

On completion, the resulting session is saved, and some tabs are refreshed.

The last tab refreshed is displayed, resulting in the ActiveObjects tab being the current tab.

```
threadValidator_x64.exe -program "c:\myProgram.exe" -saveSession "c:
\myResults\session3.tvm_x64" -exportAsHTML "c:\myResults\session3.html" -displayUI -
refreshSummary -refreshLocks -refreshActiveObjects
```

A brief explanation of each argument:

Option	**Argument**	**Description**
-program	"c:\myProgram.exe"	The target program to launch
-saveSession	"c:\myResults\session3.tvm_x64"	After the application finishes, the session should be saved in this file
-exportAsHTML	"c:\myResults\session3.html"	After the application finishes, the exported data should be saved in this file
-displayUI		Show the user interface during the thread test
-refreshSummary		These tabs should be refreshed when the test completes
-refreshLocks		
-refreshActiveObjects		

4.2 Environment variables

Environment variables can be referenced on the command line.

This allows you to set an environment variable outside of Thread Validator (cmd prompt, batch file, etc) and then reference it on the command line.

For example:

```
        -program %BUILD_DIR%\testProgram.exe
```

If the BUILD_DIR environment variable is set to e:\dev\debug the above would evaluate to –program e:\dev\debug\testProgram.exe

What if I can't set an environment variable?

There are situations where it isn't desirable, or possible to set the environment variable value prior to starting Thread Validator.

In those situations you can set the environment variable on the command line using -setenvironment.

```
        -setenvironment BUILD_DIR=e:\dev\debug -program %BUILD_DIR%\testProgram.exe
```

Problems with environment variable substitution

If you are running from a command prompt, or batch file, or any process that will handle environment variable substitution using %ENV_VAR% you will find that referencing the environment variable on the command line won't work when using -setenvironment, because by the time Thread Validator sees the command line the %ENV_VAR% values have already been substituted.

To get around this, using ENV_VAR instead of %ENV_VAR%.

```
        -setenvironment BUILD_DIR=e:\dev\debug -program $BUILD_DIR$\testProgram.exe
```

-setenvironment

Set environment variables for Thread Validator, as a series of name/value pairs.

Use this option once for each environment variable you wish to set.

Usage of **-setenvironment** for any environment variable must appear on the command line prior to any reference to that environment variable on the command line.

To pass quotes along with the string, escape a pair of inner quotes like the example below

Examples:

-setenvironment APP_FLAG=ON;
-setenvironment "APP_FAG=ON;"
-setenvironment "APP_COMMS=ON; APP_DEBUG=OFF;"
-setenvironment "APP_MSG=\"A quoted string with spaces\";"
-setenvironment BUILD_DIR=e:\dev\debug

Note that this is not the same as -environment, which allows you to specify environment values that you can pass to the program being launched.

4.3 Development environment

The following options allow you to specify the development environment you used to build the application being tested.

-devIDE

Specifies the development environment or compiler used to build the application being tested.

These values correspond to the values on the Symbol Lookup part of the settings dialog.

- clang
- codeWarrior
- cppBuilder
- delphi
- devC++
- mingw
- other
- rust
- salfordFortran
- visualBasic6
- visualStudio. If you specify this you also need to specify **-devVisualStudioVersion** or **-devVisualStudioYear**
- visualStudioCustomDLL
- visualStudioDontSet

Examples:

-devIDE visualStudio
-devIDE delphi

-devVisualStudioVersion

If **-devIDE** has been specified as **visualStudio** then **-devVisualStudioVersion** can be used to specify the version of Visual Studio.

The version of Visual Studio needs to be installed on the machine for this to work.

The following versions are valid:

- 6
- 7
- 7.1
- 8
- 9
- 10
- 11
- 12
- 14

- 15
- 16
- 17

Examples:

-devVisualStudioVersion 6
-devVisualStudioVersion 10
-devVisualStudioVersion 17

*If you use **-devVisualStudioVersion** then you don't need to use **-devVisualStudioYear**.*

-devVisualStudioYear

If **-devIDE** has been specified as **visualStudio** then **-devVisualStudioYear** can be used to specify the version of Visual Studio.

The version of Visual Studio needs to be installed on the machine for this to work.

The following years are valid:

- 1996
- 2002
- 2003
- 2005
- 2008
- 2010
- 2012
- 2013
- 2015
- 2017
- 2019
- 2022

Examples:

-devVisualStudioYear 1996
-devVisualStudioYear 2010
-devVisualStudioYear 2022

*If you use **-devVisualStudioYear** then you don't need to use **-devVisualStudioVersion**.*

4.4 Target Program & Start Modes

Resetting global settings

-resetSettings

Forces Thread Validator to reset all settings to the default state, except for any configured colours and the UI Global Hook settings which must be reset manually.

If using this option, it's recommended that you list this *first* on your command line or in your command file.

Specifying the target application

The following options let you launch a program (with various start-up modes), inject into a running program or wait for a program to start before attaching.

Launching a program

-program

Specifies the full file system path of the executable target program to be started by Thread Validator, *including any extension*.

Not compatible with -injectName, -injectID, -waitName or -monitorAService.

See -arg below to pass arguments to your program, and -directory to set where it runs.

Examples:

-program c:\testbed.exe
-program "c:\new compiler\version2\testbed.exe"

If you specify the file *without* a path then:

- If you used -directory to set a startup directory then the filename in that directory is used if it exists
- Otherwise, the directories in the PATH environment variable are used to look for the filename

-programToMonitorEXE
-programToMonitor

-programToMonitor has been replaced by **-programToMonitorEXE**. **-programToMonitor** will be honoured to provided backward compatibility.

Specifies the full path of the program from which the data is collected, but *does not change* which process is initially launched. *Include the extension*.

This program will be monitored by Thread Validator only when the program specified using -program starts it.

If no path is specified, the first child process that has the same name will be monitored.

To monitor *any* program that is launched specify `<<Any>>` as the program argument. In batch files and powershell scripts you will need to quote this to get it accepted by the file parser.

See -programToMonitorLaunchCount to change which instance of the program is monitored.

Only valid in conjunction with -program.

Examples:

-programToMonitorEXE c:\testbed-child-process.exe
-programToMonitorEXE "c:\new compiler\version2\testbedChildProcess.exe"
-programToMonitorEXE testbed-child-process.exe
-programToMonitorEXE "<<Any>>"

-program c:\testbed.exe **-programToMonitorEXE** c:\testbed-child-process.exe

In this last example `c:\testbed.exe` is launched but not monitored. Only when `testbed.exe` launches a child process `c:\testbed-child-process.exe` is that child process monitored.

-programToMonitorDLL

This option provides a qualifying DLL to identify different .Net Core processes, which are typically launched using the same .Net Core runtime. *Include the dll extension.*

Only valid in conjunction with -program and -programToMonitorEXE.

Examples:

-programToMonitorDLL c:\test\dotNetCoreApp.dll

-program c:\testbed.exe **-programToMonitorEXE "**c:\program files\dotnet\dotnet.exe" **-programToMonitorDLL** c:\test\dotNetCoreApp.dll

In this last example `c:\testbed.exe` is launched but not monitored. Only when `testbed.exe` launches a child process `c:\program files\dotnet\dotnet.exe` to run the application `c:\test\dotNetCoreApp.dll` is that child process monitored.

-programToMonitorLaunchCount

Specify the nth invocation of the child program specified by -programToMonitor which is to have its data collected.

Any value which is invalid (including anything less than 1) will default to 1.

Only valid in conjunction with -programToMonitor and consequently also -program.

Examples:

-programToMonitorLaunchCount 1
-programToMonitorLaunchCount 34

-program c:\testbed.exe **-programToMonitor** c:\testbed-child-process.exe **-programToMonitorLaunchCount 1**

In the above example c:\testbed.exe is launched but not monitored. As soon as testbed.exe launches a child process c:\testbed-child-process.exe then that child process is monitored.

If the value 1 was changed to a 2, then only the second invocation of c:\testbed-child-process.exe would get monitored, with the first invocation being ignored.

-environment

Environment variables for program, as a series of name/value pairs. Not to be confused with -setenvironment.

Use this option once for each environment variable you wish to set.

To pass quotes along with the string, escape a pair of inner quotes like the example below

Only valid with: -program

Examples:

```
-environment APP_FLAG=ON;
-environment "APP_FAG=ON;"
-environment "APP_COMMS=ON; APP_DEBUG=OFF;"
-environment "APP_MSG=\"A quoted string with spaces\";"
```

-arg

Passes the following element on the command line to the target program.

-arg can be used multiple times, or you can use -allArgs

To pass quotes along with the string, escape a pair of inner quotes like the example below

Only valid with: -program

Examples:

```
-arg myProgram.exe
-arg "c:\Program Files\myApp\myProgram.exe"
-arg "-in infile -out outfile"
-arg "\"a quoted string\""
```

-allArgs

Passes the remainder of the command line (after -allArgs) to the program being launched.

Unlike -arg above, there is no need to escape the quotes as the content is passed verbatim.

Only valid with: -program

Example:

-allArgs anything put here is passed to the target program "even stuff in quotes" is passed

-directory

Sets the working directory in which the program is executed. If -directory is not specified the program is run in its current directory.

Only valid with: -program

Examples:

-directory c:\development\
-directory "c:\research and development\"

-stdin

Specifies a file to be read and piped to the standard input of the application being tested.

If the filename contains spaces, the filename should be quoted.

An error occurs if the file does not exist. See **-ignoreMissingStdin** to avoid this error.

Examples:

-stdin c:\settings\input.txt
-stdin "c:\reg tests settings\input.txt"

-stdout

Specifies a file to be written with data piped from the standard output of the application being tested.

If the filename contains spaces, the filename should be quoted.

An error occurs if the file does not exist. See **-ignoreMissingStdout** to avoid this error.

Examples:

-stdout c:\settings\output.txt
-stdout "c:\reg tests results\output.txt"

-ignoreMissingStdin

If this flag is specified and the file specified by **-stdin** does not exist, no error is reported.

-ignoreMissingStdout

If this flag is specified and the file specified by **-stdout** does not exist, no error is reported.

Injecting into a program

-injectName

Sets the name of the process for Thread Validator to attach to.

Not compatible with -program, -injectID, -waitName or -monitorAService.

Examples:

-injectName c:\testbed.exe
-injectName "c:\new compiler\version2\testbed.exe"

-injectID

Sets the numeric id of a process for Thread Validator to attach to.

Not compatible with -program, -injectName, -waitName or -monitorAService.

Example:

-injectID 1032

Waiting for a program

-waitNameEXE
-waitName

-waitName has been replaced by **-waitNameEXE**. **-waitName** will be honoured to provided backwards compatibility.

Sets the name of a process that Thread Validator will wait for.

When the named process starts Thread Validator will attach to the process.

Not compatible with -program, -injectName, -injectID or -monitorAService.

Examples:

-waitNameEXE c:\testbed.exe
-waitNameEXE "c:\new compiler\version2\testbed.exe"

-waitNameDLL

Sets the name of a process DLL that Thread Validator will wait for.

When the named process starts Thread Validator will attach to the process.

Examples:

-waitNameDLL c:\dotNetApp.dll
-waitNameDLL "c:\new compiler\version2\dotNetApp.dll"

For use with -waitNameEXE when you want to wait for .Net Core applications.

-waitNameEXE "c:\program files\dotnet\dotnet.exe" **-waitNameDLL** "c:\testApps\dotNetCoreApp\release\dotNetCoreApp.dll"

Monitoring a service

-monitorAService

Sets the full file system path of a service *including any extension*.

Thread Validator will wait for the service to start and attach to it.

Not compatible with -program, -injectName, -injectID or -waitName.

Examples:

-monitorAService c:\service.exe
-monitorAService "c:\new compiler\version2\service.exe"

Data Collection

-collectData

Enables or disables the collection of flow tracing data

Examples:

-collectData:On
-collectData:Off

-collectStdout

Enables or disables the collection of standard output (stdout)

Examples:

-collectStdout:On
-collectStdout:Off

4.5 User interface visibility

User interface visibility

You can choose to hide or show Thread Validator during the test, as well as the window of the target application.

-displayUI

Forces the Thread Validator user interface to be displayed during the test.

This is useful for debugging a command line session that is not working, for example inspecting the Diagnostic tab for messages related to the test.

You wouldn't normally use this option when running unattended thread tests.

-doNotInteractWithUser

Never display dialog boxes in the target application that is being profiled.

This applies even for warning and error dialog boxes.

The intended use for this option is for when you are running command line sessions on unattended computers and you have automated processes that may kill the Thread Validator user interface if something goes wrong. Actions such as this then cause the stub to recognise the user interface has gone away and display an error warning.

-hideUI

Hides the Thread Validator user interface during the test.

-launchAppHide
-launchAppHidden (for backwards compatibility only)

Hides the target application during the test.

Depending on your application, this may not work and may not even be suitable.

This is equivalent to setting the `wShowWindow` member of the `STARTUPINFO` struct to `SW_HIDE` when using the Win32 `CreateProcess()` function.

It's useful if you're testing console applications that have no user interaction, as it prevents the console/command prompt from being displayed.

For GUI applications this option very much depends on how your application works.

For interactive applications, it clearly has no use, but for some, hiding the GUI may help prevent various windows messages from being processed.

Typically, for complex applications, it's better to design this capability into your application and control it via a command line, which can be passed in from Thread Validator via the **-arg** option.

-launchAppShow

Shows the target application during the test.

This is equivalent to setting the `wShowWindow` member of the `STARTUPINFO` struct to `SW_SHOW` when using the Win32 `CreateProcess()` function.

-launchAppShowMaximized
-launchAppShowMinimized
-launchAppShowMinNoActive
-launchAppShowNA
-launchAppShowNoActivate
-launchAppShowNormal

As well as the previous two options to show or hide the target application during the test there are other options equivalent to values that can be used in the `STARTUPINFO` struct.

The options are equivalent to the setting the `wShowWindow` member to the following values

Option	wShowWindow member	Launched application is shown...
-launchAppShowMaximized	`SW_SHOWMAXIMIZED`	Maximized and activated
-launchAppShowMinimized	`SW_SHOWMINIMIZED`	Minimized and activated
-launchAppShowMinNoActive	`SW_SHOWMINNOACTIVE`	Minimized and not active
-launchAppShowNA	`SW_SHOWNA`	Shown at current size and position but not activated
-launchAppShowNoActivate	`SW_SHOWNOACTIVATE`	Show at most recent size and position but not activated
-launchAppShowNormal	`SW_SHOWNORMAL`	Show at original size and position and activated

-showCommandPrompt

-hideCommandPrompt

Causes any launched console window to be shown or hidden during the test.

-showErrorsOnCommandPrompt

If an error occurs when launching the application, the error will be reported on the command line.

Refreshing the interface after test completion

You can run automated tests that leave the user interface open after completion,

The following options are used to automatically refresh the main data tabs in Thread Validator once a test is complete.

-refreshSummary
-refreshLocks
-refreshPerThreadLocks
-refreshCurrentLocks
-refreshThreads
-refreshCoverage
-refreshActiveObjects
-refreshAnalysis
-refreshObjects
-refreshWaitChain

4.6 Session Management

Session management

The following options let you control the sessions during testing

-baseline

Loads a previous session as the baseline session against which the recorded session is compared for regressions or improvements.

Examples:

-baseline c:\baseline\testMacro1.tvm
-baseline "c:\base line\testMacro1.tvm"

Ensure your session manager is configured to hold at least 2 sessions or use **-numSessions** to specify how many sessions to use.

-numSessions

Sets the number of sessions that can be loaded at once.

This is equivalent to the same setting in the session manage and can't be less than 1.

Example:

-numSessions 2

-saveSession

Saves the session data when all data has finished being collecting from the target program.

Thread Validator 32 and 64 bit use the file extension .tvm and .tvm_64 respectively.

A missing or incorrect filename extension will be corrected automatically

Examples:

-saveSession c:\results\testMacro1.tvm
-saveSession "c:\test results\testMacro1.tvm_x64"
-saveSession c:\results\testMacro1

-sessionLoad

-sessionLoad loads a previously created session to be merged with the data from the session being *recorded*.

These options might be used when a series of tests have already been performed and their sessions saved.

Thread Validator 32 and 64 bit use the file extension .tvm and .tvm_64 respectively.

A missing or incorrect filename extension will be corrected automatically

Examples:

-sessionLoad c:\results\testMacro1.tvm
-sessionLoad "c:\test results\testMacro1.tvm"
-sessionLoad c:\results\testMacro1

Ensure your session manager is configured to hold at least 2 or 3 sessions or use **-numSessions** to specify how many sessions to use.

-sessionCompareHTML
-sessionCompareXML

Compare two sessions, producing an HTML or XML report detailing any regression and improvements.

The report is produced using the XML Export tags described in the Exporting Sessions section..

The two sessions can be loaded using one of these options:

-baseline and **-sessionLoad**

-baseline and running a program using one of **-program**, **-injectName**, **-injectID**, or **-waitName**

Examples:

-sessionCompareXML c:\regtests\testMacro1.xml
-sessionCompareHTML "c:\reg tests\testMacro1.html"

Ensure your session manager is configured to hold at least 2 sessions or use **-numSessions** to specify how many sessions to use.

4.7 Session Export Options

Session export format - HTML or XML

-exportAsHTML
-exportAsXML

Export the session data as an HTML or XML file when Thread Validator has finished collecting data from the target program.

If you merge the current session with another session, the exported HTML will be for the *merged* session.

If you disable merging with the current session the export will be for the *unmerged* session.

Example:

-exportAsHTML c:\results\html\testMacro1.html
-exportAsXML "c:\test results\xml\testMacro1.html"

Session export encoding - HTML or XML

These options allow you to export the session data as UTF-16, UTF8 or ASCII. UTF-16 and UTF-8 will add a byte order mark (BOM) to the start of the exported file.

-exportAsHTML_BOM

The exported HTML will be exported with the appropriate format.

-exportAsHTML_BOM ASCII
-exportAsHTML_BOM UTF8
-exportAsHTML_BOM UTF16

-exportAsXML_BOM

The exported XML will be exported with the appropriate format.

-exportAsXML_BOM	**ASCII**
-exportAsXML_BOM	**UTF8**
-exportAsXML_BOM	**UTF16**

4.8 File Locations

File Locations

When using the command line it's convenient to store settings and options in files that can be easily referenced.

Those files include:

- Global settings files
- File locations for source, PDB or MAP files
- DLL hook files

Each of these file types can be saved or exported from Thread Validator.

The **-settings** option is used to specify the settings to be used for the test. If the filename contains spaces, the filename should be quoted. This option is the same as **-loadSettings** and is provided for backwards compatibility.

Loading global settings from a file

Global settings are usually stored in the registry, but you can save a specific set of settings for use in thread tests:

- **Settings** menu › **Save settings...**

-loadSettings
-settings

Points to a previously saved settings file to be used for the test.

Examples:

-loadSettings c:\settings\testMacro1.tvs
-loadSettings "c:\thread test settings\testMacro1.tvs"

The -settings option is identical to -loadSettings and is provided for backwards compatibility

File locations for source, PDB or MAP files

File location files can be easily generated by exporting file locations from the File Locations page of the settings dialog.

-fileLocations

Specify a plain text file listing file locations to be used during testing. See the format of the file below.

Each set of file types (one per line) is preceded by a header line in the file.

- **[Files]** ❯ Source files
- **[Third]** ❯ Third party source files
- **[PDB]** ❯ PDB files
- **[MAP]** ❯ MAP files

Example:

-fileLocations c:\threadTests\testFileLocations1.tvxfl

Example file:

```
[Files]
c:\work\project1\
[Third]
d:\VisualStudio\VC98\Include
[PDB]
c:\work\project3\debug
c:\work\project3\release
[MAP]
c:\work\project3\debug
c:\work\project3\release
```

Files listing DLLs to hook

DLL hook files can be easily generated by exporting DLL hooks from the Hooked DLLs page in the Filters section of the settings dialog.

-dllHookFile

Points to a file listing the DLLs to be hooked for the test.

Examples:

-dllHookFile c:\settings\testMacroDLLs.tvx
-dllHookFile "c:\thread tests settings\testMacroDLLs.tvx"

The first line of text in the DLL hooks file is one of the following:

- Rule: DoNotHook › DLLs marked as enabled *will not* be hooked. All other DLLs will be hooked
- Rule: DoHook › DLLs marked as enabled *will* be hooked. All other DLLs will not be hooked
- Rule: HookAll › All DLLs will be hooked regardless of the settings in the list

Capitalization is important.

The remaining lines list one DLL filename or folder path and an enabled state on each line.

Example:

```
Rule:DoNotHook
nativeExample.exe enable=FALSE
MFC42D.DLL enable=TRUE
MSVCRTD.dll enable=TRUE
KERNEL32.dll enable=TRUE
ole32.dll enable=TRUE
```

Example:

```
Rule:DoHook
E:\OM\C\threadValidator\examples\nativeExample\DebugNonLink
enable=TRUE
```

Example:

```
Rule:DoHook
"E:\OM\C\threadValidator\examples\nativeExample with
spaces\DebugNonLink" enable=TRUE
```

Example:

```
Rule:DoHook
%ENV_VAR%\DebugNonLink enable=TRUE
```

Here, the environment variable `ENV_VAR` is used to replace the text `%ENV_VAR%` in the path definition.

The file can be ANSI or UNICODE text and paths with spaces do not need quotes.

4.9 Command Files

Using a command file

If your command line is very long, consider using **-commandFile** to specify a command file for your arguments.

-commandFile

Specify a file from which to read the command line arguments.

Useful when command lines become unwieldy or longer than the windows command size limits.

Use -- to insert comments into the file, including when commenting out option.

Examples:

-commandFile c:\threadtests\testMacro1.cf
-commandFile "c:\thread tests\testMacro1.cf"

⊟ Example command file

```
-hideUI
-program c:\testbed\testApp.exe

-- arguments for application
-arg argumentOne
-arg argumentTwo
-arg "-s wobble"
-directory c:\testbed\test1
-settings c:\testbed\settings_test1.tvs

-- do export and save of the results
-exportAsHTML c:\testbed\results\test1.html
-saveSession c:\testbed\results\test1.tvm
```

For any argument that can be supplied to a command in a command file, you can also specify an environment variable substitution.

```
-directory %DIR%
-program %DIR%\testProgram.exe
```

The environment variables must have been set prior to starting Thread Validator.

You cannot specify a command with an environment variable substitution.

4.10 Help, Errors & Return Codes

The following options may help with using and debugging the command line driven automated regression testing.

Command line help

-help
-?

Command line help is printed on the standard output.

-echoArgsToUser

Show the arguments that were supplied to the program.

Debugging command driven testing

If you're having problems with using the command line, check the following, try displaying error messages using the option below, and look at the exit return codes.

.

- separate command line arguments with spaces
- all command line options that include spaces need to have quotes around them
- some arguments are only useful in conjunction with others - check notes against each option
- some arguments are incompatible with others - check notes against each option

-showErrorsWithMessageBox

Forces errors to be displayed using a message box when running from the command line.

This can be very useful when debugging a command line that does not appear to work correctly.

Exit return codes

Thread Validator returns the following status codes when running from the command line. Some of these status codes are for internal use, or are obsolete (retired products).

- **0** › All ok
- **-1** › Unknown error. An unexpected error occurred starting the runtime
- **-2** › Application started ok. You should not see this code returned
- **-3** › Application failed to start. E.g. runtime not present, not an executable or injection dll not present,
- **-4** › Target application is not an application
- **-5** › Don't know what format the executable is, cannot process it
- **-6** › Not a 32 bit application
- **-7** › Not a 64 bit application
- **-8** › Using incorrect MSVCR(8|9).DLL that links to CoreDLL.dll (incorrect DLL is from WinCE)
- **-9** › Win16 app cannot start these because we can't inject into them
- **-10** › Win32 app - not used
- **-11** › Win64 app - not used
- **-12** › .Net application
- **-13** › User bailed out because app not linked to MSVCRT dynamically
- **-14** › Not found in launch history
- **-15** › DLL to inject was not found
- **-16** › Startup directory does not exist
- **-17** › Symbol server directory does not exist
- **-18** › Could not build a command line
- **-19** › No runtime specified, cannot execute script (or Java) (obsolete)

- **-20** › Java arguments are OK - not an error (obsolete)
- **-21** › Java agentlib supplied that is not allowed because Java Thread Validator uses it (obsolete)
- **-22** › Java xrun supplied that is not allowed because Java Thread Validator uses it (obsolete)
- **-23** › Java cp supplied that is not allowed because Java Thread Validator uses it (obsolete)
- **-24** › Java classpath supplied that is not allowed because Java Thread Validator uses it (obsolete)
- **-25** › Firefox is already running, please close it (obsolete)
- **-26** › Lua runtime DLL version is not known (obsolete)
- **-27** › Not compatible software
- **-28** › InjectUsingCreateProcess, no DLL name supplied
- **-29** › InjectUsingCreateProcess, Unable to open PE File when inspecting DLL
- **-30** › InjectUsingCreateProcess, Invalid PE File when inspecting DLL
- **-31** › InjectUsingCreateProcess, No Kernel32 DLL
- **-32** › InjectUsingCreateProcess, NULL VirtualFree() from GetProcAddress
- **-33** › InjectUsingCreateProcess, NULL GetModuleHandleW() from GetModuleHandleW
- **-34** › InjectUsingCreateProcess, NULL LoadLibraryW() from LoadLibraryW
- **-35** › InjectUsingCreateProcess, NULL FreeLibrary() from FreeLibrary
- **-36** › InjectUsingCreateProcess, NULL VirtualProtect() from GetProcAddress
- **-37** › InjectUsingCreateProcess, NULL VirtualFree() from GetProcAddress
- **-38** › InjectUsingCreateProcess, unable to find DLL load address
- **-39** › InjectUsingCreateProcess, unable to write to remote process's memory
- **-40** › InjectUsingCreateProcess, unable to read remote process's memory
- **-41** › InjectUsingCreateProcess, unable to resume a thread
- **-42** › UPX compressed - cannot process such executables
- **-43** › Java class not found in CLASSPATH
- **-44** › Failed to launch the 32 bit svlGetProcAddressHelperUtil.exe
- **-45** › Uknown error with svlGetProcAddressHelperUtil.exe
- **-46** › Couldn't load specified DLL into svlGetProcAddressHelperUtil.exe
- **-47** › Couldn't find function in the DLL svlGetProcAddressHelperUtil.exe
- **-48** › Missing DLL argument svlGetProcAddressHelperUtil.exe
- **-49** › Missing function argument svlGetProcAddressHelperUtil.exe
- **-50** › Missing svlGetProcAddressHelperUtil.exe
- **-51** › Target process has a manifest that requires elevation
- **-52** › svlInjectIntoProcessHelper_x64.exe not found
- **-53** › svlInjectIntoProcessHelper_x64.exe failed to start
- **-54** › svlInjectIntoProcessHelper_x64.exe failed to return error code
- **-55** › getImageBase() worked ok
- **-56** › ReadFile() failed in getImageBase()
- **-57** › NULL pointer when trying to allocate memory
- **-58** › CreateFile() failed in getImageBase()
- **-59** › ReadProcessMemory() failed in getImageBase()
- **-60** › VirtualQueryEx() failed in getImageBase()
- **-61** › Bad /appName argument in svlInjectIntoProcessHelper_x64.exe
- **-62** › Bad /dllName argument in svlInjectIntoProcessHelper_x64.exe
- **-63** › Bad /procId argument in svlInjectIntoProcessHelper_x64.exe
- **-64** › Failed to OpenProcess in svlInjectIntoProcessHelper_x64.exe
- **-65** › A DLL that the .exe depends upon cannot be found
- **-66** › A stdin file was specified, but Validator could not open it
- **-67** › A stdout file was specified, but Validator could not open it
- **-68** › Failed to create the child output pipe
- **-69** › Failed to create a duplicate of the output write handle for the std error write handle. This is necessary in case the child application closes one of its std output handles
- **-70** › Failed to create the child input pipe
- **-71** › Failed to create a duplicate output read temporary file

- **-72** › Failed to create a duplicate input write temporary file
- **-73** › User was trying to launch a service as an application that was linked to TV APIs. User cancelled when informed of this fact
- **-74** › Returned by Thread Validator if user performs a baseline comparison and memory leaks are detected
- **-75** › Shutdown and restart as 32 bit Thread Validator
- **-76** › Shutdown and restart as 64 bit Thread Validator
- **-77** › Entry point in executable is NULL.
- **-78** › Application is .Net Core.
- **-79** › Entry point is for a .Net application.
- **-80** › VirtualAllocEx() returned NULL
- **-81** › InjectUsingCreateProcess, NULL GetLastError() from GetProcAddress

4.11 Command Line Reference

Command line reference

The following alphabetical list provides a convenient look-up for all the command line arguments used in automated regression testing.

Option	Description
-?	Print command line help on the standard output.
-allArgs	Pass the remainder of the command line to the program being launched.
-arg	Pass command line arguments to the target program. Can be used multiple times.
-baseline	Loads a previous session as the baseline session against which the recorded session is compared for regressions or improvements.
-collectData	Turn data collection on or off
-collectStdout	Turn collection of stdout on or off
-commandFile	Specify a file from which to read the command line arguments.
-devIDE	Specify the development environment used to be the target program.
-devVisualStudioVersion	Specify which version of Visual Studio by year.
-devVisualStudioYear	Specify which version of Visual Studio by version number.
-directory	Set the working directory in which the program is executed.
-displayUI	Force the Thread Validator user interface to be displayed during the test.
-dllHookFile	Points to a file listing the DLLs to be hooked for the test.
-doNotInteractWithUser	Never display dialog boxes in the target application that is being profiled.

-echoArgsToUser	
-environment	Environment variables for program, as a series of name/value pairs
-exportAsHTML -exportAsXML	Export the session data as an HTML or XML file when Thread Validator has finished collecting data from the target program.
-exportAsHTML_BOM -exportAsXML_BOM	Specify the file encoding for the exported file
-fileLocations	Specify a plain text file listing file locations to be used during testing. See the format of the file below.
-help	Print command line help on the standard output.
-hideCommandPrompt	Any launched console window will be hidden during the test.
-hideUI	Hide the Thread Validator user interface during the test.
-injectID	Set the numeric (decimal) id of a process for Thread Validator to attach to.
-injectName	Set the name of the process for Thread Validator to attach to.
-launchAppHide	Hide the target application during the test.
-launchAppShow	Show the target application during the test.
-launchAppShowMaximized	Show the target application maximized and activated.
-launchAppShowMinNoActive	Show the target application minimized and activated.
-launchAppShowMinimized	Show the target application minimized and not active.
-launchAppShowNA	Show the target application at current size and position but not activated.
-launchAppShowNoActivate	Show the target application at most recent size and position but not activated.
-launchAppShowNormal	Show the target application at original size and position and activated.
-loadSession	Load a previously created session to be merged with the data from the session being recorded.
-loadSettings	Points to a previously saved settings file to be used for the test.
-monitorAService	Specify the full file system path to the service to monitor, including any extension. The service is not started by Thread Validator but by an external means.
-numSessions	Set the number of sessions that can be loaded at once.
-program	Specify the full file system path of the executable target program to be started by Thread Validator, including any extension.
-programToMonitor	Changes which program the data is collected from but does not change which process Thread Validator initially launches.
-programToMonitorLaunchCount	Specify the nth invocation of the programToMonitor which is to have its data collected.

-refreshActiveObjects	Automatically refresh the Active Objects tab once a test is complete.
-refreshQuery	Automatically refresh the Query tab once a test is complete.
-refreshCoverage	Automatically refresh the Coverage tab once a test is complete.
-refreshActiveLocks	Automatically refresh the Active Locks tab once a test is complete.
-refreshAllLocks	Automatically refresh the All Locks tab once a test is complete.
-refreshObjectHandles	Automatically refresh the Object Handles tab once a test is complete.
-refreshLocksPerThread	Automatically refresh the Locks Per Thread tab once a test is complete.
-refreshSummary	Automatically refresh the Summary tab once a test is complete.
-refreshThreads	Automatically refresh the Threads tab once a test is complete.
-refreshWaitChain	Automatically refresh the Wait Chain tab once a test is complete.
-resetSettings	Forces Thread Validator to reset (nearly) all settings to the default state.
-saveSession	Save the session data when all data has finished being collecting from the target program.
-sessionCompareHTML -sessionCompareXML	Compare two sessions, producing an HTML or XML report detailing any regression and improvements.
-sessionLoad	Loads a previously created session to be merged with the data from the session being *recorded*
-setenvironment	Environment variables for Thread Validator, as a series of name/value pairs
-settings	Points to a previously saved settings file to be used for the test.
-showCommandPrompt	Any launched console window will be shown during the test.
-showErrorsOnCommandPrompt	If an error occurs when launching the application, the error will be reported on the command line.
-showErrorsWithMessageBox	Force errors to be displayed using a message box when running from the command line.
-waitName	Name a process that Thread Validator will wait for.

To run 32 bit thread validator run **`C:\Program Files (x86)\Software Verify\Thread Validator x86\threadValidator.exe`**

To run 64 bit thread validator run **`C:\Program Files (x86)\Software Verify\Thread Validator x64\threadValidator_x64.exe`**

4.12 Troubleshooting

Running from the command line can cause some problems, often because you can't be sure that what you put on the command line did what you thought would do.

Ensure the arguments supplied are what you expected.

-echoArgsToUser

If you are testing a console application, make sure you can see it.

-showCommandPrompt

If an errors occur when processing the command line, make sure you can see those.

-showErrorsOnCommandPrompt

-showErrorsWithMessageBox

Look on the diagnostic tab to ensure the diagnostic data collected makes sense.

If you've got -hideUI in your command line, comment it out temporarily (make it -xhideUI so that it's not recognised).

What if the tool hangs?

If you're running from the command line, most likely you'll be running from a cmd prompt, or possibly powershell.

We've only ever had one customer report a hang with any of our tools when running from the command line.

We eventually found the problem, and it wasn't with the software tool.

The problem was that they were running the tool in hidden mode (-hideUI) from a command prompt and for unknown reasons the tool would never exit.

When they added a simple change to their command the problem went away.

They added **cmd /c** to the start of their command line. This opens a new command prompt and instructs it to launch the command line and wait for it to exit.

Problem command line:

```
"c:\program files (x86)\Software Verify\Thread Validator x64\threadValidator_x64.exe" -program c:\
```

Working command line:

```
cmd /c "c:\program files (x86)\Software Verify\Thread Validator x64\threadValidator_x64.exe" -prog
```

Part V

5 Native API

The Thread Validator API

There are some features of Thread Validator that are useful to call directly from your program, including tracking of memory in custom heap managers.

Thread Validator has an API that makes this possible; just include `svlTVAPI.c and svlTVAPI.h` to your codebase. There is no library to link to, dlls to copy.

Source files

The source files can be found in the `API` directory in the Thread Validator install directory.

```
svlTVAPI.h
svlTVAPI.c
```

Just add these files to your project and build.

If you are using precompiled headers you will need to disable them for `svlTVAPI.c`.

Working with services?

If you are working with services you to attach Thread Validator to a service and to start Thread Validator, you should use the NT Service API, not the functions in this API.

All the other functions in this API can be used with applications and with services.

Unicode or ANSI?

All the API functions are provided in Unicode and ANSI variants where strings are used. We've also provided a character width neutral #define in the same fashion that the Windows.h header files do.

For example the function for naming a heap is provided as `tvSetThreadNameA()`, `tvSetThreadNameW()` with the character width neutral `tvSetThreadName()` mapping to `tvSetThreadNameW()` for unicode and `tvSetThreadNameA()` for ANSI.

In this document we're going to use `TCHAR` like the Window.h header files do.

Deploying on a customer machine

You can use the API without incurring any dependency on Thread Validator.

If Thread Validator is not installed on the machine the software runs on, nothing will happen.

This allows you to add the Thread Validator API to your software without need to have a separate build for use with Thread Validator.

Loading the Profiler

For most use cases won't need to load the profiler, as the profiler will have been loaded when your launched your program from Thread Validator, or when you injected into your program using Inject or Wait For Application.

However if you're running your program from outside of Thread Validator and want to load the profiler from inside your program you can use `tvLoadProfiler()` to do that. You'll then need to call `tvStartProfiler()` to start it.

Starting the Profiler

To start the profiler from your API code you need to call the function `tvStartProfiler()` from your code before you call any API functions. Ideally you should call this function as early in your program as possible.

If you prefer to start the profiler from the user interface or command line you can omit the startProfiler() call. You can leave it present if you wish to start Thread Validator from your program.

Naming threads

You can name threads using the `tvSetThreadNameA()` and `tvSetThreadNameW()` functions.

Turning data collection on and off

You can turn data collection on and off using the `tvSetCollect()` function.

You can use `tvGetCollect()` to inspect the data collection status.

5.1 Native API Reference

Unicode or ANSI?

All the API functions are provided in Unicode and ANSI variants where strings are used. We've also provided a character width neutral #define in the same fashion that the Windows.h header files do.

For example the function for naming a heap is provided as `tvSetThreadNameA()`, `tvSetThreadNameW()` with the character width neutral `tvSetThreadName()` mapping to `tvSetThreadNameW()` for unicode and `tvSetThreadNameA()` for ANSI.

In this document we're going to use `TCHAR` like the Window.h header files do.

All the API functions are declared as `extern "C"`, so they can be used by C users and C++ users.

To use these functions #include `svlTVAPI.h` into your code.

tvLoadProfiler

Loads the profiler DLL into memory, but does not start the profiler.

Use this for:
32 bit applications with a 32 bit Thread Validator GUI
64 bit applications with a 64 bit Thread Validator GUI

For most use cases won't need to load the profiler, as the profiler will have been loaded when your launched your program from Thread Validator, or when you injected into your program using Inject or Wait For Application.

However if you're running your program from outside of Thread Validator and want to load the profiler from inside your program you can use `tvLoadProfiler()` to do that. You'll then need to call `tvStartProfiler()` to start it.

```
extern "C"
int tvLoadProfiler();

 Returns:
 TRUE   Successfully loaded TV DLL into target application.
 FALSE  Failed to load the TV DLL into target application.
        Check that the PATH environment variable points to the Thread Validator
 install directory contains svlThreadValidatorStub*.dll.
```

Do not use this function if you are working with services, use the ***NT Service API***.

tvLoadProfiler6432

Loads the profiler DLL into memory, but does not start the profiler.

Use this for:
32 bit applications with a 64 bit Thread Validator GUI

For most use cases won't need to load the profiler, as the profiler will have been loaded when your launched your program from Thread Validator, or when you injected into your program using Inject or Wait For Application.

However if you're running your program from outside of Thread Validator and want to load the profiler from inside your program you can use `tvLoadProfiler6432()` to do that. You'll then need to call `tvStartProfiler()` to start it.

```
extern "C"
int pvLoadProfiler6432();
```

```
Returns:
TRUE   Successfully loaded TV DLL into target application.
FALSE  Failed to load the TV DLL into target application.
        Check that the PATH environment variable points to the Thread Validator
    install directory contains svlThreadValidatorStub*.dll.
```

Do not use this function if you are working with services, use the ***NT Service API***.

tvStartProfiler

To start the profiler from your API code you need to call the function `tvStartProfiler()` from your code before you call any API functions. Ideally you should call this function as early in your program as possible.

```
extern "C"
int tvStartProfiler();

Returns:
TRUE   Successfully started TV profiler.
FALSE  Failed to start the TV profiler.
```

If you prefer to start the profiler from the user interface or command line you can omit the `tvStartProfiler()` call. You can leave it present if you wish to start Thread Validator from your program.

Do not use this function if you are working with services, use the ***NT Service API***.

tvSetThreadName()

Sets the name of a thread.

```
extern "C"
void tvSetThreadName(DWORD            threadID
                     const TCHAR      *name);
```

tvSetThreadNameA()

Sets the name of a thread.

```
extern "C"
void tvSetThreadNameA(DWORD           threadID
                      const char      *name);
```

tvSetThreadNameW()

Sets the name of a thread.

```
extern "C"
void tvSetThreadNameW(DWORD          threadID
                      const wchar_t  *name);
```

tvSetCollect()

Enables or disables data collection - i.e. whether data is sent to Thread Validator from your target application.

```
extern "C"
void tvSetCollect(int enable); // TRUE to enable, FALSE to disable
```

tvGetCollect()

Returns whether data collection is on.

```
extern "C"
int tvGetCollect();           // Returns TRUE or FALSE
```

5.2 C# API

The C# API is a wrapper around the native API.

For all of these APIs see the native API for more details.

Adding the API to your application

The C# API is provided as a source code `svlTVAPI.cs` file that you add to your application. The source file is in the API directory in the Thread Validator install directory.

The C# API does not add any dependencies to your application - if Thread Validator is present the API functions work, if Thread Validator is not present the API functions do nothing.

The C# API

The C# API is implemented by the `ThreadValidator` class in the `SoftwareVerify` namespace.

collectOn()

Turn data collection on.

```
public static void collectOn();
```

collectOff()

Turn data collection off.

```
public static void collectOff();
```

setCollect()

Turn data collection on or off.

```
public static void setCollect(bool enable);
```

getCollect()

Determine if data collection is turned on or off.

```
public static bool getCollect();
```

5.3 Calling the API via GetProcAddress

Calling API functions using GetProcAddress

If you don't want to use the svlTVAPI.c/h files you can use `GetProcAddress` to find the interface functions in the Thread Validator DLL.

The interface functions have different names and do not use C++ name mangling, but have identical parameters to the API functions.

To determine the function name take any native API name, replace the leading **tv** with **api**. For example `tvSetThreadNameW()` becomes `apiSetThreadNameW();`

Example usage

```
    typedef void (*tvSetThreadNameW_FUNC)(const TCHAR   *name);
```

```
HMODULE getValidatorModule()
{
   HMODULE   hModule;

   hModule = GetModuleHandle(_T("svlThreadValidatorStub6432.dll"));           // 32 bit DLL wi
   if (hModule == NULL)
      hModule = GetModuleHandle(_T("svlThreadValidatorStub_x64.dll"));  // 64 bit DLL with 64 
   if (hModule == NULL)
      hModule = GetModuleHandle(_T("svlThreadValidatorStub.dll"));             // 32 bit DLL wi

   return hModule;
}

HMODULE   hMod;

// get module, will only succeed if Thread Validator launched this app or is injected into thi

hMod = getValidatorModule();
if (hMod != NULL)
{
    // TV is present, lookup the function and call it to set the current thread's name

    tvSetThreadNameW_FUNC   pFunc;

    // "apiSetThreadNameW" is equivalent to linking against "tvSetThreadNameW"

    pFunc = (tvSetThreadNameW_FUNC)GetProcAddress(hMod, "apiSetThreadNameW");
    if (pFunc != NULL)
    {
        (*pFunc)(threadName);
    }
}
```

API functions and their GetProcAddress names

For any API functions not listed, try looking up the name in `svlThreadValidatorStub.dll` using [depends.exe]() or [PE File Browser]().

⊟ Show API functions and GetProcAddress names

API Name	GetProcAddress() Name
tvLoadProfiler	
tvIsValidatorPresent	
tvStartProfiler	
tvSetThreadNameA	apiSetThreadNameA
tvSetThreadNameW	apiSetThreadNameW
tvSetCollect	apiSetCollect
tvGetCollect	apiGetCollect
tvShutdownThreadVali	apiShutdownThreadValidator

Other exported functions

You may see some other functions exported from `svlThreadValidatorStub.dll(_x64).dll`.

These other functions are for Thread Validator's use. Using them may damage memory locations and/or crash your code. Best not to use them!

5.4 Convenience functions

Convenience functions

One convenience function is provided that will start the Thread Validator GUI (if it is not already running), then load the Thread Validator execution tracer into your process and start tracing it.

```
extern "C"
int loadValidatorIntoApplication();

Returns:
TRUE   Successfully loaded TV DLL into target application and successfully
started the profiler.
FALSE   Failed to load the TV DLL or failed to start the profiler.
```

To use this function #include `loadValidatorIntoApplication.h` into your code.

The source files can be found in the `API` directory in the Thread Validator install directory.

```
loadValidatorIntoApplication.h
loadValidatorIntoApplication.c
```

Just add these files to your project and build.

Part VI

6 Working with IIS and Services

When working with NT services your account must have the appropriate privileges described in the User Permissions topic.

Attaching to your service

To use Thread Validator with NT Services you need to link a small library to your application and call two functions in the library.

The NT Service API

The NT Service API is provided to enable Thread Validator to work with services.

The API works just as well with normal applications, and the same considerations outlined here also apply generally.

When the NT Service API is used, source code symbols are acquired in the stub and sent to the Thread Validator user interface.

Monitoring the service

When working with Thread Validator and services using the NT Service API you don't start the service using Thread Validator.

Instead, you start the service the way you normally start the service - e.g. with the service control manager.

The code that you have embedded into your service then contacts Thread Validator, which you should have running *before* starting the service.

Once you've exercised your service and stopped it, Thread Validator will show the usual thread data.

Examples and help

We provide some Example Service Source Code to demonstrate how to embed the service code into your service.

If you have problems using Thread Validator with services, please contact us at support@softwareverify.com.

6.1 NT Service API

The Thread Validator stub service libraries

The NT Service API is very simple, consisting of functions to load, start and unload the Thread Validator DLL.

We have also provided some debugging functions to help you debug the implementation of the NT Service API because getting data into and out of services is not always straightforward.

The stub service libraries used for this are shown in the following table:

	32 bit Thread Validator	64 bit Thread Validator

32 bit service	`svlTVStubService.lib`
	`svlTVStubServiceMD.lib`
	`svlTVStubServiceMT.lib`
		`svlTVStubService6432.lib`

64 bit service	N/A	`svlTVStubService_x64.lib`
		`svlTVStubServiceMD_x64.lib`
		`svlTVStubServiceMT_x64.lib`

All the functions exported from these libraries are exported as `extern "C"` so that C and C++ users can use them.

Library name suffixes

The MD suffix indicates the library was built with the /MD compiler switch.
The MT suffix indicates the library was built with the /MT compiler switch.

Directory Name: 2010 or 2012?

Visual Studio 6 to Visual Studio 2010
If you are using Visual Studio 2010 or earlier, use libraries from a directory with 2010 in the directory name.

Visual Studio 2010 to Visual Studio 2022
If you are using Visual Studio 2012 or later, use libraries from a directory with 2012 in the directory name.

Header files

The header files can be found in the `svlTVStubService` directory in the Thread Validator install directory.

The headers file provide an error enumeration and the NT Service API functions.

```
svlTVStubService.h
svlServiceError.h
```

Linker Problems

Some linkers cannot link the stub service library file. If you have this problem see What do I do if I cannot use svlTVStubService.lib?

Loading the Thread Validator DLL into your service

To load the Thread Validator stub dll `svlThreadValidatorStub(_x64).dll` into your service, call `svlTVStub_LoadThreadValidator()`, *not* `LoadLibrary()`.

If you are monitoring a 32 bit service with the 64 bit Thread Validator user interface you should use `svlTVStub_LoadThreadValidator6432()`.

Shutting down the Thread Validator DLL from your service.

To shutdown Thread Validator's monitoring of the service , call `svlTVStub_ShutdownThreadValidator()`.

This sends the shutting down notification and removes any hooks for your process.

Calling this function is optional. You can stop your service without calling this function.

Unloading the Thread Validator DLL from your service.

To unload the Thread Validator stub dll, call `svlTVStub_UnloadThreadValidator()`, `not FreeLibrary()`.

Calling this function is optional. You can stop your service without calling this function.

Setting a service notification callback

Once you have successfully loaded the Thread Validator DLL you can setup a service callback so that the service control manager can be kept updated during the process of starting the validator.

When a service is starting, Windows requires the service to inform the Service Control Manager (SCM) that is starting at least every ten seconds.

Failure to do so results in Windows concluding that the service has failed to start, and the service is terminated.

Instrumenting your service may well take *more* than 10 seconds, depending on the complexity and size of your service.

The solution is for *Thread Validator* to periodically call a user supplied callback from which you can regularly inform the SCM of the appropriate status.

You can set the service callback with `svlTVStub_SetServiceCallback(callback, userParam)`.

Usage

Here is an example callback which ignores the userParam.

```
void serviceCallback(void   *userParam)
{
    static DWORD dwCheckPoint = 1;

    ssStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    ssStatus.dwServiceSpecificExitCode = 0;

    ssStatus.dwControlsAccepted = 0;

    ssStatus.dwCurrentState = dwCurrentState;
    ssStatus.dwWin32ExitCode = dwWin32ExitCode;
    ssStatus.dwWaitHint = dwWaitHint;

    ssStatus.dwCheckPoint = dwCheckPoint++;

    // Report the status of the service to the service control manager.

    return SetServiceStatus(sshStatusHandle, &ssStatus);
}
```

Starting Thread Validator DLL in your service

To start Thread Validator detecting deadlocks in your service call `svlTVStub_StartThreadValidator`.

Starting Thread Validator DLL in IIS

To start Thread Validator detecting deadlocks in IIS call `svlTVStub_StartThreadValidatorForIIS()`.

Setting a filename for all logging data to be written to

To set the filename for all debugging/logging information to be written to call `svlTVStub_setLogFileName()`.

Deleting the logfile

To delete the log file call `svlTVStub_deleteLogFile()`.

Writing text to the logfile

To write a standard ANSI character string to the log file call `svlTVStub_writeToLogFileA(text)`. The ANSI string will be converted to Unicode prior to writing to the log file.

To write a Unicode character string to the log file call `svlTVStub_writeToLogFileW(text)`.

Writing error code descriptions to the logfile

To write a human readable description of the SVL_SERVICE_ERROR error code to the log file call `svlTVStub_writeToLogFile(errCode)`.

Writing LastError code descriptions to the logfile

To write a human readable description of the Windows error code to the log file call `svlTVStub_writeToLogFileLastError(errCode)`.

Dumping the PATH environment to the logfile

To write the contents of the PATH environment variable to the log file call `svlTVStub_dumpPathToLogFile()`.

This can be useful if you want to know what the search path is when trying to debug why a DLL wasn't found during an attempt to load the Validator DLL.

6.1.1 Changes to the NT Service API

API changes - February 2018

To make the API easier to use with services we made the following changes:

- Changed the API by adding many debugging functions to allow you to easily log information.

- We also extended the error enumeration to provide additional error status values.

- We also split the function of loading and starting Thread Validator into two functions - a *load* function and a *start* function.

- We split the functionality so that you could setup a service callback prior to calling the *start* function.

  The service callback allows the service control manager to be informed that the service is still active during time consuming operations, such as starting the Thread Validator when the service is non-trivial in scope.

  Failure to inform the service control manager results in the service being killed by the service control manager because it thinks the service has hung.

  This change in the API is to ensure you get better results from using our software.

What do you need to do to move from the old API to the new API?

Change all SVL_ERROR declarations to SVL_SERVICE_ERROR.

Your previous startup code probably looked like this:

```
SVL_ERROR errCode;

errCode = svlTVStub_LoadThreadValidator();
```

Change it to this:

```
SVL_SERVICE_ERROR errCode;

errCode = svlTVStub_LoadThreadValidator();

errCode = svlTVStub_SetServiceCallback(serviceCallback, NULL);

errCode = svlTVStub_StartThreadValidator();
```

The serviceCallback would look something like this:

```
void serviceCallback(void   *userParam)
{
    static DWORD dwCheckPoint = 1;

    ssStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    ssStatus.dwServiceSpecificExitCode = 0;

    ssStatus.dwControlsAccepted = 0;

    ssStatus.dwCurrentState = dwCurrentState;
    ssStatus.dwWin32ExitCode = dwWin32ExitCode;
    ssStatus.dwWaitHint = dwWaitHint;
    ssStatus.dwCheckPoint = dwCheckPoint++;

    // Report the status of the service to the service control manager.

    return SetServiceStatus(sshStatusHandle, &ssStatus);
}
```

In the code above we have omitted error handling. To see how to use the new logging function with error handling pleas

Important.

Once your service is running (rather than starting) your service callback should set the appropriate running status `SERVICE_RUNNING` rather than `SERVICE_START_PENDING`.

```
if (!ReportStatusToSCMgr(SERVICE_RUNNING,       // service state
                         NO_ERROR,              // exit code
                         0))                    // wait hint
{
   dwErr = GetLastError();
   if (bLogging)
      svlTVStub_writeToLogFileW(L"ReportStatusToSCMgr:5\r\n");
   goto cleanup;
}
```

An alternative solution is to prevent the service callback from being called once the service status has been set to running.

```
   svlTVStub_SetServiceCallback(NULL, NULL);.
```

6.1.2 NT Service API Reference

The API consists of the following functions.

SVL_SERVICE_ERROR Enumeration

```
typedef enum _svlServiceError
{
   SVL_OK,                                                    // Normal behaviour
   SVL_ALREADY_LOADED,                                        // Stub DLL already loaded into serv
   SVL_LOAD_FAILED,                                           // Failed to load stub DLL into serv
   SVL_FAILED_TO_ENABLE_STUB_SYMBOLS,                         // Loaded DLL, but failed to enable
   SVL_NOT_LOADED,                                            // Couldn't unload DLL because DLL n
   SVL_FAIL_UNLOAD,                                           // Couldn't unload DLL because could
   SVL_FAIL_TO_CLEANUP_INTERNAL_HEAP,                         // Couldn't get the internal stub he
   SVL_FAIL_MODULE_HANDLE                                     // Couldn't get the stub DLL handle
   SVL_FAIL_SETSERVICECALLBACK,                               // Couldn't call the set service cal
   SVL_FAIL_COULD_NOT_FIND_ENTRY_POINT,                       // Couldn't find the DLL entry point
   SVL_FAIL_TO_START,                                         // Failed to start the Validator
   SVL_FAIL_SETSERVICECALLBACKTHRESHOLD,                      // Couldn't call the set service cal
   SVL_FAIL_PATHS_DO_NOT_MATCH,                               // Path to service in env vars doesn
   SVL_FAIL_INCORRECT_PRODUCT_PREFIX,                         // Wrong validator
   SVL_FAIL_X86_VALIDATOR_FOUND_EXPECTED_X64_VALIDATOR,       // Found wrong bit depth validator
   SVL_FAIL_X64_VALIDATOR_FOUND_EXPECTED_X86_VALIDATOR,       // Found wrong bit depth validator
   SVL_FAIL_DID_YOU_MONITOR_A_SERVICE_FROM_VALIDATOR,         // Looks like Monitor A Service wasn
   SVL_FAIL_ENV_VAR_NOT_FOUND,                                // Env Var not found
   SVL_FAIL_VALIDATOR_ENV_VAR_NOT_FOUND,                      // Env Var identifying validator not
   SVL_FAIL_VALIDATOR_ID_NOT_SPECIFIED,                       // Validator process not specified
   SVL_FAIL_VALIDATOR_ID_NOT_A_PROCESS,                       // Validator process identified does
   SVL_FAIL_VALIDATOR_NOT_FOUND,                              // Validator process identified does
} SVL_SERVICE_ERROR;
```

svlTVStub_LoadThreadValidator

```
extern "C"
SVL_SERVICE_ERROR svlTVStub_LoadThreadValidator();
```

To load the Thread Validator stub `svlThreadValidatorStub.dll` into your service, use `svlTVStub_LoadThreadValidator()`, *not* `LoadLibrary()`.

This loads the DLL and sets up a few internal variables in the DLL to ensure that symbols are sent from the stub to the Thread Validator user interface.

This is necessary because the Thread Validator user interface can't open a process handle to a service and so is unable to get symbols from the process.

To solve this, symbols are sent from the stub to the user interface as needed.

If you just call `LoadLibrary()` on the DLL, symbols will *not* be sent to the Thread Validator user interface and you won't get meaningful function names in your stack traces.

This function can be used when monitoring:

- 32 bit services or applications with Thread Validator
- 64 bit services or applications with Thread Validator x64

If you are monitoring 32 bit applications with Thread Validator x64 you should use `svlTVStub_LoadThreadValidator6432()`.

Which function you should call is shown in the table below.

	32 bit Thread Validator	64 bit Thread Validator
32 bit service	`svlTVStub_LoadThreadValidator()`	`svlTVStub_LoadThreadValidator6432()`
64 bit service	N/A	`svlTVStub_LoadThreadValidator()`

svlTVStub_LoadThreadValidator6432

```
extern "C"
SVL_SERVICE_ERROR svlTVStub_LoadThreadValidator6432();
```

To load the Thread Validator stub `svlThreadValidatorStub6432.dll` into your service, use `svlTVStub_LoadThreadValidator6432()`, *not* `LoadLibrary()`.

This loads the DLL and sets up a few internal variables in the DLL to ensure that symbols are sent from the stub to the Thread Validator user interface.

This is necessary because the Thread Validator user interface can't open a process handle to a service and so is unable to get symbols from the process.

To solve this, symbols are sent from the stub to the user interface as needed.

If you just call `LoadLibrary()` on the DLL, symbols will *not* be sent to the Thread Validator user interface and you won't get meaningful function names in your stack traces.

This function should only be used when monitoring 32 bit services or applications with Thread Validator x64.

svlTVStub_StartThreadValidator

```
extern "C"
SVL_SERVICE_ERROR svlTVStub_StartThreadValidator();
```

To start Thread Validator inspecting the service call `svlTVStub_StartThreadValidator()`.

svlTVStub_StartThreadValidatorForIIS

```
extern "C"
SVL_SERVICE_ERROR svlTVStub_StartThreadValidatorForIIS();
```

To start Thread Validator inspecting IIS call `svlTVStub_StartThreadValidatorForIIS()`.

Example usage.

svlTVStub_ShutdownThreadValidator

```
extern "C"
SVL_SERVICE_ERROR svlTVStub_ShutdownThreadValidator();
```

To stop Thread Validator inspecing the service call `svlTVStub_Shutdown`Thread`Validator()`.

This sends the shutting down notification and removes any hooks for your process.

Calling this function is optional. ***You can stop your service without calling this function.***

svlTVStub_UnloadThreadValidator

```
extern "C"
SVL_SERVICE_ERROR svlTVStub_UnloadThreadValidator();
```

To unload Thread Validator call `svlTVStub_UnloadThreadValidator()`, *do not call* `FreeLibrary()`.

Calling this function is optional. ***You can stop your service without calling this function.***

svlTVStub_SetServiceCallback

```
extern "C"
SVL_SERVICE_ERROR svlTVStub_SetServiceCallback(serviceCallback_FUNC callback,
                                               void*                userParam);
```

`svlTVStub_SetServiceCallback` is used to setup a service callback that is used to inform the Windows service con

`userParam` is a value you can supply which will then be passed to the callback every time the callback is called during instrumentation.

Why is a service callback needed?

When a service is starting, Windows requires the service to inform the Service Control Manager (SCM) that is starting at least every ten seconds.

Failure to do so results in Windows concluding that the service has failed to start, and the service is terminated.

Instrumenting your service may well take *more* than 10 seconds, depending on the complexity and size of your service.

The solution is for *Thread Validator* to periodically call a user supplied callback from which you can regularly inform the SCM of the appropriate status.

We strongly recommend that you setup a service callback. Not setting a service callback can result in failure of your

Debugging functions

The following functions are provided to help you log information about the progress, success or failure of the NT Service API attaching Thread Validator to your service.

We strongly recommend that you use these logging functions so that you can understand why Thread Validator might fail to connect to a service.

To see example usage of these debugging functions please look in `service.cpp` in the `examples\service` directory in the Thread Validator install directory.

svlTVStub_setLogFileName

```
extern "C"
void svlTVStub_setLogFileName(const wchar_t* fileName);
```

Call `svlTVStub_setLogFileName` to set the name of the filename used for logging.

This function must be called before you can use any of the other debugging functions.

Setting this filename also sets the filename used by some of these API functions - you will find additional logging data from those functions that will help debug any issues with the service.

svlTVStub_deleteLogFile

```
extern "C"
void svlTVStub_deleteLogFile();
```

This function deletes the log file.

svlTVStub_writeToLogFileA

```
extern "C"
void svlTVStub_writeToLogFileA(const char* text);
```

This function writes a standard ANSI character string to the log file.

The ANSI string will be converted to Unicode prior to writing to the log file.

svlTVStub_writeToLogFileW

```
extern "C"
void svlTVStub_writeToLogFileW(const wchar_t* text);
```

This function writes a Unicode character string to the log file.

svlTVStub_writeToLogFile

```
extern "C"
void svlTVStub_writeToLogFile(SVL_SERVICE_ERROR errCode);
```

This function writes a human readable description of the SVL_SERVICE_ERROR error code to the log file.

svlTVStub_writeToLogFileLastError

```
extern "C"
void svlTVStub_writeToLogFileLastError(DWORD errCode);
```

This function writes a human readable description of the Windows error code to the log file.

The errCode parameter is the error code returned from GetLastError().

svlTVStub_dumpPathToLogFile

```
extern "C"
void svlTVStub_dumpPathToLogFile();
```

This function writes the contents of the PATH environment variable to the log file.

This can be useful if you want to know what the search path is when trying to debug why a DLL wasn't found during an attempt to load the Validator DLL.

6.1.3 Troubleshooting

Troubleshooting - Service fails to start

If a service takes too long to start the service control manager kills the service.

The way to stop this is for a service to call `ReportStatusToSCMgr()` to tell the service control manager that the service is still OK.

Thread Validator can't do this for you as the call requires some data from any earlier call you have made.

The solution is that you provide a callback using `svlTVStub_SetServiceCallback()` that Thread Validator can call during the process of attaching to the service, and you can call the appropriate function.

Example code to set the callback:

```
       errCode = svlTVStub_SetServiceCallback(serviceCallback,              // the
callback
                                              NULL);                        // some
user data (we don't have any, so set NULL)
       if (bLogging)
       {
              if (errCode != SVL_OK)
              {
                     svlTVStub_writeToLogFileW(L"Setting service callback failed.
\r\n");
                     svlTVStub_writeToLogFile(errCode);
              }

              svlTVStub_writeToLogFileW(L"Starting Thread Validator\r\n");
       }
```

Example callback:

```
static void serviceCallback(void  *userParam)
{
       // just tell the Service Control Manager that we are still busy
       // in this example userParam is not used
       //
       // note that prior to the Validator loading it's DLL ssStatus.dwCurrentState
must have been initialised, most likely to SERVICE_START_PENDING
       // you could pass a fixed value here, but it would need to change once the
service has finished starting up so that you don't unintentionally change the service
state
       // when this callback is called. This callback is called whenever
instrumentation happens (when a DLL is loaded). Thus you can't assume this is only
called during service startup,
       // it may also get called later in the service lifetime.

       ReportStatusToSCMgr(ssStatus.dwCurrentState,    // service state
                           NO_ERROR,                  // exit code
                           3000);                     // wait hint
}
```

We strongly recommend that you set a service callback. It won't harm your program and it will remove any likelihood of your service being killed by the service control manager.

Troubleshooting - Service starts, Thread Validator gets no data

If you have problems getting Thread Validator to monitor your service you'll need to find out what's failing.

Until Thread Validator loads correctly and successfully connects to the graphical user interface you have no way of knowing what is happening.

The solution is to set a log file that Thread Validator can write status messages to. You can also write your own status messages to this log file.

Set the log file using `svlTVStub_setLogFileName`. Write to it using `svlTVStub_writeToLogFile()`, `svlTVStub_writeToLogFileA()`, `svlTVStub_writeToLogFileW()`.

Then when things are not working as expected take a look at the log file to see the errors. The Thread Validator will often suggest what the problem is.

We strongly recommend that you configure the log file and use it when working with services. It has saved us a lot of time.

6.2 Working with IIS

Configuring IIS for use with ISAPI

We assume that you are familiar with IIS. This is not a topic we can provide advice for.

That said, we wrote a blog article about configuring IIS for use with ISAPI.

Example ISAPI

We have provided an example ISAPI extension configured for use with Thread Validator.

This example is provided as source code and project files. You will need to build it yourself, you may need to change an include path to find the appropriate headers. The resulting ISAPI will need to be copied to your website for testing and the website configured to allow the ISAPI to execute (please see the above mentioned blog article for details on that).

You can find the example ISAPI in the **isapiExample** folder in the Thread Validator installation directory.

Using Thread Validator with IIS

IIS is a service application. It runs as one of the more restricted applications on Microsoft Windows.

Thread mapped files created by IIS cannot be opened by user mode programs (Thread Validator, for example). DLLs, executables and files cannot be opened by IIS except if they are in directories which IIS has access to. These are security measures intended to make your computer secure from attack.

These security measures make it hard for tools like Thread Validator to work.

- We have to communicate settings information to Thread Validator via text file
- All DLLs and helper programs we want to use need to be copied to the web root (or a subdirectory within the web root) so that they can be used.
- We need to have our own data transport because our usual high speed memory mapped data transport is not available.

It's also not possible to launch IIS or inject into a running IIS instance.

The only way to work with IIS is by using the NT Service API, and using the `svlTVStub_StartThreadValidatorForIIS()` function instead of `svlTVStub_StartThreadValidator()`.

We've provide some example code to show you how to attach and detach from your ISAPI extension.

Workflow

1) Start monitoring your ISAPI by using the Monitor ISAPI dialog.

Launch menu › **IIS** menu › **Monitor ISAPI...**

2) When you have finished interacting with the web pages that use the ISAPI component shutdown IIS, wait for Thread Validator's status to indicate "Ready" and examine the results.

Launch menu › **IIS** menu › **Stop IIS**

6.3 Example Source Code

Service Example

Example demonstrating how to monitor a service.

Also see the example service that ships with Thread Validator.

You can find this in the `\examples\service` directory in the Thread Validator install directory.

Also see the example service and child process that ships with Thread Validator.

You can find this in the `\examples\serviceWithAChildProcess` directory in the Thread Validator install directory.

IIS Example

Example demonstrating how to monitor an ISAPI DLL.

Also see the example ISAPI DLL that ships with Thread Validator.

You can find this in the `\examples\isapiExample` directory in the Thread Validator install directory.

6.3.1 Example Service Source Code

Where to put your code

When you use the functions to load and unload Thread Validator from your service, it is important that you put the function calls in the correct place in your software.

The correct place to put them is in a 'balanced' location, such that you would expect no memory leaks to occur between the load and the unload function call, assuming the service was working correctly.

Typically, this means that Thread Validator is:

- loaded as the first action in the `service_main()` function
- unloaded just before the service control manager is informed of the stopped status

The source code shown below shows an example `service_main()` function used in a service, demonstrating where to load and unload Thread Validator.

The long comment covers problems with the way services are stopped and what may be displayed in a debugger if this happens.

➡ The code is extracted from `service\service.cpp`, part of the full example of an NT service, client and a utility for controlling whether the service uses Thread Validator.

⊟ Show the C++ example service_main() function

```
void serviceCallback(void   *userParam)
{
   // just tell the Service Control Manager that we are still busy
   // in this example userParam is not used

   static DWORD dwCheckPoint = 1;

   ssStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
   ssStatus.dwServiceSpecificExitCode = 0;

   ssStatus.dwControlsAccepted = 0;

   ssStatus.dwCurrentState = dwCurrentState;
   ssStatus.dwWin32ExitCode = dwWin32ExitCode;
   ssStatus.dwWaitHint = dwWaitHint;
   ssStatus.dwCheckPoint = dwCheckPoint++;

   // Report the status of the service to the service control manager.

   return SetServiceStatus(sshStatusHandle, &ssStatus);
}
```

```
//-NAME---------------------------------
// service_main
//.DESCRIPTION..........................
//
// Initializes the service, then calls the function to do the work.
// This function is typically where you will load and unload Thread Validator
//
//.PARAMETERS...........................
//
// dwArgc   - number of command line arguments
// lpszArgv - array of command line arguments
//
//.RETURN.CODES.........................
//--------------------------------------

static CRITICAL_SECTION   nativeExampleCritSect;

void WINAPI service_main(DWORD   dwArgc,
                   LPTSTR   *lpszArgv)
{
   if (bLogging)
   {
      svlTVStub_setLogFileName(SZLOGFILENAME);
      svlTVStub_deleteLogFile();
   }

   InitializeCriticalSection(&nativeExampleCritSect);
   InitializeCriticalSection(&testCritSect);

   // register our service control handler:

   sshStatusHandle = RegisterServiceCtrlHandler(TEXT(SZSERVICENAME), service_ctrl);
   if (sshStatusHandle != 0)
   {
      DWORD   dwErr = 0;

      // **TV_EXAMPLE** start

      if (bThreadValidator)
      {
         // load Thread Validator (but if monitoring a 32 bit service with Thread
Validator x64 use svlTVStub_LoadThreadValidator6432())

         if (bLogging)
         {
            svlTVStub_writeToLogFileW(_T("About to load Thread Validator\r\n"));
         }

         SVL_SERVICE_ERROR   errCode;

#ifdef IS6432
         // x86 with x64 GUI
         errCode = svlTVStub_LoadThreadValidator6432();
#else   //#ifdef IS6432
         // x86 with x86 GUI
         // x64 with x64 GUI
         errCode = svlTVStub_LoadThreadValidator();
#endif   //#ifdef IS6432
         if (bLogging)
         {
            if (errCode != SVL_OK)
            {
               DWORD   lastError;

               lastError = GetLastError();
               svlTVStub_writeToLogFileW(_T("Thread Validator load failed. \r\n"));
               svlTVStub_writeToLogFileLastError(lastError);
               svlTVStub_writeToLogFile(errCode);

               svlTVStub_dumpPathToLogFile();
            }
            else
            {
               svlTVStub_writeToLogFileW(_T("Thread Validator load success. \r\n"));
            }
         }

         // setup a service callback so that the Service Control Manager knows the
service
         // is starting up even if instrumentation takes longer than 10 seconds (which
it will
         // for a non-trivial application)

         if (bLogging)
            svlTVStub_writeToLogFileW(_T("Setting service callback Thread
Validator\r\n"));

         errCode = svlTVStub_SetServiceCallback(serviceCallback,      // the callback
                                                NULL);               // some user data
(we don't have any, so set NULL)
         if (bLogging)
         {
            if (errCode != SVL_OK)
            {
               svlTVStub_writeToLogFileW(_T("Setting service callback failed. \r\n"));
               svlTVStub_writeToLogFile(errCode);
            }

            svlTVStub_writeToLogFileW(_T("Starting Thread Validator\r\n"));
         }

         errCode = svlTVStub_StartThreadValidator();
         if (bLogging)
         {
            if (errCode != SVL_OK)
            {
               DWORD   lastError;

               lastError = GetLastError();
               svlTVStub_writeToLogFileW(_T("Starting Thread Validator failed.
\r\n"));
```

```
               svlTVStub_writeToLogFileLastError(lastError);
               svlTVStub_writeToLogFile(errCode);
            }

            svlTVStub_writeToLogFileW(_T("Finished loading Thread Validator\r\n"));
         }
      }
      else
      {
         if (bLogging)
            svlTVStub_writeToLogFileW(_T("Not using Thread Validator, DLL will not be
loaded\r\n"));
      }

      // **TV_EXAMPLE** end

      // SERVICE_STATUS members that don't change in example

      ssStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
      ssStatus.dwServiceSpecificExitCode = 0;

      // report the status to the service control manager.

      if (ReportStatusToSCMgr(SERVICE_START_PENDING, // service state
                              NO_ERROR,              // exit code
                              3000))                 // wait hint
      {
         // deliberately enter a critical section so that we can see that with Thread
Validator

         EnterCriticalSection(&nativeExampleCritSect);

         // do work

         dwErr = ServiceStart(dwArgc, lpszArgv);

         // finished doing work

         LeaveCriticalSection(&nativeExampleCritSect);
      }

      // **TV_EXAMPLE** start

      if (bThreadValidator)
      {
         // unload Thread Validator here
         // IMPORTANT.
         // Because of the way services work, you can find that this thread which is
trying to gracefully unload
         // ThreadValidator is ripped from under you by the operating system. This
prevents Thread Validator from
         // removing all its hooks successfully. If Thread Validator does not remove
all of its hooks successfully
         // because this happens, then you may get a crash when the service stops.
         //
         // An alternative fix is to spawn another thread which then unloads Thread
Validator.
         // See the code for ServiceStop() for comments relating to this.
         //
         // A callstack for such a crash is shown below. If you see this type of crash
you need to put you code to
         // unload Thread Validator somewhere else. The stack trace may be different,
but a fundamental point is the
         // code calling through doexit(), exit() and ExitProcess()
         //
         //NTDLL! 77f64e70()
         //SVLTHREADVALIDATORSTUB!
         //MSVCRT! 78001436()
         //MSVCRT! 7800578c()
         //DBGHELP! 6d55da25()
         //DBGHELP! 6d55de83()
         //DBGHELP! 6d53705d()
         //DBGHELP! 6d51cc69()
         //DBGHELP! 6d51f6e8()
         //DBGHELP! 6d524ebf()
         //DBGHELP! 6d52a7b0()
         //DBGHELP! 6d52b00a()
         //DBGHELP! 6d526487()
         //DBGHELP! 6d5264d7()
         //DBGHELP! 6d5264f7()
         //SVLTHREADVALIDATORSTUB!
         //SVLTHREADVALIDATORSTUB!
         //SVLTHREADVALIDATORSTUB!
         //SVLTHREADVALIDATORSTUB!
         //SVLTHREADVALIDATORSTUB!
         //SVLTHREADVALIDATORSTUB!
         //SVLTHREADVALIDATORSTUB!
         //SVLTHREADVALIDATORSTUB!
         //MSVCRT! 78001436()
         //MSVCRT! 780057db()
         //KERNEL32! 77f19fdb()
         //SVLTHREADVALIDATORSTUB! ExitProcess hook
         //doexit(int 0x00000000, int 0x00000000, int 0x00000000) line 392
         //exit(int 0x00000000) line 279 + 13 bytes
         //mainCRTStartup() line 345
         //KERNEL32! 77f1b9ea()

         svlTVStub_UnloadThreadValidator();
      }

      // **TV_EXAMPLE** end

      // try to report the stopped status to the service control manager.

      (VOID)ReportStatusToSCMgr(SERVICE_STOPPED, dwErr, 0);
   }

   DeleteCriticalSection(&testCritSect);
   DeleteCriticalSection(&nativeExampleCritSect);
   return;
}




void WINAPI service_main(DWORD dwArgc, LPTSTR *lpszArgv)
{
       // load Thread Validator here

       svlTVStub_LoadThreadValidator();

       // register our service control handler:

       sshStatusHandle = RegisterServiceCtrlHandler(TEXT(SZSERVICENAME),
service_ctrl);

       // SERVICE_STATUS members that don't change in example

       ssStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
       ssStatus.dwServiceSpecificExitCode = 0;

       // report the status to the service control manager.

       if (!ReportStatusToSCMgr(SERVICE_START_PENDING, NO_ERROR, 3000))
       {
              if (sshStatusHandle)
                     ReportStatusToSCMgr(SERVICE_STOPPED, dwErr, 0);
       }

       // do the work of the service

       doWork(dwArgc, lpszArgv);

       // unload Thread Validator here
       // IMPORTANT.
       // Because of the way services work, you can find that this thread which is
trying to gracefully unload
       // ThreadValidator is ripped from under you by the operating system. This
prevents Thread Validator from
       // removing all its hooks successfully. If Thread Validator does not remove all
of its hooks successfully
       // because this happens, then you may get a crash when the service stops.
       //
       // A callstack for such a crash is shown below. If you see this type of crash
you need to put your code to
       // unload Thread Validator somewhere else. The stack trace may be different,
but a fundamental point is the
       // code calling through doexit(), exit() and ExitProcess()
       //
```

```
        //NTDLL! 77f64e70()
        //SVLTHREADVALIDATORSTUB!
        //MSVCRT! 78001436()
        //MSVCRT! 7800578c()
        //DBGHELP! 6d55da25()
        //DBGHELP! 6d55de83()
        //DBGHELP! 6d53705d()
        //DBGHELP! 6d51cc69()
        //DBGHELP! 6d51f6e8()
        //DBGHELP! 6d524ebf()
        //DBGHELP! 6d52a7b0()
        //DBGHELP! 6d52b00a()
        //DBGHELP! 6d526487()
        //DBGHELP! 6d5264d7()
        //DBGHELP! 6d5264f7()
        //SVLTHREADVALIDATORSTUB!
        //SVLTHREADVALIDATORSTUB!
        //SVLTHREADVALIDATORSTUB!
        //SVLTHREADVALIDATORSTUB!
        //SVLTHREADVALIDATORSTUB!
        //SVLTHREADVALIDATORSTUB!
        //SVLTHREADVALIDATORSTUB!
        //SVLTHREADVALIDATORSTUB!
        //MSVCRT! 78001436()
        //MSVCRT! 780057db()
        //KERNEL32! 77f19fdb()
        //SVLTHREADVALIDATORSTUB! ExitProcess hook
        //doexit(int 0x00000000, int 0x00000000, int 0x00000000) line 392
        //exit(int 0x00000000) line 279 + 13 bytes
        //mainCRTStartup() line 345
        //KERNEL32! 77f1b9ea()

        svlTVStub_UnloadThreadValidator();

        // try to report the stopped status to the service control manager.

        if (sshStatusHandle)
                ReportStatusToSCMgr(SERVICE_STOPPED, dwErr, 0);

        // tried putting the call to svlTVStub_UnloadThreadValidator(); here but often
the thread
        // was pulled from under it by the operating system

        return;
}
```

6.3.2 Example ISAPI Source Code

Where to put your code

When you use the functions to load and unload Thread Validator from your service, it is important that you put the function calls in the correct place in your ISAPI extension.

Typically, this means that Thread Validator is:

- loaded as the first action in the `GetExtensionVersion()` function of your ISAPI extension.
- unloaded in the `TerminateExtension()` function of your ISAPI extension.

Example source code

The source code shown below shows an example `GetExtensionVersion()` and an example `TerminateExtension()` used in an ISAPI, demonstrating where to load and unload Thread Validator.

This example code logs errors. We strongly recommend that you do this in your example. Because IIS is a protected process that can't communicate to the outside world except via HTTP/HTTPS when anything fails during the loading and start of Thread Validator the only means we have of communicating that failure to you is via the log file. Please use the log file, it will make debugging any mistakes very much easier, simpler and quicker than any other method.

This process is almost identical to working with a regular service, except that `svlTVStub_StartThreadValidator()` is replaced with `svlTVStub_StartThreadValidatorForIIS()`.

This example assumes the web root is located `C:\\testISAPIWebsite`

⊟ Show the C++ example ISAPI functions

```
#include "svlTVStubService.h"
#include "svlServiceError.h"

BOOL WINAPI GetExtensionVersion(HSE_VERSION_INFO *pVer)
{
       // some setup work to define what the extension is

       pVer->dwExtensionVersion = HSE_VERSION;
       strncpy(pVer->lpszExtensionDesc, "Validate ISAPI Extension",
HSE_MAX_EXT_DLL_NAME_LEN);

       // load Validator here

       svlTVStub_setLogFileName(L"C:\\testISAPIWebsite\\svl_TV_log.txt");
       svlTVStub_deleteLogFile();

       SVL_SERVICE_ERROR   errCode;
#ifdef IS6432
       // x86 with x64 GUI
       errCode = svlTVStub_LoadThreadValidator6432();
#else   //#ifdef IS6432
       // x86 with x86 GUI
```

```
    // x64 with x64 GUI
    errCode = svlTVStub_LoadThreadValidator();
#endif   //#ifdef IS6432
    if (errCode != SVL_OK)
    {
        DWORD   lastError;

        lastError = GetLastError();
        svlTVStub_writeToLogFileW(L"Thread Validator load failed. \r\n");
        svlTVStub_writeToLogFileLastError(lastError);
        svlTVStub_writeToLogFile(errCode);

        svlTVStub_dumpPathToLogFile();
    }
    else
    {
        svlTVStub_writeToLogFileW(L"Thread Validator load success. \r\n");

        errCode = svlTVStub_StartThreadValidatorForIIS();
        if (errCode != SVL_OK)
        {
            DWORD   lastError;

            lastError = GetLastError();
            svlTVStub_writeToLogFileW(L"Starting Thread Validator failed.
\r\n");
            svlTVStub_writeToLogFileLastError(lastError);
            svlTVStub_writeToLogFile(errCode);
        }

        svlTVStub_writeToLogFileW(L"Finished starting Thread Validator\r\n");
    }

    return TRUE;
}


BOOL WINAPI TerminateExtension(DWORD     dwFlags)
{
    // unload Validator here

    svlTVStub_UnloadThreadValidator();

    return TRUE;
}
```

Part VII

7 Examples

The need for examples

We know Thread Validator is a complex product, but the programs that need to be tested are often even more complex, and are certainly all different.

For this reason, it's important to be able to test and demonstrate the features of Thread Validator in an easy and repeatable way.

The example application provides a safe test demonstration:

- It lets you trigger time consuming calculations in your own time so you can observe performance hotspots
- It provides source code to demonstrate usage, correctly or otherwise!

This section has help for the example application followed by some examples of using it in conjunction with Thread Validator.

Some additional projects provide examples of using NT Services.

All example projects are supplied as source code and projects. You'll need to build the examples or services before you can use them.

7.1 Example Application

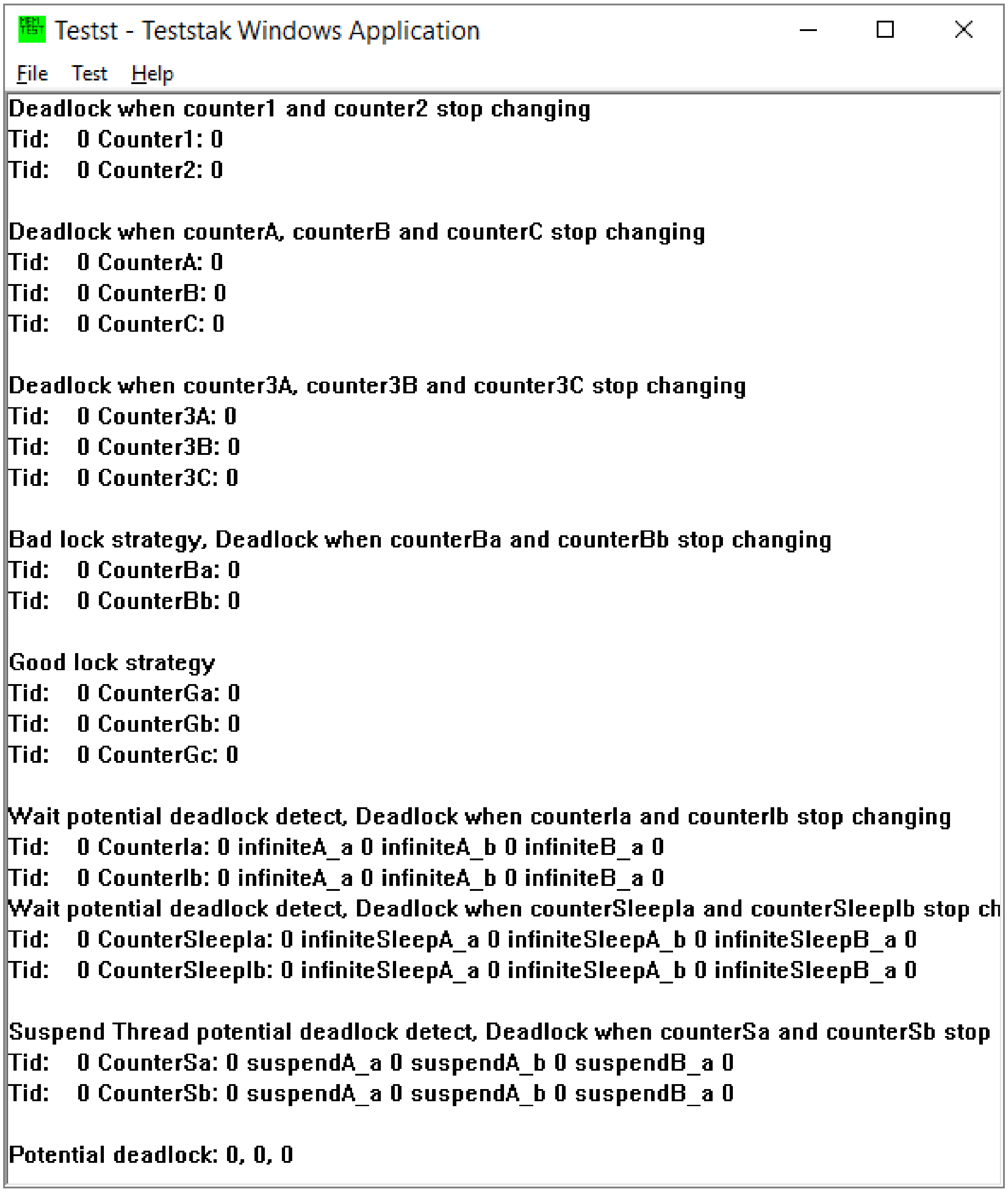
The example application

The example application is a great way to explore the capabilities of Thread Validator.

The source and projects are included in the installation, but you'll need to build the example application yourself.

You can then use nativeExample.exe in conjunction with Thread Validator to generate thread errors, waits, deadlocks and potential deadlocks, and monitor the behaviour of the application as you use it.

After launching the example application from Thread Validator, the following dialog appears showing counters reflecting the status of various tests you can run from the **Test** menu.



```
Deadlock when counter1 and counter2 stop changing
Tid:    0 Counter1: 0
Tid:    0 Counter2: 0

Deadlock when counterA, counterB and counterC stop changing
Tid:    0 CounterA: 0
Tid:    0 CounterB: 0
Tid:    0 CounterC: 0

Deadlock when counter3A, counter3B and counter3C stop changing
Tid:    0 Counter3A: 0
Tid:    0 Counter3B: 0
Tid:    0 Counter3C: 0

Bad lock strategy, Deadlock when counterBa and counterBb stop changing
Tid:    0 CounterBa: 0
Tid:    0 CounterBb: 0

Good lock strategy
Tid:    0 CounterGa: 0
Tid:    0 CounterGb: 0
Tid:    0 CounterGc: 0

Wait potential deadlock detect, Deadlock when counterIa and counterIb stop changing
Tid:    0 CounterIa: 0 infiniteA_a 0 infiniteA_b 0 infiniteB_a 0
Tid:    0 CounterIb: 0 infiniteA_a 0 infiniteA_b 0 infiniteB_a 0
Wait potential deadlock detect, Deadlock when counterSleepIa and counterSleepIb stop ch
Tid:    0 CounterSleepIa: 0 infiniteSleepA_a 0 infiniteSleepA_b 0 infiniteSleepB_a 0
Tid:    0 CounterSleepIb: 0 infiniteSleepA_a 0 infiniteSleepA_b 0 infiniteSleepB_a 0

Suspend Thread potential deadlock detect, Deadlock when counterSa and counterSb stop
Tid:    0 CounterSa: 0 suspendA_a 0 suspendA_b 0 suspendB_a 0
Tid:    0 CounterSb: 0 suspendA_a 0 suspendA_b 0 suspendB_a 0

Potential deadlock: 0, 0, 0
```

How to use these examples

The best way to understand how Thread Validator works is by example.

We recommend launching the example application from Thread Validator and observing how the menu actions affect threads, locks and synchronization objects.

Examining the source code is the best way to see what's going on in the example application.

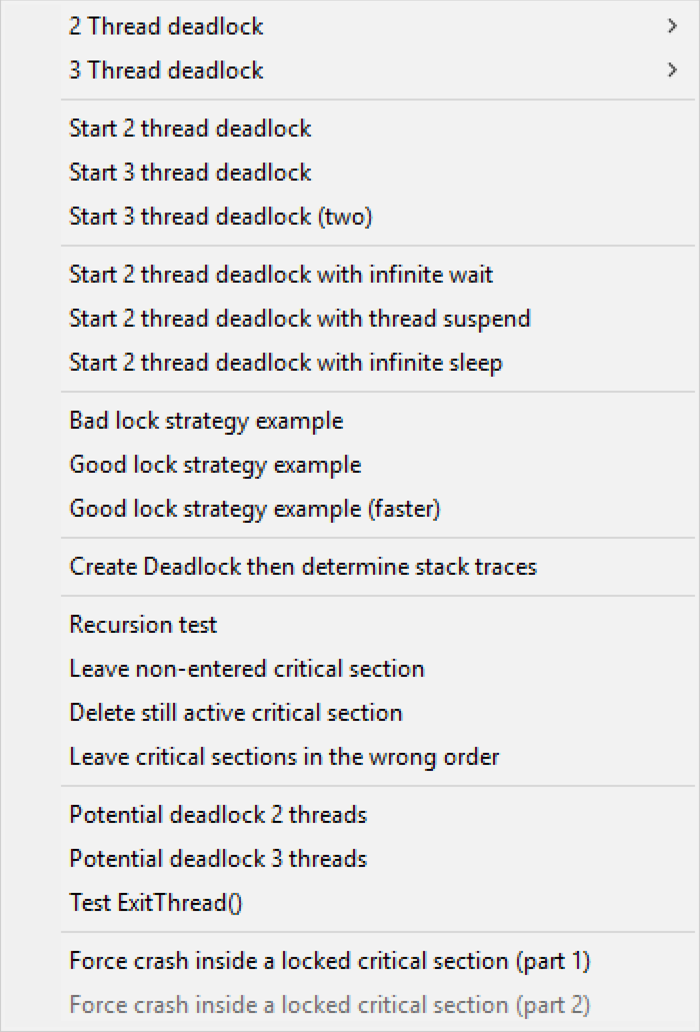
The examples create threads and engage critical sections in different scenarios. For convenience, below we have provided the source locations where each menu action runs a test.

Most test locations are in the `CTeststakView` class of `nativeExample\TESTSVW.CPP`

File menu

File menu › **Exit** › closes the example application

Test menu


2 Thread deadlock
3 Thread deadlock
Start 2 thread deadlock
Start 3 thread deadlock
Start 3 thread deadlock (two)
Start 2 thread deadlock with infinite wait
Start 2 thread deadlock with thread suspend
Start 2 thread deadlock with infinite sleep
Bad lock strategy example
Good lock strategy example
Good lock strategy example (faster)
Create Deadlock then determine stack traces
Recursion test
Leave non-entered critical section
Delete still active critical section
Leave critical sections in the wrong order
Potential deadlock 2 threads
Potential deadlock 3 threads
Test ExitThread()
Force crash inside a locked critical section (part 1)
Force crash inside a locked critical section (part 2)

Test menu › **...**

2 Thread deadlock

```
OnHandlesStartthread1()
OnHandlesStartthread2()
```

Creates two named threads that increment counter1 and counter 2 in the midst of repeatedly locking and unlocking two critical sections with an interim Sleep.

Sooner or later the two threads will deadlock and the counters being updated in the view will stop.

3 Thread deadlock

```
OnHandlesStartthreadA()
OnHandlesStartthreadB()
OnHandlesStartthreadC()
```

Creates three named threads that increment counterA,B,C in the midst of repeatedly locking and unlocking three critical sections with interim Sleeps.

Sooner or later the threads will deadlock and the counters being updated in the view will stop.

Start 2 Thread deadlock

```
OnTestStart2ThreadDeadlock()
```

Combines the above *2 Thread deadlock* operations into a single function.

Start 3 Thread deadlock

```
OnTestStart3threaddeadlock()
```

Creates three named threads that increment counter3A,3B,3C in the midst of repeatedly locking and unlocking two of three critical sections with no interim Sleeps, and with a circular dependency.

The threads will deadlock and the counters being updated in the view will stop.

Start 3 Thread deadlock (two)

```
OnTestStart3threaddeadlock2()
```

Similar to above but with interim Sleeps and no circular dependency.

The threads will still soon deadlock and the counters being updated in the view will stop.

Start 2 Thread deadlock with infinite wait

```
OnTestStart3threaddeadlockwithinfinitewait()
```

See comments in this function for an explanation.
Deadlock occurs when Counterla,lb stop incrementing.

Start 2 Thread deadlock with thread suspend

```
OnTestStart3threaddeadlockwiththreadsuspend()
```

See comments in this function for an explanation.
Deadlock occurs when CounterSa,Sb stop incrementing.

Start 2 Thread deadlock with infinite sleep

```
OnTestStart3threaddeadlockwithinfinitesleep()
```

See comments in this function for an explanation.

Help menu

Help menu › **...**

› **About Thread Validator Tester**	`nativeExampleApp::OnAppAbout()` Shows a simple information dialog using code in nativeExample.cpp

7.1.1 Building the example application

Where to find the example applications

The example projects are in the **examples** sub-directory of the Thread Validator installation directory.

If the directory is not present, reinstall your software and choose *custom* or *full* installation to include the examples.

To avoid permission issues creating some of the output files, you may need to copy the project out of the Program Files folder, if that's where it's installed.

Solutions and projects

For modern Visual Studio use examples.sln to build all the examples. Some may not build on the first pass because a dependency hasn't yet been built. Two or three builds should resolve that.

- **examples.sln** › for Microsoft® Visual Studio / .net

If you're using Visual Studio 6, build **nativeExample.dsp** and the dependent projects in **dllADependentOnB.dsp**, **dllBDependentOnC.dsp**, **dllC.dsp**.

Configurations

There are only two configurations in each project:

- **Debug Non Link** / **Release Non Link** › with the `wWinMainCRTStartup` unicode entry point

Using Visual Studio Express?

You might find you can't build the example application with Express versions of Visual Studio because it doesn't provide all the necessary libraries.

If that's the case, try searching for the missing libraries in one of the freely available Windows SDKs from the Microsoft website.

If you use Visual Studio Express to build your ***own*** application, Thread Validator will still work with it just fine.

7.2 Example NT Service

The example NT Service

As well as the example application, an example *service* is provided along with details about building it.

There's also an example client.

The example service demonstrates how to use the NT Service API to call the two functions required to use Thread Validator with NT Services.

The following tasks are performed when the service is started:

- Loads the Thread Validator stub DLL into the service
- Performs the normal work of the service until it's stopped
- Unloads the Thread Validator stub DLL from the service
- Informs the service control manager that a stop is pending

➡ Read more about working with NT Services.

7.2.1 Building the example service

Example service project files

The example project can be found in the `service` sub-directory in the directory where Thread Validator was installed.

If the directory is not present, reinstall your software and choose custom or full installation.

There are two project files in the directory:

- **service.dsp** › for Microsoft® Developer Studio® 6.0
- **service.vcproj** › for Microsoft® Visual Studio / .net

Configurations

There are just two simple configurations in each project:

- **Debug / Release** › dynamically links to the `svlTVStubService(_x64).lib` demonstrating use with the NT Service API

Using the service

To run the service, you will need administrator privileges.

Once the service is installed and started, you can use the provided serviceClient to interact with it.

The service is named **SVL x86/x64 TV Simple Service** in the control panel **Services** dialog.

SVL x86 TV Simple Service Manual Local System

Once you start the service, it will show as 'Running'

SVL x86 TV Simple Service Running Manual Local System

The service provides the following command line options:

- **-install** › Install the service
- **-remove** › Uninstall the service
- **-start** › Start the service
- **-stop** › Stop the service
- **-debug** › Run as a console application for debugging
- **-?** › Display the help message
- **-help** › Display the help message

Open a cmd prompt in administrator mode, navigate to the location of the service executable, and use one of these commands to install, remove, start, stop the service.

Examples:

```
        serviceTV.exe -install

        serviceTV.exe -start

        serviceTV.exe -stop

        serviceTV.exe -remove
```

7.2.2 Building the example client

If you've already built the sample service, the build process is very similar.

Project files

The example project can be found in the `serviceClient` sub-directory in the directory where Thread Validator was installed.

If the directory is not present, reinstall your software and choose custom or full installation.

There are two project files in the directory:

- **serviceClient.dsp** › for Microsoft® Developer Studio® 6.0
- **serviceClient.vcproj** › for Microsoft® Visual Studio / .net

Configurations

There are two configurations in each project:

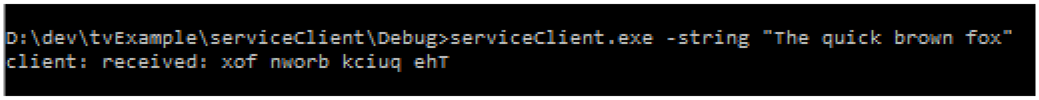
- **Debug / Release** › dynamically links to the `svlTVStubService(_x64).lib` demonstrating use with the NT Service API

Using serviceClient

The client interacts with the running service and provides the following command line options:

- **-string** › Sends the text that follows the option (quoted if it has spaces) to the service

  If the service is running the service will return the string in reverse order:



```
D:\dev\tvExample\serviceClient\Debug>serviceClient.exe -string "The quick brown fox"
client: received: xof nworb kciuq ehT
```

- **-?** › Display the help message (usage)
- **-help** › Display the help message (usage)

If the service is not running you'll see a message including something like "`CallNamedPipe failed`".

7.2.3 Building the example service utility

The serviceMutex project demonstrates a way of controlling whether Thread Validator is used without having to rebuild your service.

Project files

The example project can be found in the `serviceMutex` sub-directory in the directory where Thread Validator was installed.

If the directory is not present, reinstall your software and choose custom or full installation.

There are two project files in the directory:

- **serviceMutex.dsp** › for Microsoft® Developer Studio® 6.0
- **serviceMutex.vcproj** › for Microsoft® Visual Studio / .net

Configurations

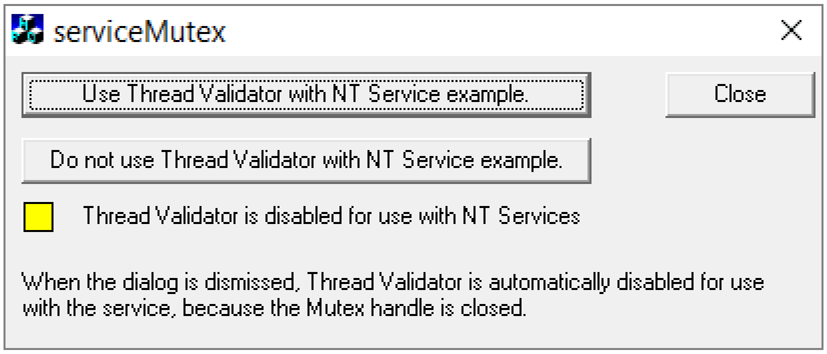
There are just two configurations in each project:

- **Debug / Release** › dynamically links to the `svlTVStubService(_x64).lib` demonstrating use with the NT Service API

Using the service utility

The utility provides a dialog box interface to allow the control over the creation of a mutex object with the name specified in the `service.h` header file.

Only if the service is started with the mutex created, does the service load Thread Validator.


serviceMutex
Use Thread Validator with NT Service example.
Close
Do not use Thread Validator with NT Service example.

Thread Validator is disabled for use with NT Services

When the dialog is dismissed, Thread Validator is automatically disabled for use with the service, because the Mutex handle is closed.

If you don't like using mutexes in this way, you could change the code in the service and the utility to communicate through shared memory, a registry setting, or another method of your choice.

7.2.4 Monitoring the service

Once the example service and example client has been built, the next step is to test them using Thread Validator.

Installing the service

If you haven't installed the service, do the following:

- open an administrator mode cmd prompt
- navigate to the directory containing the serviceTV.exe to install
- serviceTV.exe -install

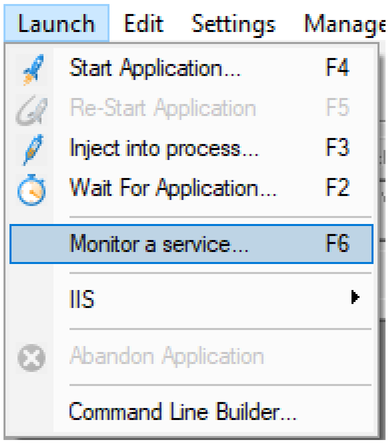
Monitoring the service

Prerequisites

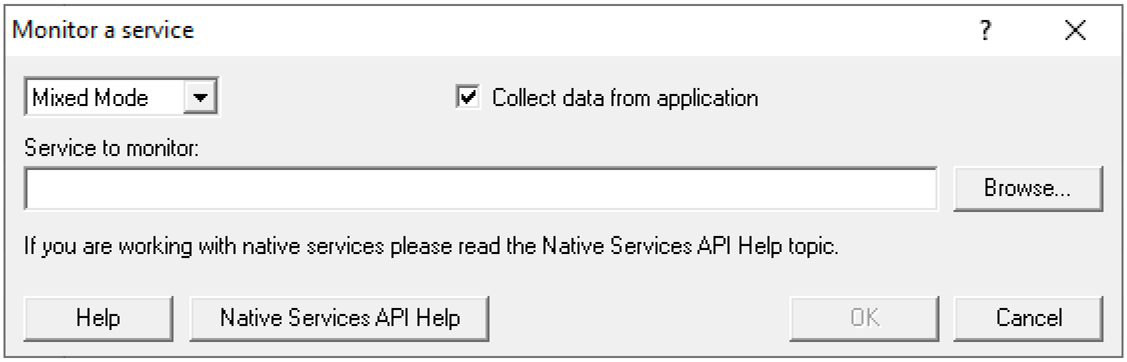
- example service has been installed, but not started (if service has been started, stop the service)
- example service and example client have been built

The following process is used to monitor the application launched by the service:

- From the Launch menu choose `Services > Monitor a Service...`



- The Monitor a service dialog is displayed



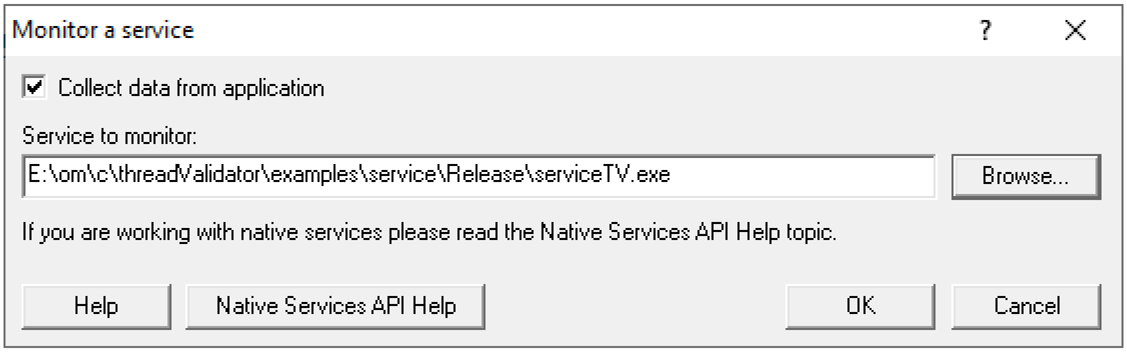
Mixed Mode

☑ Collect data from application

Service to monitor:

If you are working with native services please read the Native Services API Help topic.

- Use **Browse...** to open the file chooser dialog and choose the service that will be monitored by Thread Validator.



Monitor a service

☑ Collect data from application

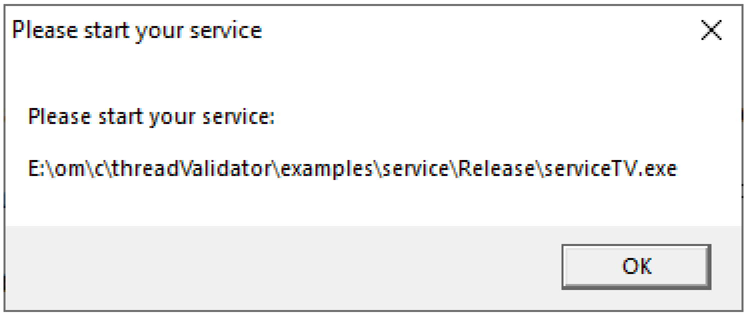
Service to monitor:

E:\om\c\threadValidator\examples\service\Release\serviceTV.exe

If you are working with native services please read the Native Services API Help topic.

- Click **OK**

- Thread Validator sets up a variety of parameters then displays a dialog box asking you to start you service. Click **OK** to dismiss the dialog



Please start your service

Please start your service:

E:\om\c\threadValidator\examples\service\Release\serviceTV.exe

- Start your service. For the example `serviceTV.exe` do the following
  - open an administrator mode cmd prompt
  - navigate to the directory containing the `serviceTV.exe` to start
  - `serviceTV.exe -start`
  - `serviceTV.exe` starts will be monitored by Thread Validator

- The target application contacts Thread Validator

- Data is collected until the service finishes executing

- Thread Validator displays the results

7.3 Example Application Launched from a Service

The example Application launched from a Service

This pair of projects create an application that is launched from a service.

The purpose of this example is to show how to monitor the application that is launched from the service. This is also the same process for monitoring an application launched by an application launched from a service.

This process is subtly different to the method for working with services (see the example service for that).

Service

The service project is `serviceWithAChildProcess.vcxproj`

The following tasks are performed when the service is started:

- the test application is launched from the service

Application

The application project is `serviceChildProcess.vcxproj`

The application's first task is to load Thread Validator into the application.

- Loads the Thread Validator stub DLL into the application
- Configures the NT Service API to communicate to Thread Validator
- Does some work that can be monitored by Thread Validator
- Exits

Implementation Details

For implementation details see `attachToThreadValidator();` in `serviceChildProcess.cpp`.

The application will need to link to the NT Service API, for example `..\..\..\svlTVStubService\release_2010_x64\svlTVStubService_x64.lib` (for a release x64 EXE/DLL).

Important. Call `attachToThreadValidator() as close to the start of your application as possible, before any threads have been created.`

➡ Read more about working with NT Services.

7.3.1 Building the service and application

Example solution files

The example solution can be found in the `examples\serviceWithAChildProcess` subdirectory in the directory where Thread Validator was installed.

If the directory is not present, reinstall your software and choose custom or full installation.

Example project files

The example projects can be found in the subdirectories in the directory where Thread Validator was installed.

`examples\serviceWithAChildProcess\serviceWithAChildProcess`

- **serviceWithAChildProcess.vcproj** › for Microsoft® Visual Studio / .net

`examples\serviceWithAChildProcess\serviceChildProcess`

- **serviceChildProcess.vcproj** › for Microsoft® Visual Studio / .net

Configurations

There are a small number of configurations in each project:

- **Debug / Release** › dynamically links to the `svlTVStubService(_x64).lib` demonstrating use with the NT Service API

Using the service

The service is named **SVL *** TV Child Process** in the control panel services dialog (*** changes depending on the build configuration), and provides the following command line options:

- **-install** › Install the service
- **-remove** › Uninstall the service
- **-start** › Start the service
- **-stop** › Stop the service
- **-debug** › Run as a console application for debugging
- **-?** › Display the help message
- **-help** › Display the help message

Open a cmd prompt in administrator mode, navigate to the location of the service executable, and use one of these commands to install, remove, start, stop the service.

Examples:

```
serviceWithAChildProcess.exe -install

serviceWithAChildProcess.exe -start

serviceWithAChildProcess.exe -stop

serviceWithAChildProcess.exe -remove
```

7.3.2 Monitoring the application launched from the service

Once the example service and example application are built, the next step is to test them using Thread Validator.

Installing the service

If you haven't installed the service, do the following:

- open an administrator mode cmd prompt
- navigate to the directory containing the serviceWithAProcess.exe to install
- serviceWithAProcess.exe -install

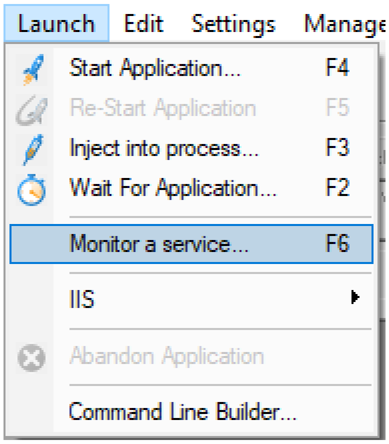
Monitoring the application launched by the service

Prerequisites

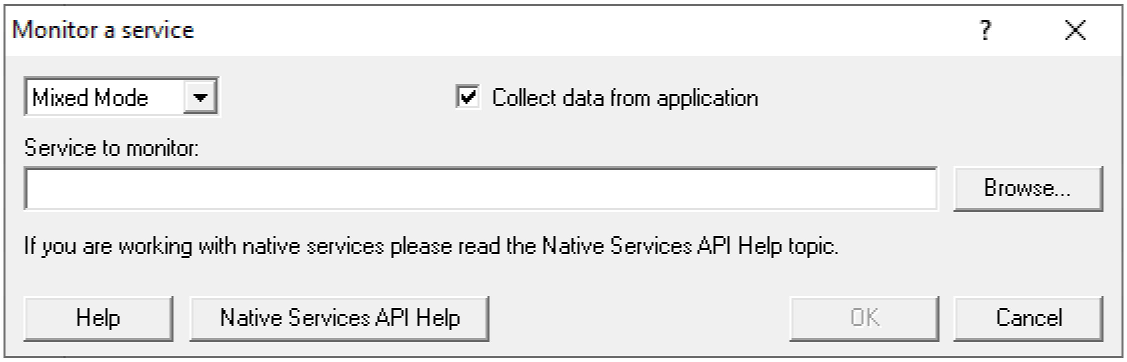
- example service has been installed, but not started (if service has been started, stop the service)
- example service and example application have been built (application must use the NT Service API as demonstrated in attachToThreadValidator())
- example application executable is in the same directory as the example service (this is only a requirement for the example)

The following process is used to monitor the application launched by the service:

- From the Launch menu choose `Services > Monitor a Service...`



- The Monitor a service dialog is displayed



Monitor a service

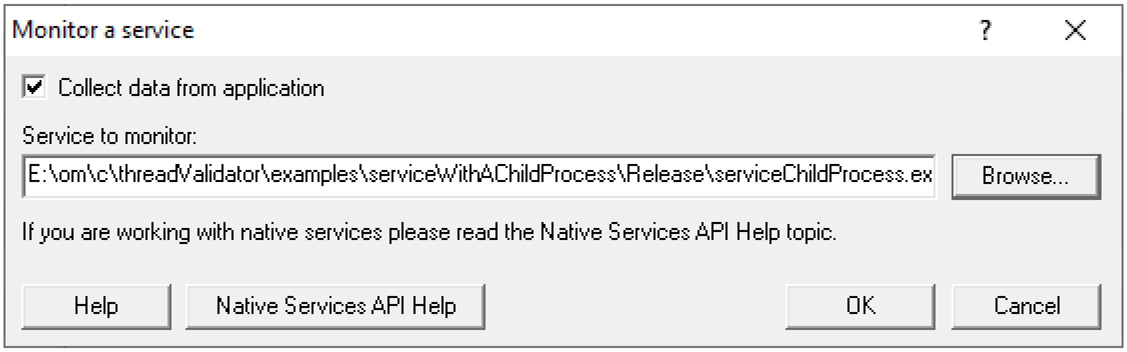
Mixed Mode

☑ Collect data from application

Service to monitor:

If you are working with native services please read the Native Services API Help topic.

- Use **Browse...** to open the file chooser dialog and choose the application that will be monitored by Thread Validator. This is the application that is launched by the service. Do not choose the service



Monitor a service

☑ Collect data from application

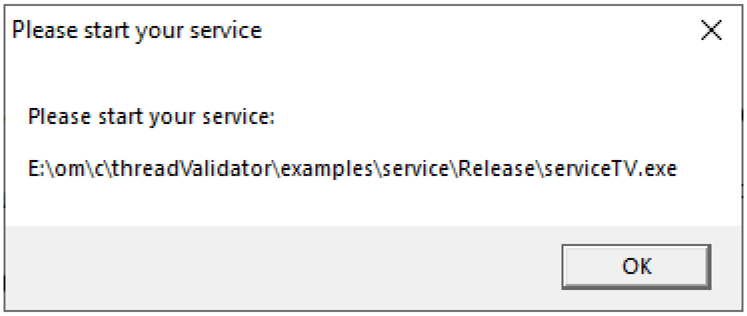
Service to monitor:

E:\om\c\threadValidator\examples\serviceWithAChildProcess\Release\serviceChildProcess.ex

If you are working with native services please read the Native Services API Help topic.

- Click **OK**

- Thread Validator sets up a variety of parameters then displays a dialog box asking you to start you service. Click **OK** to dismiss the dialog



Please start your service

Please start your service:

E:\om\c\threadValidator\examples\service\Release\serviceTV.exe

- Start your service. For the example `serviceWithAChildProcess.exe` do the following
  - open an administrator mode cmd prompt
  - navigate to the directory containing the `serviceWithAProcess.exe` to start
  - `serviceWithAProcess.exe -start`
  - `serviceWithAProcess.exe` starts and launches the child process `serviceChildProcess.exe` that will be monitored

- The target application contacts Thread Validator

- Data is collected until the target process finishes executing

- Thread Validator displays the results

Part VIII

8 Hook Reference

Enabling and disabling the hooks

The lists on the following page show the hooks used by Thread Validator and are for reference only.

New hooks may be added in later versions of the software.

You can individually enable and disable the hook settings as well as whole groups at a time.

Hooking DLLS

You can enable or disable hooks on a per-DLL basis using the Hooked DLLs settings.

8.1 Synchronization Hooks

These hooks are used by Thread Validator and are for reference only.

New hooks may be added in later versions of the software.

You can individually enable and disable the hook settings as well as whole groups at a time.

Critical sections

- DeleteCriticalSection()
- EnterCriticalSection()
- InitializeCriticalSection()
- InitializeCriticalSectionEx()
- InitializeCriticalSectionAndSpinCount()
- LeaveCriticalSection()
- SetCriticalSectionSpinCount()
- TryEnterCriticalSection()

Events

- CreateEventA()
- CreateEventW()
- OpenEventA()
- OpenEventW()
- PulseEvent()
- SetEvent()
- ResetEvent()

File

- CreateFileA()
- CreateFileW()
- FindCloseChangeNotification()
- FindFirstChangeNotificationA()
- FindFirstChangeNotificationW()

Jobs

- CreateJobObjectA()
- CreateJobObjectW()

Misc

- QueueUserAPC()
- Sleep()
- SleepEx()

Mutexes

- CreateMutexA()
- CreateMutexW()
- OpenMutexA()
- OpenMutexW()
- ReleaseMutex()

Process

- CreateProcessA()
- CreateProcessW()
- CreateProcessAsUserA()
- CreateProcessAsUserW()
- CreateProcessWithLogonW()
- DuplicateHandle()
- FreeLibraryAndExitThread()
- OpenProcess()
- TerminateProcess()

Semaphores

- CreateSemaphoreA()
- CreateSemaphoreW()
- OpenSemaphoreA()
- OpenSemaphoreW()
- ReleaseSemaphore()

Threads

- CloseHandle()
- CreateRemoteThread()
- CreateThread()
- ExitThread()

- OpenThread()
- ResumeThread()
- SuspendThread()
- TerminateThread()

Timers

- CancelWaitableTimer()
- CreateWaitableTimerA()
- CreateWaitableTimerW()
- OpenWaitableTimerA()
- OpenWaitableTimerW()
- SetWaitableTimer()

Timer Queues

- ChangeTimerQueueTimer()
- CreateTimerQueue()
- CreateTimerQueueTimer()
- DeleteTimerQueue()
- DeleteTimerQueueEx()
- DeleteTimerQueueTimer()

Waits

- MsgWaitForMultipleObjects()
- MsgWaitForMultipleObjectsEx()
- SignalObjectAndWait()
- RegisterWaitForSingleObject()
- RegisterWaitForSingleObjectEx()
- UnregisterWait()
- UnregisterWaitEx()
- WaitForSingleObject()
- WaitForMultipleObjects()
- WaitForSingleObjectEx()
- WaitForMultipleObjectsEx()

Part IX

9 Debug Information, Symbols, Filenames, Line Numbers

Depending on which IDE or compiler/linker combination the process to create debug information to ensure that you have symbols, filenames and line numbers is different.

This section shows you what to do to ensure you have symbols for your compiler and linker.

9.1 Visual Studio

Enabling debug information in Visual Studio has changed over the years depending on the version of Visual Studio you are using.

It's generally the same, but there have been some changes in recent versions that can cause confusion.

By default debug configurations create debug information, but for some versions of Visual Studio, release configurations do not create debug information.

You need to set both compiler and linker settings to get debug information. ***Setting just one or the other will not give you debug information you can use.***

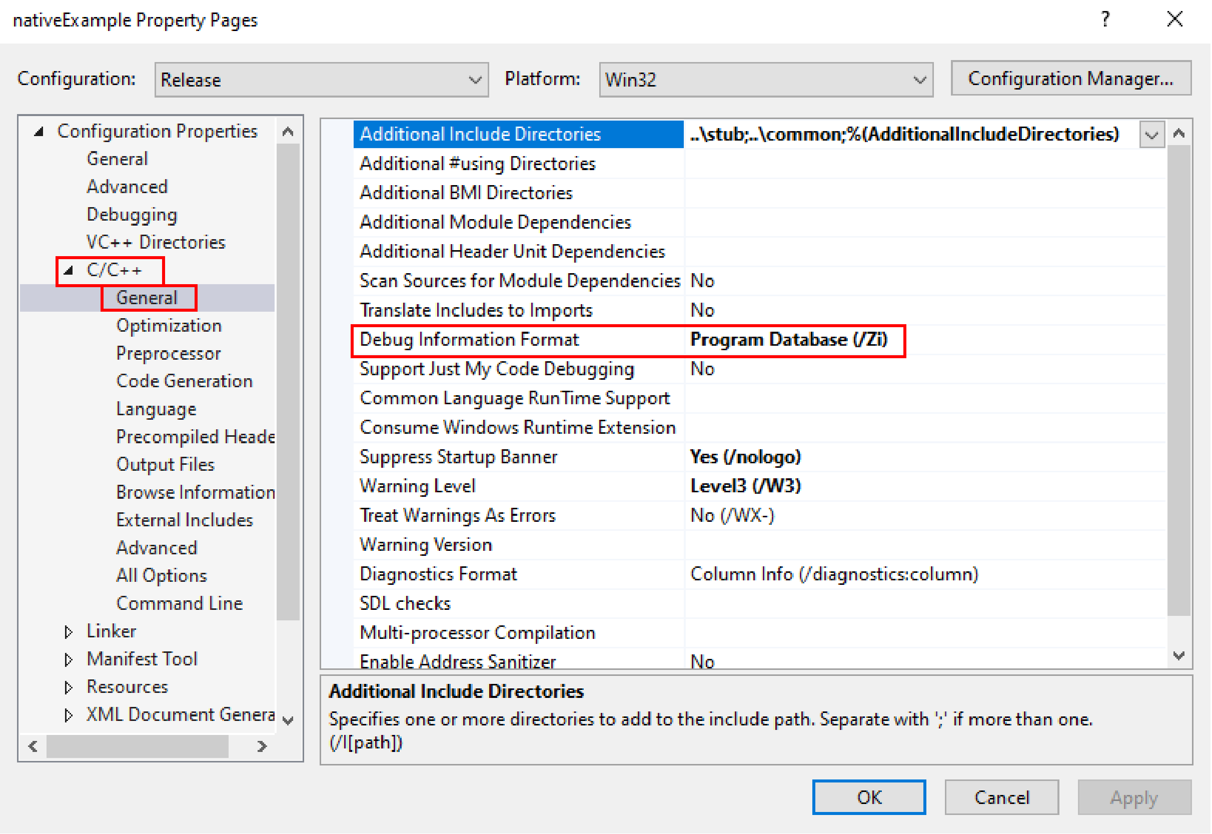
Configurations

In the help below we show you how to modify one configuration, for example Release | Win32.

You need to modify all configurations appropriately. Release, Debug, Win32, Win64 and any other configurations you are using.

Visual Studio 2017 - 2021

Compiler Settings



Configuration: Release

Platform: Win32

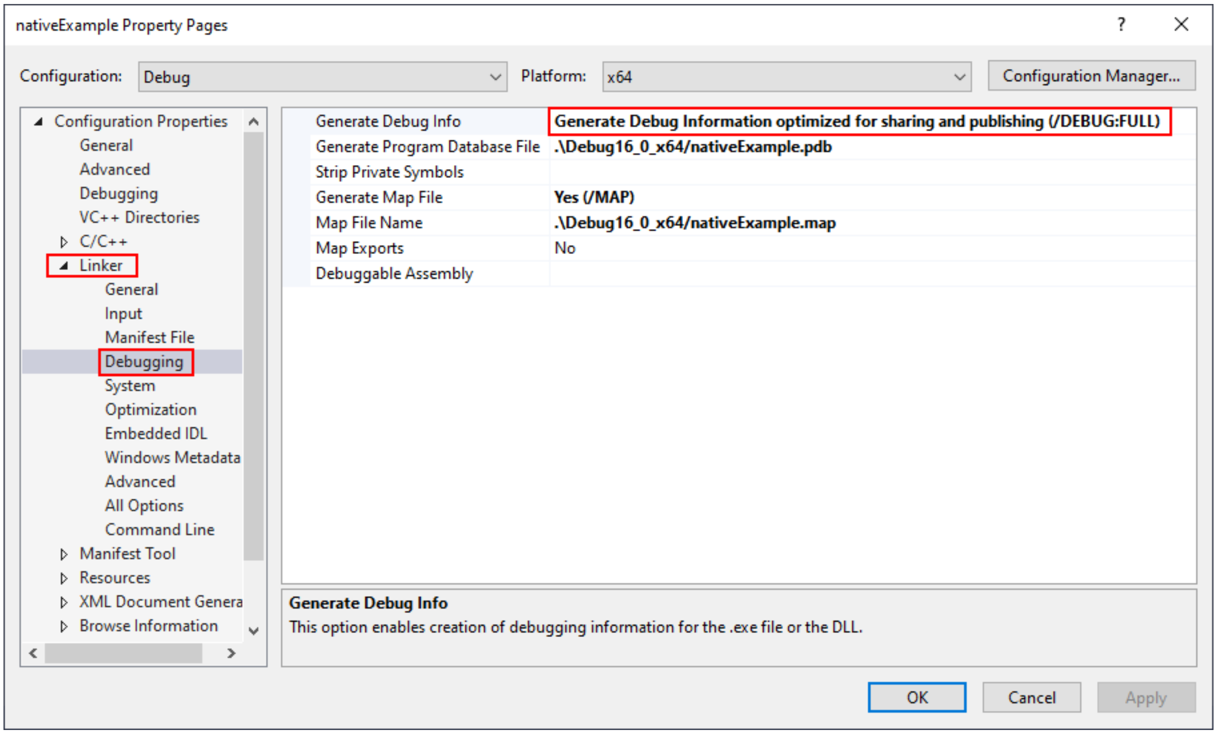
Additional Include Directories	..\stub;..\common;%(AdditionalIncludeDirectories)
Additional #using Directories	
Additional BMI Directories	
Additional Module Dependencies	
Additional Header Unit Dependencies	
Scan Sources for Module Dependencies	No
Translate Includes to Imports	No
Debug Information Format	Program Database (/Zi)
Support Just My Code Debugging	No
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level3 (/W3)
Treat Warnings As Errors	No (/WX-)
Warning Version	
Diagnostics Format	Column Info (/diagnostics:column)
SDL checks	
Multi-processor Compilation	
Enable Address Sanitizer	No

Additional Include Directories

Specifies one or more directories to add to the include path. Separate with ';' if more than one. (/I[path])

Linker Settings

If you're building on a different machine to the machine you're working on (for example a build server), you should choose **/DEBUG:FULL**, not /DEBUG or /DEBUG:FASTLINK.



Configuration: Debug

Platform: x64

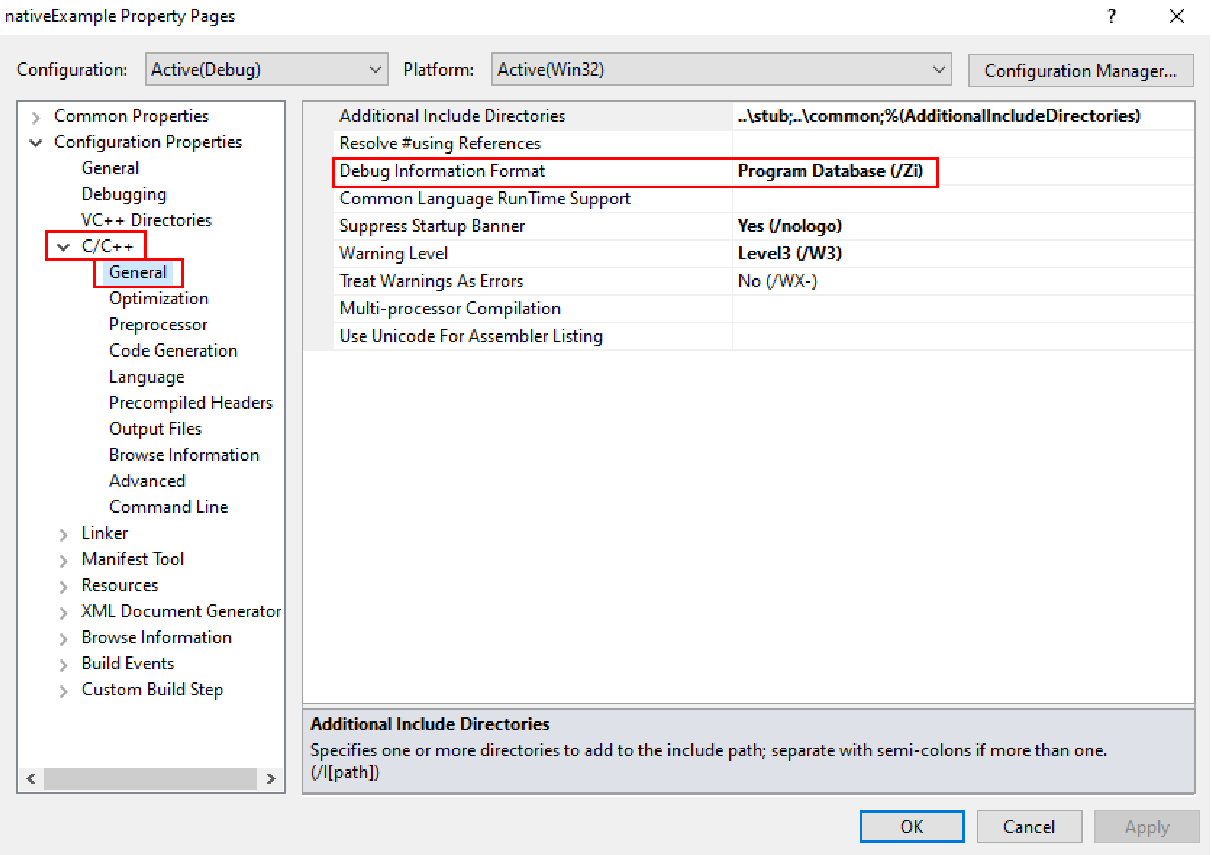
Generate Debug Info	Generate Debug Information optimized for sharing and publishing (/DEBUG:FULL)
Generate Program Database File	.\Debug16_0_x64/nativeExample.pdb
Strip Private Symbols	
Generate Map File	Yes (/MAP)
Map File Name	.\Debug16_0_x64/nativeExample.map
Map Exports	No
Debuggable Assembly	

Generate Debug Info
This option enables creation of debugging information for the .exe file or the DLL.

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Visual Studio 2010 - 2015

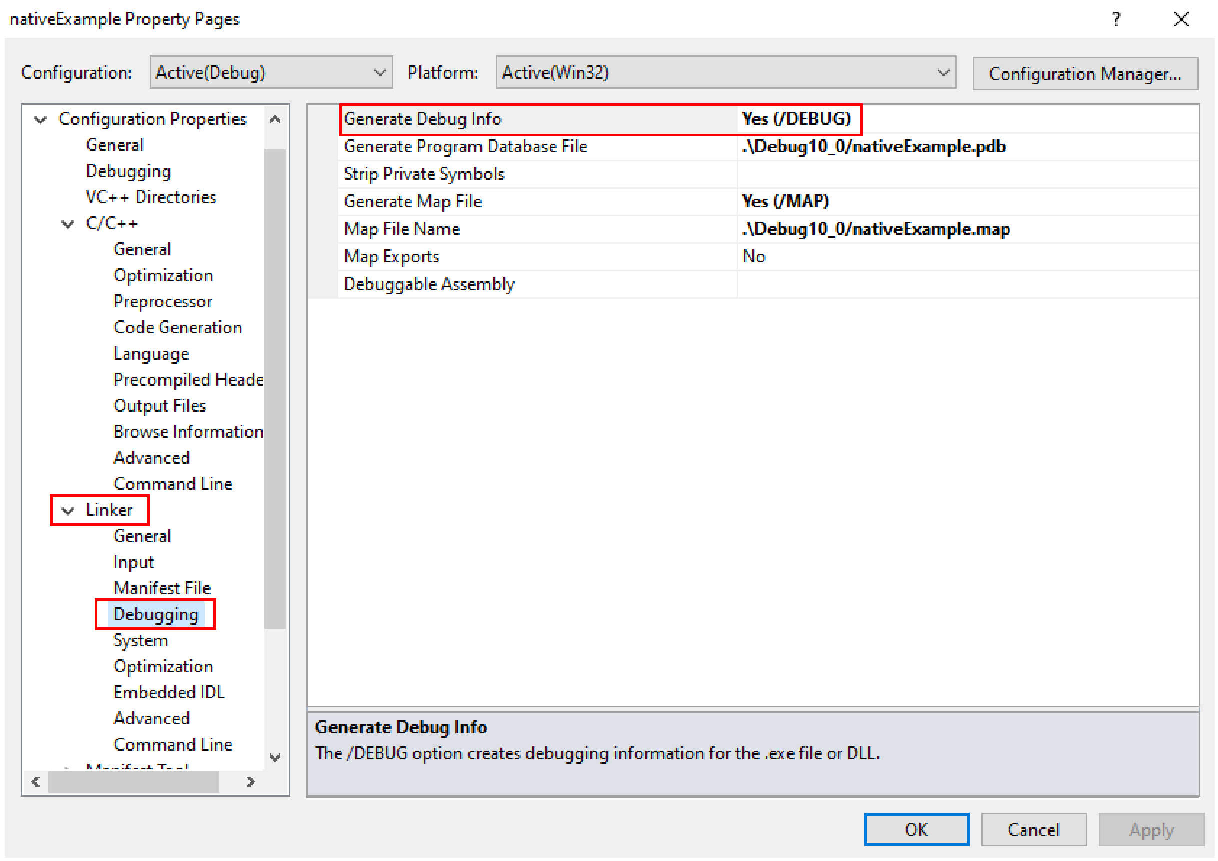
Compiler Settings



Additional Include Directories	..\stub;..\common;%(AdditionalIncludeDirectories)
Resolve #using References	
Debug Information Format	Program Database (/Zi)
Common Language RunTime Support	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level3 (/W3)
Treat Warnings As Errors	No (/WX-)
Multi-processor Compilation	
Use Unicode For Assembler Listing	

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

Linker Settings



Configuration: Active(Debug) Platform: Active(Win32)

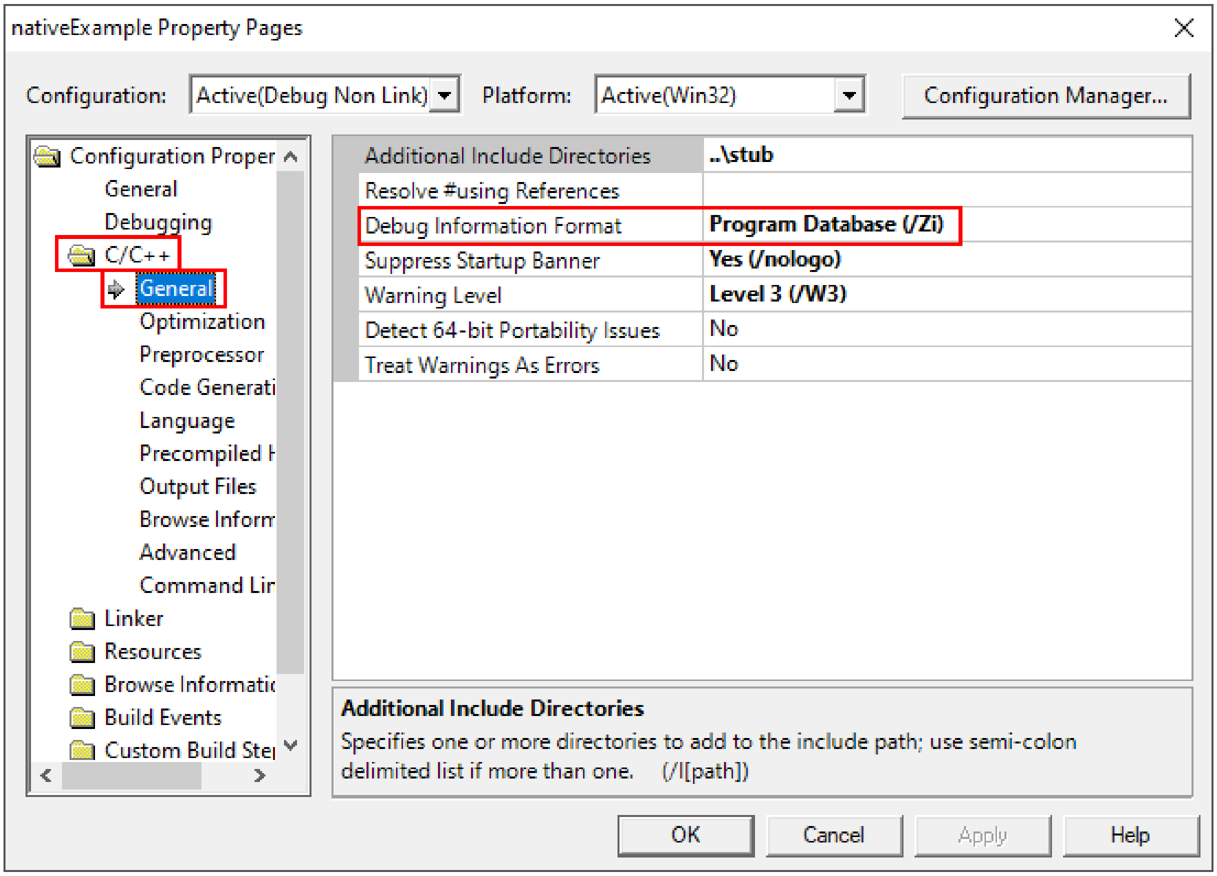
Generate Debug Info	Yes (/DEBUG)
Generate Program Database File	.\Debug10_0/nativeExample.pdb
Strip Private Symbols	
Generate Map File	Yes (/MAP)
Map File Name	.\Debug10_0/nativeExample.map
Map Exports	No
Debuggable Assembly	

Generate Debug Info
The /DEBUG option creates debugging information for the .exe file or DLL.

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Visual Studio 2002 - 2008

Compiler Settings



nativeExample Property Pages

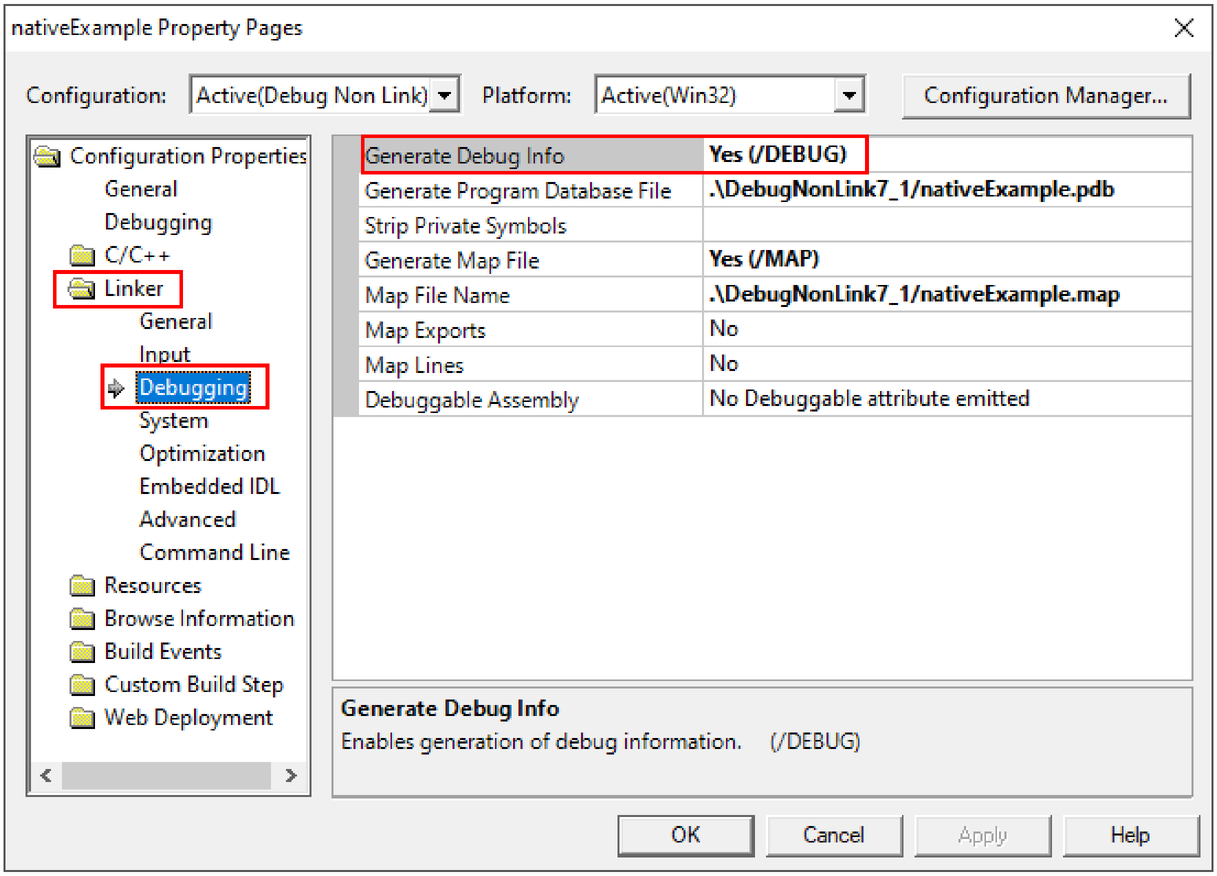
Configuration: Active(Debug Non Link)

Platform: Active(Win32)

Additional Include Directories	..\stub
Resolve #using References	
Debug Information Format	**Program Database (/Zi)**
Suppress Startup Banner	**Yes (/nologo)**
Warning Level	**Level 3 (/W3)**
Detect 64-bit Portability Issues	No
Treat Warnings As Errors	No

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

Linker Settings



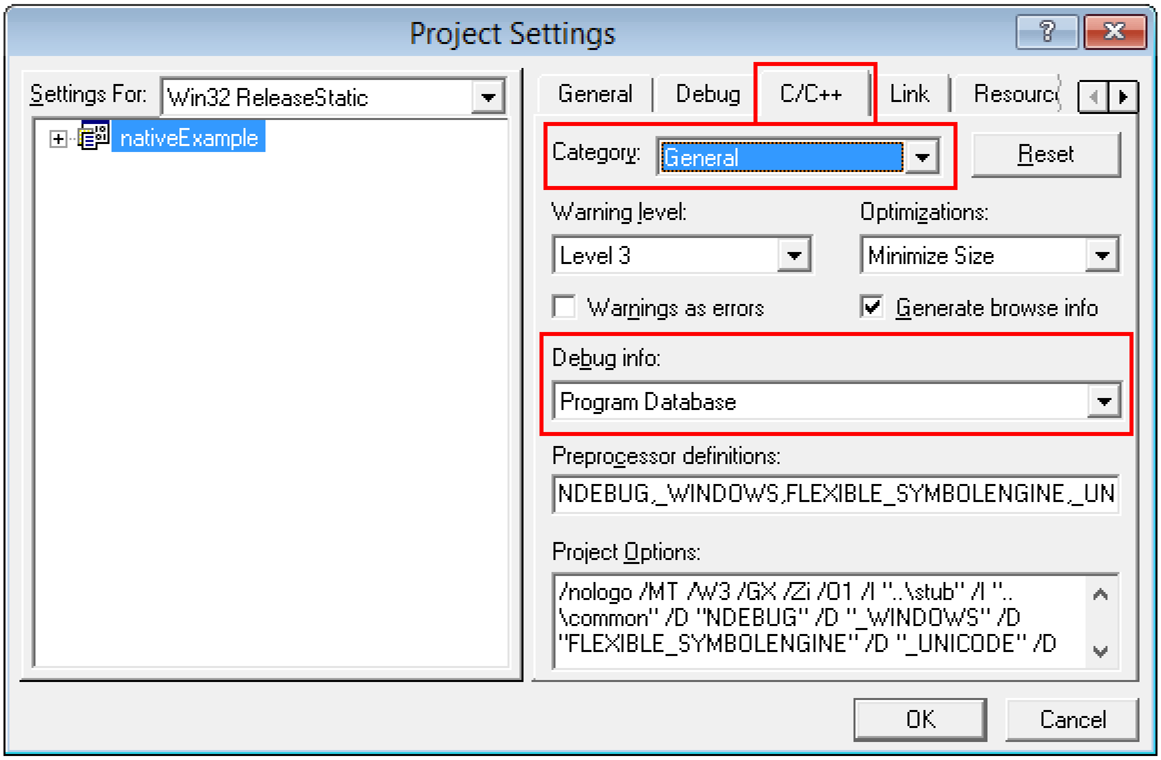
Generate Debug Info	Yes (/DEBUG)
Generate Program Database File	.\DebugNonLink7_1/nativeExample.pdb
Strip Private Symbols	
Generate Map File	Yes (/MAP)
Map File Name	.\DebugNonLink7_1/nativeExample.map
Map Exports	No
Map Lines	No
Debuggable Assembly	No Debuggable attribute emitted

Generate Debug Info
Enables generation of debug information. (/DEBUG)

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Visual Studio 6.0

Compiler Settings



Project Settings

Settings For: Win32 ReleaseStatic

nativeExample

Category: General

Warning level: Level 3

Optimizations: Minimize Size

□ Warnings as errors

☑ Generate browse info

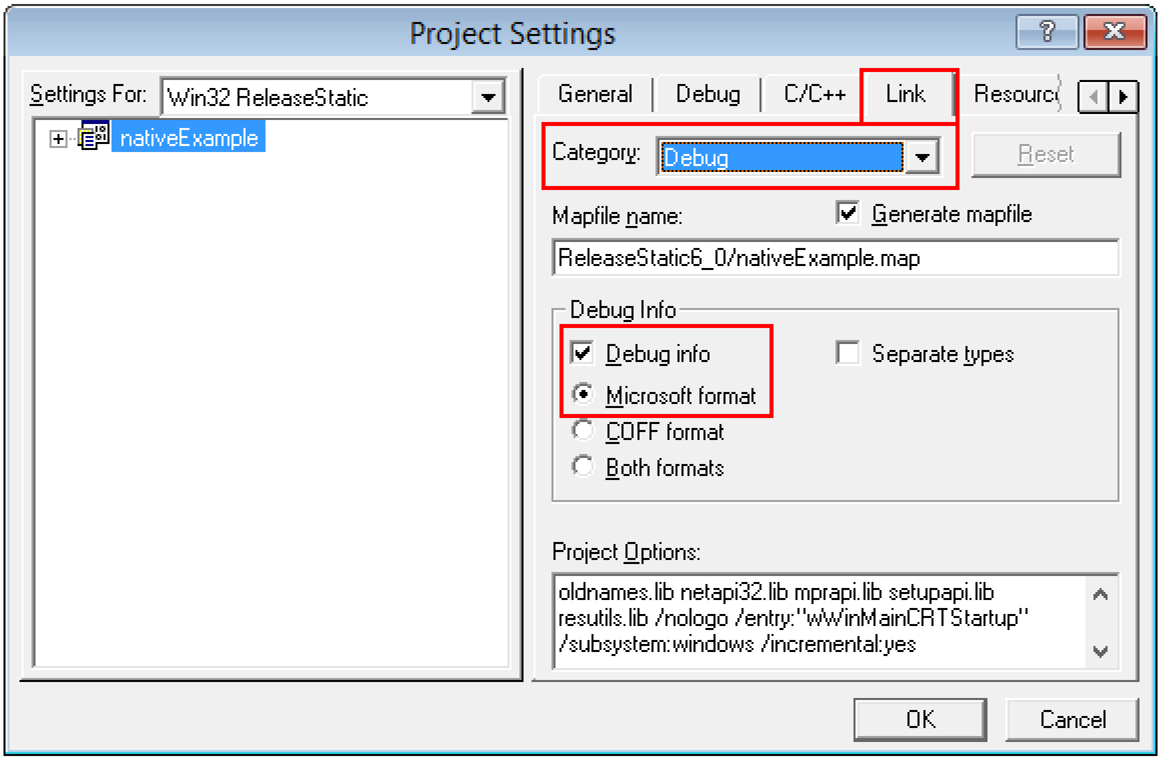
Debug info: Program Database

Preprocessor definitions: NDEBUG,_WINDOWS,FLEXIBLE_SYMBOLENGINE,_UN

Project Options:

```
/nologo /MT /W3 /GX /Zi /O1 /I "..\stub" /I "..
\common" /D "NDEBUG" /D "_WINDOWS" /D
"FLEXIBLE_SYMBOLENGINE" /D "_UNICODE" /D
```

Linker Settings



Project Settings

Settings For: Win32 ReleaseStatic

nativeExample

Category: Debug

Mapfile name: ReleaseStatic6_0/nativeExample.map

☑ Generate mapfile

Debug Info

☑ Debug info

● Microsoft format

○ COFF format

○ Both formats

□ Separate types

Project Options:

```
oldnames.lib netapi32.lib mprapi.lib setupapi.lib
resutils.lib /nologo /entry:"wWinMainCRTStartup"
/subsystem:windows /incremental:yes
```

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

9.2 C++ Builder

Debug information can be provided using two methods.

- Debugging information (TDS or DWARF format)
- MAP files

Debugging Information

Debug configurations of C++ Builder projects automatically generate debug information that provides symbols, filenames and line numbers.

However the release configurations of C++ Builder projects do not automatically generate debug information. You need to configure that yourself.

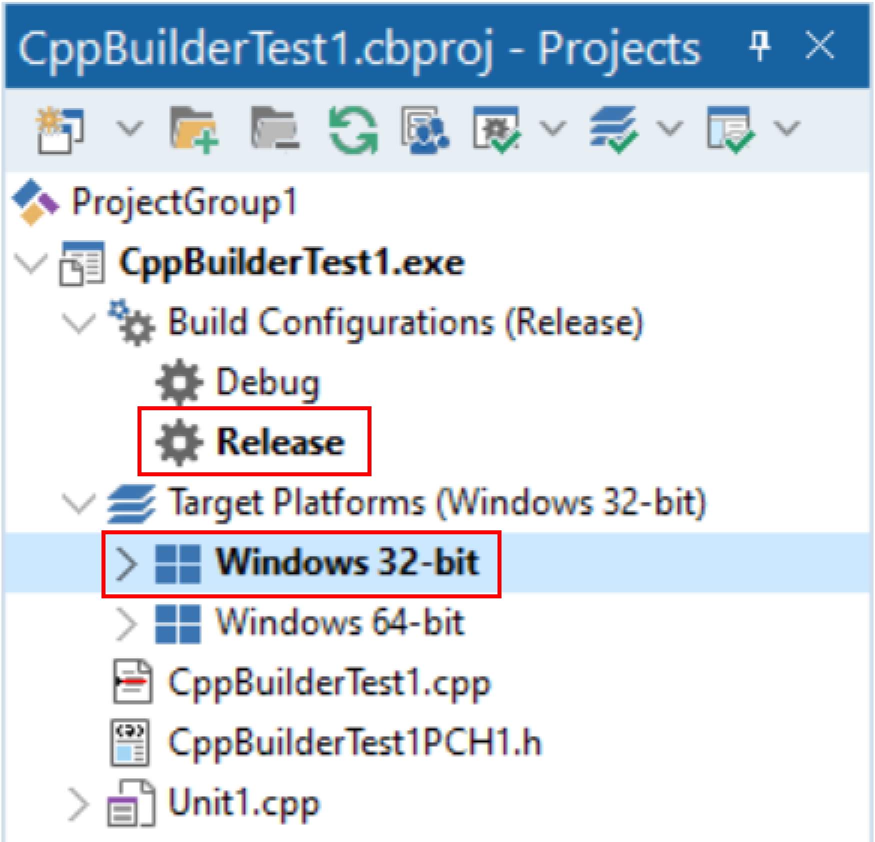
Here's how you do that. It's slightly different if you're building 32 bit applications rather 64 bit applications.

You need to set both compiler and linker settings to get debug information. Setting just one or the other will not give you debug information you can use.

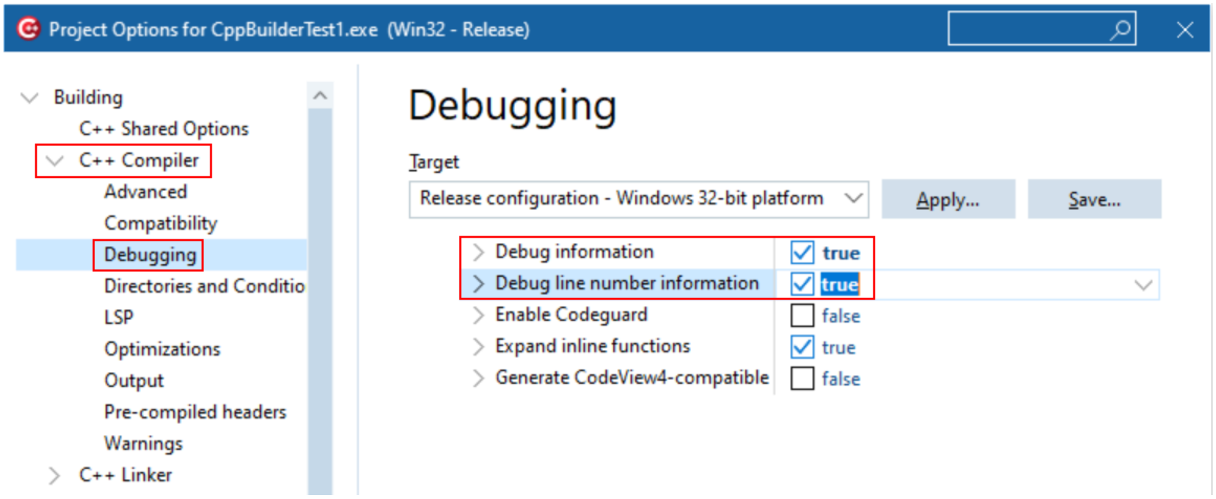
32 bit C++ Builder

Project Configuration

Change your project settings to target 32 bit builds.



Compiler Settings



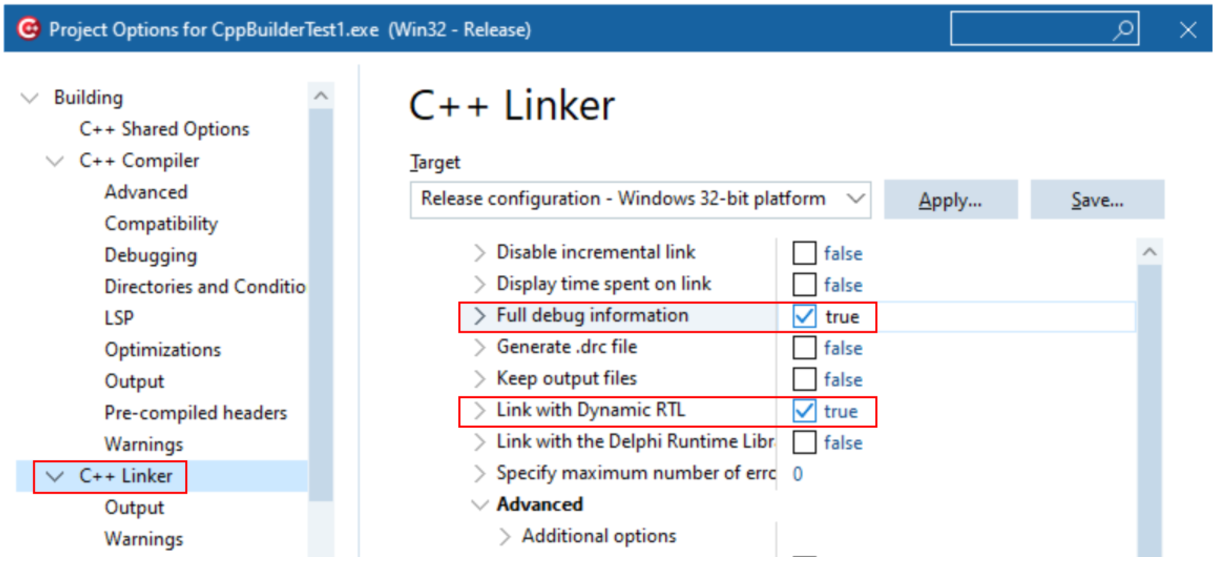
Project Options for CppBuilderTest1.exe (Win32 - Release)

Debugging

Target: Release configuration - Windows 32-bit platform

Debug information	☑ true
Debug line number information	☑ true
Enable Codeguard	□ false
Expand inline functions	☑ true
Generate CodeView4-compatible	□ false

Linker Settings



Project Options for CppBuilderTest1.exe (Win32 - Release)

C++ Linker

Target: Release configuration - Windows 32-bit platform

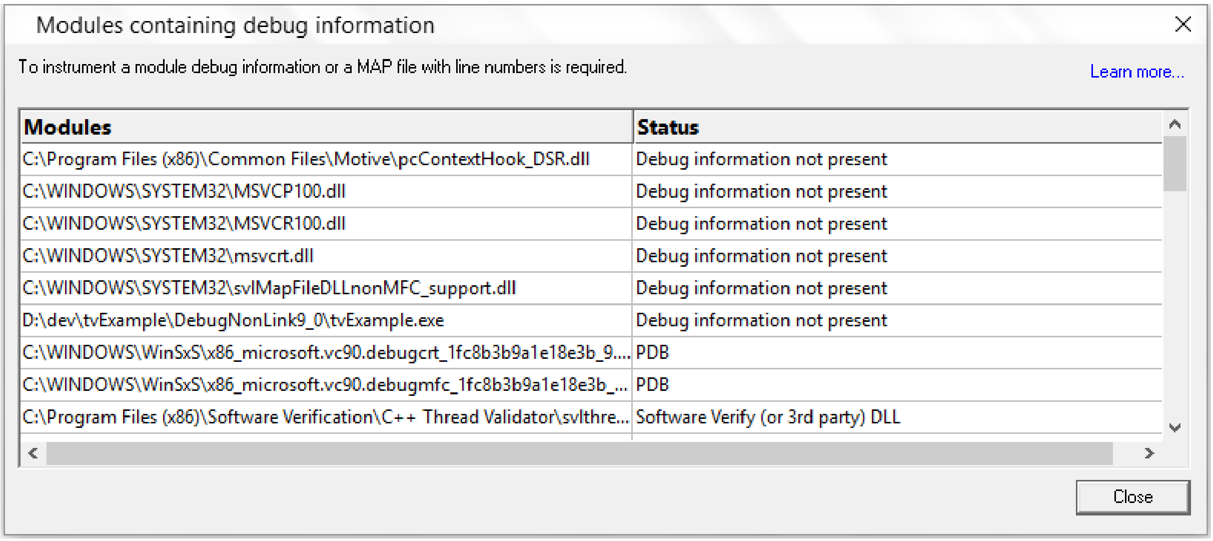
- Disable incremental link □ false
- Display time spent on link □ false
- Full debug information ☑ true
- Generate .drc file □ false
- Keep output files □ false
- Link with Dynamic RTL ☑ true
- Link with the Delphi Runtime Libr □ false
- Specify maximum number of errc 0
- Advanced
  - Additional options

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

64 bit C++ Builder

Project Configuration

Change your project settings to target 64 bit builds.

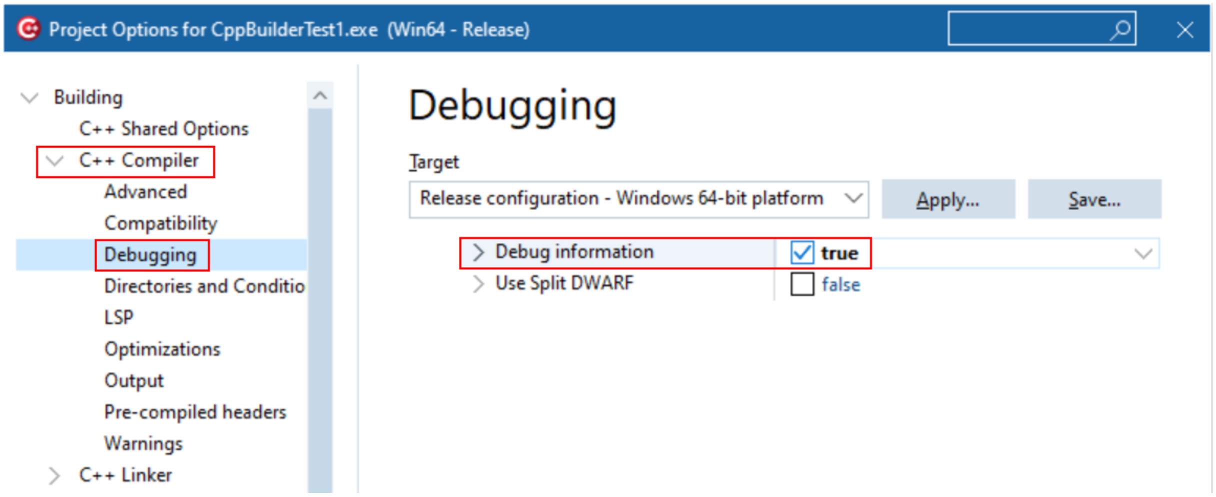


Modules containing debug information

To instrument a module debug information or a MAP file with line numbers is required.

Modules	Status
C:\Program Files (x86)\Common Files\Motive\pcContextHook_DSR.dll	Debug information not present
C:\WINDOWS\SYSTEM32\MSVCP100.dll	Debug information not present
C:\WINDOWS\SYSTEM32\MSVCR100.dll	Debug information not present
C:\WINDOWS\SYSTEM32\msvcrt.dll	Debug information not present
C:\WINDOWS\SYSTEM32\svlMapFileDLLnonMFC_support.dll	Debug information not present
D:\dev\tvExample\DebugNonLink9_0\tvExample.exe	Debug information not present
C:\WINDOWS\WinSxS\x86_microsoft.vc90.debugcrt_1fc8b3b9a1e18e3b_9....	PDB
C:\WINDOWS\WinSxS\x86_microsoft.vc90.debugmfc_1fc8b3b9a1e18e3b_...	PDB
C:\Program Files (x86)\Software Verification\C++ Thread Validator\svlthre...	Software Verify (or 3rd party) DLL

Compiler Settings



Project Options for CppBuilderTest1.exe (Win64 - Release)

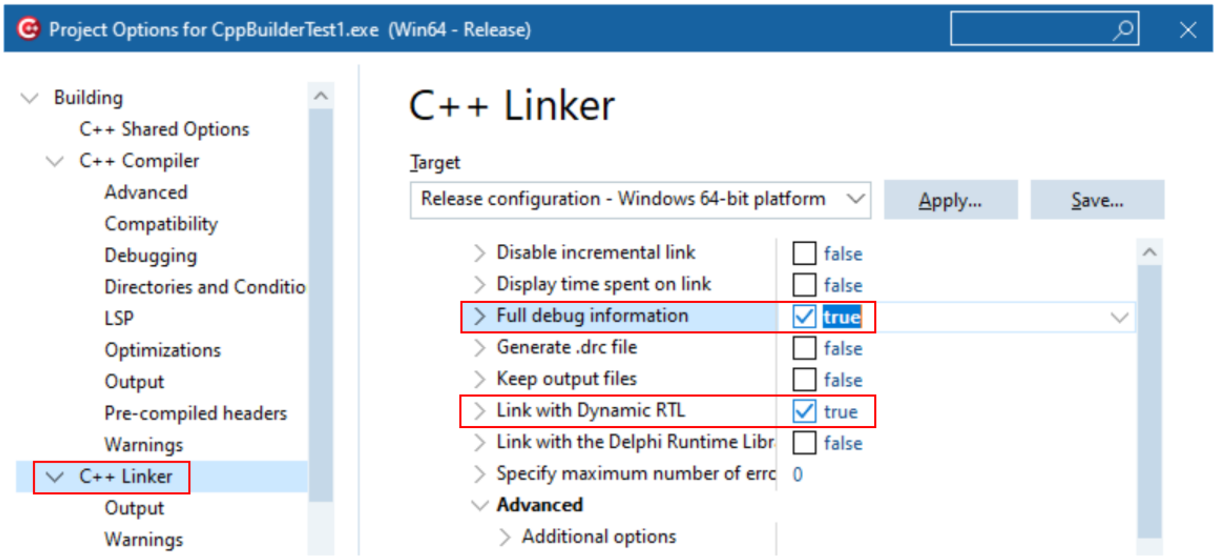
- Building
  - C++ Shared Options
  - C++ Compiler
    - Advanced
    - Compatibility
    - Debugging
    - Directories and Conditio
    - LSP
    - Optimizations
    - Output
    - Pre-compiled headers
    - Warnings
  - C++ Linker

Debugging

Target: Release configuration - Windows 64-bit platform

Debug information	☑ true
Use Split DWARF	□ false

Linker Settings



Project Options for CppBuilderTest1.exe (Win64 - Release)

- Building
  - C++ Shared Options
  - C++ Compiler
    - Advanced
    - Compatibility
    - Debugging
    - Directories and Conditio
    - LSP
    - Optimizations
    - Output
    - Pre-compiled headers
    - Warnings
  - C++ Linker
    - Output
    - Warnings

C++ Linker

Target: Release configuration - Windows 64-bit platform

Disable incremental link	□ false
Display time spent on link	□ false
Full debug information	☑ true
Generate .drc file	□ false
Keep output files	□ false
Link with Dynamic RTL	☑ true
Link with the Delphi Runtime Libr	□ false
Specify maximum number of errc	0
Advanced	
Additional options	

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

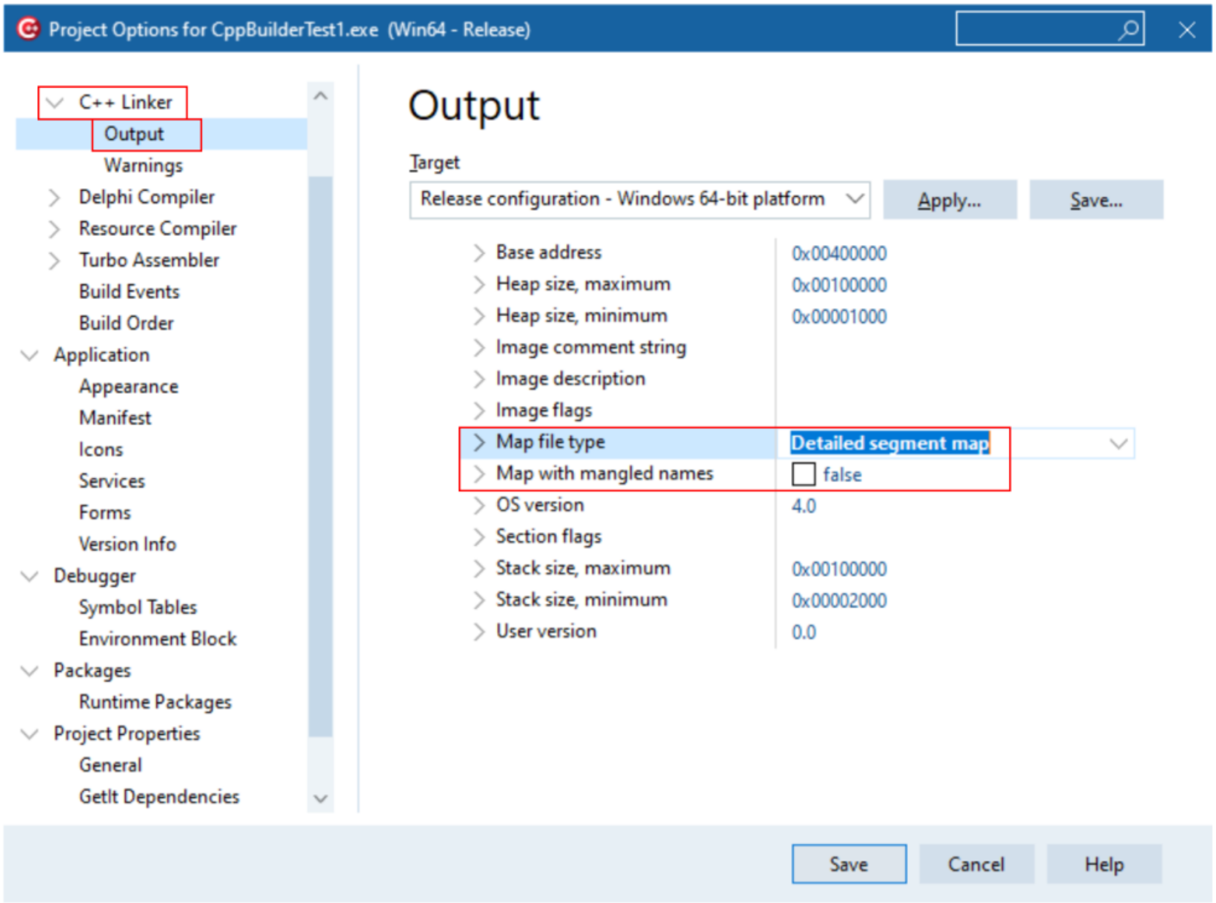
MAP files

MAP files are not generated by default. You need to enable the option to generate a detailed map file.

The method is the same for 32 bit and 64 bit C++ Builder.

Select the project configuration as shown in the Debugging Information section above, then modify the C++ Linker, Output settings.

Linker Settings



Base address	0x00400000
Heap size, maximum	0x00100000
Heap size, minimum	0x00001000
Image comment string	
Image description	
Image flags	
Map file type	Detailed segment map
Map with mangled names	□ false
OS version	4.0
Section flags	
Stack size, maximum	0x00100000
Stack size, minimum	0x00002000
User version	0.0

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Debugging Information or MAP files?

If you can create both debugging information and MAP files which should I use?

Thread Validator uses this information to provide symbols, filenames and line numbers in stack traces.

For this purpose it doesn't matter whether you use Debugging Information or MAP files.

9.3 Delphi

Debug information can be provided using two methods.

- Debugging information (TDS format)
- MAP files

Debugging Information

Debug configurations of Delphi projects automatically generate debug information that provides symbols, filenames and line numbers.

However the release configurations of Delphi projects do not automatically generate debug information. You need to configure that yourself.

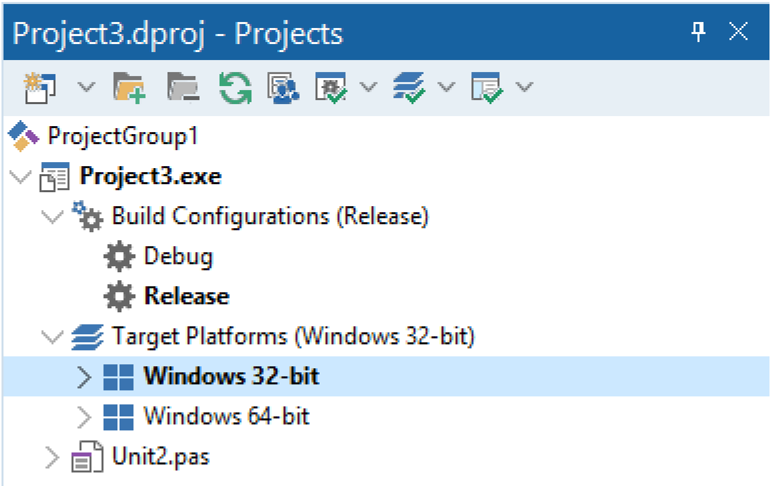
Here's how you do that. It's slightly different if you're building 32 bit applications rather 64 bit applications.

You need to set both compiler and linker settings to get debug information. Setting just one or the other will not give you debug information you can use.

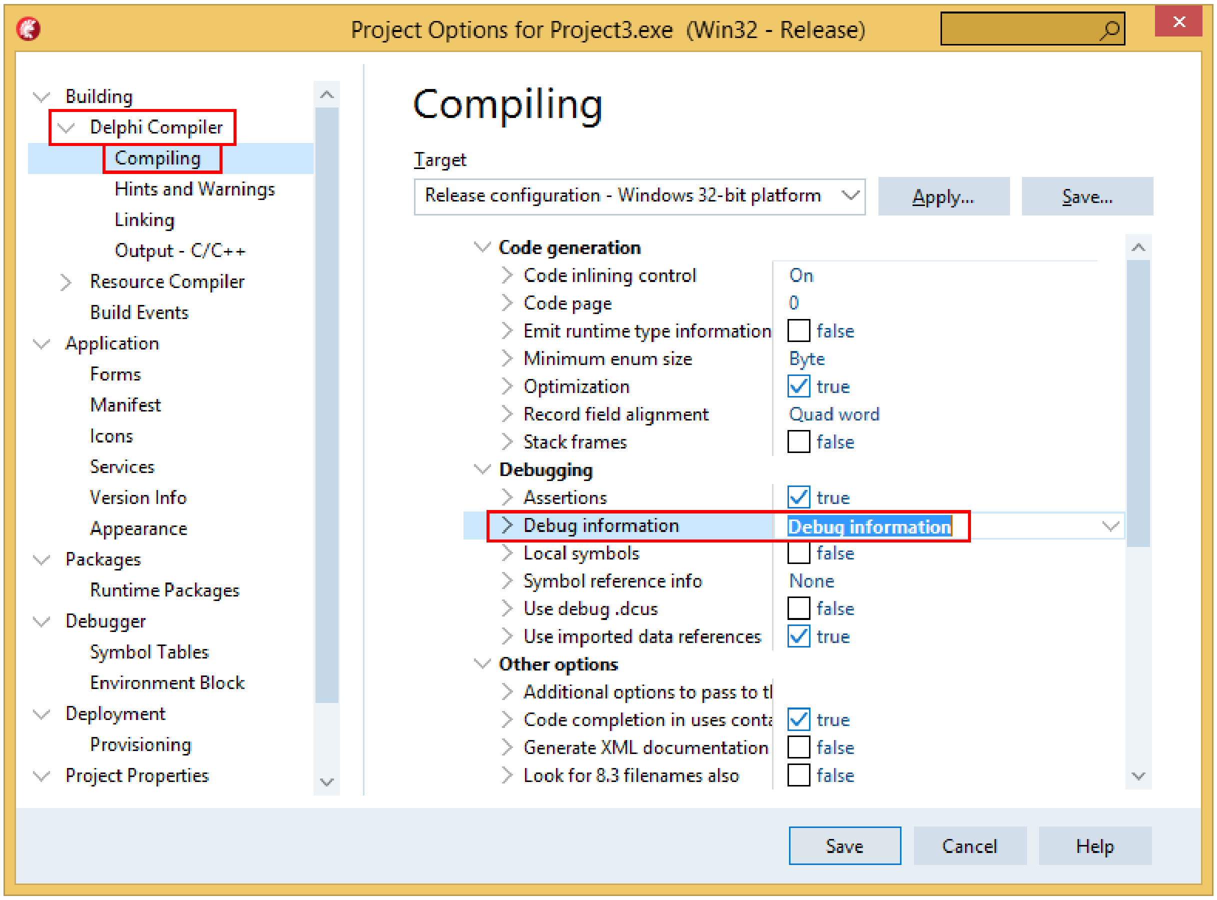
32 bit Delphi

Project Configuration

Change your project settings to target 32 bit builds.


Project3.dproj - Projects
ProjectGroup1
Project3.exe
Build Configurations (Release)
Debug
Release
Target Platforms (Windows 32-bit)
Windows 32-bit
Windows 64-bit
Unit2.pas

Compiler Settings



Compiling

Target: Release configuration - Windows 32-bit platform

Code generation

Code inlining control	On
Code page	0
Emit runtime type information	□ false
Minimum enum size	Byte
Optimization	☑ true
Record field alignment	Quad word
Stack frames	□ false

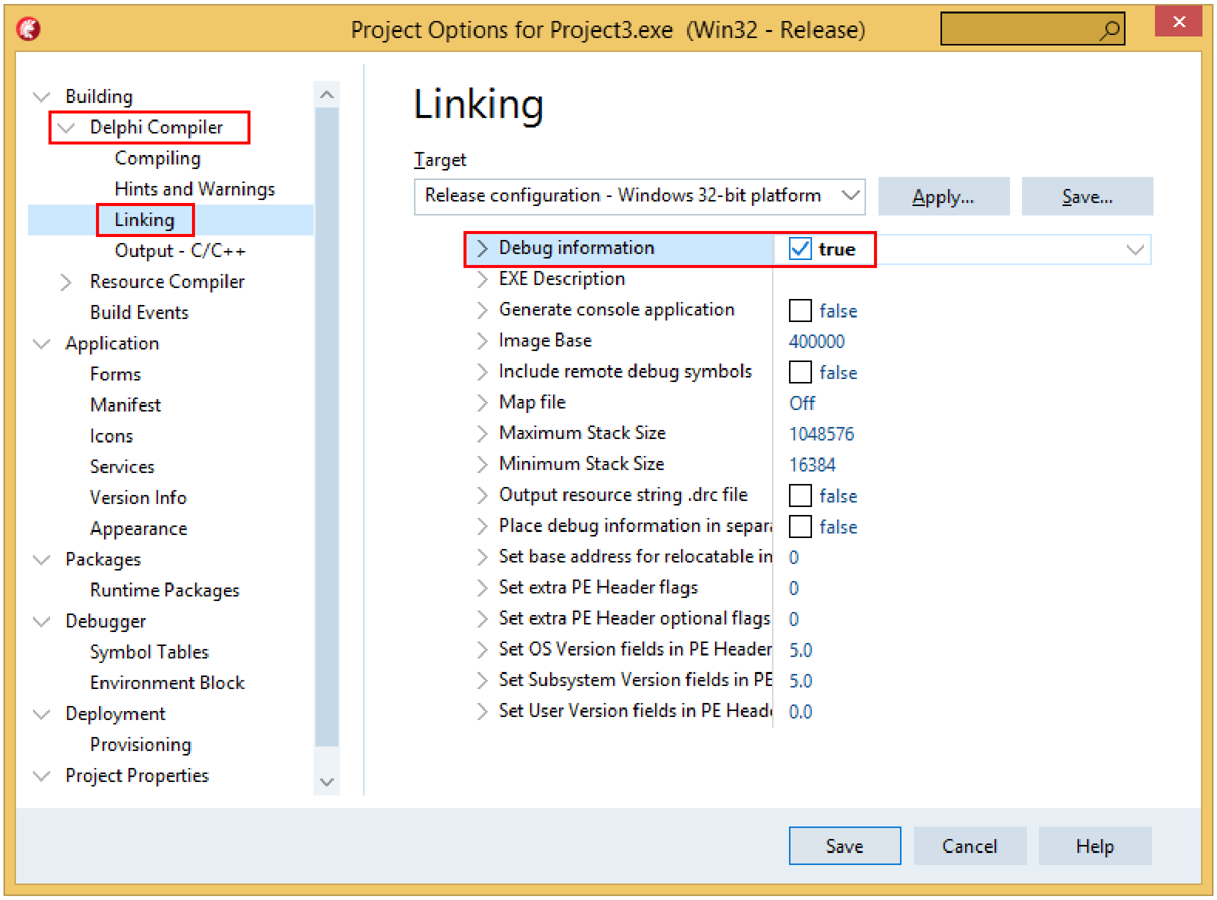
Debugging

Assertions	☑ true
Debug information	Debug information
Local symbols	□ false
Symbol reference info	None
Use debug .dcus	□ false
Use imported data references	☑ true

Other options

Additional options to pass to tl	
Code completion in uses conta	☑ true
Generate XML documentation	□ false
Look for 8.3 filenames also	□ false

Linker Settings



Linking

Target: Release configuration - Windows 32-bit platform

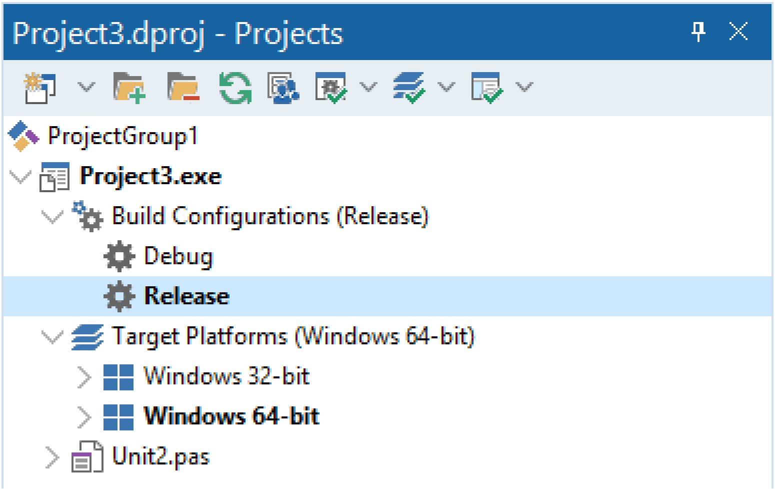
Debug information	☑ true
EXE Description	
Generate console application	□ false
Image Base	400000
Include remote debug symbols	□ false
Map file	Off
Maximum Stack Size	1048576
Minimum Stack Size	16384
Output resource string .drc file	□ false
Place debug information in separa	□ false
Set base address for relocatable in	0
Set extra PE Header flags	0
Set extra PE Header optional flags	0
Set OS Version fields in PE Header	5.0
Set Subsystem Version fields in PE	5.0
Set User Version fields in PE Head	0.0

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

64 bit Delphi

Project Configuration

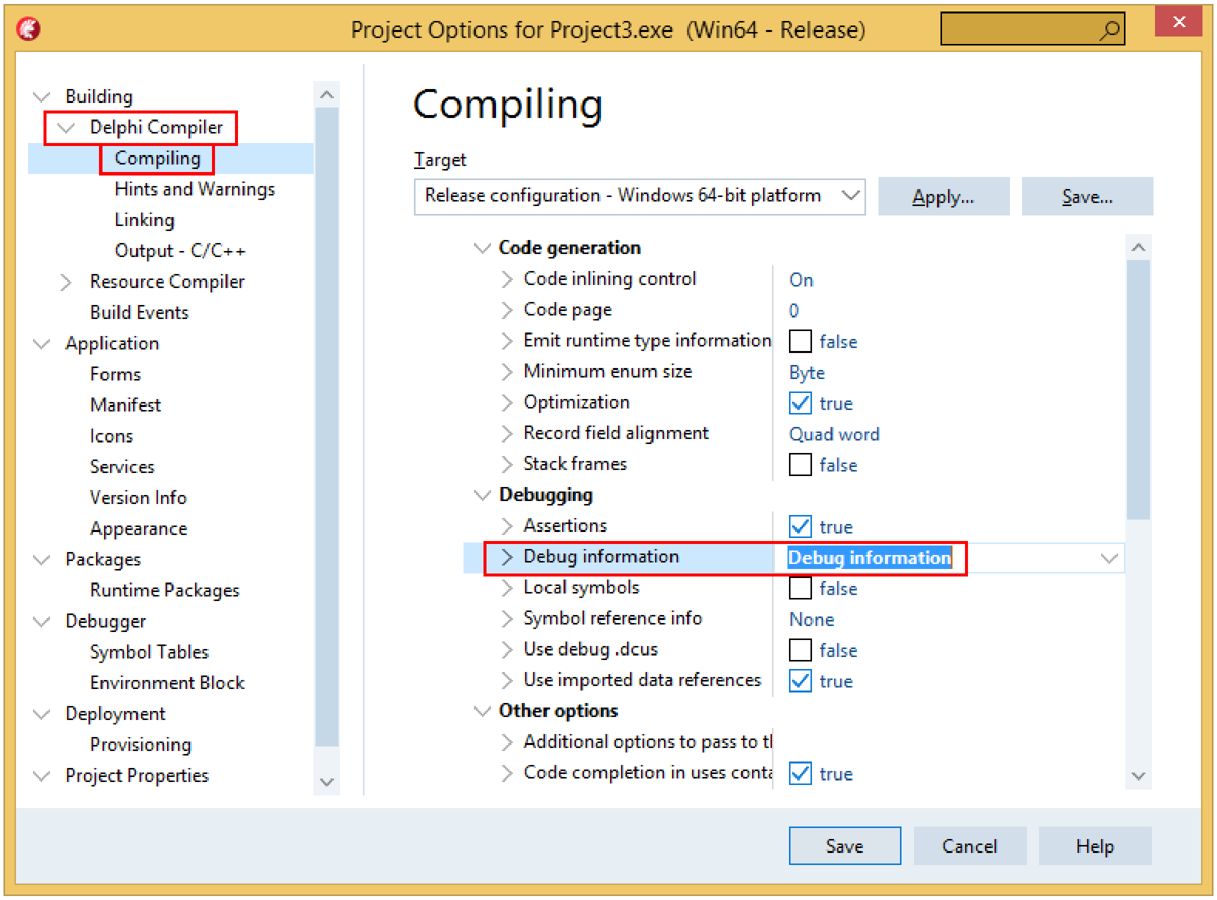
Change your project settings to target 64 bit builds.



Project3.dproj - Projects

- ProjectGroup1
  - Project3.exe
    - Build Configurations (Release)
      - Debug
      - Release
    - Target Platforms (Windows 64-bit)
      - Windows 32-bit
      - Windows 64-bit
    - Unit2.pas

Compiler Settings



Project Options for Project3.exe (Win64 - Release)

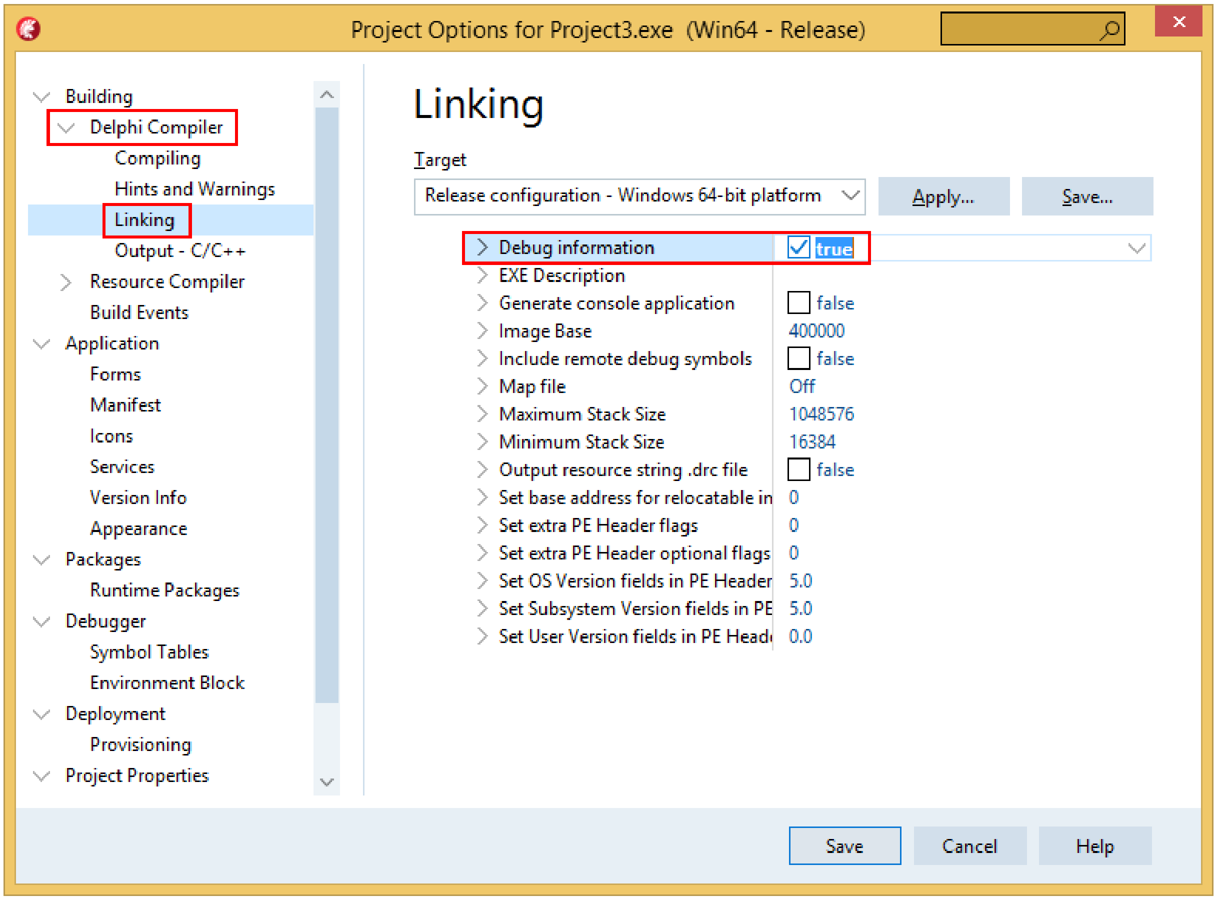
- Building
  - Delphi Compiler
    - Compiling
    - Hints and Warnings
    - Linking
    - Output - C/C++
  - Resource Compiler
  - Build Events
- Application
  - Forms
  - Manifest
  - Icons
  - Services
  - Version Info
  - Appearance
- Packages
  - Runtime Packages
- Debugger
  - Symbol Tables
  - Environment Block
- Deployment
  - Provisioning
- Project Properties

Compiling

Target: Release configuration - Windows 64-bit platform

Code generation	
Code inlining control	On
Code page	0
Emit runtime type information	□ false
Minimum enum size	Byte
Optimization	☑ true
Record field alignment	Quad word
Stack frames	□ false
Debugging	
Assertions	☑ true
Debug information	Debug information
Local symbols	□ false
Symbol reference info	None
Use debug .dcus	□ false
Use imported data references	☑ true
Other options	
Additional options to pass to tl	
Code completion in uses conta	☑ true

Linker Settings



Linking

Target: Release configuration - Windows 64-bit platform

Debug information	☑ true
EXE Description	
Generate console application	□ false
Image Base	400000
Include remote debug symbols	□ false
Map file	Off
Maximum Stack Size	1048576
Minimum Stack Size	16384
Output resource string .drc file	□ false
Set base address for relocatable in	0
Set extra PE Header flags	0
Set extra PE Header optional flags	0
Set OS Version fields in PE Header	5.0
Set Subsystem Version fields in PE	5.0
Set User Version fields in PE Head	0.0

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

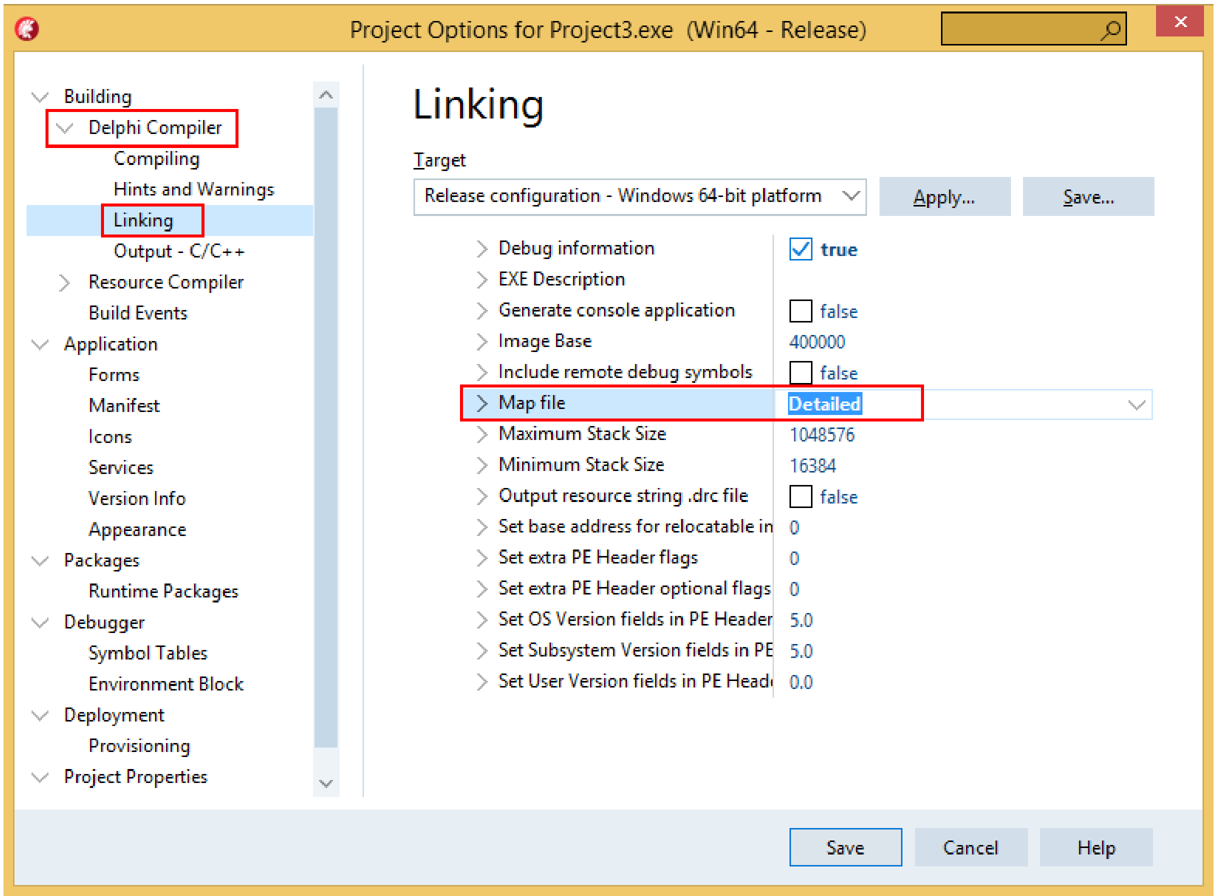
MAP files

MAP files are not generated by default. You need to enable the option to generate a detailed map file.

The method is the same for 32 bit and 64 bit Delphi.

Select the project configuration as shown in the Debugging Information section above, then modify the Delphi Compiler, Linking settings.

Linker Settings



Linking

Target: Release configuration - Windows 64-bit platform

Debug information	☑ true
EXE Description	
Generate console application	□ false
Image Base	400000
Include remote debug symbols	□ false
Map file	Detailed
Maximum Stack Size	1048576
Minimum Stack Size	16384
Output resource string .drc file	□ false
Set base address for relocatable in	0
Set extra PE Header flags	0
Set extra PE Header optional flags	0
Set OS Version fields in PE Header	5.0
Set Subsystem Version fields in PE	5.0
Set User Version fields in PE Head	0.0

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Debugging Information or MAP files?

If you can create both debugging information and MAP files which should I use?

Thread Validator uses this information to provide symbols, filenames and line numbers in stack traces.

For this purpose it doesn't matter whether you use Debugging Information or MAP files.

9.4 MingW, gcc, g++

The following compiler options are available if you are using MingW, gcc or g++.

-g
This is the default debug format. This will normally choose the DWARF symbol format.

-gdwarf
The DWARF symbol format.

-gstabs
The STABS symbol format.

-gCoff
The COFF symbol format. This does create a lot of unnecessary symbols, making symbol parsing slower.

9.5 Dev C++

Dev C++ uses the gcc and g++ compilers.

The following compiler options are available if you are using gcc or g++.

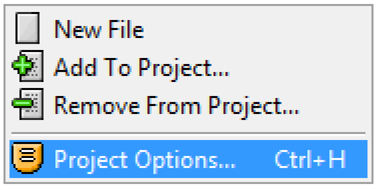
-g
This is the default debug format. This will normally choose the DWARF symbol format.

-gdwarf
The DWARF symbol format.

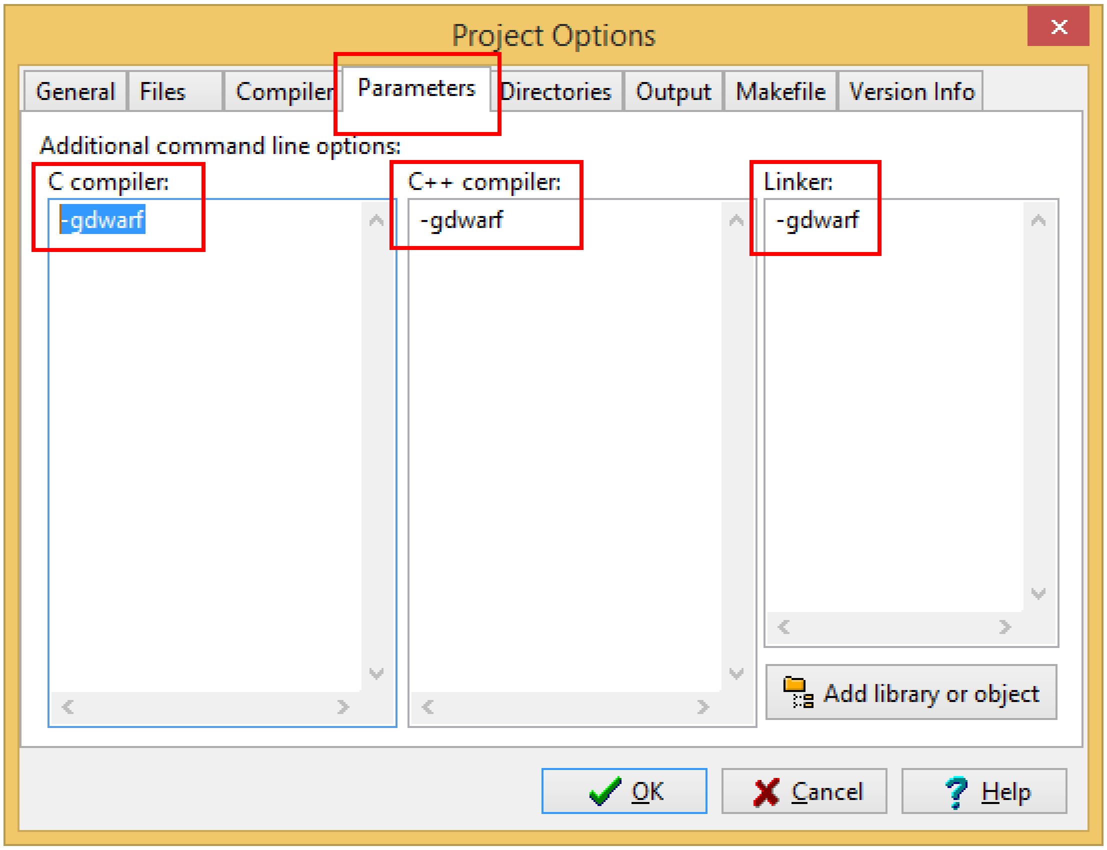
-gstabs
The STABS symbol format.

-gCoff
The COFF symbol format. This does create a lot of unnecessary symbols, making symbol parsing slower.

You can edit the compiler and linker options by choosing **Project Options...** from the **Project** menu.



Compiler and Linker options



Additional command line options:

C compiler: -gdwarf

C++ compiler: -gdwarf

Linker: -gdwarf

When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

9.6 Salford Software FORTRAN 95

Salford FORTRAN95 symbolic information is embedded in the .exe/.dll as COFF (Common Object File Format) information, with some proprietary extensions to Salford Software (which they have kindly shared with us).

Please consult the documentation for Salford FORTRAN95 to include debug information (including filenames and line numbers) in the COFF information.

If you still have problems, please contact us giving as much detail as possible, including what you've tried.

9.7 Metrowerks

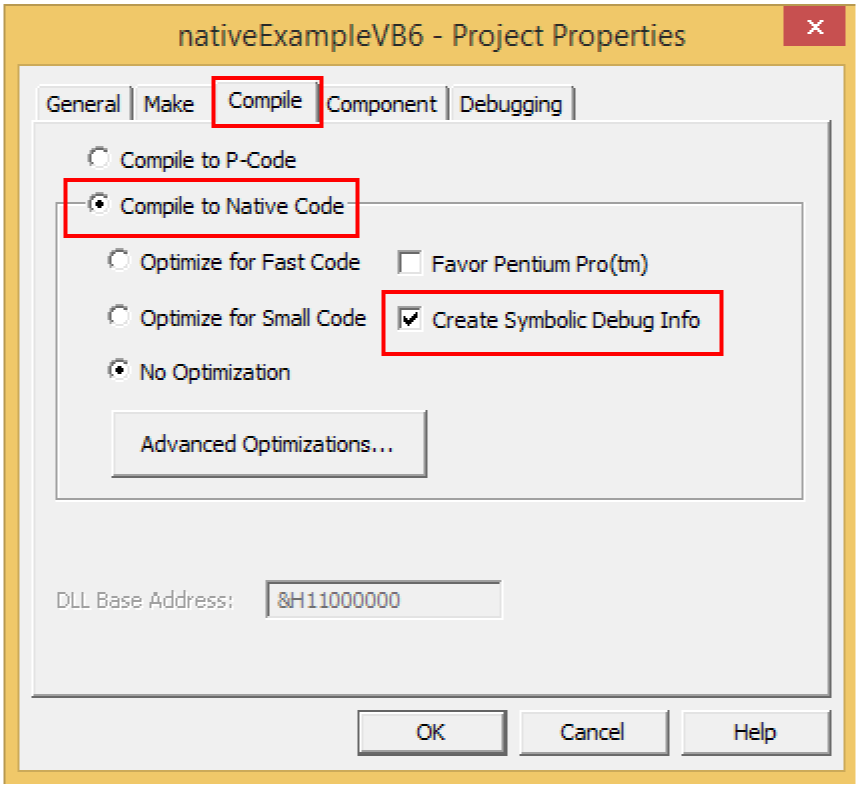
Metrowerks symbolic information is embedded in the .exe/.dll as CodeView information.

Please consult the documentation for CodeWarrior in order to include debug information (including filenames and line numbers) in the CodeView information.

If you still have problems, please contact us giving as much detail as possible, including what you've tried.

9.8 Visual Basic 6

To get debug symbols for Visual Basic you need to open the **Properties** dialog box from the **Project** menu (you'll find it at the bottom of the menu).


nativeExampleVB6 - Project Properties
General
Make
Compile
Component
Debugging
Advanced Optimizations...
OK
Cancel
Help

○ Compile to P-Code
● Compile to Native Code
○ Optimize for Fast Code
○ Optimize for Small Code
● No Optimization
□ Favor Pentium Pro(tm)
☑ Create Symbolic Debug Info

DLL Base Address: &H11000000

When you have changed your project properties you need to build the application.

Go to the **File** menu and choose **Make <projectname.exe>**.

Part X

10 Frequently Asked Questions

Here's a brief description about the type of question included in each of the following sections:

- General questions

  How Thread Validator works and how to do a few of the more common tasks.

- Not getting results

  Missing or unhooked data and not finding the data or callstacks you expected.

- Seeing unexpected data

  The data you are finding looks wrong or is unexpected.

- Crashes and error reports

  Your program crashes with Thread Validator or Thread Validator itself has a problem.

- DbgHelp

  Troubleshooting search paths for DbgHelp.dll, and finding or installing different versions.

- System and environment

  Your environment on the machine you are using Thread Validator with.

10.1 General Questions

⊟ Does Thread Validator work with NT Services?

Absolutely. There is a help section on working with NT Services.

⊟ Why might Inject or Launch fail?

Not using CreateProcess

The Inject and Wait for Application to Start functionality use `CreateRemoteThread` to inject into an application.

For the reasons below, injection using `CreateRemoteThread` does not always work.

Common reasons for injection failure

- A missing DLL in your application

  Check your application is complete.

- The target application is a .NET application or .NET service

  Check your application or service is not written using .NET technology.

- A missing DLL in Thread Validator

  Check Thread Validator is installed correctly.

- The application may have started and finished before the DLL could be injected

  This only applies if you are *launching* the application.

- The application security settings do not allow process handles to be opened

- The application is a service and is running with different privileges than Thread Validator

  If the application being injected into is a service it is recommended that the service and Thread Validator are both run on the same user account. See the topic on working with NT services.

Application Specific Reasons for Failure

A small percentage of applications/services will not allow any DLL to be injected into them.

The reasons for this are unknown, but our testing shows that the reason for failure to inject is a combination of application, operating system and hardware that causes an inconsistency during injection (we think it is a timing issue) that causes a failure.

Our tests show that on NT 4 about 1% of all applications fail to inject, 2% on Windows 2000 rising to 5% with Windows XP.

We expect that subsequent operating systems (Windows 2003 and Windows Vista) will have higher failure rates.

⊟ How do I name a thread?

Some features such as the Callstack tab can use thread names to make things a bit more intuitive.

Windows 10

Use the SetThreadDescription() Win32 API (or the methods described in the next section).

Earlier versions of Windows

From within your application you can provide a name for use by a debugger or debugging tool by using the Win32 RaiseException() API.

Add the function below to your application. This is based on an example from Microsoft and there are other examples available on the web; some specify a buffer size of 8 characters and one terminator, others specify no strict buffer size limit.

⊟ Show code

```
// This function is documented as being callable from outside of the thread which
// named, however it appears that it works more reliably if called from within th
// the thread being name, passing a threadId of -1 to indicate "current thread"

#ifndef DWORD_PTR

#ifdef _WIN64
#define DWORD_PTR   unsigned __int64
#else   //#ifdef _WIN64
#define DWORD_PTR   DWORD
#endif   //#ifdef _WIN64

#endif   //#ifndef DWORD_PTR

void CTeststakApp::nameThread(const DWORD   threadId,
                        const char   *name)
{
   // You can name your threads by using the following code.
   // Thread Validator will intercept the exception and pass it along (so if you
   // under a debugger the debugger will also see the exception and read the thre

   // NOTE: this is for 'unmanaged' C++ ONLY!

   #define MS_VC_EXCEPTION 0x406D1388
   #define BUFFER_LEN      16

   typedef struct tagTHREADNAME_INFO
   {
      DWORD   dwType;      // must be 0x1000
      LPCSTR   szName;      // pointer to name (in user addr space) buffer must b
      DWORD   dwThreadID;   // thread ID (-1 == caller thread)
      DWORD   dwFlags;   // reserved for future use, must be zero
   } THREADNAME_INFO;

   THREADNAME_INFO   ThreadInfo;
   char         szSafeThreadName[BUFFER_LEN];   // buffer can be any size, just m

   memset(szSafeThreadName, 0, sizeof(szSafeThreadName));   // ensure all charact
   strncpy(szSafeThreadName, name, BUFFER_LEN - 1);      // copying name
   //szSafeThreadName[BUFFER_LEN - 1] = '\0';

   ThreadInfo.dwType = 0x1000;
   ThreadInfo.szName = szSafeThreadName;
   ThreadInfo.dwThreadID = threadId;
   ThreadInfo.dwFlags = 0;

   __try
   {
      RaiseException(MS_VC_EXCEPTION, 0, sizeof(ThreadInfo) / sizeof(DWORD_PTR),
   }
   __except(EXCEPTION_EXECUTE_HANDLER)
   {
      // do nothing, just catch the exception so that you don't terminate the app
   }
}
```

After adding this function declaration you can call it from inside the thread procedure of any executing thread to name.

```
    nameThread(-1, "example");
```

To name a thread from *outside* of the thread procedure pass the thread id instead of -1.

The example application shipped with Thread Validator demonstrates how to use nameThread. See `nativeExample.cpp`.

⊟ How do I clear the symbol cache?

To clear Thread Validator's in-memory symbol cache, delete all sessions first:

- **Managers** Menu › **Session Manager** › **Delete All** › **Close**

Then flush the cache:

- **Settings** Menu › **Edit Settings** › **General** › **File Cache / Subst Drives** › click **Flush Cache** button › **OK**

Flush Cache disabled? Delete all the sessions *first.*

➡ See also: Clearing coverage cache.

⊟ I have an idea for a feature, can it be added to Thread Validator?

We have tried to add as many features to Thread Validator that we thought would be useful to our users.

In fact, every feature in Thread Validator has been used to solve problems and bugs for clients who consult us, and in our own business, so we know the features we have are useful.

However, maybe we overlooked a feature that you would find very useful.

We'll happily consider most ideas for new features to Thread Validator. But no Quake, FlightSim or Flappy Bird Easter eggs though, sorry!

Please contact us to let us know your thoughts.

10.2 Not getting results

⊟ Why don't I get any information about critical sections?

If you have started (or re-started) your application with data collection turned off, then Thread Validator will not collect or show any data.

⊟ Why can't I get callstacks for sleeping or suspended threads?

Thread Validator *can* display callstacks for sleeping or suspended threads.

To do this Thread Validator must hook the `SuspendThread`, `Sleep` and `SleepEx` functions.

Thread Validator will only hook these functions if you enable **Functions allocating waitable handles** on the Hook Insertion settings dialog and check the required options.

In addition, callstack collection must be enabled. See the Collect and Callstack tabs.

⊟ Why are my ordinal to symbol conversions not working?

If you have used the Ordinal Handling utility, you may still find the ordinal names have not been converted in symbol names,

- **You haven't defined the ordinal to symbol conversion for the correct DLL in which the ordinals are defined**

  Double check the DLL for which you defined the conversions really include those ordinals.

  ➡ See the topic on Ordinal Handling

10.3 Seeing unexpected data

⊟ Why do I get <UNKNOWN> symbols?

Some callstacks may not have a symbol name and can display the value <UNKNOWN>.

There are several reasons this may happen:

- **The program you are monitoring has no debugging information**

  You'll need to enable debugging information in your program.

  Debugging information is controlled from the Linker Tab on your Visual C++ project settings, and is available for Debug *and* Release builds.

- **The PDB files with the debug information can't be found**

  The program you are monitoring may have the debug information but Thread Validator can't find the debug information if it's stored in PDB files that are not in the current directory.

Use the File Paths dialog to set where the PDB files can be found.

If you don't have PDB files for a particular DLL, but do have MAP files, you can also set the location of these too.

- **A stack trace contains a location not present inside a DLL**

  This sometimes happens when hooks cause the program to jump to dynamically allocated memory holding the hook.

  These hooks will not have any debugging information referencing them.

- **The DLL has no filename and line-number data in the debugging information**

  This is the case for some release mode DLLs from Microsoft such as mfc42(u).dll and mfc80(u).dll.

  These only have symbol name information available, with filename and line-number removed.

  If you have this problem, you could try and get more up to date symbol information from Microsoft using the symbol server support page in the global settings dialog.

  Note that this will only work if the symbol server symbols *do actually contain* filename and line-number information - they might not!

If none of the above solves your problem and **all** symbols are still displayed as <UNKNOWN> please drop us a line. We have found that newer versions of Visual Studio sometimes change the debug information format and need a newer version of DbgHelp.dll. The version of DbgHelp.dll that is shipped with Thread Validator is compatible with Visual Studio.net and all previous versions of Visual Studio.

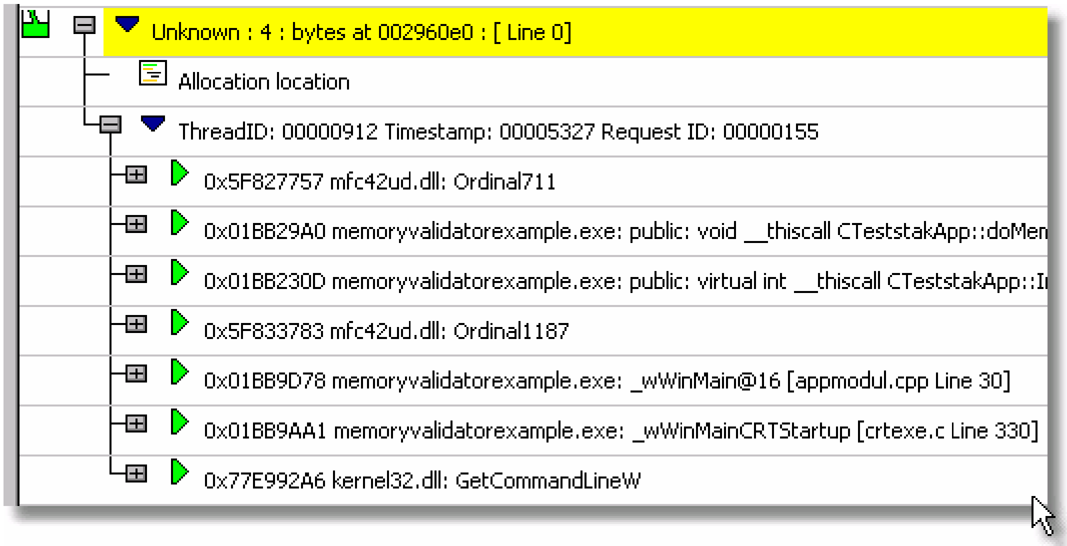
Some symbols are displayed as Ordinalxxx, why?

If Thread Validator can't find debug information for DLLs that have their functions exported as ordinal values, the functions are named `OrdinalXXX`, using the decimal number of the function.

These ordinal values *can* be displayed as function names, but only if the linker definition (.def) file that refers to each relevant DLL is known.

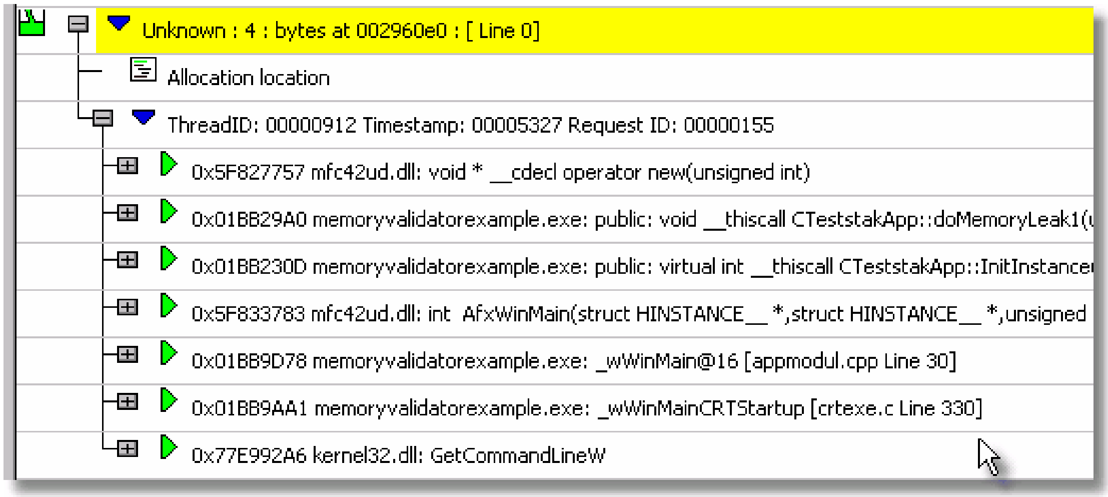
You can define the Ordinal Handling, but don't forget to also select the **Convert DLL exported function ordinals to symbols** check box on the Symbols and Warnings page of the global settings dialog.

Here's an example of Ordinal function names without debug information: `Ordinal711` and `Ordinal187`



```
Unknown : 4 : bytes at 002960e0 : [ Line 0]
  Allocation location
  ThreadID: 00000912 Timestamp: 00005327 Request ID: 00000155
    0x5F827757 mfc42ud.dll: Ordinal711
    0x01BB29A0 memoryvalidatorexample.exe: public: void __thiscall CTeststakApp::doMen
    0x01BB230D memoryvalidatorexample.exe: public: virtual int __thiscall CTeststakApp::I
    0x5F833783 mfc42ud.dll: Ordinal1187
    0x01BB9D78 memoryvalidatorexample.exe: _wWinMain@16 [appmodul.cpp Line 30]
    0x01BB9AA1 memoryvalidatorexample.exe: _wWinMainCRTStartup [crtexe.c Line 330]
    0x77E992A6 kernel32.dll: GetCommandLineW
```

And here's the same thing without debug information but after ordinal to symbol conversion, showing `__cdecl operator new()` and `int AfxWinMain()`



```
Unknown : 4 : bytes at 002960e0 : [ Line 0]
  Allocation location
  ThreadID: 00000912 Timestamp: 00005327 Request ID: 00000155
    0x5F827757 mfc42ud.dll: void * __cdecl operator new(unsigned int)
    0x01BB29A0 memoryvalidatorexample.exe: public: void __thiscall CTeststakApp::doMemoryLeak1(
    0x01BB230D memoryvalidatorexample.exe: public: virtual int __thiscall CTeststakApp::InitInstance
    0x5F833783 mfc42ud.dll: int  AfxWinMain(struct HINSTANCE__ *,struct HINSTANCE__ *,unsigned
    0x01BB9D78 memoryvalidatorexample.exe: _wWinMain@16 [appmodul.cpp Line 30]
    0x01BB9AA1 memoryvalidatorexample.exe: _wWinMainCRTStartup [crtexe.c Line 330]
    0x77E992A6 kernel32.dll: GetCommandLineW
```

⊟ What is address 0x006d0065?

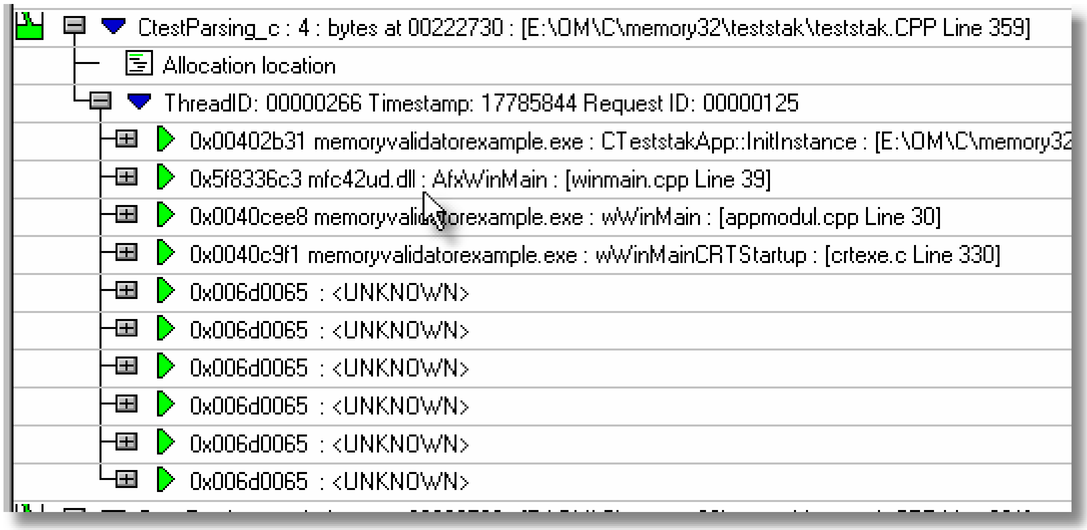
This is relevant to one version of the VisualStudio.net DbgHelp library that sometimes does not correctly identify the end of a stack walk.

When this happens, a stack trace can have numerous addresses of value `0x006d0065` tacked onto the end of it, even if the stack walking callback informs DbgHelp that the address is not valid.

This bug will not affect Thread Validator. All stack traces shown will be valid. A few stack traces may have extra data, but no data will be missing.

We don't filter these addresses out, in case a valid DLL does get loaded and uses this address space, producing symbols for this address.

Callstacks that contain this error look like this:



```
CtestParsing_c : 4 : bytes at 00222730 : [E:\OM\C\memory32\teststak\teststak.CPP Line 359]
    Allocation location
    ThreadID: 00000266 Timestamp: 17785844 Request ID: 00000125
        0x00402b31 memoryvalidatorexample.exe : CTeststakApp::InitInstance : [E:\OM\C\memory32
        0x5f8336c3 mfc42ud.dll : AfxWinMain : [winmain.cpp Line 39]
        0x0040cee8 memoryvalidatorexample.exe : wWinMain : [appmodul.cpp Line 30]
        0x0040c9f1 memoryvalidatorexample.exe : wWinMainCRTStartup : [crtexe.c Line 330]
        0x006d0065  : <UNKNOWN>
        0x006d0065  : <UNKNOWN>
        0x006d0065  : <UNKNOWN>
        0x006d0065  : <UNKNOWN>
        0x006d0065  : <UNKNOWN>
        0x006d0065  : <UNKNOWN>
```

As you can see from the image, the program started at the UNICODE entry point wWinMainCRTStartup, so there should be no symbols (other than GetPriorityBoost or other kernel32.dll symbols) after this entry.

Subsequent versions of the DbgHelp.dll from Microsoft fix this bug.

10.4 Crashes and error reports

⊟ The program I'm trying to monitor keeps crashing, why?

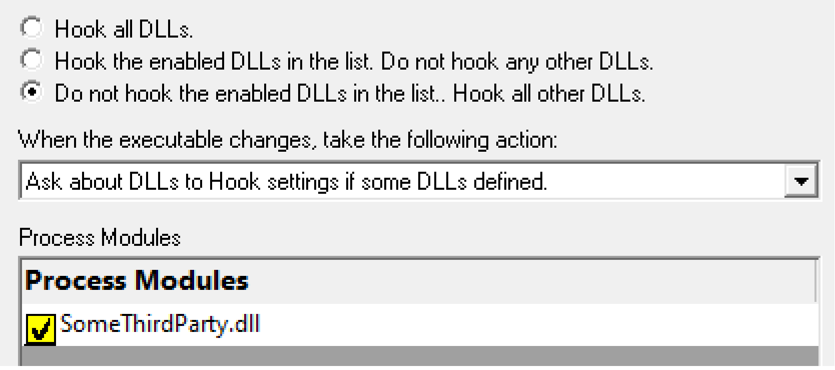
The following assumes your crash is one that only happens when using Thread Validator.

Here's a few scenarios in which your program *might* crash:

- **Third party DLLs are using system wide hooks**

  Some DLLs from third party vendors use system wide hooks and do not interact with Thread Validator and the target program very well.

  If you can identify such DLLs, prevent them being hooked by adding the DLL name to the Hooked DLLs page of the global settings dialog as in the example below.



○ Hook all DLLs.
○ Hook the enabled DLLs in the list. Do not hook any other DLLs.
● Do not hook the enabled DLLs in the list.. Hook all other DLLs.

When the executable changes, take the following action:

Ask about DLLs to Hook settings if some DLLs defined.

Process Modules

Process Modules
☑ SomeThirdParty.dll

- **Third party DLLs are using global hooks**

  A global hook DLL from a third party vendor could be adversely affecting Thread Validator when hooking your program.

  Read about handling global hooks on the Global Hooks page of the settings dialog.

  Judging by multiple independent error reports, we believe there may be an incompatibility between Thread Validator and the global hooks that come with the Matrox G400 and the Matrox Millenium II PCI video cards released in the late 1990's.

- **There may be a bug in Thread Validator**

  It happens. We've tried to make Thread Validator as robust as possible, but bugs and new scenarios do occur.

  First, ensure that the crash never happens if you are *not* using Thread Validator.

  Second, check all the suggestions above.

  Then drop us a line sending details of the error and we'll try to reproduce the crash with a view to fixing any bugs found in as timely a manner as possible.

⊟ Thread Validator gives an Unrecoverable Error?

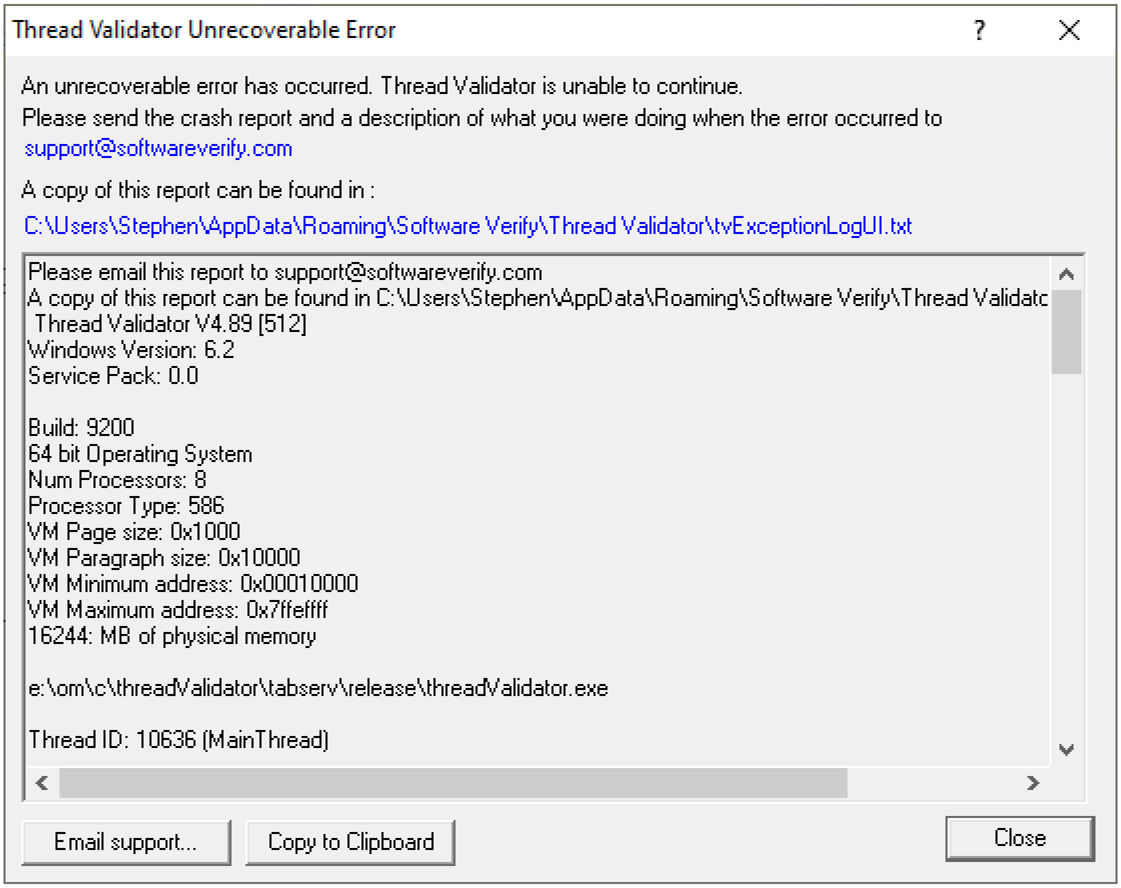
The Thread Validator Unrecoverable Error dialog is displayed when an unexpected internal error means Thread Validator cannot continue to execute.

A stack trace and register dump is shown and you can **Copy to Clipboard** so that the data can be sent to us with a description of the activities that caused the error.

We'll aim to fix any problems in as timely manner as possible.

The data shown in the dialog is also written to `c:\users\<user name>\AppData\Roaming\Software Verify\Thread Validator\tvExceptionLogUI.txt`.

The picture below shows a stack overflow exception report (artificially generated).



An unrecoverable error has occurred. Thread Validator is unable to continue.
Please send the crash report and a description of what you were doing when the error occurred to
support@softwareverify.com

A copy of this report can be found in :
C:\Users\Stephen\AppData\Roaming\Software Verify\Thread Validator\tvExceptionLogUI.txt

```
Please email this report to support@softwareverify.com
A copy of this report can be found in C:\Users\Stephen\AppData\Roaming\Software Verify\Thread Validatc
 Thread Validator V4.89 [512]
Windows Version: 6.2
Service Pack: 0.0

Build: 9200
64 bit Operating System
Num Processors: 8
Processor Type: 586
VM Page size: 0x1000
VM Paragraph size: 0x10000
VM Minimum address: 0x00010000
VM Maximum address: 0x7ffeffff
16244: MB of physical memory

e:\om\c\threadValidator\tabserv\release\threadValidator.exe

Thread ID: 10636 (MainThread)
```

What is in tvExceptionLogUI.txt and tvExceptionLog.txt?

In the event of a crash, the file `c:\users\<user name>\AppData\Roaming\Software Verify\Thread Validator\tvExceptionLogUI.txt` contains information that identifies where Thread Validator was executing when it crashed.

In the event of a crash, the file `c:\users\<user name>\AppData\Roaming\Software Verify\Thread Validator\tvExceptionLog.txt` contains information that identifies where the target process was executing when it crashed. This crash may be caused by an error in the target process or an error in the Thread Validator's instrumentation code.

The file contains a stack trace and register dump and is the same information that is displayed in the Unrecoverable Error dialog (above) when a crash occurred.

The file contains only the data for the *most recent* exception.

10.5 Debug symbols and DbgHelp

Why does Thread Validator fail to load my symbols?

In a few cases Thread Validator will fail to load symbols for a DLL that you believe you have provided symbols for.

This topic describes the possible causes. Please read the suggested course of action for each compiler.

⊟ Microsoft Visual Studio or Developer Studio

Symbols are defined in PDB files with the same name as the .exe or .dll to which it refers.

Thread Validator uses the Microsoft supplied DbgHelp.dll to perform all symbol handling activities.

Correct PDB name and location?

To ensure that the correct PDB is found to match a DLL the following must be true:

- The DLL and PDB file have the same name, except for the extension

  For example test.pdb matches for test.dll or test.exe.

- The first matching PDB file in the PDB search path has the correct checksum

  If DbgHelp finds a PDB file with a different checksum, loading symbols will fail but the search will still stop.

Verify that there are no PDB files with the same file name that are on the PDB search path, except for the PDB file you expect to be used.

You can check the DbgHelp symbol search path to troubleshoot symbol loading failures relating to the symbol search path.

Are compiler and linker producing symbols?

If DbgHelp is still failing to load your symbols, check the following:

- Your program is **compiled** to include symbol information

- Your program is **linked** to include symbol information

  Linker options are different to the compiler options

Running correct version of DLL?

Check that you are using:

- The **most recent** version of your DLL

- The **correct build** version of your DLL

  For example release DLL with release builds, debug DLL with debug builds

Checking for correctly loaded modules

When your application is running, check the modules being loaded by the application.

In Thread Validator, you can check the modules by using the Loaded Modules dialog, or by inspecting the Diagnostics tab.

You need to be sure that your application is not loading a different DLL with the same name from a different directory that is on the search path.

Correct version of DbgHelp.dll?

Try checking the version of DbgHelp.dll used by your Visual Studio installation and the version of DbgHelp.dll distributed with Thread Validator.

If the version used by Visual Studio is higher, it's possible Microsoft changed the PDB file format, making the symbols unreadable by Thread Validator.

To fix this:

- Copy the DbgHelp.dll from Visual Studio to the Thread Validator installation directory

- Remove any DbgHelp.dll from your application directory

  When Thread Validator launches an application it copies Thread Validator's DbgHelp.dll to the directory of the executable.

  This ensures that the DbgHelp.dll used is more recent than the default `system32\dbghelp.dll` which may not get updated.

  You need to find and remove these dlls - e.g. `c:\myapplication\debug\DbgHelp.dll` etc.

If all else fails...

Sometimes symbolic information will not load for unknown reasons.

In this circumstance, after trying the above suggestions, try changing the location in which symbols are sourced.

You could also try flushing and disabling the caching of symbols.

If you still have problems, please contact us giving as much detail as possible, including what you've tried.

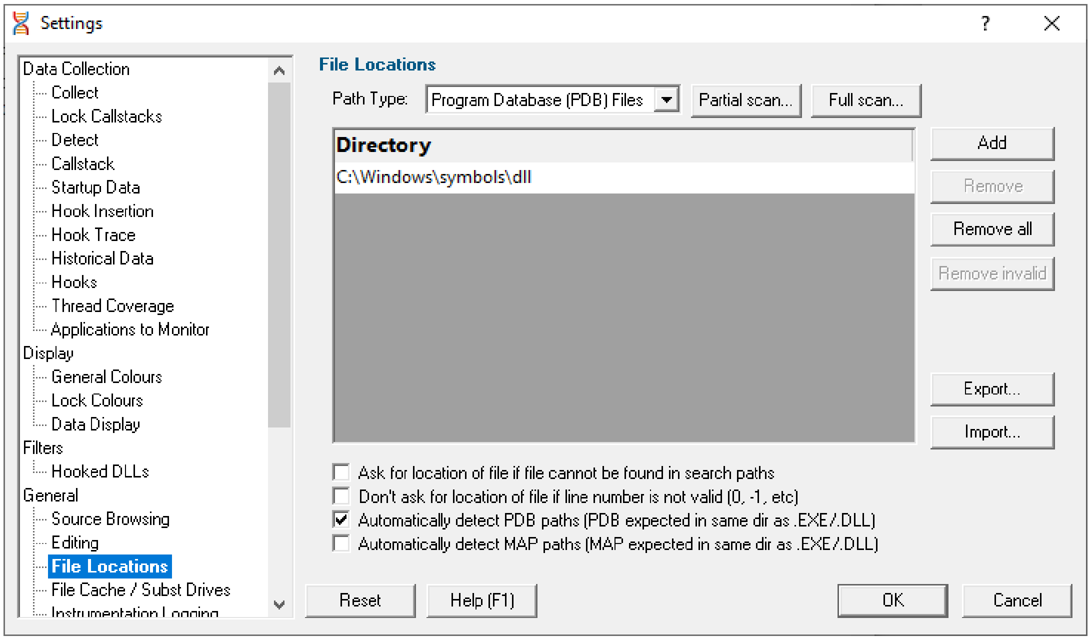
⊟ Visual Studio 2005 (8.0) and later versions

You may find that symbols for the `msvcr80.dll`, `msvcr80d.dll`, `mf80.dll`, `mfc80u.dll`, `mfc80d.dll` and `mfc80ud.dll` DLLs are not loaded.

The reason for this is that these symbols are stored in `c:\windows\symbols\dll` rather than with the DLLs themselves.

This is due to the Windows.NET Side-by-Side (WinSxS) DLL/assembly loading.

To resolve this, add the path **c:\windows\symbols\dll** to the list of paths for Program Database (PDB) Files on the File Locations tab:



Path Type: Program Database (PDB) Files

Directory
C:\Windows\symbols\dll

☐ Ask for location of file if file cannot be found in search paths
☐ Don't ask for location of file if line number is not valid (0, -1, etc)
☑ Automatically detect PDB paths (PDB expected in same dir as .EXE/.DLL)
☐ Automatically detect MAP paths (MAP expected in same dir as .EXE/.DLL)

You may need to restart Thread Validator to get valid symbols for `MFC80(u)(d).dll` if you have already recorded a session for which you did not get symbols.

Alternatively follow the instructions in the question on how to clear the symbol cache:

Metrowerks CodeWarrior for Windows V8 / V9

Metrowerks symbolic information is embedded in the .exe/.dll as CodeView information.

Please consult the documentation for CodeWarrior in order to include debug information (including filenames and line numbers) in the CodeView information.

If you still have problems, please contact us giving as much detail as possible, including what you've tried.

Salford Software Fortran 95

Salford Fortran 95 symbolic information is embedded in the .exe/.dll as COFF (Common Object File Format) information, with some proprietary extensions to Salford Software (which they have kindly shared with us).

Please consult the documentation for Salford FORTRAN95 to include debug information (including filenames and line numbers) in the COFF information.

If you still have problems, please contact us giving as much detail as possible, including what you've tried.

MingW compiler

We recommend compiling your software with `-gstabs` to create stabs debugging information.

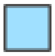
The `-gCoff` option is also supported, but this does create a lot of unnecessary symbols, making symbol parsing slower.

Troubleshooting DbgHelp.dll

Thread Validator uses the Microsoft Debugging DLL, DbgHelp.dll, copying the correct private version to your application's directory as your program is started.

However, there are cases where your application can be started independently, and you must ensure that your application uses the correct DbgHelp.dll.

Diagnostic error messages appear on the Diagnostics tab as in the example below detailing which version of DbgHelp.dll was expected and what was actually loaded.



If you see any DbgHelp warning dialogs, or get diagnostic errors, ensure the correct DbgHelp.dll is used by:

- **Copy (don't move) DbgHelp.dll**

  from: the Thread Validator install directory

  to: the location of the application being tested (the same directory as the .exe).

  Rerun your test.

- **Try updating the versions of DbgHelp.dll** in:

  `c:\windows\system32`

  and

  `c:\windows\system32\dllcache`

  Accept any Windows permission warnings if you try to do this.

  Rerun your test.

If you *still* continue to have problems, please drop us a line via our support email.

How do I examine (and fix) the DbgHelp symbol search path?

It can sometimes be quite confusing to see why symbols fail to load for modules built with compilers that generate PDB files, e.g.: Microsoft, Intel.

There are typically three reasons for failure: the PDB file is...

- **missing**, for example it was not provided with the executable
- **in the wrong place**, so the debugging library can't find it
- **the wrong version**, for example from a different build

The diagnostic tab

The Diagnostic tab of Thread Validator displays lots of messages that can help diagnose many problems.

To show only DbgHelp debug information, use the message filter drop down at the top of the diagnostic tab. This lets you examine where DbgHelp.dll looks for symbols.

Examine the output to see if it's finding the PDB file you think it should, and if it rejects the contents of any PDB file it finds.

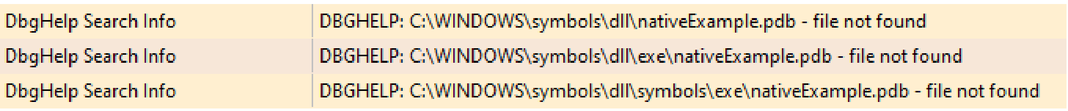
Output for alternate modules is shown in alternating coloursets, and the messages are the exact same output from the DbgHelp.dll debugging stream.

Examples of examining the diagnostics

Below we show three examples using nativeExample.exe and nativeExample.pdb from our example application.

- **Correct symbol file found**

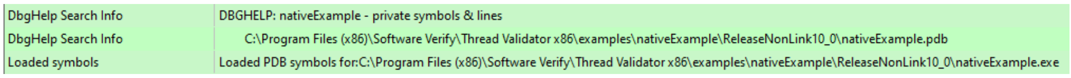
  DbgHelp first searches in various places looking for nativeExample.pdb



DbgHelp Search Info	DBGHELP: C:\WINDOWS\symbols\dll\nativeExample.pdb - file not found
DbgHelp Search Info	DBGHELP: C:\WINDOWS\symbols\dll\exe\nativeExample.pdb - file not found
DbgHelp Search Info	DBGHELP: C:\WINDOWS\symbols\dll\symbols\exe\nativeExample.pdb - file not found

Depending on your machine, there may be other search paths included.

Finally nativeExample.pdb is found in the same directory as the .exe file of the target program



DbgHelp Search Info	DBGHELP: nativeExample - private symbols & lines
DbgHelp Search Info	C:\Program Files (x86)\Software Verify\Thread Validator x86\examples\nativeExample\ReleaseNonLink10_0\nativeExample.pdb
Loaded symbols	Loaded PDB symbols for:C:\Program Files (x86)\Software Verify\Thread Validator x86\examples\nativeExample\ReleaseNonLink10_0\nativeExample.exe

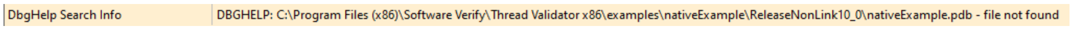
DbgHelp loads private symbols and lines, (the alternative being that DbgHelp loads public symbols).

Outcome:
Success. Symbols are loaded.

- **Missing symbol file**

  As before, DbgHelp first searches in various places looking for nativeExample.pdb

  But, nativeExample.pdb doesn't get found in the same directory as the .exe file of the target program.



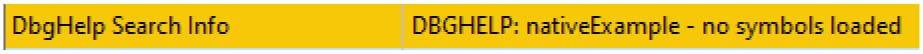
```
DbgHelp Search Info    DBGHELP: C:\Program Files (x86)\Software Verify\Thread Validator x86\examples\nativeExample\ReleaseNonLink10_0\nativeExample.pdb - file not found
```

  nativeExample.pdb never gets found on the search path.

  SymSrv might then look for additional locations for nativeExample.pdb, but has no luck.

  DbgHelp might find some COFF symbols in the executable, however these don't contain filename or line number information.

  Finally all options are exhausted.



```
DbgHelp Search Info    DBGHELP: nativeExample - no symbols loaded
```

  Outcome:
  Failure. The PDB file could not be found. Some default symbols are loaded but are not of much use.

  Resolution:
  Check the File Locations PDB paths to ensure that all the possible paths for PDB files are listed.

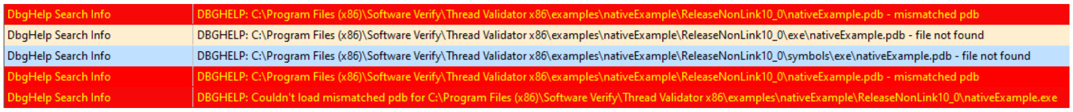
- **Incorrect symbol file**

  As before, DbgHelp first searches in various places looking for nativeExample.pdb

  This time, nativeExample.pdb *does* get found in the same directory as the .exe file of the target program.

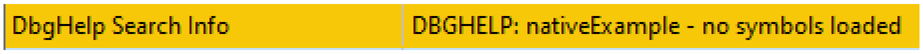
  DbgHelp tries to load the symbols but fails - the checksum inside the PDB file does not match the module.

  This might be because the symbols are for a different build of the software, or it's an incorrectly named PDB file belonging to another program.



```
DbgHelp Search Info    DBGHELP: C:\Program Files (x86)\Software Verify\Thread Validator x86\examples\nativeExample\ReleaseNonLink10_0\nativeExample.pdb - mismatched pdb
DbgHelp Search Info    DBGHELP: C:\Program Files (x86)\Software Verify\Thread Validator x86\examples\nativeExample\ReleaseNonLink10_0\exe\nativeExample.pdb - file not found
DbgHelp Search Info    DBGHELP: C:\Program Files (x86)\Software Verify\Thread Validator x86\examples\nativeExample\ReleaseNonLink10_0\symbols\exe\nativeExample.pdb - file not found
DbgHelp Search Info    DBGHELP: C:\Program Files (x86)\Software Verify\Thread Validator x86\examples\nativeExample\ReleaseNonLink10_0\nativeExample.pdb - mismatched pdb
DbgHelp Search Info    DBGHELP: Couldn't load mismatched pdb for C:\Program Files (x86)\Software Verify\Thread Validator x86\examples\nativeExample\ReleaseNonLink10_0\nativeExample.exe
```

  Finally all options are exhausted.



```
DbgHelp Search Info    DBGHELP: nativeExample - no symbols loaded
```

Outcome:
Failure. A PDB file was found, but it was not the right one.

Resolutions:
Double check the PDB is the correct one for the build you are running.
When copying builds from another machine (or from a build server), make sure to copy the correct PDB as well.
Check the File Locations PDB paths to ensure that all the possible paths for PDB files are listed.
Check the order of those PDB paths in case there are multiple paths resulting in the wrong PDB being found first.

How can I create a map file with line numbers

If you don't have the ability to use .PDB files for debug information, you may be able to use .MAP files with line information.

The following is only applicable to Debug builds. Map files for Release builds can't have line number data.

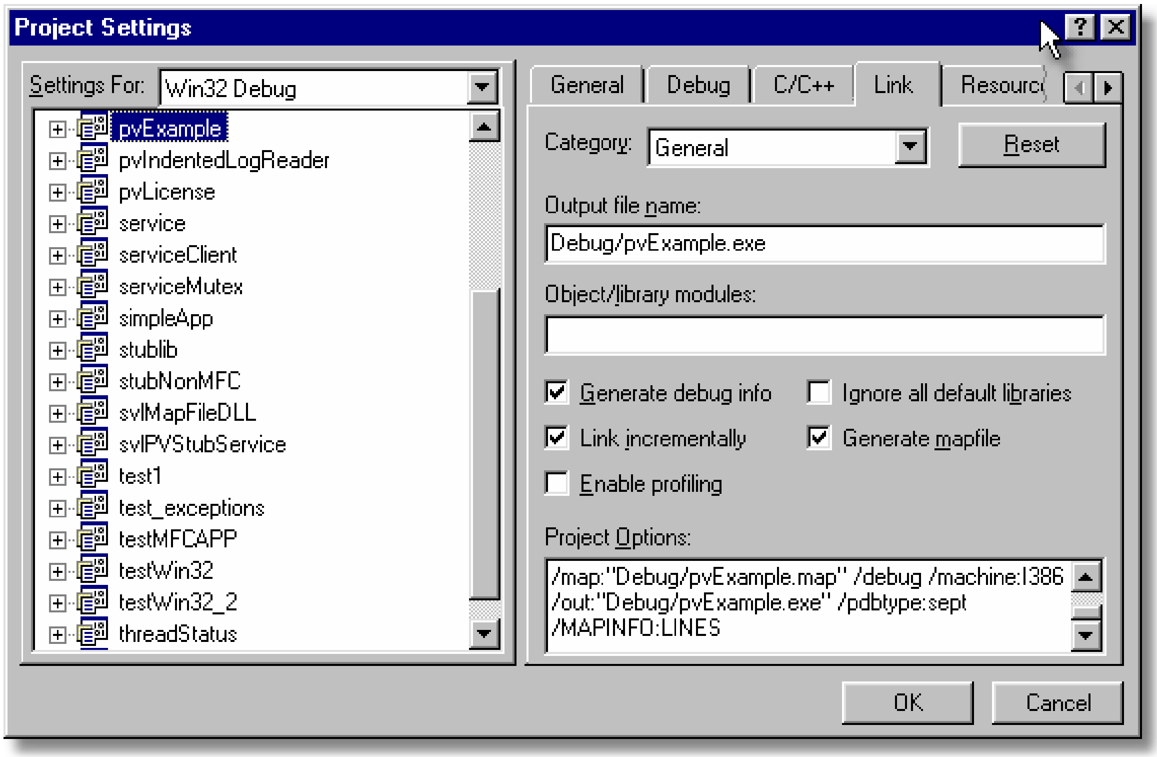
Microsoft discontinued support for including line information in .MAP files with Visual Studio 8.0 (2005). There is no easy workaround to this.

To select the `/MAPINFO:LINES` option for Visual Studio 6.0 use the following steps. If you are using Visual Studio 7.0, 7.1 (i.e .NET 2002 or 2003) the project settings user interface is slightly different, but the basic principle remains the same.

In Visual Studio:

Project Menu › **Settings...** › Select project › Shows project settings

The example image below shows a project called **nativeExample**.



Settings For: Win32 Debug

Category: General

Output file name: Debug/pvExample.exe

Object/library modules:

☑ Generate debug info ☐ Ignore all default libraries

☑ Link incrementally ☑ Generate mapfile

☐ Enable profiling

Project Options:

```
/map:"Debug/pvExample.map" /debug /machine:I386
/out:"Debug/pvExample.exe" /pdbtype:sept
/MAPINFO:LINES
```

- **Generate mapfile** > check option to request MAP file output
- **Project Options** > add the text `/MAPINFO:LINES` to add line information to the file > **OK**

Save your project workspace and build your project.

Due to daylight saving times it is possible for a MAP file to have an embedded timestamp that is different than the DLL timestamp by an hour. In these situations Thread Validator will not recognise the MAP as valid. The solution to this problem is to rebuild the application.

10.6 Extensions, services and tools

Including stublib.h in my project doesn't compile. Why?

You may encounter problems when including `stublib.h` in order to link directly with Thread Validator.

Include path problems

Ensure that your project **C preprocessor include paths** reference both of the **stub** and **stublib** subdirectories in the installation directory of Thread Validator.

For example, if Thread Validator is installed in:

```
C:\Program Files (x86)\Software Verify\Thread Validator
```

Then add the following paths for *all* configurations; Debug, Release, etc:

```
C:\Program Files (x86)\Software Verify\Thread Validator\stub
C:\Program Files (x86)\Software Verify\Thread Validator\stublib
```

Compiler errors

If you include `stublib.h`, your project must have included `windows.h` first, (or see below for an alternative).

If you fail to include `windows.h` then `stublib.h` will refer to some none-existent datatypes, causing compiler errors similar to the ones shown below.

Here's an example program that will not compile:

```
#include "stdafx.h"
#include "stublib.h"

int main(int argc, char* argv[])
{
   return 0;
}
```

⊟ See the compiler errors from the above code

```
--------------------Configuration: testMV_allEnum - Win32
Debug--------------------
Compiling...
testMV_allEnum.cpp
c:\program files\software verification\thread validator\stub\allenum.h(70) :
error C2146: syntax error : missing ';' before identifier 'lRequest'
c:\program files\software verification\thread validator\stub\allenum.h(70) :
error C2501: 'LONG' : missing storage-class or type specifiers
c:\program files\software verification\thread validator\stub\allenum.h(70) :
error C2501: 'lRequest' : missing storage-class or type specifiers
c:\program files\software verification\thread validator\stub\allenum.h(71) :
error C2146: syntax error : missing ';' before identifier 'reserved3'
c:\program files\software verification\thread validator\stub\allenum.h(71) :
error C2501: 'DWORD' : missing storage-class or type specifiers
c:\program files\software verification\thread validator\stub\allenum.h(71) :
error C2501: 'reserved3' : missing storage-class or type specifiers
c:\program files\software verification\thread validator\stub\allenum.h(73) :
error C2143: syntax error : missing ';' before '*'
c:\program files\software verification\thread validator\stub\allenum.h(73) :
error C2501: 'BYTE' : missing storage-class or type specifiers
c:\program files\software verification\thread validator\stub\allenum.h(74) :
error C2501: 'dde_pbData' : missing storage-class or type specifiers
```

To fix this problem simply include `windows.h` before `stublib.h`

```
#include "stdafx.h"
#include <windows.h>      // new line to fix compile errors
#include "stublib.h"

int main(int argc, char* argv[])
{
   return 0;
}
```

Can't include windows.h?

If including `windows.h` is not an option, you can just define the following types:

```
#define LONG long
#define DWORD unsigned long
#define BYTE unsigned char
#define HANDLE void *
```

⊟ What do I do if I cannot use svlMVStubService.lib?

You may find that you can't use **svlTVStubService.lib / svlTVStubService_x64.lib** because your linker doesn't understand the format of the lib file.

If that happens you can use the code below to compile the two functions that would be provided by those libraries.

⊟ See the header file

```
#ifndef _SVL_TVSTUB_SERVICE_H
#define _SVL_TVSTUB_SERVICE_H
```

```
#include "svlServiceError.h"

// IMPORTANT.
// If you use svlTVStub_LoadThreadValidator() to load svlThreadValidatorStub.dll in
// application, you must also use svlTVStub_UnloadThreadValidator() to unload the D
// your application being closed down. Failure to do so will almost certainly resul
// It does not matter how the application is closed down, you must ensure that you
// svlTVStub_UnloadThreadValidator() to unload the DLL if you have loaded it.
//
// The DLL prepares itself in different ways and shuts itself down differently depe
// it is:-
// a) Directly linked to the application for use with the API or injected with Thre
//    When the DLL is used in this manner to DLL expects to oversee and manage the
//    shutdown.
// b) Loaded by using svlTVStub_LoadThreadValidator().
//    When the DLL is used in this manner to DLL expects to be removed prior to app
//    and the behaviour of the DLL is undefined once you enter the program shutdown
//
//     This difference in behaviour is intentional and is done to allow the use of
//     services.

#ifdef __cplusplus
extern "C" {
#endif

SVL_SERVICE_ERROR svlTVStub_LoadThreadValidator(serviceCallback_FUNC callback,
                                                void                 *userParam);

SVL_SERVICE_ERROR svlTVStub_UnloadThreadValidator();

#ifdef __cplusplus
}
#endif

#endif
```

⊟ See the implementation file

```
#include "svlTVStubService.h"

#include <windows.h>
#include <tchar.h>

//-NAME---------------------------------
//.DESCRIPTION..........................
//.PARAMETERS...........................
//.RETURN.CODES.........................
//--------------------------------------

static HMODULE   hModule = NULL;

//-NAME---------------------------------
//.DESCRIPTION..........................
//.PARAMETERS...........................
//.RETURN.CODES.........................
//--------------------------------------

typedef void (*ENABLE_STUB_SYMBOL_FUNC)();

SVL_SERVICE_ERROR svlTVStub_LoadThreadValidator(serviceCallback_FUNC callback,
                                                void                 *userParam)
{
   SVL_SERVICE_ERROR   errCode = SVL_OK;

   if (hModule == NULL)
   {
      hModule = LoadLibraryW(L"svlThreadValidatorStub.dll");     // change this to
      if (hModule != NULL)
      {
         // DLL loaded, set the service callback function

         SETCALLBACK_FUNC   setCallbackFunc;

         setCallbackFunc = (SETCALLBACK_FUNC)GetProcAddress(hModule, "apiSetService
         if (setCallbackFunc != NULL)
         {
            (*setCallbackFunc)(callback, userParam);
         }

         // now start the profiler

         PROC *p;

         p = GetProcAddress(hModule, "startProfiler");
         if (p != NULL)
             (*p)();
         // now turn on provision of symbols by the stub

         ENABLE_STUB_SYMBOL_FUNC   enableSymbolFunc;

         enableSymbolFunc = (ENABLE_STUB_SYMBOL_FUNC)GetProcAddress(hModule, "apiEn
         if (enableSymbolFunc != NULL)
         {
            (*enableSymbolFunc)();
         }
         else
         {
            errCode = SVL_FAILED_TO_ENABLE_STUB_SYMBOLS;
         }
      }
      else
      {
         errCode = SVL_LOAD_FAILED;
      }
   }
   else
   {
      errCode = SVL_ALREADY_LOADED;
   }

   return errCode;
}

//-NAME---------------------------------
//.DESCRIPTION..........................
//.PARAMETERS...........................
//.RETURN.CODES.........................
//--------------------------------------

typedef void (*UNLOAD_FUNC)();
typedef HANDLE (*GET_STUB_HEAP_FUNC)();

SVL_SERVICE_ERROR svlTVStub_UnloadThreadValidator()
{
   SVL_SERVICE_ERROR   errCode = SVL_OK;

   if (hModule != NULL)
   {
      // get the stub heap before we shut down the DLL

      HANDLE           hStubHeap = NULL;
      GET_STUB_HEAP_FUNC   getHeapFunc;

      getHeapFunc = (GET_STUB_HEAP_FUNC)GetProcAddress(hModule, "apiGetInternalMVst
      if (getHeapFunc != NULL)
      {
         hStubHeap = (*getHeapFunc)();
      }

      // get the unload stub function
      UNLOAD_FUNC   unloadFunc;

      unloadFunc = (UNLOAD_FUNC)GetProcAddress(hModule, "apiShutdownThreadValidator
      if (unloadFunc != NULL)
      {
         (*unloadFunc)();

         // get the function

         HMODULE   hModule;

         hModule = GetModuleHandleW(L"svlThreadValidatorStub.dll");
         if (hModule != NULL)
         {
            // unload the stub

            FreeLibrary(hModule);

            // destroy the stub's heap (which was still in use whilst FreeLibrary()

            if (hStubHeap != NULL)
               HeapDestroy(hStubHeap);
            else
            {
               if (errCode == SVL_OK)
                  errCode = SVL_FAIL_TO_CLEANUP_INTERNAL_HEAP;
            }
         }
         else
         {
            errCode = SVL_FAIL_MODULE_HANDLE;
         }
      }
      else
      {
         errCode = SVL_FAIL_UNLOAD;
      }

      hModule = NULL;
   }
   else
      errCode = SVL_NOT_LOADED;

  return errCode;
}
```

10.7 System and environment

⊟ How do I create a Power User on Windows XP?

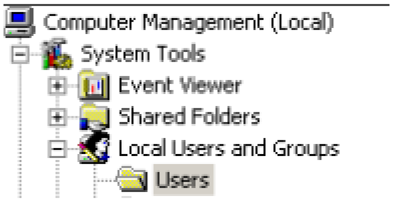
Windows 2000 and Windows XP Pro allow Power User accounts that stop short of full Administrator permissions.

To make an existing user (say **Test User**) a Power User do the following:

- **Start Menu** > **Right click** on **My Computer** > **Manage**

  The Computer Management window appears

- On the left, expand **System Tools** > **Local Users and Groups** > **Users**



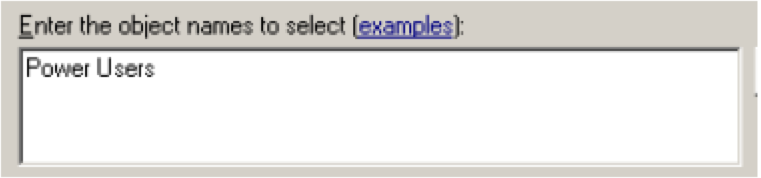
- On the right, select and **Right click** on '**Test User**' > **Properties**

  The User Properties dialog appears

- Select the **Member Of** tab > **Add...**

  The Select Groups dialog appears

- In the bottom box, **type** `Power Users` > **OK**



Enter the object names to select (examples):

Power Users

- In the user properties dialog select **Users** > **Remove** > **OK**

- Close **Computer Management**

Your **Test User** is now a member of the Power Users group - and probably not really a 'Test' User any more!

⊟ What file extensions does Thread Validator use?

Most configuration data is stored in the registry, but some information is file-based such as settings, coverage, hook and filter data.

Thread Validator uses the following extensions:

Sessions, Settings, Hooks and File Locations

tvm	Session files for 32 bit or 64 bit Thread Validator
tvm_x64	
tvs	Settings for 32 bit or 64 bit
tvs_x64	
tvx	Hooked DLLs
tvxfl	File locations

Session Export

html	HTML export files
xml	XML export files

Program Launch, Extensions

dll	Extension DLLs
exe	Program files

Part XI

11 Installing Floating Licensing

How to install floating licences

Floating licences float globally. Your team members in an office on the other side of the world can share a floating licence with you.

If you have floating licences install the software on all machines in your business unit that wish to use the software.

For an overview of how floating licences work, please read this.

Floating licence server

The floating licence server is managed by Software Verify.

No server to setup, no licences to misconfigure. All the things that are bad about floating licences, we've removed all that.

If you need to acquire a licences or release a licence, see the Floating Licences tab.

Floating licence help

If you have problems with the floating licences please contact support@softwareverify.com

If you need to purchase additional floating licences for a new floating licence please visit https://www.softwareverify.com/purchasing/.

If you need to purchase additional floating licences to add to an existing floating licence please contact sales@softwareverify.com.

Part XII

12 Copyright notices

12.1 Udis86

This software uses the library svlUdis86.dll and svlUdis86_x64.dll. These libraries are modified binary versions of the open source disassembler udis86.

udis86 was hosted at http://udis86.sourceforge.net/
udis86 is currently hosted at https://github.com/vmt/udis86 although the current distribution (at the time of writing) appears to be missing some files required to compile.

The 1.7.0 version of the udis source code contains this copyright notice: Copyright (c) 2005, 2006, Vivek Mohan
The 1.7.2 version of the udis source code contains this copyright notice: Copyright (c) 2002-2009 Vivek Thampi

These copyright notices appear to conflict and the latter copyright notice completely ignores the claims set forth in the 1.7.0 copyright notice.

In accordance with the license terms in the 1.7.2 software we include this binary license.

Index