Thread Validator Help

USER MANUAL

Copyright © 2003-2017 Software Verify Limited
Software Verify
# Table of Contents

1. **Overview**
   1.1 Notation used in this help ................................................................. 8
   1.2 Introducing Thread Validator .............................................................. 10
   1.3 Why Thread Validator? ........................................................................ 11
   1.4 What do you need to run Thread Validator? ........................................ 13
   1.5 Buying Thread Validator and Support ................................................ 15
   1.6 How does Thread Validator work? ...................................................... 15
   1.7 Supported Compilers .......................................................................... 16
   1.8 User Permissions ................................................................................ 18

2. **Getting Started**
   2.1 Enabling Debugging ........................................................................... 25
      2.1.1 Microsoft Compilers ....................................................................... 25
      2.1.2 Borland Compilers (C++ and Delphi) ............................................. 26
      2.1.3 MinGW - gcc/g++ ........................................................................... 26
      2.1.4 QtCreator ....................................................................................... 26
      2.1.5 Other Compilers ........................................................................... 27
   2.2 Quick Start ........................................................................................... 27

3. **The User Interface**
   3.1 First run configuration ....................................................................... 33
   3.2 Menu Reference .................................................................................. 42
      3.2.1 File menu ....................................................................................... 43
      3.2.2 Edit menu ....................................................................................... 44
      3.2.3 Settings menu ............................................................................... 44
      3.2.4 Managers menu ............................................................................. 45
      3.2.5 Query menu .................................................................................. 45
      3.2.6 Tools menu ................................................................................... 46
      3.2.7 Data Views menu .......................................................................... 46
      3.2.8 Software Updates menu ................................................................. 47
      3.2.9 Help menu .................................................................................... 48
   3.3 Toolbar Reference ................................................................................ 48

   3.4 The status bar ..................................................................................... 50

   3.5 Keyboard Shortcuts ............................................................................ 51

   3.6 Icons ................................................................................................... 52

   3.7 The main display ............................................................................... 52
      3.7.1 Summary ....................................................................................... 53
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9.2</td>
<td>Loading and Saving Settings</td>
<td>172</td>
</tr>
<tr>
<td>3.9.3</td>
<td>User Permissions Warnings</td>
<td>172</td>
</tr>
<tr>
<td>3.9.4</td>
<td>Symbol Path Truncated Warning</td>
<td>173</td>
</tr>
<tr>
<td>3.10</td>
<td>Ordinal Handling</td>
<td>176</td>
</tr>
<tr>
<td>3.11</td>
<td>Managers</td>
<td>180</td>
</tr>
<tr>
<td>3.11.1</td>
<td>Session Manager</td>
<td>180</td>
</tr>
<tr>
<td>3.11.2</td>
<td>Thread Filter</td>
<td>182</td>
</tr>
<tr>
<td>3.11.3</td>
<td>Watermarks</td>
<td>183</td>
</tr>
<tr>
<td>3.11.4</td>
<td>Bookmarks</td>
<td>186</td>
</tr>
<tr>
<td>3.12</td>
<td>Query and Search</td>
<td>188</td>
</tr>
<tr>
<td>3.12.1</td>
<td>Finding synchronization objects</td>
<td>188</td>
</tr>
<tr>
<td>3.12.2</td>
<td>Finding functions</td>
<td>192</td>
</tr>
<tr>
<td>3.12.3</td>
<td>Finding critical sections only used in one thread</td>
<td>195</td>
</tr>
<tr>
<td>3.12.4</td>
<td>Fetching callstacks for all threads</td>
<td>197</td>
</tr>
<tr>
<td>3.12.5</td>
<td>Deadlock detection</td>
<td>200</td>
</tr>
<tr>
<td>3.12.6</td>
<td>Potential deadlock detection</td>
<td>202</td>
</tr>
<tr>
<td>3.12.7</td>
<td>Display lock order</td>
<td>205</td>
</tr>
<tr>
<td>3.13</td>
<td>Tools</td>
<td>210</td>
</tr>
<tr>
<td>3.13.1</td>
<td>Colour coded source code editor</td>
<td>211</td>
</tr>
<tr>
<td>3.13.2</td>
<td>Refresh and Refresh All</td>
<td>215</td>
</tr>
<tr>
<td>3.13.3</td>
<td>Loaded Modules</td>
<td>216</td>
</tr>
<tr>
<td>3.13.4</td>
<td>DLL Debug Information</td>
<td>217</td>
</tr>
<tr>
<td>3.13.5</td>
<td>Instrumentation Logging Data</td>
<td>223</td>
</tr>
<tr>
<td>3.14</td>
<td>Software Updates</td>
<td>224</td>
</tr>
<tr>
<td>3.15</td>
<td>Loading, Saving, Exporting, Closing</td>
<td>230</td>
</tr>
<tr>
<td>3.15.1</td>
<td>Loading &amp; Saving Sessions</td>
<td>231</td>
</tr>
<tr>
<td>3.15.2</td>
<td>Exporting Sessions</td>
<td>232</td>
</tr>
<tr>
<td>3.15.2.1</td>
<td>XML Export Tags</td>
<td>234</td>
</tr>
<tr>
<td>3.16</td>
<td>Starting your target program</td>
<td>237</td>
</tr>
<tr>
<td>3.16.1</td>
<td>Launching the program</td>
<td>238</td>
</tr>
<tr>
<td>3.16.2</td>
<td>Re-Launching the program</td>
<td>246</td>
</tr>
<tr>
<td>3.16.3</td>
<td>Injecting into a running program</td>
<td>246</td>
</tr>
<tr>
<td>3.16.4</td>
<td>Waiting for a program</td>
<td>251</td>
</tr>
<tr>
<td>3.16.5</td>
<td>Monitor a service</td>
<td>257</td>
</tr>
<tr>
<td>3.16.6</td>
<td>Linking to a program</td>
<td>259</td>
</tr>
<tr>
<td>3.16.7</td>
<td>Environment Variables</td>
<td>259</td>
</tr>
<tr>
<td>3.17</td>
<td>Stopping your target program</td>
<td>261</td>
</tr>
<tr>
<td>3.18</td>
<td>Data Collection</td>
<td>261</td>
</tr>
<tr>
<td>3.19</td>
<td>Help</td>
<td>262</td>
</tr>
</tbody>
</table>
# Table of Contents

4. **Command Line Interface** 266  
   4.1 Target Program & Start Modes ................................................................. 268  
   4.2 User interface visibility .............................................................................. 273  
   4.3 Session Management .................................................................................. 276  
   4.4 Session Export Options ............................................................................ 277  
   4.5 File Locations ............................................................................................. 278  
   4.6 Command Files ........................................................................................... 280  
   4.7 Help, Errors & Return Codes ..................................................................... 281  
   4.8 Command Line Reference ......................................................................... 283  

5. **Working with NT Services** 287  
   5.1 NT Service API .......................................................................................... 289  
   5.2 Example Service Source Code .................................................................... 295  

6. **Examples** 303  
   6.1 Example Application .................................................................................. 304  
      6.1.1 Building the example application ......................................................... 308  
   6.2 Example NT Service .................................................................................... 309  
      6.2.1 Building the example service ............................................................... 309  
      6.2.2 Building the example client ................................................................. 310  
      6.2.3 Building the example service utility .................................................... 311  

7. **Hook Reference** 313  
   7.1 Synchronization Hooks .............................................................................. 314  

8. **Frequently Asked Questions** 317  
   8.1 General Questions ..................................................................................... 318  
   8.2 Not getting results ..................................................................................... 321  
   8.3 Seeing unexpected data ............................................................................ 322  
   8.4 Crashes and error reports ......................................................................... 325  
   8.5 Debug symbols and DbgHelp .................................................................... 327  
   8.6 Extensions, services and tools .................................................................. 336  
   8.7 System and environment ......................................................................... 343  

9. **Copyright notices** 345  
   9.1 Udis86 ...................................................................................................... 346
| Index | 347 |
Overview
1 Overview

Hi, welcome to the Thread Validator help manual.

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

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 Verify 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?

C++ 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 C++ 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.
Overview

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**: Shows a table summarising what each critical section is doing, including any WaitForSingleObject and WaitForMultipleObject calls.
- **Per Thread Locks**: Displays information about the critical sections used by each thread in the application.
- **Current Locks**: Displays information about the critical sections currently locked or waiting for each thread in the application.
- **Threads**: This display provides a graphical view of thread activity over time.
- **Coverage**: Displays source code containing thread synchronization operations and lets you find untested areas. Requires coverage data collection to be enabled.
- **Active Objects**: This view displays information for each type of synchronization object and synchronization API call.
- **Analysis**: Provides the ability to perform queries on the collected data and to display related information.
- **Objects**: Lists all the objects in the program, their type and number allocated.
- **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 affect 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:
Overview

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.
Overview

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**
Overview

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](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
Overview

- 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
- etc...

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

» Microsoft Compilers 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.

Borland http://www.borland.com

- Borland C++ Builder 5.0
- Borland C++ Builder 5.5
- Borland C++ Builder 6.0
- Borland C++ free command line tools
- Borland Delphi 6.0
- Borland Delphi 7.0

Some later versions of Borland C++ and Borland Delphi are also supported - if the software can produce TDS format debugging data our software tools can work with them.

» Borland Compilers in the Getting Started section.
Overview

**MinGW** [http://www.mingw.org](http://www.mingw.org)

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

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

⇒ [MinGW compiler](#) in the *Getting Started* section.

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

- **QtCreator**
  
  ⇒ [QtCreator](#) in the *Getting Started* section.

**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 Intel Visual Fortran product may be compatible with Thread Validator. If you are using Intel 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.

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
Overview

Click Add... > Show Users

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
Overview

- 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:

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

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]

Insufficient user privileges

The following dialog is displayed if a user has insufficient privileges to use the software correctly.
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](#)
Getting Started
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

#### 2.1.1 Microsoft Compilers

You must ensure that your application is built with debugging information.

**Enabling debugging information**

Debugging information must be enabled for debug builds (the default) and for release builds (disabled by default).

To enable debugging information for a DLL/EXE:

- open the project settings/solution for that DLL/EXE
- select the build type (debug or release)
- modify the compiler settings to enable debugging information
  e.g. **Configuration Properties** → **C/C++** → **General** → **Debug Information Format** → e.g. **Program Database (Zi)**
- modify the linker settings to enable debugging information
  e.g. **Configuration Properties** → **Linker** → **Debugging** → **Generate Debug Info** → e.g. **Yes (/ DEBUG)**

You must enable both compiler and linker settings to generate debugging information.
It is not possible to create an .exe or .dll that contains debug information. Debug information is always placed in a .pdb file.

⇒ MSDN page on Generating Debug Info

⇒ Enabling debugging information with Borland, MinGW, QtCreator and other compilers.

2.1.2 Borland Compilers (C++ and Delphi)

You must ensure that your application is built with debugging information.

Enabling debugging information

Debugging information must be enabled for debug builds (the default) and for release builds (disabled by default).

You should modify your project settings so that debugging information is created in the TDS format.

TDS debugging data can be stored in the DLL/EXE being built or in a separate TDS file.

⇒ Enabling debugging information with Microsoft, MinGW, QtCreator and other compilers.

2.1.3 MinGW - gcc/g++

You must ensure that your application is built with debugging information.

Enabling debugging information

Debugging information must be enabled for debug builds and for release builds.

Debugging information must be in the STABS format or COFF format.

You can enable STABS format debugging information by adding the -gstabs option to your compiler command line.

You can enable COFF format debugging information by adding the -gcoff option to your compiler command line.

Be sure that any other -g directives do not override this option.

For example:

```
  gcc -gstabs main.c  
gcc -gcoff main.c
```

⇒ Enabling debugging information with Microsoft, Borland, QtCreator and other compilers.

2.1.4 QtCreator

When working with applications built using the QtCreator IDE (using the MingW compiler) you must ensure that your application is built with debugging information.
Enabling debugging information

QtCreator by default uses the STABS debugging format which Thread Validator understands.

Debugging information must be enabled for debug builds and for release builds.

Enable debugging information with Microsoft, Borland, MinGW and other compilers.

2.1.5 Other Compilers

You must ensure that your application is built with debugging information.

Enabling debugging information

Thread Validator can understand debugging information in the following formats:

- Microsoft Program Database (PDB)
- Borland Turbo Debugger System (TDS)
- CodeView NB10
- COFF

If you're using a compiler/linker combination that produces debugging information in a different format, please contact us for advice.

Enable debugging information with Microsoft, Borland, MinGW and QtCreator compilers.

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 - tvExample.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 Microsoft, Borland, MinGW, QtCreator 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...
...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 ➤ Go!
Getting Started

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.

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.
The User Interface
The User Interface

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
The User Interface

users) and ends with a short video tutorial.

Let's set up C++ Thread Validator

Before you start using C++ 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 C++ 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.
**The User Interface**

- **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.
The User Interface

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.
The User Interface

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.
The User Interface

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

- **Critical Section** initialize, modify and destroy
- **Event** create, open and destroy
- **Mutex** create, open and destroy
- **Semaphore** create, open and destroy

---

**Copyright © 2003-2017 Software Verify Limited**
### The User Interface

- **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.
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:

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

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 resets 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.

Video overview of application

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

The video has audio
The User Interface

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 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 Per Thread Locks tab might show:

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Lock</th>
<th>Recursion</th>
<th>Contention</th>
<th>C/L Ratio</th>
<th>Wait Time</th>
<th>Sequence</th>
<th>Locked</th>
<th>Waiting</th>
<th>Owning Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThreadId:15880</td>
<td>17,907 2</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>WrUserRequest</td>
<td>StateWait Ctx:36,899</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x5ef61df0</td>
<td>17,904 0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0x5ebed724 : f:\dd\vctools\vc7\ship\atlmfc\src\mfc\afxcrit.cpp Line 94</td>
</tr>
<tr>
<td>ThreadId:15048</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>WrQueue</td>
<td>StateWait Ctx:44 (Not thread stalled)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThreadId:69280</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>WrQueue</td>
<td>StateWait Ctx:116 (Not thread stalled)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThreadId:21280</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>WrQueue</td>
<td>StateWait Ctx:133</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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

```
Address   Lock   Recursion Contention C/L Ratio Wait Time Sequence Locked Waiting Owning Module
ThreadId:15880 17,907 2 0 0.00% - WrUserRequest - StateWait Ctx:36,899
0x5ef61df0 17,904 0 0 0.00% - - - - 0x5ebed724 : f:\dd\vctools\vc7\ship\atlmfc\src\mfc\afxcrit.cpp Line 94
0x0041a388 3 2 0 0.00% - 3 Locked - 0x004097d1 : d:\dev\tvexample\tvexample.cpp Line 138
ThreadId:15048 0 0 0.00% - WrQueue - StateWait Ctx:44 (Not thread stalled)
ThreadId:69280 0 0 0.00% - WrQueue - StateWait Ctx:116 (Not thread stalled)
ThreadId:21280 0 0 0.00% - WrQueue - StateWait Ctx:133
```

3.2.3 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.
The User Interface

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

![Menu with options]

### 3.2.4 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:

![Manager menu options]

### 3.2.5 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**.
The User Interface

3.2.6 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

3.2.7 Data Views menu

Data Views

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

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** shows 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.

Hidden views are remembered between sessions.

### 3.2.8 Software Updates menu

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

3.2.9 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:

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:
The User Interface

Standard toolbar

- Load session
- Save session
- Help

Session toolbar

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

Program information

The target program name, start time and status is displayed when a target program is active, otherwise the area is blank or No active session is shown.

```
tvExample.exe: Sun Sep 27 22:09:53 2015 (Ready)
```

The status can include the following:

- **(Starting)**  the target program is in the process of starting up and being attached to
- **(Running)**   the program is being monitored
- **(Post Processing)** the collected data is being processed after termination
- **(Ready)**    the program has completed, or a previous session has been loaded

The background colour turns green for a few seconds when the program status changes.

```
tvExample.exe (Running)
tvExample.exe: Sun Sep 27 22:12:32 2015 (Post Processing)
```

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.
The User Interface

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

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** Settings
### 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
- 🔒 Critical section allocation
- ⚖ Critical section deallocation
- 🕒 Trace message, or `OutputDebugString()` message
- ⚔ Trace from an exception
- ⚒ 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**
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.

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.
The User Interface

- 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 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 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 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 94 which was displayed separately on the Locks panel)
- the number of contended threads is 4, shown as a fraction of the 17 threads being monitored in the application (displayed on the Threads panel)
- the total number of contentions is 7,676, of which one of the locks is responsible for 6,844 of those contentions
- the maximum contention ratio is 10%.
The User Interface

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 45 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](#)
The User Interface

Error Detection:
- Threading errors
  - Edit... Detect settings

Filter Summary:
- DLLs Hooked
  - Edit... Hooked DLLs 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 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.
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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Address of the critical section, the handle for a <code>WaitForSingleObject</code> call, or the number of handles for a <code>WaitForMultipleObjects</code> call.</td>
</tr>
<tr>
<td>Lock</td>
<td>Number of times the critical section has been locked by any thread.</td>
</tr>
<tr>
<td>Recursion</td>
<td>Number of times the critical section has been re-entered whilst locked by the current thread.</td>
</tr>
<tr>
<td>Contention</td>
<td>Number of attempts a thread has made to acquire the critical section but has had to wait because it is owned by another thread.</td>
</tr>
<tr>
<td>Wait Time</td>
<td>Total time spent waiting by all threads to gain access to this critical section.</td>
</tr>
<tr>
<td>Sequence</td>
<td>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.</td>
</tr>
<tr>
<td>Thread</td>
<td>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.</td>
</tr>
<tr>
<td>State</td>
<td>Current state of the thread, obtained by polling the operating system for data at regular intervals.</td>
</tr>
</tbody>
</table>
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.

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

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.
Updating the display

- **Update Interval** 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 Dialog]

  **Highlighting settings:**
The User Interface

- **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 contended 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.
The User Interface

Thread Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread Id</td>
<td>118564</td>
</tr>
<tr>
<td>Deadlock Count</td>
<td>0</td>
</tr>
<tr>
<td>Potential Deadlock Count</td>
<td>0</td>
</tr>
<tr>
<td>Exit Not Entered Count</td>
<td>0</td>
</tr>
<tr>
<td>Exit Out Of Order Count</td>
<td>0</td>
</tr>
<tr>
<td>Wait Time</td>
<td>18,446,744,073,709,551,615ms</td>
</tr>
</tbody>
</table>

CriticalSection Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>0x0041a388</td>
</tr>
<tr>
<td>Lock Symbol Name</td>
<td>CTeststakApp::critSectRecurseHeldLongTime</td>
</tr>
<tr>
<td>Lock Count</td>
<td>3</td>
</tr>
<tr>
<td>Recursion Count</td>
<td>2</td>
</tr>
<tr>
<td>Contention Count</td>
<td>0</td>
</tr>
<tr>
<td>Wait Time</td>
<td>0ms</td>
</tr>
<tr>
<td>Thread Id</td>
<td>117768</td>
</tr>
<tr>
<td>Sequence Id</td>
<td>29</td>
</tr>
</tbody>
</table>

WaitForSingleObject Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Handle</td>
<td>0x00000248</td>
</tr>
<tr>
<td>Handle Type</td>
<td>Thread</td>
</tr>
<tr>
<td>Wait Timeout</td>
<td>INFINITE</td>
</tr>
<tr>
<td>Wait Time</td>
<td>5,311ms</td>
</tr>
<tr>
<td>Thread Id</td>
<td>113240</td>
</tr>
<tr>
<td>Sequence Id</td>
<td>71,081,472</td>
</tr>
<tr>
<td>Handle Creation Callstack</td>
<td>0x00407111 calls...</td>
</tr>
</tbody>
</table>
The User Interface

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
The User Interface

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.

### 3.7.3 Per Thread Locks

The **Per Thread Locks** 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.

![Image of Per Thread Locks View]

**The Per-Thread Locks View**

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

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

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

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 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Address of the critical section, the handle for a WaitForSingleObject call, or the number of handles for a WaitForMultipleObjects call.</td>
</tr>
<tr>
<td>Lock</td>
<td>Number of times the critical section has been locked by any thread.</td>
</tr>
<tr>
<td>Recursion</td>
<td>Number of times the critical section has been re-entered whilst locked by the current thread.</td>
</tr>
<tr>
<td>Contention</td>
<td>Number of attempts a thread has made to acquire the critical section but has had to wait because it is owned by another thread.</td>
</tr>
<tr>
<td>Wait Time</td>
<td>Total time spent waiting by all threads to gain access to this critical section.</td>
</tr>
<tr>
<td>Sequence</td>
<td>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.</td>
</tr>
<tr>
<td>Locked</td>
<td>Shows &quot;Locked&quot; if the critical section is locked.</td>
</tr>
<tr>
<td>Waiting</td>
<td>Shows &quot;Waiting&quot; if the critical section is waiting.</td>
</tr>
</tbody>
</table>
- **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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Address</strong></td>
<td>Thread Id</td>
</tr>
<tr>
<td><strong>Lock</strong></td>
<td>The total number of locks acquired by this thread</td>
</tr>
<tr>
<td><strong>Recursion</strong></td>
<td>The total number of context switches performed by this thread</td>
</tr>
<tr>
<td><strong>Contention</strong></td>
<td>The total number of critical section contentions for this thread</td>
</tr>
<tr>
<td><strong>C/L Ratio</strong></td>
<td>The percentage of the total contentions over the total number of locks for this thread</td>
</tr>
<tr>
<td><strong>Wait Time</strong></td>
<td>The total number time spent waiting for access to critical sections</td>
</tr>
<tr>
<td><strong>Sequence</strong></td>
<td>Not used</td>
</tr>
<tr>
<td><strong>Locked</strong></td>
<td>Not used</td>
</tr>
<tr>
<td><strong>Waiting</strong></td>
<td>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).</td>
</tr>
<tr>
<td><strong>Owning Module</strong></td>
<td>Information about thread errors. As an example, in the image at the top of this page the thread entry lines (starting with <code>ThreadId:nnn</code> in the Address column), you can see that the data for deadlocked threads reads <code>[DEADLOCKED]</code> and the data for threads with</td>
</tr>
</tbody>
</table>
The User Interface

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.

<table>
<thead>
<tr>
<th>ThreadId</th>
<th>5,270,139</th>
<th>2,635,069</th>
<th>2,051,809</th>
<th>39.60%</th>
<th>1,709,331,615ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00311c24</td>
<td>2,635,069</td>
<td>0</td>
<td>1,254,831</td>
<td>47.62%</td>
<td>0ms</td>
</tr>
<tr>
<td>0x00311c68</td>
<td>2,635,068</td>
<td>0</td>
<td>836,878</td>
<td>31.76%</td>
<td>0ms</td>
</tr>
<tr>
<td>0x00688a85</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
</tr>
<tr>
<td>0x00638c0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
</tr>
</tbody>
</table>

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** 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.
The User Interface

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

  ![Per Thread Locks Settings dialog](image)

  **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

**Per Thread 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.
The User Interface

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 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
The User Interface

- **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.

![Recurse](image)

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.

### 3.7.4 Current Locks

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

Click a part of the image below to jump straight to the help for that area.
The User Interface

The Current 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 Per-Thread Locks 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 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 Current Locks view are identical to the columns on the Per-Thread Locks tab.
Note that like the Per-Thread Locks, the thread header rows display different information to the critical section rows.

**Display settings**

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

![Current Locks Settings dialog]

**Current Locks menu options**

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

3.7.5 **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.
The User Interface

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 dialog]

  - **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
The User Interface

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:

<table>
<thead>
<tr>
<th>1000</th>
<th>Buffer Overflow!</th>
<th>Data Items Lost: 494</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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 Per-Thread Locks 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 Current 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 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 Per-Thread Locks tab (and the Current Locks tab).

Note that like the Per-Thread Locks, 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 Per-Thread Locks tab.

**Locks tab display settings**

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

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

### 3.7.6 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.

#### 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:

<table>
<thead>
<tr>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Totals</td>
</tr>
<tr>
<td>Thread Coverage data collection not enabled.</td>
</tr>
<tr>
<td>See &quot;Thread Coverage&quot; on settings dialog (Intermediate/Expert UI mode).</td>
</tr>
</tbody>
</table>

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 C++ 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.

Coverage options

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

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** 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

3.7.7 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.

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

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:
The User Interface

To edit the source code, double click on any part of the lines of source code displayed or use the 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

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
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:

- **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](#).
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.
The User Interface

Updating the display

- **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.

- **Add Bookmark...** adds a bookmark for the selected item
- **Add Watermark...** adds a watermark for the selected item
- **Edit Source Code...** opens the default or preferred editor to edit the source code
- **Collapse Trace**
- **Expand Trace**
- **Collapse All**
- **Expand All**

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 User Interface

- **Collapse** or **Expand All** hides or shows all the available callstack information for every listed item.

### 3.7.8 Analysis

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.

#### 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:
The User Interface

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

Predefined searches display:

- locked or waiting 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
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.

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).
The User Interface

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:
The User Interface

- **Locked**  finds all locked critical sections

```
Locked critical sections...
- Critical Section 0x0041a388, thread 27636
- Critical Section 0x0ef31878, thread 27636 [ERROR: Deleted whilst in use (still locked)]
- Critical Section 0x0ef3167c, thread 37596
```

- **Waiting**  all waiting critical sections

```
Waiting critical sections and WaitForXXXObject(s)(Ex)
- Critical Section 0x0ef3167c, thread 38632
- Critical Section 0x0ef3167c, thread 36304
- Critical Section 0x0ef315c4, thread 31988 [ERROR: Deadlock count (264491)]
```

- **Deadlocked**  finds deadlock locations

```
Deadlocks...
- Critical Section 0x0ef315a0, thread 31988 [ERROR: Deadlock count (263357)]
- Critical Section 0x0ef315c4, thread 31988 [ERROR: Deadlock count (265245)]
- Critical Section 0x0ef315c4, thread 38676 [ERROR: Deadlock count (265245)]
```

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

```
Potential deadlocks...
- Critical Section 0x02fd196c, thread 36648 [ERROR: Potential deadlock count (4)]
- Critical Section 0x02fd196c, thread 38968 [ERROR: Potential deadlock count (4)]
- Critical Section 0x02fd199c, thread 36648 [ERROR: Potential deadlock count (4)]
```

You'll need to enable the detection of potential deadlocks otherwise you'll just see this warning.
The User Interface

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

  - Critical Section 0x02fd1860, thread 40620 [ERROR: Leaving unentered critical section]
  - Critical Section 0x02fd1860, thread 40620 [ERROR: Leaving unentered critical section]
  - Critical Section 0x02fd1860, thread 40620 [ERROR: Leaving unentered critical section]

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

  - Critical Section (Leave) 0x02fd1860 Thread ID 40620 () [debug
text
test
cpp Line 1597] > Leave critical section that was not locked by this thread
  - Critical Section (Leave) 0x02fd1860 Thread ID 40620 () [debug
text
test
cpp Line 1597] > Leave critical section that was not locked by this thread
  - Critical Section (Leave) 0x02fd1860 Thread ID 40620 () [debug
text
test
cpp Line 1597] > Leave critical section that was not locked by this thread
  - Critical Section (Delete) 0x02fd1878 Thread ID 40620 () [debug
text
test
cpp Line 1617] > Deleting critical section that is still in use

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

  - f:\debug\text\test\src\dbgheap.c(1491) : Assertion failed: pUserData != NULL
  - Process 0x00000524 (tvExample.exe), created: 0:0:0, exit: 0:0:0, user: 0:0:0, kernel: 0:0:0

**Analysis tab options - lower window**

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

- **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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same address</td>
<td>Finds any other critical sections using the same memory address</td>
</tr>
<tr>
<td>Same handle</td>
<td>Finds any other WaitForSingleObject using the same resource handle</td>
</tr>
<tr>
<td>Same object/handle type</td>
<td>Critical sections and wait calls that have the same object type or handle type.</td>
</tr>
<tr>
<td>Same location, same callstack</td>
<td>Finds other critical sections that have the same usage location and:</td>
</tr>
<tr>
<td>same callstack</td>
<td>using the same callstack</td>
</tr>
<tr>
<td>different callstack</td>
<td>using different callstacks</td>
</tr>
<tr>
<td>all callstacks</td>
<td>regardless of callstack</td>
</tr>
<tr>
<td>Same source file</td>
<td>All critical sections in the same source file...</td>
</tr>
<tr>
<td>Same DLL</td>
<td>or the same DLL</td>
</tr>
<tr>
<td>Same class</td>
<td>All critical sections from the same C++ class</td>
</tr>
<tr>
<td>Entries prior</td>
<td>For critical section events, finds the previous 5, 10 or 20 events</td>
</tr>
<tr>
<td>Entries after</td>
<td>or the next 5, 10 or 20 events</td>
</tr>
</tbody>
</table>

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.
The User Interface

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

 Dagger Menu option: editing source code

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

Dagger 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

3.7.9 Objects

The Objects view displays a list of all the waitable handles the target program is using (except those used by Thread Validator).

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)** 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.

### 3.7.10 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**
The User Interface

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
- Function hook success or failure
- Delay loaded function hooked
- Possible hook found
- Function already hooked
- Hook at address

Other information

- DLL load address
- DbgHelp searching
- 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 > Symbols and Warnings 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:
### The User Interface

- **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
### The User Interface

<table>
<thead>
<tr>
<th>Diagnostic</th>
<th>Stdout</th>
<th>Copy</th>
<th>Clear</th>
<th>Display Most Recent</th>
<th>62 Lines</th>
</tr>
</thead>
</table>

#### Stdout and Stderr

```
ytilbtr oetf renhilou gnoic s'tinUppC
----------------------------------------

orcam esu daetsni, ecapseman tinUppC eralced ylticilpxe t'nod -
.DNE_SN_TUNUPPC dna NIGEB_SN_TUNUPPC

'ecapseman tinUppC ni ssalc ot refer ot 'tinUppC' esu ylticilpxe t'nod -
ro 'tinUppC' rehtie ot sdnapxe hcihw SN_TUNUPPC orcam esu daetsni
.noitarugfnoc eht no gnidneped gnithon

'LTS roF.noitacifluaq lluf esu syawla, 'evtcerid gnisu' eht esu t'nod -
::dts esu syawla

orcam tsac s'tinUppC esu daetsni ,yltcerid tsac elyts ++C esu t'nod -
.)TSAC_TSONC_TUNUPPC(

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.

### 3.7.11 Floating Licence

The **Floating Licence** view displays information about the computers using the floating licence.
The User Interface

This view is only displayed if a floating licence has been purchased. Evaluation users will not see this view.

The screenshot above show two computers using the same 2 user floating licence, that has maintenance id 9348. 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.

**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.

- **Licence ID**
  The licence id for the software.

- **ID**
  The maintenance id for the software.

- **Software Tool**
  The software tool and version of the software that is running on that computer.

- **Operating System**
  The operating system running on that computer.

**Unlicenced users**

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 a salmon pink 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:
The user interface mode affects the following tasks:

- **Attaching to an application** (Injection)
- **Launching an application**
- **Wait for application to start**

### 3.9 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**
3.9.1 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:

Keyboard shortcuts:

F6 Settings

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.
The User Interface

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.
Restoring the default settings

The settings dialog has a **Reset** button near the bottom left of the dialog which you can use to reset all global settings back to their default values.

This **Reset** button resets nearly **all global settings** in Thread Validator, not just the settings visible on the current tab of the dialog.

3.9.1.1 Data Collection

3.9.1.1.1 Collect

The **Collect** tab allows you to specify the types of data that will be collected for display by Thread

...
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 locks summary** ✔ collect data for the **Summary** and **Locks** Summary tab
- **Collect data for overall per-thread locks summary** ✔ collect data for the **Per Thread Locks** tab
- **Collect data for current locks-per-thread summary** ✔ collect data for the **Current 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** - Resets all global settings, not just those on the current page.

### 3.9.1.1.2 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**

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.
The User Interface

- 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 options are only checked if you have a need for the information about all callstacks.

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

3.9.1.1.3 Detect

The Detect tab allows you to specify how the automatic detection of errors is handled.

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

### 3.9.1.1.4 Callstack

The **Callstack** tab allows you to specify how the callstack is collected, and how information about the callstack is displayed.
The User Interface

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
The User Interface

- **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 build).

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. Less frames than the value specified are collected, the alternative stack walk will be used.

- **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:

- **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 guarantee the alternative stack walk will be correct.

### Stack Walking Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always use DbgHelp StackWalk()</td>
<td></td>
</tr>
<tr>
<td>Always use alternative StackWalk()</td>
<td></td>
</tr>
<tr>
<td>Use DbgHelp StackWalk, but when not enough stack frames are collected use the alternative stack walk.</td>
<td></td>
</tr>
<tr>
<td>Specify the number of stack frames which will cause the alternative stack walk to be used.</td>
<td>5</td>
</tr>
<tr>
<td>When using alternative stack walk, use fast option.</td>
<td></td>
</tr>
<tr>
<td>When using alternative stack walk with fast option, do extra consistency checks.</td>
<td></td>
</tr>
<tr>
<td>When using alternative stack walk, relative CALL instructions must be within byte range.</td>
<td>8192</td>
</tr>
</tbody>
</table>

### Stack Walking Methods

- **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.
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.

### 3.9.1.1.5 Startup Data

The **Startup Data** tab controls how to manage Critical Sections in DLLs loaded prior to Thread Validator's stub being loaded.

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

**3.9.1.1.6 Hook Insertion**

The **Hook Insertion** tab allows you to control which functions are monitored by Thread Validator.
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.
The User Interface

- **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

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

### 3.9.1.1.7 Hook Trace

The **Hook Trace** tab allows you to control which debug or trace functions are monitored by Thread Validator.
The User Interface

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 remains unchanged. If the time between context switches exceeds the time threshold the thread is marked as stalled.

- **Detect stalled threads**
- **Time threshold (ms)**: 10000

OK | Cancel
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 \texttt{Sleep()} or \texttt{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\,\footnote{\textbf{Detect stalled threads} \hspace{1em} enable thread stall detection} as defined on the \textit{Lock Colours} settings dialog.

- \textbf{Detect stalled threads} \hspace{1em} enable thread stall detection
- \textbf{Time threshold (mS)} \hspace{1em} enter the threshold time (in milliseconds)

  The default is 10 seconds (10000ms)

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

### 3.9.1.1.8 Historical Data

The \textit{Historical Data} tab allows you to control how many of the most recent stack traces are kept by Thread Validator.
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** - Resets all global settings, not just those on the current page.

### 3.9.1.1.9 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](#).
The User Interface

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

### 3.9.1.1.10 Thread Coverage

The **Thread Coverage** tab allows you to control collection of coverage statistics for synchronization functions.

![Thread Coverage settings](image-url)

**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
The User Interface

thread synchronization code.

<table>
<thead>
<tr>
<th>File Location</th>
<th>% Visited</th>
<th>Num Lines</th>
<th>Num Visited</th>
<th>Visit Count</th>
<th>DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.39%</td>
<td>406</td>
<td>30</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>57.14%</td>
<td>7</td>
<td>4</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>50.00%</td>
<td>14</td>
<td>7</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>50.00%</td>
<td>4</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>50.00%</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>42.02%</td>
<td>7</td>
<td>3</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25.00%</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16.07%</td>
<td>10</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.00%</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.00%</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.00%</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.00%</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.00%</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

See the section at the end of this page about tracking waitable handles.

When the collection of thread coverage data is enabled, there 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.
The User Interface

- **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 Dialog](image)

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
The User Interface

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

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
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 saves the auto-merge session in the specified directory

  For example, if you run the application tvExample.exe, and specify a central session directory of e:\threadCoverageResults, the central session will be saved in a file named e:\threadCoverageResults\tvExample.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)
The User Interface

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  No
```

- **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:
The User Interface

OK displays the Hook Insertion settings to choose which waitable handles you want to track.

Pick which waitable handle functions you're interested in for coverage and click OK.

Alternatively if you don't want to get coverage after all, Cancel and then uncheck Collect thread coverage data to prevent these warnings.

Reset - Resets all global settings, not just those on the current page. Any thread coverage filters added as above are also removed.
3.9.1.11 Applications to Monitor

If your target program launches other child applications then the Applications to Monitor page lets you choose which ones to monitor.

Monitoring child applications

You may have a case where the program you need to start is not the one you're actually 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 ones 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.

- **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 vtest.console.exe.

### The Application to Monitor dialog

The **Application to Monitor** dialog lets you define or edit a launch program and its 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 dialog](image)

- **Application to Launch** ➤ edit or **Browse**... 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.

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

### 3.9.1.2 Display

#### 3.9.1.2.1 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:
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 to pick one of 16 predefined colours below

<table>
<thead>
<tr>
<th>Colour</th>
<th>Hex Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>aqua</td>
<td>#00FFFF</td>
</tr>
<tr>
<td>black</td>
<td>#000000</td>
</tr>
<tr>
<td>blue</td>
<td>#0000FF</td>
</tr>
<tr>
<td>fuchsia</td>
<td>#C000C0</td>
</tr>
<tr>
<td>gray</td>
<td>#808080</td>
</tr>
<tr>
<td>green</td>
<td>#008000</td>
</tr>
<tr>
<td>lime</td>
<td>#00FF00</td>
</tr>
<tr>
<td>maroon</td>
<td>#800000</td>
</tr>
<tr>
<td>navy</td>
<td>#000080</td>
</tr>
<tr>
<td>olive</td>
<td>#80C000</td>
</tr>
<tr>
<td>purple</td>
<td>#800080</td>
</tr>
<tr>
<td>red</td>
<td>#FF0000</td>
</tr>
<tr>
<td>silver</td>
<td>#C0C0C0</td>
</tr>
<tr>
<td>teal</td>
<td>#008080</td>
</tr>
<tr>
<td>white</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td>yellow</td>
<td>#FFFF00</td>
</tr>
</tbody>
</table>

- Click the button to edit the colour using the standard colour dialog:
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** - Resets all global settings, not just those on the current page.

### 3.9.1.2.2 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:
The User Interface

Changing lock colours

For each colour you can choose a predefined colour or make your own:

- Use the drop-down list to pick one of 16 predefined colours below:

  - aqua
  - black
  - blue
  - fuchsia
  - gray
  - green
  - lime
  - maroon
  - navy
  - olive
  - purple
  - red
  - silver
  - teal
  - white
  - yellow

- Click the button to edit the colour using the standard colour dialog:
**Reset** - Resets all global settings, not just those on the current page.

### 3.9.1.2.3 Data Display

The **Data Display** page allows you to specify how numeric data is formatted in the various displays.
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 - Resets all global settings, not just those on the current page.

3.9.1.2.4 Active Objects Display

The Active Objects Display tab allows you to choose which data items are displayed on the Active Objects tab of the main display.
The User Interface

Showing data on the Active Objects tab

Tick any of the three boxes to show the corresponding data on the Active Objects tab:

- Synchronization objects
- Handles
- Trace messages

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

3.9.1.3 Filters

3.9.1.3.1 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.
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
The User Interface

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

<table>
<thead>
<tr>
<th>Process Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Gradients.exe</td>
</tr>
<tr>
<td>✓ KERNEL32.dll</td>
</tr>
<tr>
<td>✓ MSVCR90D.dll</td>
</tr>
<tr>
<td>✓ QtCore4.dll</td>
</tr>
<tr>
<td>✓ QtGui4.dll</td>
</tr>
<tr>
<td>✓ QtSvg4.dll</td>
</tr>
</tbody>
</table>

**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 included or excluded 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
The User Interface

- **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...
The User Interface

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 dialog](image)

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

### 3.9.1.4 General

### 3.9.1.4.1 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

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:

```cpp
if (index < 0)
    return;

// get current expand/collapsed status
BOOL fExpanded = tree->GetItemExpand(index);
```

- **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
The User Interface

- **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:

```plaintext
Thread (Create Handle 0x000002b4, Thread ID 32052) [d:\dev\tveexample\testsvv.cpp Line 580]
Allocation location
ThreadID: 00034588 Timestamp: 352244625

0x00404f8a tExample.exe CTeststakView::OnHandlesStartthreadA [d:\dev\tveexample\testsvv.cpp Line 580]

575:
576: void CTeststakView::OnHandlesStartthreadA()
577:
578:    hThreadA = CreateThread(NULL, 0,
579:    threadProcA, this,
580:    0, &threadIdA);
581:    CTeststakApp::nameThread(threadIdA, "STA");
582:
583:
584:    void CTeststakView::OnUpdateHandlesStartthreadA(CCmdUI* pCmdUI)
585:
```

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

### 3.9.1.4.2 Editing

The **Editing** tab allows you to configure which editor Thread Validator will use for editing source code.
The default settings are shown below:

![Settings dialog]

**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:
### The User Interface

<table>
<thead>
<tr>
<th>C++ Thread Validator editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>User defined editor</td>
</tr>
<tr>
<td>Visual Studio 2015 [Running Instance]</td>
</tr>
<tr>
<td>Visual Studio 2015 [New Instance]</td>
</tr>
<tr>
<td>Visual Studio 2013 [Running Instance]</td>
</tr>
<tr>
<td>Visual Studio 2013 [New Instance]</td>
</tr>
<tr>
<td>Visual Studio 2012 [Running Instance]</td>
</tr>
<tr>
<td>Visual Studio 2012 [New Instance]</td>
</tr>
<tr>
<td>Visual Studio 2010 [Running Instance]</td>
</tr>
<tr>
<td>Visual Studio 2010 [New Instance]</td>
</tr>
<tr>
<td>Visual Studio 2008 [Running Instance]</td>
</tr>
<tr>
<td>Visual Studio 2008 [New Instance]</td>
</tr>
<tr>
<td>Visual Studio 2005 [Running Instance]</td>
</tr>
<tr>
<td>Visual Studio 2005 [New Instance]</td>
</tr>
<tr>
<td>Visual Studio .Net 2002 [Running Instance]</td>
</tr>
<tr>
<td>Visual Studio 5.0 [Running Instance]</td>
</tr>
<tr>
<td>Visual Studio 5.0 [New Instance]</td>
</tr>
<tr>
<td>SCiTE editor [Running Instance]</td>
</tr>
<tr>
<td>SCiTE editor [New Instance]</td>
</tr>
</tbody>
</table>

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.

 SCIȚE 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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td></td>
</tr>
<tr>
<td>(Space) Filename</td>
<td></td>
</tr>
<tr>
<td>Filename</td>
<td></td>
</tr>
<tr>
<td>(Space) Line Number</td>
<td></td>
</tr>
<tr>
<td>Line Number</td>
<td></td>
</tr>
<tr>
<td>Space</td>
<td></td>
</tr>
<tr>
<td>(Space) Other</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
</tbody>
</table>

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

#### 3.9.1.4.3 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:
The User Interface

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.
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
- **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

There are two options for scanning your computer looking for directories containing files of the current path type:
The User Interface

- **Partial scan...**  
  displays a directory browser  
  navigate to a location you want to scan within  
  OK  
  starts a scan for directories

- **Full scan...**  
  starts a scan of all drives for directories containing files

Both 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:

![File Scan dialog](image)

Once the scan is complete you'll see the **File Paths** dialog showing you the scan results:

![File Paths dialog](image)

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:**

```markdown
[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** - Resets *almost 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.

### 3.9.1.4.4 File Cache / Subst Drives

The **File Cache / Subst Drives** tab allows you to specify the following:

- set where cached information is stored and when it gets cleared
- mappings between drive names and source code locations

The default settings are shown below:
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.

**Substitute drives**

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.

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 **Subst Drives** table > enter the name of a disk drive in the **Drive** column > click in the **Path** column > enter the path that is to be used to substitute for the drive name

  For example, you might enter X:\ for the drive and e:\version2_1\release for the path.

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

- **Remove** removes selected substitutions from the list

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

**Reset** - Resets most global settings including those on other pages, but not the settings on this page.

### 3.9.1.4.5 Source Lookup

The **Source Lookup** tab allows you control how source code is parsed and what the default behaviour is when symbol debugging information is not found.

The default options are shown below:
Source lookup

When debugging information is missing or unavailable, you can choose one or more of the following methods to try and resolve the symbols:

- **Use map files to find file and line numbers** use linker .map files. Faster than using .Bsc files.
- **Map Ordinals to function names** use linker definition .def files. Fast, though not usually needed in addition to the other two options.
- **Use Bsc files to find file and line numbers** use compiler .Bsc source code browser files.

See also: topics on File Locations and Ordinal Handling.

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

### 3.9.1.4.6 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 Communication Settings](image)

**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:

\[ 1 + \text{numThreads} \times (1 + \text{numWaitForXXXObjects} + (\text{numCriticalSections} \times 3)) \]

Don't set values overly large as this may also adversely affects memory usage. The default is $10^5$.

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

### 3.9.1.4.7 Symbols and Warnings

The **Symbols and Warnings** tab allows you to set an assortment of symbol and diagnostic warning data.
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.

Missing PDB and MAP file warnings

If Thread Validator can't find debug information while instrumenting a module then a warning can be displayed.

The debug information for each module should be found in a .pdb file or a .map file which must include line information.

This warning doesn't apply to system modules as they frequently don't have the debug information available.

- **Display Debug Information Warning Dialog** shows the DLL Debug Information dialog if no debug information can be found.

  The dialog lists which modules were loaded and their debug information status.

  In the example below, the sixth line shows no PDB or MAP file was found for the `tvExample` application.

Convert Ordinals into symbols

- **Convert DLL exported function ordinals to symbols** enable the ordinal to function name
The User Interface

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

**ExitProcess breakpoint**

Some bugs can manifest themselves by the process suddenly exiting for an unknown reason.

There are two common causes of this:

- The program encountered a serious error, did not warn the user and called `exit()` or `ExitProcess()`

  If the program simply calls `exit()`, the code will still be routed via `ExitProcess()`

- The program exhausted its stack space and could not continue

  When this happens, the program may or may not have enough stack space to warn the user that the stack space has been exhausted.

  When the program runs out of stack space, Thread Validator cannot help.

When the target program is exiting in 'unusual circumstances', Thread Validator can be used to cause a breakpoint instruction to be executed when the target program reaches `ExitProcess()`.

- **Breakpoint at ExitProcess()** cause a breakpoint instruction to be executed on `ExitProcess()`

  When the target program reaches `ExitProcess()`, a dialog box will be displayed.

  ![C++ Thread Validator [STUB] ExitProc... ×](#)

  - **Debug** enter the debugger and determine what caused the process to exit
  - **Continue** allow the program to exit as normal

**Diagnostics**

A lot of diagnostic information is collected and displayed on the [diagnostic tab](#) when attaching to a...
The User Interface

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)

![Warning Dialog](example.png)

- **Help...** will open the manual for [User Permissions Warnings](#)

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

### 3.9.1.4.8 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.
Symbol lookup for Microsoft / Intel compilers

Symbols for your application are read using an appropriate symbol handler for the type of debugging information you have.

A selection of different DbgHelp.dll versions are provided by Thread Validator:

- **We can provide a DbgHelp.dll...** automatically provides the appropriate (most recent) version of Visual Studio (the default)

  You can manually override which of the provided handlers is used via the list of Visual Studio versions.

Alternatively, you may wish to specify a DbgHelp.dll from disk, rather than use the provided ones.

- **...locate a version of DbgHelp.dll that best matches your build** enter a path or **Browse** to a location to specify your own handler

**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 the MinGW compiler or the Delphi or C++ Builder, choose the appropriate option.

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

After selecting the compiler, clicking the links will show the information below

For MinGW:

**MinGW Compiler Debug Information**

For MinGW compilers, both STABS and COFF debug formats are supported.

To generate STABS debug information use -gstabs when building your software.
To generate COFF debug information use -gCoff when building your software.
We recommend using -gstabs

Close

For Delphi or C++ Builder:

**Borland Compiler Debug Information**

For Delphi and C++ Builder you need to enable TDS format debugging information.
If this does not work, please enable MAP file generation, enabling any options that generate more detailed MAP files.

Close

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

**3.9.1.4.9 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 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.

Managing symbol servers

- Add... displays the symbol server dialog described below
The User Interface

- **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.

- **Symbol Server DLL** enter or browse to specify the name of the DLL (with or without path) that will be handling the symbol server negotiation

  Typically this is *symsrv.dll*, but if you have your own custom DLL implementation you should enter that **and** ensure that a copy of the DLL is present in the Thread Validator installation directory.

- **Symbol Store Directory** has been specified and exists.
- **Symbol Server** has been specified (this value will not be checked for correctness).

The following three entries must be set to enable the **OK** button and define the symbol server.

- **Enable Symbol Server** enable or disable this server

  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).
The User Interface

- **Set Default** uses the current entry as the default next time the dialog is opened

- **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

- **Symbol Server** enter the URL of the symbol server you wish to use - the Microsoft server is initially set as the default

  - **Set Default** uses the current entry as the default next time the dialog is opened

You can optionally associate a directory to scan when you are getting symbols (below)

- **Exe/DLL 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](image)

**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 dialog]

Prerequisites for pre-fetching symbols

The pre-fetching of symbols requires the installation of [Microsoft's Debugging Tools](https://developer.microsoft.com/en-us/windows/debug). 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/symbols
```

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

### 3.9.1.5 Third Party DLLs

3.9.1.5.1 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.
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)
The User Interface

- **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.
The User Interface

Global hook is trying to load into Thread Validator

Thread Validator has detected that the global hook:

cintcap.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)

Do you want to allow this DLL to load?

- **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** - Resets most global settings including those on other pages.

### 3.9.2 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

### 3.9.3 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:
The User Interface

Settings menu  ▶ User Permissions Warnings...  ▶ shows the User Permissions Warnings dialog below

The Help button displays the User Permissions help topic.

See also, the question about creating Power User accounts on Windows XP.

3.9.4 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.
The display lists each path with its 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\releaseldll`
3.10 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:

```plaintext
??0CBrush@@QEAA@K@Z @ 332 NONAME
??0CBrush@@QEAA@PEAVCBitmap@@Z @ 333 NONAME
??0CByteArray@@QEAA@XZ @ 334 NONAME
??0CChevronOwnerDrawMenu@@QEAA@Z @ 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
The User Interface

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:
The User Interface

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.
The User Interface

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 9.0\Common7\Tools\Def
Number of 44
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\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\atlmfc\src\mfc\AMD64
```

- **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
The User Interface

- **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

### 3.11 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**

#### 3.11.1 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.

The **Managers menu** \(\Rightarrow\) **Session Manager...** \(\Rightarrow\) shows the **Session Chooser** dialog below, highlighting the current session.

![Session Chooser dialog](image)
The User Interface

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

  ![](image)

  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](image)

  The alias is added before the session details:

  ![Session](image)

  · **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
3.11.2 Thread Filter

Thread filtering affects data displayed on Locks, Per Thread Locks, Current 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.
The User Interface

- **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.

**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?

### 3.11.3 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 **Session Toolbar**:

![Image of Session Toolbar with a highlighted option]

- 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 **Session 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.

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.

### 3.11.4 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 **Session Toolbar**:

![Session Toolbar](image)

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
The User Interface

Or use the Session Toolbar option:

Unlike watermarks, bookmarks are shown in the order you add them.

![Bookmarks dialog](image)

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.
3.12 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** use the Find Synchronization 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

3.12.1 Finding synchronization objects

Searching for synchronization objects

Using the Find Synchronization Object dialog below, you can search for objects in the Active Objects Tab.
The User Interface

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.

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**
The User Interface

Search for type `CriticalSection`, in the range 0x67c20000 to 0x67c22000, inclusively. All other search criteria are disabled.

The results are shown below - all `CriticalSection` objects are selected.

The search is repeated using same search criteria as above, but with the exclusive option selected.
The results are shown below and only \texttt{CriticalSection} objects in the specified address range are now selected.

3.12.2 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:

- \texttt{Query} menu \to choose \texttt{Find Function...} \to displays the Find Function dialog

Or use the following icon on the \texttt{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

ียว 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

ียว 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
The User Interface

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

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
3.12.3 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**.
The User Interface

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:

```
Dead Thread 1: Start Deadlock Thread 1
Tid: 2 Thread deadlock
Tid: 3 Thread deadlock
```

After starting deadlock Thread 1, that menu option is disabled and clicking Find on the Single Thread dialog shows two results:

![Single Thread Critical Section Detector](image)

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.

3.12.4 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**.

![Deadlocked Thread PostMortem dialog](image)

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.

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.

### '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](#).
3.12.5 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 Panel](image)

   In the snapshots view of the **Threads** tab

   ![Threads Panel](image)

   Or in the Locks tab:

   ![Locks Panel](image)

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 panel](image)

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)

![Threads snapshots](image)

Or in the Locks tab, where the locked threads are also now highlighted in red:

![Locks tab](image)

### 3.12.6 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:
In the snapshots view of the **Threads** tab

<table>
<thead>
<tr>
<th>Address</th>
<th>Lock</th>
<th>Recursion</th>
<th>Contention</th>
<th>C/L Ratio</th>
<th>Wait Time</th>
<th>Sequence</th>
<th>Thread</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:0544eef0</td>
<td>11,660</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>14,352</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0:0372ebc4</td>
<td>332</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>14,350</td>
<td>3472 (tpot2A)</td>
<td>StateWait</td>
</tr>
<tr>
<td>0:0372ebec</td>
<td>332</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>14,348</td>
<td>3472 (tpot2A)</td>
<td>StateWait</td>
</tr>
<tr>
<td>0:0372e9d4</td>
<td>332</td>
<td>0</td>
<td>196</td>
<td>59.04%</td>
<td>677ms</td>
<td>14,342</td>
<td>3472 (1) (tpot2A)</td>
<td>StateWait</td>
</tr>
<tr>
<td>0:0372ebc</td>
<td>331</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>14,330</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0:0041e3b8</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>3</td>
<td>14788 (UIThread)</td>
<td>StateWait (WithUserRequest)</td>
</tr>
<tr>
<td>0:0372eb8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>17</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0:0372e720</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>18</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

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:
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 Locks tab, where the locked threads are also now highlighted in pink:

<table>
<thead>
<tr>
<th>Address</th>
<th>Lock</th>
<th>Recursion</th>
<th>Contention</th>
<th>C/L Ratio</th>
<th>Wait Time</th>
<th>Sequence</th>
<th>Thread</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x034019F0</td>
<td>14076</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>18,070</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x037520c4</td>
<td>495</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>18,048</td>
<td>1930 (tpot28)</td>
<td>StateWait(DelayExecution)</td>
</tr>
<tr>
<td>0x03752194</td>
<td>495</td>
<td>0</td>
<td>295</td>
<td>59.40%</td>
<td>624ms</td>
<td>18,039</td>
<td>1930 (tpot28)</td>
<td>StateWait(DelayExecution)</td>
</tr>
<tr>
<td>0x037520b0</td>
<td>494</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>18,053</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x037520c0</td>
<td>494</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>18,047</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x037a3798</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>3</td>
<td>14788 (UThread)</td>
<td>StateWait(WaitRequest)</td>
</tr>
<tr>
<td>0x037a2768</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>17</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x037a2750</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
<td>18</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### 3.12.7 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:

- **Locks tab** → 📋 popup menu → Lock Acquisition Order...
The User Interface

- **Query menu ▸ Display lock 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

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
The User Interface

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.

<table>
<thead>
<tr>
<th>Address</th>
<th>Sequence</th>
<th>Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x002e290c</td>
<td>10</td>
<td>Waiting</td>
<td>view-&gt;sect3_b</td>
</tr>
<tr>
<td>0x002e290c</td>
<td>10</td>
<td>Locked</td>
<td>view-&gt;sect3_b</td>
</tr>
<tr>
<td>0x002e28b5</td>
<td>10</td>
<td>Locked</td>
<td>view-&gt;sect3_a</td>
</tr>
<tr>
<td>0x002e28b5</td>
<td>10</td>
<td>Waiting</td>
<td>view-&gt;sect3_a</td>
</tr>
<tr>
<td>0x002e2930</td>
<td>10</td>
<td>Locked</td>
<td>view-&gt;sect3_c</td>
</tr>
<tr>
<td>0x002e2930</td>
<td>10</td>
<td>Waiting</td>
<td>view-&gt;sect3_c</td>
</tr>
</tbody>
</table>

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**.
The User Interface

**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**

**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).

**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:
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.
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

3.13 Tools

Click on an item in the picture of the Tools Menu below to jump to the relevant topic:
The User Interface

3.13.1 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:
The User Interface

**File menu**

The file options need no explanation:

- Load...
- Save...
- Save As...
- Exit

**Edit menu**

All the following edit options should also be familiar:
The User Interface

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
The User Interface

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 Dialog

- Display Left Symbols
- Display Right Symbols
- Display Tooltips
- Allow vertical scrollbar
- Allow horizontal scrollbar
- Allow Drag Open
- Allow Drag Close
- Allow Line Highlighting

Character handling
- Tab length: [ ]
- Add tabs as spaces
- Replace tabs with spaces when deleting
- Translate 'v to '\n
Undo/Redo
- Allow Line Collapsing
- Line Collapse Colour: silver
- Allow Quoted Text Colouring
- Quoted Text Colour: teal
- Allow Punctuation Colouring
- Punctuation Colour: olive
- Maximum number of undo operations: [ ]
- Store movement undos: [ ]
- Store editing undos: [ ]

Options...
- set tab length and other options

Wrap Width...
- changes the column width at which lines will wrap in the display
**The User Interface**

**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:

```c
void CTeststakView::OnTestGoodlockstrategyexample()
{
    goodDelay = 250;
    handle_GA = CreateThread(NULL, 0,
        threadProc_goodA, this,
        0, &threadId3_goodA);
    CTeststakApp::nameThread(threadId3_goodA, "tpgoodA");
    handle_GB = CreateThread(NULL, 0,
        threadProc_goodB, this,
        0, &threadId3_goodB);
    CTeststakApp::nameThread(threadId3_goodB, "tpgoodB");
    handle_GC = CreateThread(NULL, 0,
        threadProc_goodC, this,
        0, &threadId3_goodC);
    CTeststakApp::nameThread(threadId3_goodC, "tpgoodC");
}
```

Collapsed:

```c
 void CTeststakView::OnTestGoodlockstrategyexample()
{
}
```

(Line collapsing is *temporary* and not remembered between edit sessions.)

**3.13.2 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.
The User Interface

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.

### 3.13.3 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 **Path** the module was loaded from
3.13.4 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...](image)

The dialog shows:

- the path from which Modules (DLL or EXE) were loaded
- the debug Status (below)

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
The User Interface

- 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.
The User Interface

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
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.
Provides the relevant debug compiler and linker flags. An example for Visual Studio 2017 to 2015 is below:
The User Interface

- 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.
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 symbols server options...** shows the **Symbol Servers settings** page

### 3.13.5 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** shows the **Instrumentation Logging Data...** dialog.
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

### 3.14 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.
The User Interface

**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:

![No new updates are available for C++ Thread Validator](image)

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](image)

- **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.
The User Interface

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:
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.

Managing software maintenance

Software Updates menu › Renew software updates › shows the Software update maintenance renewal dialog
The User Interface

- **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

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:

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

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.

Software update directory

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

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

Software Updates menu ➤ Set software update directory ➤ shows the Software update download directory dialog

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

Example reasons for invalid directories include:

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

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

- Reset ➤ reverts the download location to the user's TMP directory

The default location is c:\users\{username}\AppData\Local\Temp

3.15 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:
The User Interface

- 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.

3.15.1 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
The User Interface

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.

3.15.2 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.
The User Interface

Export Session

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** \( \text{TRACE()} \) and \( \text{OutputDebugString()} \) 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 User Interface

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

  Defaults to the menu option selected, but included here to more easily export one format and then the other.

- **Format** set whether exporting HTML or XML

  Check the **overwrite existing file** option if you want to be warned about overwrites.

3.15.2.1 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
<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>
</XML>
```

Thread and Lock information

Thread and lock information is listing the following tag pairs:

```xml
<LOCK_DATA>...</LOCK_DATA>
```

Example lock data might be as follows:
Lock data may contain the following:

- **OBJECT**: Handle of waited object
- **TIMEOUT**: Timeout in milliseconds
- **WAITING**: Number of waiting processor cycles
- **THREAD**: Thread id
- **STATUS**: Thread status
- **LOCATION**: Source code line where object was allocated, locked or waiting
- **CALLSTACK**: List of callstack entry tag pairs
- **NUM_HANDLES**: Number of handles waited for a multiple wait
- **OBJECTS**: List of object handle tag pairs
- **OBJ**: Object handle
- **CRITICAL_SECTION**: Critical section address
- **ENTERED**: Entered count
- **RECURSE**: Recursion count
- **CONTENDED**: Contended count
- **CONTENDRATIO**: Contention ratio (contention vs entered)
- **THREAD_WAITING**: Number of waiting threads

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.
Allocation events

The next level of tags are shown in optional `<EVENT>` tags.

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\tvexample\testsvw.cpp</File>
  <Line>226</Line>
  <Synchronization>CriticalSection</Synchronization>
  <StackTrace>
    <Symbol>0x00402d1d tvExample.exe  CTeststakView::CTeststakView : [c:\program files (x86)\software verification\c++ thread validator\tvexample\testsvw.cpp Line 226]
    <Symbol>0x004025a7 tvExample.exe  CTeststakView::CreateObject : [c:\program files (x86)\software verification\c++ thread validator\tvexample\testsvw.cpp Line 53]
    <Symbol>0x6242b6a7 mfc90ud.dll  CRuntimeClass::CreateObject : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\objcore.cpp Line 142]
    <Symbol>0x620d3ece mfc90ud.dll  CFrameWnd::CreateView : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 627]
    <Symbol>0x620d4003 mfc90ud.dll  CFrameWnd::OnCreateClient : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 659]
    <Symbol>0x620d40c4 mfc90ud.dll  CFrameWnd::OnCreateHelper : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 678]
    <Symbol>0x620d4073 mfc90ud.dll  CFrameWnd::OnCreate : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\winfrm.cpp Line 669]
    <Symbol>0x620c9337 mfc90ud.dll  CWnd::OnWndMsg : [f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\wincore.cpp Line 2014]
  </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
- `<File>` the source file location of the allocation event
- `<Line>` the source line number in the file
- `<Type>` a string indicating the datatype of the allocated object, if known
- `<Handle>` the value of the allocated handle
- `<HandleType>` the type of handle allocated, if known
- `<Synchronization>` the type of synchronization object
- `<Thread>` the id of the thread in which the allocation was made (shown as a decimal value)
- `<Time>` the timestamp of the allocation
  
  This is a relative 'ticker' time rather than an absolute time, and is not measured in hours/mins/secs.
- `<ReportType>` type of trace message
- `<Message>` a message from a TRACE() macro or OutputDebugString()
The User Interface

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 tvExample.exe CTeststakView::CTeststakView : [c:\program files (x86)\software verification\c++ thread validator\tvexample\testsvw.cpp Line 226]`  

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/>`

3.16 Starting your target program

Starting options

There are four 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
The User Interface

- **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
- **Link a library** (provided) to your program which will cause Thread Validator to be started whenever the program is started

**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.

### 3.16.1 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
The User Interface

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](image)

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.
  
  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 n\(^{th}\) 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.
The User Interface

- **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 thumbnail 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.
The User Interface

- 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 Coverage 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
The User Interface

Administrator privileges in wizard mode

If administrator privileges are required you'll be reminded of the need to restart here:
The User Interface

- **Start Application...** shows the Administrator Privileges Required confirmation dialog before restarting

![Administrator Privileges Required dialog](image)

**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.

- **Go!** start your program and attach Thread Validator to it

  Double clicking a program in the list will also start it immediately.
The User Interface

Administrator privileges in dialog mode

If administrator privileges are required, the Go! button will show the privileges icon reminding you of the need to restart.

- **Go!** shows the Administrator Privileges Required confirmation dialog before restarting.

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
The User Interface

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

### 3.16.2 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.

### 3.16.3 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.
The User Interface

Or use the shortcut

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.

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:

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.

<table>
<thead>
<tr>
<th>ID</th>
<th>Admin</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>7820</td>
<td></td>
<td>EgsUpdate.exe</td>
</tr>
<tr>
<td>5644</td>
<td></td>
<td>ExpressTray.exe</td>
</tr>
<tr>
<td>7268</td>
<td></td>
<td>Fitbit Connect.exe</td>
</tr>
<tr>
<td>2396</td>
<td></td>
<td>FitbitConnectService.exe</td>
</tr>
<tr>
<td>10115</td>
<td></td>
<td>GarminService.exe</td>
</tr>
<tr>
<td>912</td>
<td></td>
<td>Gradients.exe</td>
</tr>
<tr>
<td>7452</td>
<td></td>
<td>HELPMAN.EXE</td>
</tr>
<tr>
<td>8120</td>
<td></td>
<td>HotkeyUtility.exe</td>
</tr>
<tr>
<td>732</td>
<td></td>
<td>i_view32.exe</td>
</tr>
</tbody>
</table>

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.

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.
The User Interface

- **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.
3.16.4 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 User Interface

The following applies only if you did not start Thread Validator in administrator mode.

When choosing the 'wait for program' method described in this topic, a restart of Thread Validator with administrator privileges will be required to proceed.

This is the case whether you are in Wizard or Dialog user interface mode.

**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.

**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 located in slightly different places.

In this section we'll cover the Wizard mode first and the Dialog mode later.

**The wait for application wizard**

The first page of the wizard lets you specify the application or choose one that you've waited for previously.
Choose the process and click **Next >>** for the next page of the wizard.

**Page 1: Choosing the process**

- **Application to wait for**  
  Type or **Browse** to set the application name to launch

  You can also choose a batch file and the first executable started in the batch file will be launched.

- **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

**Page 2: Data collection**

The second page controls when to start collecting information.
Depending on your application, and what you want to test, 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.

- **Collect data from application** if it’s the startup Thread you want to monitor, then obviously start collecting data from launch

---

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 confirms the application and prompts you to start waiting:

- **Wait for Process...** starts waiting and then injects Thread Validator into the specified process, showing progress status

  The button changes to show **Stop Waiting**. Click this to abandon the wait.

### 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]

Timing issues are inherit with 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]

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
The User Interface

The User Interface

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.

Dialog mode

In Dialog mode, all the settings are in one dialog which 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 Wait For Process button.
3.16.5 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

  There is no toolbar or shortcut option.

**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 dialog](image)

- 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](image)

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.

3.16.6 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.

3.16.7 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`

- **Add...** adds a new item to the list
- **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
3.17 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.

3.18 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.
The User Interface

3.19 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

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.
The User Interface

About box

Help menu ➤ About Thread Validator... ➤ shows contact and copyright information, as well as details of your license

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®.

**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

**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®.

**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

You can find topic specific tutorials on the main Tutorials tab and the Help menu.
Command Line Interface
4 Command Line Interface

Thread Validator provides a command line interface to allow you to perform automated critical section data collection.

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
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.

---

### Example - running a session

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.

```
-program "c:\myProgram.exe" -saveSession "c:\myResults\session1.tvm" -displayUI -refreshSummary -refreshLocks -refreshActiveObjects
```

A brief explanation of each argument:

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-program</td>
<td>&quot;c:\myProgram.exe&quot;</td>
<td>The target program to launch</td>
</tr>
<tr>
<td>-saveSession</td>
<td>&quot;c:\myResults\session1.tvm&quot;</td>
<td>After the application finishes, the session should be saved in this file</td>
</tr>
<tr>
<td>-displayUI</td>
<td></td>
<td>Show the user interface during the thread test</td>
</tr>
<tr>
<td>-refreshSummary</td>
<td></td>
<td>These tabs should be refreshed when the test completes</td>
</tr>
<tr>
<td>-refreshLocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-refreshActiveObjects</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.1 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
Specify 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

-programToMonitor

Specifies which program the data is collected from, but does not change which process is initially launched. Include the extension.

The child program will be monitored by Thread Validator when the initial 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 you will need to quote this to get it accepted by the batch file parser.

Only valid in conjunction with -program.

Examples:

-programToMonitor c:\testbed-child-process.exe
-programToMonitor testbed-child-process.exe
-programToMonitor "<<Any>>"
**Command Line Interface**

- **-program** `c:\testbed.exe` **-programToMonitor** `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.

- **-programToMonitorLaunchCount**

  Specify the \text{n}th 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 <variables>**

  Environment variables for program, as a series of name/value pairs.

  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;**
  - **-arg "APP_FLAG=ON;"**
  - **-arg "APP_COMMS=ON; APP_DEBUG=OFF;"**
  - **-arg "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\n
-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**
Command Line Interface

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

-waitName
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:

```
-waitName c:\testbed.exe
-waitName "c:\new compiler\version2\testbed.exe"
```

**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"
```

## 4.2 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 setting the `wShowWindow` member to the following values
## Command Line Interface

<table>
<thead>
<tr>
<th>Option</th>
<th>wShowWindow member</th>
<th>Launched application is shown...</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>launchAppShowMaximized</td>
<td>SW_SHOWMAXIMIZED</td>
<td>Maximized and activated</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>launchAppShowMinimized</td>
<td>SW_SHOWMINIMIZED</td>
<td>Minimized and activated</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>launchAppShowMinNoActive</td>
<td>SW_SHOWMINNOACTIVE</td>
<td>Minimized and not active</td>
</tr>
<tr>
<td>-launchAppShowNA</td>
<td>SW_SHOWNA</td>
<td>Shown at current size and position but not activated</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>launchAppShowNoActivate</td>
<td>SW_SHOWNOACTIVATE</td>
<td>Show at most recent size and position but not activated</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>launchAppShowNormal</td>
<td>SW_SHOWNORMAL</td>
<td>Show at original size and position and activated</td>
</tr>
</tbody>
</table>

- **-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**
4.3 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.mvm
-baseline "c:\base line\testMacro1.mvm"

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 manager 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.
Command Line Interface

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:

```
-sessioonLoad c:\results\testMacro1.tvm
-sessioonLoad "c:\test results\testMacro1.tvm"
-sessioonLoad 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:

```
-sessioonCompareXML c:\regtests\testMacro1.xml
-sessioonCompareHTML "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.

4.4 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"
```

### 4.5 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"
```
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\n[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"
```
Command Line Interface

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
tvExample.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\tvExample\DebugNonLink enable=TRUE
```

Example:

```
Rule:DoHook
"E:\OM\C\threadValidator\tvExample 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.

4.6 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.mvs

--; do export and save of the results
-exportAsHTML c:\testbed\results\test1.html
-saveSession c:\testbed\results\test1.mvm
```

## 4.7 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`

### 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
Command Line Interface

- 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.

- **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 Bug Validator uses it (obsolete)
- **-22** ➤ Java xrun supplied that is not allowed because Java Bug Validator uses it (obsolete)
- **-23** ➤ Java cp supplied that is not allowed because Java Bug Validator uses it (obsolete)
- **-24** ➤ Java classpath supplied that is not allowed because Java Bug 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
## Command Line Reference

### Command line reference

The following alphabetical list provides a convenient look-up for all the command line arguments used in
# Command Line Interface

 automated regression testing.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-?</code></td>
<td>Print command line help on the standard output.</td>
</tr>
<tr>
<td><code>-allArgs</code></td>
<td>Pass the remainder of the command line to the program being launched.</td>
</tr>
<tr>
<td><code>-arg</code></td>
<td>Pass command line arguments to the target program. Can be used multiple times.</td>
</tr>
<tr>
<td><code>-baseline</code></td>
<td>Loads a previous session as the baseline session against which the recorded session is compared for regressions or improvements.</td>
</tr>
<tr>
<td><code>-commandFile</code></td>
<td>Specify a file from which to read the command line arguments.</td>
</tr>
<tr>
<td><code>-directory</code></td>
<td>Set the working directory in which the program is executed.</td>
</tr>
<tr>
<td><code>-displayUI</code></td>
<td>Force the Thread Validator user interface to be displayed during the test.</td>
</tr>
<tr>
<td><code>-dllHookFile</code></td>
<td>Points to a file listing the DLLs to be hooked for the test.</td>
</tr>
<tr>
<td><code>-doNotInteractWithUser</code></td>
<td>Never display dialog boxes in the target application that is being profiled.</td>
</tr>
<tr>
<td><code>-echoArgsToUser</code></td>
<td>Environment variables for program, as a series of name/value pairs</td>
</tr>
<tr>
<td><code>-exportAsHTML</code></td>
<td>Export the session data as an HTML or XML file when Thread Validator has finished collecting data from the target program.</td>
</tr>
<tr>
<td><code>-exportAsXML</code></td>
<td>Specify a plain text file listing file locations to be used during testing. See the format of the file below.</td>
</tr>
<tr>
<td><code>-fileLocations</code></td>
<td></td>
</tr>
<tr>
<td><code>-help</code></td>
<td>Print command line help on the standard output.</td>
</tr>
<tr>
<td><code>-hideCommandPrompt</code></td>
<td>Any launched console window will be hidden during the test.</td>
</tr>
<tr>
<td><code>-hideUI</code></td>
<td>Hide the Thread Validator user interface during the test.</td>
</tr>
<tr>
<td><code>-injectID</code></td>
<td>Set the numeric (decimal) id of a process for Thread Validator to attach to.</td>
</tr>
<tr>
<td><code>-injectName</code></td>
<td>Set the name of the process for Thread Validator to attach to.</td>
</tr>
<tr>
<td><code>-launchAppHide</code></td>
<td>Hide the target application during the test.</td>
</tr>
<tr>
<td><code>-launchAppShow</code></td>
<td>Show the target application during the test.</td>
</tr>
<tr>
<td><code>-launchAppShowMaximized</code></td>
<td>Show the target application maximized and activated.</td>
</tr>
<tr>
<td><code>-launchAppShowMinNoActive</code></td>
<td>Show the target application minimized and activated.</td>
</tr>
<tr>
<td><code>-launchAppShowMinimized</code></td>
<td>Show the target application minimized and not active.</td>
</tr>
<tr>
<td><code>-launchAppShowNA</code></td>
<td>Show the target application at current size and position but not activated.</td>
</tr>
<tr>
<td><code>-launchAppShowNoActivate</code></td>
<td>Show the target application at most recent size and position but not activated.</td>
</tr>
<tr>
<td><code>-launchAppShowNormal</code></td>
<td>Show the target application at original size and position and</td>
</tr>
</tbody>
</table>

Thread Validator Help                                                                                                     Copyright © 2003-2017 Software Verify Limited
### Command Line Interface

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-loadSession</strong></td>
<td>Load a previously created session to be merged with the data from the session being recorded.</td>
</tr>
<tr>
<td><strong>-loadSettings</strong></td>
<td>Points to a previously saved settings file to be used for the test.</td>
</tr>
<tr>
<td><strong>-monitorAService</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>-numSessions</strong></td>
<td>Set the number of sessions that can be loaded at once.</td>
</tr>
<tr>
<td><strong>-program</strong></td>
<td>Specify the full file system path of the executable target program to be started by Thread Validator, including any extension.</td>
</tr>
<tr>
<td><strong>-programToMonitor</strong></td>
<td>Changes which program the data is collected from but does not change which process Thread Validator initially launches.</td>
</tr>
<tr>
<td><strong>-programToMonitorLaunchCount</strong></td>
<td>Specify the nth invocation of the programToMonitor which is to have its data collected.</td>
</tr>
<tr>
<td><strong>-refreshActiveObjects</strong></td>
<td>Automatically refresh the <strong>Active Objects tab</strong> once a test is complete.</td>
</tr>
<tr>
<td><strong>-refreshAnalysis</strong></td>
<td>Automatically refresh the <strong>Analysis tab</strong> once a test is complete.</td>
</tr>
<tr>
<td><strong>-refreshCoverage</strong></td>
<td>Automatically refresh the <strong>Coverage tab</strong> once a test is complete.</td>
</tr>
<tr>
<td><strong>-refreshCurrentLocks</strong></td>
<td>Automatically refresh the <strong>Current Locks tab</strong> once a test is complete.</td>
</tr>
<tr>
<td><strong>-refreshLocks</strong></td>
<td>Automatically refresh the <strong>Locks tab</strong> once a test is complete.</td>
</tr>
<tr>
<td><strong>-refreshObjects</strong></td>
<td>Automatically refresh the <strong>Objects tab</strong> once a test is complete.</td>
</tr>
<tr>
<td><strong>-refreshPerThreadLocks</strong></td>
<td>Automatically refresh the <strong>Per Thread Locks tab</strong> once a test is complete.</td>
</tr>
<tr>
<td><strong>-refreshSummary</strong></td>
<td>Automatically refresh the <strong>Summary tab</strong> once a test is complete.</td>
</tr>
<tr>
<td><strong>-refreshThreads</strong></td>
<td>Automatically refresh the <strong>Threads tab</strong> once a test is complete.</td>
</tr>
<tr>
<td><strong>-resetSettings</strong></td>
<td>Forces Thread Validator to reset (nearly) all settings to the default state.</td>
</tr>
<tr>
<td><strong>-saveSession</strong></td>
<td>Save the session data when all data has finished being collecting from the target program.</td>
</tr>
<tr>
<td><strong>-sessionCompareHTML</strong></td>
<td>Compare two sessions, producing an HTML or XML report detailing any regression and improvements.</td>
</tr>
<tr>
<td><strong>-sessionCompareXML</strong></td>
<td>Loads a previously created session to be merged with the data from the session being recorded.</td>
</tr>
<tr>
<td><strong>-settings</strong></td>
<td>Points to a previously saved settings file to be used for the test.</td>
</tr>
<tr>
<td><strong>-showCommandPrompt</strong></td>
<td>Any launched console window will be shown during the test.</td>
</tr>
<tr>
<td><strong>-showErrorsOnCommandPrompt</strong></td>
<td>If an error occurs when launching the application, the error will be reported on the command line.</td>
</tr>
<tr>
<td><strong>-showErrorsWithMessageBox</strong></td>
<td>Force errors to be displayed using a message box when</td>
</tr>
</tbody>
</table>
-waitName
Name a process that Thread Validator will wait for.
Working with NT Services
5 Working with NT 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.

Service Account & User Account

If at all possible, run your service on the same user account that you run the Thread Validator user interface.

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.
5.1 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:

<table>
<thead>
<tr>
<th></th>
<th>32 bit Thread Validator</th>
<th>64 bit Thread Validator</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bit service</td>
<td>svlTVStubService.lib</td>
<td>svlTVStubService.lib</td>
</tr>
<tr>
<td>64 bit service</td>
<td>N/A</td>
<td>svlTVStubService_x64.lib</td>
</tr>
</tbody>
</table>

All the functions exported from these libraries are exported as extern "C" so that C and C++ users can use them.

The simple header file svlTVStubService.h and svlServiceError.h is found in the svlTVStubService directory of the install area.

The header files provides an error enumeration and the API functions.

**API changes - February 2018**

To make the API easier to use with services we 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 the 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 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_SERVICE_ERROR declarations to SVL_SERVICE_ERROR.

Your previous startup code probably looked like this:

```c
SVL_SERVICE_ERROR errCode;

errCode = svlTVStub_LoadThreadValidator();
```
Change it to this:

```c
SVL_SERVICE_ERROR errCode;
errCode = svlTVStub_LoadThreadValidator();
errCode = svlTVStub_SetServiceCallback(serviceCallback, NULL);
errCode = svlTVStub_StartThreadValidator();
```

The `serviceCallback` would look something like this:

```c
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 please examine the source code `service.cpp` in the example `service` project.

**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`. An alternative solution is to prevent the service callback from being called once the service status has been set to running. Do this by calling `svlCVStub_SetServiceCallback(NULL, NULL);`.

**API Reference**

**SVL_SERVICE_ERROR Enumeration**
typedef enum _svlServiceError
{
    SVL_OK,                              // Normal behaviour
    SVL_ALREADY_LOADED,                  // Stub DLL already loaded into service
    SVL_LOAD_FAILED,                     // Failed to load stub DLL into service
    SVL_FAILED_TO_ENABLE_STUB_SYMBOLS,   // Loaded DLL, but failed to enable stub symbols because of
    SVL_NOT_LOADED,                      // Couldn't unload DLL because DLL not loaded
    SVL_FAIL_UNLOAD,                     // Couldn't unload DLL because couldn't find function
    SVL_FAIL_TO_CLEANUP_INTERNAL_HEAP,   // Couldn't get the internal stub heap and thus couldn't continue
    SVL_FAIL_MODULE_HANDLE               // Couldn't get the stub DLL handle so couldn't continue
    SVL_FAIL_COULD_NOT_FIND_ENTRY_POINT, // Couldn't find the DLL entry point to start the validator
    SVL_FAIL_TO_START                    // Failed to start the Validator
} SVL_SERVICE_ERROR;

The API consists of the following functions. Each function will be documented in detail after the list of functions.

extern "C" SVL_SERVICE_ERROR svlTVStub_LoadThreadValidator();
extern "C" SVL_SERVICE_ERROR svlTVStub_LoadThreadValidator6432();
extern "C" SVL_SERVICE_ERROR svlTVStub_StartThreadValidator();
extern "C" SVL_SERVICE_ERROR svlTVStub_UnloadThreadValidator();
extern "C" SVL_SERVICE_ERROR svlTVStub_SetServiceCallback(serviceCallback_FUNC callback,
                                                        void*                userParam);

// debugging functions
extern "C" void svlTVStub_setLogFileName(const wchar_t* fileName);
extern "C" void svlTVStub_deleteLogFile();
extern "C" void svlTVStub_writeToLogFileA(const char* text);
extern "C" void svlTVStub_writeToLogFileW(const wchar_t* text);
extern "C" void svlTVStub_writeToLogFile(SVL_SERVICE_ERROR errCode);
extern "C" void svlTVStub_writeToLogFileLastError(DWORD errCode);
extern "C" void svlTVStub_dumpPathToLogFile();

Which function you should call is shown in the table below.

<table>
<thead>
<tr>
<th></th>
<th>32 bit Thread Validator</th>
<th>64 bit Thread Validator</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bit service</td>
<td>svlTVStub_LoadThreadValidator()</td>
<td>svlTVStub_LoadThreadValidator6432()</td>
</tr>
<tr>
<td>64 bit service</td>
<td>N/A</td>
<td>svlTVStub_LoadThreadValidator()</td>
</tr>
</tbody>
</table>

Loading the Thread Validator DLL into your service
To load the Thread Validator stub dll `svlThreadValidatorStub(_x64).dll` into your service, use the function below, *not* `LoadLibrary()`.

```c
extern "C" SVL_SERVICE_ERROR svlTVStub_LoadThreadValidator();
```

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 trace information.

This function can be used when monitoring:

- 32 bit services or applications with Thread Validator
- 64 bit services or applications with Thread Validator x64

There is another version of the function that you should use when monitoring 32 bit builds with Thread Validator x64

```c
extern "C" SVL_SERVICE_ERROR svlTVStub_LoadThreadValidator6432();
```

### Unloading the Thread Validator DLL from your service.

Similarly to the loading, you'll need to use the following to unload the Thread Validator stub dll, and *not* `FreeLibrary()`.

```c
extern "C" SVL_SERVICE_ERROR svlTVStub_UnloadThreadValidator();
```

This removes all hooks from your service, unloads the DLL and destroys the stub heap after the DLL has unloaded.

If you just call `FreeLibrary()`, any hooks installed in your service will remain in the service and may lead to a crash once the DLL is unloaded, and the stub heap workspace will not be freed, resulting in a memory leak.

If you use the load method above, you must also use the unload method prior to your application being closed down. It doesn't matter how the application is closed down but failure to do this will almost certainly result in a crash.

### 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.

```
extern "C" SVL_SERVICE_ERROR svlTVStub_SetServiceCallback(serviceCallback_FUNC callback, void *userParam);
```

`userParam` is a value you can supply which will then be passed to the callback every time the callback is called during instrumentation.

Here is an example callback which ignores the `userParam`.

```c
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

Once you have successfully loaded the Thread Validator DLL you can start the Validator inspecting the service.

```
extern "C" SVL_SERVICE_ERROR svlTVStub_StartThreadValidator();
```

### Setting a filename for all logging data to be written to

To use any of the following logging functions you first need to set the name of the filename used for logging. 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.
extern "C" void svlTVStub_setLogFileName(const wchar_t* fileName);

Deleting the logfile

This function deletes the log file.

extern "C" void svlTVStub_deleteLogFile();

Writing ANSI text to the logfile

Call this function to write a standard ANSI character string to the log file.

extern "C" void svlTVStub_writeToLogFileA(const char* text);

Write UNICODE text to the logfile

Call this function to write a unicode character string to the log file. The characters are cast to ANSI prior to writing. As such you can't write Korean to the log file. This is simply a convenience function to write wide characters as simply as ANSI characters.

extern "C" void svlTVStub_writeToLogFileW(const wchar_t* text);

Writing error code descriptions to the logfile

Call this function to write a human readable description of the SVL_SERVICE_ERROR error code to the log file.

extern "C" void svlTVStub_writeToLogFile(SVL_SERVICE_ERROR errCode);

Writing LastError code descriptions to the logfile

Call this function to write a human readable description of the Windows error code to the log file.

extern "C" void svlTVStub_writeToLogFileLastError(DWORD errCode);

Dumping the PATH environment to the logfile

Call this function to write 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.

extern "C" void svlTVStub_dumpPathToLogFile();

Thread Validator DLL behaviour

The DLL prepares itself in different ways and shuts itself down differently depending on whether:
Working with NT Services

- The DLL is directly linked to the application for use with the API or injected with Thread Validator
  - The DLL expects to oversee and manage the application shutdown.
- The DLL is loaded by using the function above
  - The DLL expects to be removed prior to application shutdown and the its behaviour is undefined once you enter the program shutdown sequence.

This difference in behaviour is intentional and is done to allow the use of the stub DLL in services.

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 svTVStubService.lib?

5.2 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   tvExampleCritSect;

void WINAPI service_main(DWORD   dwArgc,
                         LPTSTR   *lpszArgv)
{
    if (bLogging)
    {
        svlTVStub_setLogFileName(SZLOGFILENAME);
        svlTVStub_deleteLogFile();
    }

    InitializeCriticalSection(&tvExampleCritSect);
    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 C++ Thread Validator x64 use svlTVStub_LoadThreadValidator6432())

    if (bLogging)
    {
        svlTVStub_writeToLogFileW(_T("About to load C++ 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("C++ Thread Validator load failed. \r\n"));
            svlTVStub_writeToLogFileLastError(lastError);
            svlTVStub_writeToLogFile(errCode);
            svlTVStub_dumpPathToLogFile();
        }
        else
        {
            svlTVStub_writeToLogFileW(_T("C++ 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 C++ Thread Validator\r\n"));
        errCode = svlTVStub_SetServiceCallback(serviceCallback,  // the callback
(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 C++ Thread Validator\r\n"));
}

erCode = svlTVStub_StartThreadValidator();
if (bLogging)
{
    if (errCode != SVL_OK)
    {
        DWORD lastError;

        lastError = GetLastError();
        svlTVStub_writeToLogFileW(_T("Starting C++ Thread Validator failed. \r
\n"));
        svlTVStub_writeToLogFileLastError(lastError);
        svlTVStub_writeToLogFile(errCode);
    }

    svlTVStub_writeToLogFileW(_T("Finished loading C++ Thread Validator\r
\n"));
}
else
{
    if (bLogging)
        svlTVStub_writeToLogFileW(_T("Not using C++ 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(&tvExampleCritSect);
// do work

dwErr = ServiceStart(dwArgc, lpszArgv);

// finished doing work

LeaveCriticalSection(&tvExampleCritSect);

} // **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!
    //MSVCR7! 78001436()
    //MSVCR7! 7800578c()
    //DBGHELP! 6d55da25()
    //DBGHELP! 6d55de83()
    //DBGHELP! 6d53705d()
    //DBGHELP! 6d51cc69()
    //DBGHELP! 6d51f6e8()
    //DBGHELP! 6d524ebf()
    //DBGHELP! 6d52a7b0()
    //DBGHELP! 6d52b00a()
    //DBGHELP! 6d526487()
    //DBGHELP! 6d526d47()
    //DBGHELP! 6d5264f7()
    //SVLTHREADVALIDATORSTUB!
    //SVLTHREADVALIDATORSTUB!
    //SVLTHREADVALIDATORSTUB!
    //SVLTHREADVALIDATORSTUB!
    //SVLTHREADVALIDATORSTUB!
    //SVLTHREADVALIDATORSTUB!
void WINAPI service_main(DWORD dwArgc, LPTSTR *lpszArgv)
{
    // load C++ 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);
    }
    else
    {
        ReportStatusToSCMgr(SERVICE_RUNNING, dwErr, 0);
    }

    DeleteCriticalSection(&testCritSect);
    DeleteCriticalSection(&tvExampleCritSect);

    return;
}
// do the work of the service

doWork(dwArgc, lpszArgv);

// unload C++ 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 C++ Thread Validator from
// removing all its hooks successfully. If C++ 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 C++ 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!
// 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;
Examples
6 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 example or services before you can use them.

6.1 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 tvExample.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.
### Examples

<table>
<thead>
<tr>
<th>Testst - Teststak Windows Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
</tr>
</tbody>
</table>

- **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 counter3a and counter3b stop changing**
  - Tid: 0 Counter3a: 0
  - Tid: 0 Counter3b: 0

- **Good lock strategy**
  - Tid: 0 CounterGa: 0
  - Tid: 0 CounterGb: 0
  - Tid: 0 CounterGc: 0

- **Wait potential deadlock detect, Deadlock when counterA and counterB stop changing**
  - Tid: 0 CounterA: 0 infiniteA_a 0 infiniteA_b 0 infiniteB_a 0
  - Tid: 0 CounterB: 0 infiniteA_a 0 infiniteA_b 0 infiniteB_a 0

- **Wait potential deadlock detect, Deadlock when counterSleepA and counterSleepB stop changing**
  - Tid: 0 CounterSleepA: 0 infiniteSleepA_a 0 infiniteSleepA_b 0 infiniteSleepB_a 0
  - Tid: 0 CounterSleepB: 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 `tvExample\TESTSVW.CPP`

**File menu**

- **File menu ➔ Exit ➔** closes the example application

**Test menu**

<table>
<thead>
<tr>
<th>Test menu option</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Thread deadlock</td>
</tr>
<tr>
<td>3 Thread deadlock</td>
</tr>
<tr>
<td>Start 2 thread deadlock</td>
</tr>
<tr>
<td>Start 3 thread deadlock</td>
</tr>
<tr>
<td>Start 3 thread deadlock (two)</td>
</tr>
<tr>
<td>Start 2 thread deadlock with infinite wait</td>
</tr>
<tr>
<td>Start 2 thread deadlock with thread suspend</td>
</tr>
<tr>
<td>Start 2 thread deadlock with infinite sleep</td>
</tr>
<tr>
<td>Bad lock strategy example</td>
</tr>
<tr>
<td>Good lock strategy example</td>
</tr>
<tr>
<td>Good lock strategy example (faster)</td>
</tr>
<tr>
<td>Create Deadlock then determine stack traces</td>
</tr>
<tr>
<td>Recursion test</td>
</tr>
<tr>
<td>Leave non-entered critical section</td>
</tr>
<tr>
<td>Delete still active critical section</td>
</tr>
<tr>
<td>Leave critical sections in the wrong order</td>
</tr>
<tr>
<td>Potential deadlock 2 threads</td>
</tr>
<tr>
<td>Potential deadlock 3 threads</td>
</tr>
<tr>
<td>Test ExitThread()</td>
</tr>
<tr>
<td>Force crash inside a locked critical section (part 1)</td>
</tr>
<tr>
<td>Force crash inside a locked critical section (part 2)</td>
</tr>
</tbody>
</table>

- **Test menu ➔ ...**
Examples

**2 Thread deadlock**

OnHandlesStartthread1()
OnHandlesStartthread2()

Creates two named threads that increment counter 1 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 counter A, 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 counter 3A, 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**

OnTestStart3threaddeadlockwithinfiniterwait()

See comments in this function for an explanation. Deadlock occurs when Counter Ia, Ib stop incrementing.

**Start 2 Thread deadlock with thread suspend**

OnTestStart3threaddeadlockwiththreadssuspend()

See comments in this function for an explanation. Deadlock occurs when Counter Sa, Sb stop incrementing.
Help menu

Help menu » ...

About Thread Validator
TvExampleApp::OnAppAbout()
Shows a simple information dialog using code in tvExample.cpp

6.1.1 Building the example application

Where to find the example application

The example project is in the tvExample 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

There are a variety of solutions and projects for different versions:

- tvExample.dsp » for Microsoft® Developer Studio® 6.0
- tvExample_VSx_x.sln » for Microsoft® Visual Studio / .net

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.

6.2 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.

6.2.1 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 svtVSStubService(_x64).lib demonstrating use
Examples 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.

Once you start the service, it will show as 'Running'

The service provides the following command line options:

- **-install**  Install the service
- **-remove**  Uninstall the service
- **-debug**  Run as a console application for debugging
- **-?**  Display the help message
- **-help**  Display the help message

Examples when run from the cmd prompt (as administrator):

```
D:\dev\tvExample\service\Debug>service.exe -?
service -install to install the service
service -remove to remove the service
service -debug <params> to run as a console app for debugging
service -? display this message
service -help display this message
```

6.2.2 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".

**6.2.3 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
Examples

- 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.

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.
Hook Reference
# 7 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.

## 7.1 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()
- InitializeCriticalSectionAndSpinCount()
- LeaveCriticalSection()
- SetCriticalSectionSpinCount()
- TryEnterCriticalSection()

### Events

- CreateEventA()
- CreateEventW()
- OpenEventA()
- OpenEventW()
- PulseEvent()
- SetEvent()
- ResetEvent()

### File

- CreateFileA()
- CreateFileW()
Hook Reference

- 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()
Hook Reference

- SuspendThread()
- TerminateThread()

Timers

- CancelWaitableTimer()
- CreateWaitableTimerA()
- CreateWaitableTimerW()
- OpenWaitableTimerA()
- OpenWaitableTimerW()
- SetWaitableTimer()

Timer Queues

- ChangeTimerQueueTimer()
- CreateTimerQueue()
- CreateTimerQueueTimer()
- DeleteTimerQueue()
- DeleteTimerQueueEx()
- DeleteTimerQueueTimer()

Waits

- MsgWaitForMultipleObjects()
- MsgWaitForMultipleObjectsEx()
- SignalAndWait()
- RegisterWaitForSingleObject()
- RegisterWaitForSingleObjectEx()
- UnregisterWait()
- UnregisterWaitEx()
- WaitForSingleObject()
- WaitForMultipleObjects()
- WaitForSingleObjectEx()
- WaitForMultipleObjectsEx()
Frequently Asked Questions
8 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.

8.1 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.
Frequently Asked Questions

- 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.

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
#endif

#ifndef _WIN64
#define DWORD_PTR   unsigned __int64
#else
#define DWORD_PTR   DWORD
#endif

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 be 8 chars + 1 terminator
    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 character
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.

```c
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 tvExample.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.

8.2 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.

Here's a couple of reasons:

- You may not have enabled ordinal handling
  
  Use the Source Parsing page on the global settings dialog to set the Map Ordinals to function names option

- 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

8.3 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 Map Ordinals to function names check box on the Source Lookup page of the global settings dialog.

  Here's an example of Ordinal function names without debug information: Ordinal711 and Ordinal187
Frequently Asked Questions

And here's the same thing without debug information but after ordinal to symbol conversion, showing `_cdecl operator new()` and `int AfxWinMain()`

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:
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.

8.4 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.
Frequently Asked Questions

- **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:\svlTVExceptionReport.txt.

The picture below shows an breakpoint exception report (artificially generated).
What is in svlTVExceptionReport.txt?

In the event of a crash, the file `c:\svlTVExceptionReport.txt` contains information that identifies where Thread Validator was executing when it crashed.

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.

8.5 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:
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.

<table>
<thead>
<tr>
<th>DbgHelp.dll version</th>
<th>DiagHelp.dll version loaded into target</th>
<th>DbgHelp.dll version expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.11.1.493</td>
<td>6.11.1.493</td>
<td>6.11.1.493</td>
</tr>
</tbody>
</table>

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 does installing DbgHelp.dll work under Windows File Protection?

The following applies to older operating systems.

Thread Validator uses the Microsoft Debugging DLL, DbgHelp.dll, to provide debugging information about your test application.

Microsoft constantly upgrade DbgHelp.dll to provide an improved, richer debugging API, and some of the API features we use are only available in more recent versions of DbgHelp.dll.
DbgHelp.dll on Windows NT

DbgHelp.dll is not installed by default on Windows NT.

However, Windows NT does not have Windows File Protection and installing DbgHelp.dll on Windows NT is straightforward.

DbgHelp.dll on Windows 2000

On Windows 2000 machines, the standard version of DbgHelp.dll needs to be upgraded as it does not have all the necessary API features we need.

See the Installation section below.

DbgHelp.dll on Windows XP

On Windows XP machines, the standard version of DbgHelp.dll does not need to be upgraded, and Thread Validator installation is straightforward.

DbgHelp.dll Installation

Our license program checks for the correct version of DbgHelp.dll and if necessary, overwrites c:\windows\system32\dllCache\dbgHelp.dll so that Windows File Protection cannot restore it to c:\windows\system32\dbgHelp.dll.

On Windows 2000 and Windows XP, Windows File Protection may complicate things depending on whether your machine has backup operating system files on it, ready for re-installs.

- **Backup installation files are on the hard disk**

  When your machine has the backup installation files already on the hard disk, Windows helpfully and silently restores the files without asking you!

  We have modified our products so that when that happens, the correct version of DbgHelp.dll will still be used, so long as your application is not stored in the Windows System directory.

  Even if a manual inspection of c:\windows\system32\dbgHelp.dll shows it has not been upgraded, our tools will still use the correct dbghelp.dll version installed with the product.

- **Backup installation files not on the hard disk**

  When Windows notices that both the files have changed, it asks you for the installation media (CD/DVD) so that it can restore the files.

  You can choose to cancel the restoration so that the more recent version of DbgHelp.dll is used.

  It's worth noting, that even though DbgHelp.dll is completely backwards compatible, you can't necessarily rely on Microsoft automatically upgrading DbgHelp.dll, as Microsoft's own Debugging tools fails to upgrade DbgHelp.dll for example.
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 tvExample.exe and tvExample.pdb from our example application.

- **Correct symbol file found**

  DbgHelp first searches in various places looking for tvExample.pdb

<table>
<thead>
<tr>
<th>DbgHelp Search Info</th>
<th>DBGHELP: C:\WINDOWS\symbols\dll\tvExample.pdb - file not found</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbgHelp Search Info</td>
<td>DBGHELP: C:\WINDOWS\symbols\dll\symbols\exe\tvExample.pdb - file not found</td>
</tr>
<tr>
<td>DbgHelp Search Info</td>
<td>DBGHELP: C:\WINDOWS\symbols\dll\exe\tvExample.pdb - file not found</td>
</tr>
</tbody>
</table>

  Depending on your machine, there may be other search paths included.

  Finally tvExample.pdb is found in the same directory as the .exe file of the target program

<table>
<thead>
<tr>
<th>DbgHelp Search Info</th>
<th>DBGHELP: tvExample - private symbols &amp; lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbgHelp Search Info</td>
<td>D:\dev\tvExample\Debug\NonLinkS\OtvExample.pdb</td>
</tr>
<tr>
<td>Loaded symbols</td>
<td>Loaded PDB symbols for D:\dev\tvExample\Debug\NonLinkS\OtvExample.exe</td>
</tr>
</tbody>
</table>

  DbgHelp loads private symbols and lines, (the alternative being that DbgHelp loads public symbols).

  **Outcome:**
Frequently Asked Questions

Success. Symbols are loaded.

- **Missing symbol file**

  As before, DbgHelp first searches in various places looking for tvExample.pdb

  But, tvExample.pdb doesn't get found in the same directory as the .exe file of the target program.

  ![DbgHelp Search Info]

  tvExample.pdb never gets found on the search path.

  SymSrv might then look for additional locations for tvExample.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]

  **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 tvExample.pdb

  This time, tvExample.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 Search Info]

  ![DbgHelp Search Info]

  ![DbgHelp Search Info]

  ![DbgHelp Search Info]

  Finally all options are exhausted.
Frequently Asked Questions

| DbgHelp Search Info | DBGHELP: tvExample - export symbols not 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 tvExample.
Frequently Asked Questions

• **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.

### 8.6 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:
Then add the following paths for all configurations; Debug, Release, etc:

C:\Program Files (x86)\Software Verify\C++ Thread Validator\stub
C:\Program Files (x86)\Software Verify\C++ 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:

```c++
#include "stdafx.h"
#include "stublib.h"

int main(int argc, char* argv[])
{
    return 0;
}
```

To fix this problem simply include windows.h before stublib.h

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(71) : error C2146: syntax error : missing ';' before identifier 'reserved3'
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
#endif

#ifndef _SVL_TVSTUBSERVICE_H
    #define _SVL_TVSTUBSERVICE_H
#endif
#include "svlServiceError.h"

// IMPORTANT.
// If you use svlTVStub_LoadThreadValidator() to load svlThreadValidatorStub.dll into your application, you must also use svlTVStub_UnloadThreadValidator() to unload the DLL prior to your application being closed down. Failure to do so will almost certainly result in a crash.
// It does not matter how the application is closed down, you must ensure that you use svlTVStub_UnloadThreadValidator() to unload the DLL if you have loaded it.
// The DLL prepares itself in different ways and shuts itself down differently depending on if:
// a) Directly linked to the application for use with the API or injected with Thread Validator.
// 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 application shutdown and the behaviour of the DLL is undefined once you enter the program shutdown sequence.
// 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

See the implementation file
Frequently Asked Questions

#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 svlThreadValidatorStub6432.dll or svlThreadValidatorStub_x64.dll as appropriate
        if (hModule != NULL)
        {
            // DLL loaded, set the service callback function
            SETCALLBACK_FUNC setCallbackFunc;

            setCallbackFunc = (SETCALLBACK_FUNC)GetProcAddress(hModule, "apiSetServiceCallback");
            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, "apiEnableStubSymbols");

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, "apiGetInternalMVstubHeap");

        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_UNLOAD;
}

hModule = NULL;
}
else
    errCode = SVL_NOT_LOADED;

return errCode;
8.7 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**

- 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
Copyright notices

9 Copyright notices

9.1 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.

* 1.7.2 Copyright (c) 2002-2009 Vivek Thampi
  * All rights reserved.
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *
  * Redistributions of source code must retain the above copyright notice,
  * this list of conditions and the following disclaimer.
  * Redistributions in binary form must reproduce the above copyright notice,
  * this list of conditions and the following disclaimer in the documentation
  * and/or other materials provided with the distribution.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
  * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
  * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
### Index

- **A**
  - About box 262
  - Acquiring threads at startup 118
  - Acquisition order (locks) 205
  - Active objects display
    - settings 140
  - Active objects tab
    - finding synchronization objects 188
      - overview 10
      - user interface 84
  - Address 0x006d0065 (faq) 322
  - Administrator
    - privileges 238
    - running as 238
  - Alignment of numerical data 139
  - Allocations (related) 90
  - Analysis
    - results 90
  - Analysis tab
    - overview 10
    - query form 90
    - user interface 90
  - API
    - NT Service 289
  - Applications to monitor
    - settings 133
  - -arg 268
  - arg (command line) 268
  - args (command line) 268
  - Arguments
    - command line overview 267
  - Attaching to a process 246
  - Auto merge
    - thread coverage 126
  - Automatic error detection 111
  - Auto-purging sessions 180
  - Bad lock strategies 90
  - -baseline 276
  - Bookmarks 90
    - active objects 84
    - adding 186
    - dialog 186
    - using 186
  - Bookmarks (editor) 211
  - Borland
    - enabling debugging 26
    - supported compilers 16
  - Breakpoint at ExitProcess 159
  - Buffer size
    - for inter process communication 158
  - Built in editor 211
  - Cache directory 155
  - Caching
    - symbols 162
  - Caching thread coverage data 126
  - Call stack depth 112
  - Callstack
    - current locks 72
    - FAQ 321
    - locks 57
    - per thread locks 65
    - sampled thread lock order 74
    - sampled thread lock states 74
    - settings 112
  - Callstack monitoring 112
  - Callstacks
    - analysis 90
    - for all threads 197
    - for deadlocked threads 197
    - relations 90
  - Child application monitoring 133
  - Class and function filters (faq) 318
Class relations 90
Clearing coverage data 126
Clipboard use 44
Closing sessions 180
Code editing 211
CodeWarrior
  supported compilers 16
Coff debug format 162
Collapsing lines
  in the editor 211
Collect
  settings 108
Collected data 90
Collecting lock data 108
Colours 135, 137
  analysis tab 90
  coverage 80
Command description (status bar) 50
Command files
  command line 280
  example 280
Command line
  examples 267
  interface overview 267
  reference 283
  refreshing display 273
  return codes 281
  start modes 268
  unrecognised arguments 267
  user interface 273
Command line arguments
  command files 280
  DLL hooks 278
  echoing 281
  editor 147
  errors 281
  export format 277
  export options 277
  file locations 278
  global settings 278
  help 281
  launching a program 268
  load settings 278
  MAP files 278
  PDB files 278
  pipe warnings 281
  resetting global settings 268
  session export 277
  -commandFile 280
Communication
  between stub and ui 15
Communication buffer size 158
Compaq
  supported compilers 16
Compilers
  enabling debugging 27
  supported 13, 16
  symbol lookup 162
Configure menu 44
Contact us 15
Contended locks
  callstack collection 110
Contention callstack 57, 65, 72
Contentions
  maximum 53
  maximum ratio 53
  summary 53
  total 53
Converting ordinals to functions 176
Copy and paste 44
Coverage
  summary 53
Coverage data
  caching 126
  collection 126
  filters 126
Coverage tab 80
  overview 10
  user interface 80
Crashes (faq) 325
Creation callstack
  current locks 72
  locks 57
  per thread locks 65
  sampled thread states 74
Critical section
  address 57, 65, 72, 74
  callstack collection 110
  code coverage 80
  contention 57, 65, 72, 74
  current locks 72
  locks 57, 65
  recursion 57, 65, 72, 74
  sampled thread states 74
  wait time 57, 65, 72, 74
Critical section information (faq) 321
Critical sections
active 84
data collection 108
in DllMain 118
used in one thread 195
within watermarks 84
Current lock
statistics 72
Current locks data 72
Current locks tab 72
data collection 108
overview 10
user interface 72
Custom queries 90

- D -
Data alignment 139
Data collection 238, 246, 251, 257
Applications to monitor 133
Callstack 112
Collect 108
Detect 111
Historical data 123
Hook insertion 119
Hook trace 121
Hooks 125
Lock callstacks 110
Startup data 118
statistics on the status bar 50
summary 53
Thread coverage 126
Data collection (faq) 321
Data display
settings 139
Data format 139
Data views menu 46
DbgHelp (faq) 327
DbgHelp messages 99
Deadlock detection 111, 200
Deadlock errors 53
Deadlocked thread callstacks 197
Deadlocks
analysis tab 90
lock order 205
Deallocated data discarding 123

Debug
DLL information 217
instrumentation log data 223
status 217
Debug enabling
Borland compilers 26
Microsoft compilers 25
MinGW compilers 26
QtCreator 26
Debugging tools for windows 164
Default settings 106
Deferred symbol loading 159
Delay loaded DLLs 141
Deleting sessions 180
Delphi symbol lookup 162
Detect
settings 111
Detection intervals 111
Diagnostic information 99
Diagnostic tab
user interface 99
Diagnostics
.net warnings 159
data collection 159
Dialog mode 104, 238, 246, 251
Dialogs 205
About 262
Administrator privileges required 238, 246
Advanced Stack Walk 112
Applications to Monitor 133
Attach to running process wizard 246
Auto Merge 126
bookmarks 186
Borland compiler debug information 162
Check for software updates 224
Compiler debug information 162
Contended Critical Section Callstacks 57, 65, 72, 74
Critical Section Information 57, 65, 72, 74
CriticalSection information 205
Current Locks Settings 72
Deadlocked Thread PostMortem 197
DLLs needing ordinal to function resolution 176
Downloading 224
Edit session alias 180
Editor 211
Environment variables 259
Export session 232
Index

Dialogs 205
File Locations Settings 84
File paths 151
File scan 151, 176
Filter objects by thread id 182
Find function 90, 192
Find Source File 84
Find synchronization object 90, 188
First run configuration 33
Global hook warning 169
Hook insertion settings 126
Inject into service warning 246
Inject validator into running process 246
instrumentation log data 223
Launch different application 141
Loaded modules 216
Lock Acquisition Order 57, 65, 72, 74, 84
Locks and waits in sequence order 205
Locks Settings 57, 84
MinGW compiler debug information 162
Modules containing debug information 159, 217
Monitor a service 257
No selected waitable handles 126
Not tracking waitable handles 126
Options (editor) 211
Options colour (editor) 211
Ordinal to function converter 176
PDB and MAP information 217
Per Thread Locks Settings 65
Please start your process 257
Prefetch symbols 164
Recursing Critical Section Callstacks 57, 65, 72, 74, 84
Save session 231
Scanning for files 176
Searching For Source File 84
Settings 106
Single Thread Critical Section Detector 195
Software update download confirmation 224
Software update maintenance has expired 224
Software update maintenance renewal 224
Software update schedule 224
Start an application and inject validator into the process 238
Start application wizard 238
Symbol Handling Environment Variables 164
Symbol server 164
Thread Coverage Filters 80, 126
Thread History Locks Settings 74
Thread History Settings 74
Thread information 57, 65, 72, 74, 84, 205
Tips 262
tvExample application 304
User interface chooser 104
User permissions warnings 172
Wait for application wizard 251
Wait for process to start then inject validator into process 251
WaitForSingleObject information 57, 65, 72, 74, 84, 205
watermarks 183
Diagnostics tab
overview 10
-directory 268
Directory (working)
command line 268
working 268
Disabling hooks 125
Discarding historical data 123
Display
callstack information 112
Data display 139
General colours 135
Lock colours 137
on Active objects tab 140
refreshing 215
source code context 145
tab views 46
update 57, 65, 72, 74, 80, 90, 98
updating via command line 273
watermarks 183
Display lock order 205
Displaying tabs 52
-displayUI 273
DLL
.def association 176
data debug information 217
instrumentation log data 223
DLL information 99
-dllHookFile 278
DllMain 118
DLLs
delay loaded 141
hooking 141
-doNotInteractWithUser 273
Downloading updates 224
## E

- Edit menu 44
- Editing
  - settings 147
- Editing source code 57, 65, 72, 74, 80, 84, 90, 147
- Editor 147, 211
- Environment information 99
- Environment variables 259
- Error detection 111
  - summary 53
- Error notifications 18
- Errors
  - command line 281
  - deadlock 53
  - potential deadlock 53
  - summary 53
  - threading 53
- Errors (faq) 325
- Examining source code 57, 65, 72, 74, 80, 84, 90
- Example application
  - building 308
  - getting started 27
  - overview 304
  - usage 304
- Example NT service
  - building 309
  - building sample client 310
  - building sample service utility 311
  - overview 309
- Examples
  - command line 267
  - deadlock checking 200
  - example application 304
  - finding allocations in functions 192
  - finding objects used in one thread 195
  - finding synchronization objects 188
  - how to use 304
  - service source code 295
- ExitProcess breakpoint 159
- Expired maintenance 224
- Export
  - command line options 277
  - file locations 151
  - formats 232
  - hooked DLLs 141
  - HTML or XML 232
  - sessions 232
- -exportAs options 277
- Extensions (faq) 343

## F

- F1 help 262
- Failed symbol load (faq) 321
- File and line numbers 157
- File cache
  - settings 155
- File extensions (faq) 343
- File locations 151
  - active objects tab 84
  - command line 278
  - settings 151
- File menu 43
- File scan 151
- -fileLocations 278
- Filtering diagnostics
  - analysis tab 90
- Filters
  - Hooked DLLs 141
  - summary 53
- Find
  - bad lock strategies 90
  - deadlocks 90
  - lock errors 90
  - locked critical sections 90
  - potential deadlocks 90
  - trace messages 90
  - waiting critical sections 90
- Finding
  - critical sections used in one thread 195
  - deadlocked thread callstacks 197
  - functions with allocations 192
  - synchronization objects 188
  - thread callstacks 197
- Finding source files
  - active objects tab 84
- First run configuration 33
- First watermark 183
- Flushing caches 155
- Forcing a deadlock check 200
- Format
  - file locations 151
  - numerical data 139
Index

Format
  XML session export  234
Formatting (editor)  211
Fortran 95
  supported compilers  16
Freed handles  123
Frequently asked questions
  clearing symbol cache  318
  crashes and error reports  325
DbgHelp  327
  extensions  343
  failing to launch  318
  general  318
  ideas  318
  not getting results  321
NT services  318
  overview  318
  power users  343
  unexpected results  322
Function keys  51
Function search  90
Functions
  finding allocations in  192
  finding synchronization objects  188
Functions with waitable handles  119

- G -

GDI handle search  90
General colours
  settings  135
General settings
  Editing  147
  File cache  155
  File locations  151
  Inter process communication  158
  Source browsing  145
  Source lookup  157
  Substitute drive  155
  Symbols and warnings  159
  Symbols lookup  162
  Symbols server  164
Getting started  25
Global settings
  (see Settings)  105
  command line  278
  resetting  106
  Global settings dialog  106
  Goto bookmark  186

- H -

Handle search  90
Handles
  average  53
  discarding  123
  maximum  53
  total  53
Help  281
  notation  8
  -help command line  281
Help menu  48, 262
  -hideUI  273
Hiding tabs  52
Highlighting
  active objects display  84
  coverage tab display  80
  current locks tab display  72
  locks tab display  57
  per thread locks tab display  65
  threads tab display  74
Historical data
  settings  123
History of threads  74
Hook groups  125
Hook insertion
  functions with waitable handles  126
  settings  119
  summary  53
Hook reference
  overview  314
  Synchronization hooks  314
Hook trace
  settings  121
Hooked DLLs
  settings  141
Hooking messages  99
Hooking trace messages  121
Hooks
  settings  125
HTML help  262
HTML session export
  user interface  232
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I</strong></td>
<td></td>
</tr>
<tr>
<td>Icons</td>
<td>52</td>
</tr>
<tr>
<td>Ideas for features (faq)</td>
<td>318</td>
</tr>
<tr>
<td>Import</td>
<td></td>
</tr>
<tr>
<td>file locations</td>
<td>151</td>
</tr>
<tr>
<td>hooked DLLs</td>
<td>141</td>
</tr>
<tr>
<td>Information about lock or wait</td>
<td>57, 65, 72, 74</td>
</tr>
<tr>
<td>Injecting</td>
<td></td>
</tr>
<tr>
<td>command line</td>
<td>268</td>
</tr>
<tr>
<td>into running process</td>
<td>246</td>
</tr>
<tr>
<td>reasons for failure (faq)</td>
<td>318</td>
</tr>
<tr>
<td>Instrumentation log data</td>
<td>223</td>
</tr>
<tr>
<td>Intel</td>
<td></td>
</tr>
<tr>
<td>supported compilers</td>
<td>16</td>
</tr>
<tr>
<td>Intel symbols</td>
<td>162</td>
</tr>
<tr>
<td>Inter process communication settings</td>
<td>158</td>
</tr>
<tr>
<td>Introduction</td>
<td>10</td>
</tr>
<tr>
<td><strong>K</strong></td>
<td></td>
</tr>
<tr>
<td>Keyboard shortcuts</td>
<td>51</td>
</tr>
<tr>
<td><strong>L</strong></td>
<td></td>
</tr>
<tr>
<td>Last watermark</td>
<td>183</td>
</tr>
<tr>
<td>Launch</td>
<td></td>
</tr>
<tr>
<td>dialog</td>
<td>27</td>
</tr>
<tr>
<td>environment variables</td>
<td>259</td>
</tr>
<tr>
<td>methods</td>
<td>238</td>
</tr>
<tr>
<td>reasons for failure (faq)</td>
<td>318</td>
</tr>
<tr>
<td>wizard</td>
<td>27</td>
</tr>
<tr>
<td>-launchApp options (command line)</td>
<td>273</td>
</tr>
<tr>
<td>Launching a program</td>
<td>238</td>
</tr>
<tr>
<td>command line</td>
<td>268</td>
</tr>
<tr>
<td>hooks during</td>
<td>141</td>
</tr>
<tr>
<td>quick start</td>
<td>27</td>
</tr>
<tr>
<td>Licensing</td>
<td>15</td>
</tr>
<tr>
<td>Limiting number of sessions</td>
<td>180</td>
</tr>
<tr>
<td>Line collapsing in the editor</td>
<td>211</td>
</tr>
<tr>
<td>Line numbers</td>
<td>157</td>
</tr>
<tr>
<td>Line numbers in MAP file (faq)</td>
<td>327</td>
</tr>
<tr>
<td>Line time collection</td>
<td>238</td>
</tr>
<tr>
<td>Linkers</td>
<td>16</td>
</tr>
<tr>
<td>Linking Thread Validator</td>
<td>259</td>
</tr>
<tr>
<td>Loaded modules</td>
<td>216</td>
</tr>
<tr>
<td>Loading Coverage Validator into NT service</td>
<td>289</td>
</tr>
<tr>
<td>Loading sessions command line</td>
<td>276</td>
</tr>
<tr>
<td>menu option</td>
<td>231</td>
</tr>
<tr>
<td>Loading settings</td>
<td>172</td>
</tr>
<tr>
<td>-loadSettings</td>
<td>278</td>
</tr>
<tr>
<td>Local filters</td>
<td>80</td>
</tr>
<tr>
<td>Local settings</td>
<td>105</td>
</tr>
<tr>
<td>Lock</td>
<td></td>
</tr>
<tr>
<td>statistics</td>
<td>57</td>
</tr>
<tr>
<td>Lock acquisition order</td>
<td>57, 65, 72, 74, 205</td>
</tr>
<tr>
<td>Lock callstacks settings</td>
<td>110</td>
</tr>
<tr>
<td>Lock colours settings</td>
<td>137</td>
</tr>
<tr>
<td>Lock errors</td>
<td>90</td>
</tr>
<tr>
<td>Lock order</td>
<td>205</td>
</tr>
<tr>
<td>Locked critical sections</td>
<td>90</td>
</tr>
<tr>
<td>Locking error detection</td>
<td>111</td>
</tr>
<tr>
<td>Locks analysis tab</td>
<td>90</td>
</tr>
<tr>
<td>contended</td>
<td>53</td>
</tr>
<tr>
<td>locked</td>
<td>53</td>
</tr>
<tr>
<td>locked at any time</td>
<td>53</td>
</tr>
<tr>
<td>locked by any thread</td>
<td>53</td>
</tr>
<tr>
<td>re-entered</td>
<td>53</td>
</tr>
<tr>
<td>slow</td>
<td>53</td>
</tr>
<tr>
<td>summary</td>
<td>53</td>
</tr>
<tr>
<td>visited and unvisited</td>
<td>53</td>
</tr>
<tr>
<td>Locks data</td>
<td>57</td>
</tr>
<tr>
<td>Locks tab</td>
<td>57</td>
</tr>
<tr>
<td>data collection</td>
<td>108</td>
</tr>
<tr>
<td>overview</td>
<td>10</td>
</tr>
<tr>
<td>user interface</td>
<td>57</td>
</tr>
<tr>
<td>Looking up symbols</td>
<td>162</td>
</tr>
</tbody>
</table>

**- M -**

<p>| Maintenance of software | 224 |
| Managers |      |
| bookmarks | 186 |
| menu | 45 |
| software maintenance | 224 |
| software updates | 224 |
| thread filters | 182 |
| watermarks | 183 |</p>
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP data in modules                        237</td>
</tr>
<tr>
<td>MAP file</td>
</tr>
<tr>
<td>command line                              278</td>
</tr>
<tr>
<td>line numbers (faq)                         327</td>
</tr>
<tr>
<td>locations                                 278</td>
</tr>
<tr>
<td>MAP file warnings                          159</td>
</tr>
<tr>
<td>Map files                                  157</td>
</tr>
<tr>
<td>MAP information                           217</td>
</tr>
<tr>
<td>Mapping ordinals                           176</td>
</tr>
<tr>
<td>Mapping ordinals to function names         157</td>
</tr>
<tr>
<td>Menus</td>
</tr>
<tr>
<td>configure                                  44</td>
</tr>
<tr>
<td>data views                                 46</td>
</tr>
<tr>
<td>edit                                       44</td>
</tr>
<tr>
<td>file                                       43</td>
</tr>
<tr>
<td>help                                       48</td>
</tr>
<tr>
<td>managers                                   45</td>
</tr>
<tr>
<td>query                                      45</td>
</tr>
<tr>
<td>software updates                          47</td>
</tr>
<tr>
<td>tools                                      46</td>
</tr>
<tr>
<td>Merging session coverage data              126</td>
</tr>
<tr>
<td>Message area (status bar)                  50</td>
</tr>
<tr>
<td>Metrowerks</td>
</tr>
<tr>
<td>supported compilers                       16</td>
</tr>
<tr>
<td>Microsoft</td>
</tr>
<tr>
<td>enabling debugging                        25</td>
</tr>
<tr>
<td>supported compilers                       16</td>
</tr>
<tr>
<td>symbols                                    162</td>
</tr>
<tr>
<td>MinGW</td>
</tr>
<tr>
<td>enabling debugging                        26</td>
</tr>
<tr>
<td>supported compilers                       16</td>
</tr>
<tr>
<td>symbol lookup                             162</td>
</tr>
<tr>
<td>Miscellaneous locking errors              90</td>
</tr>
<tr>
<td>Missing PDB/MAP files                      159</td>
</tr>
<tr>
<td>Mixed mode services                        257</td>
</tr>
<tr>
<td>Module PDB and MAP data                    237</td>
</tr>
<tr>
<td>Module search                              90</td>
</tr>
<tr>
<td>Modules</td>
</tr>
<tr>
<td>finding synchronization objects            188</td>
</tr>
<tr>
<td>hooking                                    141</td>
</tr>
<tr>
<td>loaded list                                216</td>
</tr>
<tr>
<td>manual addition                            141</td>
</tr>
<tr>
<td>Monitor a service                          257</td>
</tr>
<tr>
<td>Monitoring child applications              133</td>
</tr>
<tr>
<td>Monitoring services                        288</td>
</tr>
<tr>
<td>Modules</td>
</tr>
<tr>
<td>summary                                    53</td>
</tr>
<tr>
<td>Multiple sessions                          180</td>
</tr>
</tbody>
</table>

- N -

Naming
  threads                                  182                    |
  Naming threads (how to)                   318                    |
Native services                            257                    |
Notation used in help                       8                      |
NT services                                 |
  API                                       289                    |
  working with                              288                    |
Null handles                                119                    |
Number of sessions                          |
  command line                              276                    |
Numerical data                              |
  alignment                                 139                    |
  format                                    139                    |
  -numSessions                              276                    |

- O -

Object
  flags                                     98                     |
  handles                                    98                     |
  kernel address                             98                     |
Objects tab                                 |
  overview                                   10                    |
  user interface                             98                     |
Operating system                            |
  requirements                              13, 16                 |
Operating system information                99                     |
Options (editor)                            211                    |
Order of lock acquisition                   205                    |
Ordinal conversion to symbols               159                    |
Ordinal handling                            176                    |
Ordinal mapping (faq)                       321                    |
Ordinal maps                                 176                    |
OrdinalXXX symbols (faq)                    322                    |
Orientation                                 80                     |
Out of order detection                      111                    |
OutputDebugString messages                  121                    |

- P -

Paused start mode                           238                    |
PDB data in modules                         237                    |
PDB file
Index

PDB file
  command line 278
  debg information 217
  locations 278
PDB file warnings 159
PDF help 262
Per thread lock
  statistics 65
Per thread locks data 65
Per thread locks tab 65
  data collection 108
  overview 10
  user interface 65
Percentage bars
  current locks tab 72
  locks tab 57
  per thread locks tab 65
  threads tab 74
Performance collectors (faq) 318
Permissions 18
Pipe warnings
  command line 281
Potential deadlock detection 111
Potential deadlock errors 53
Potential deadlocks 90
Power user
  creating (faq) 343
Power user accounts 172
Predefined queries
  analysis tab 90
Prefetching symbols 164
Privileges 13, 18, 238
  -program 268
  Program information (status bar) 50
  -programToMonitor 268
  -programToMonitorLaunchCount 268
Promoting analysis data 90
Purchasing Thread Validator 15
Purging sessions 180

- Q -
Qt
  supported compilers 16
QtCreator
    enabling debugging 26
Query and search
  overview 188
Query menu 45
Questions 318
Quick start 27, 238

- R -
Readme 262
Recurring locks
  callstack collection 110
Recursion callstack 57, 65, 72, 74
Recursions
  maximum re-entries 53
  summary 53
Reference of command line options 283
Refresh 215
Refresh All 215
-refresh tab (command line) 273
Registry access 13, 18, 172
Related objects
  analysis tab 90
Related threads 205
Relations tab
  sending statistics data 90
Relaunching a program 246
Renewing maintenance 224
Report
  exporting 232
  format 232
  -resetSettings 268
Resetting
  default settings 106
Restart required 238
Restoring settings 106
Result promotion 90
Results in analysis tab 90
Return codes
  command line 281
Running data queries 90

- S -
Sales 15
Salford
  supported compilers 16
Sampled thread
  statistics 74
Sampled threads 74
Index

Saving
  sessions 231
  settings 172
Scanning
  for files 151
Scheduling software updates 224
Select all 44
Servers (symbols) 164
Service
  injecting into 246
Service account (NT services) 288
Service notification callback 289
Services
  example source code 295
Session export
  command line 277
Session management 180
  command line 276
  -sessionCompare options 276
  -sessionLoad 276
Sessions
  alias 180
  choosing 180
  closing 180
  deleting 180
  hook changes 119
  limiting 180
  loading and saving 231
  managing 180
  purging 180
Setting up 33
Settings 106, 151
  Active objects display 140
  Applications to monitor 133
  Callstack 112
  Collect 108
  Data display 139
  Detect 111
  display tabs 52
  Editing 147
  File cache 155
  File locations 151
  General colours 135
  global and local 105
  Historical data 123
  Hook insertion 119
  Hook trace 121
  Hooked DLLs 141
  Hooks 125
    in the editor 211
  Inter process communication 158
  loading and saving 172
  Lock callstacks 110
  Lock colours 137
  Source browsing 145
  Source lookup 157
  Startup data 118
  Substitute drive 155
  Symbols and warnings 159
  Symbols lookup 162
  Symbols server 164
  Thread coverage 126
  UI global hooks 169
  -settings 278
Settings (editor) 211
Shortcuts 51
  -showErrorsOnCommandPrompt 273
  -showErrorsWithMessageBox 281
  Showing tabs 52
  Single thread critical section detection 195
Sleep hooks 119
Sleeping threads 205
Snapshots of threads 74
Software updates
  credentials 33, 224
  download location 33, 224
  Software updates menu 47
Source browsing
  settings 145
Source code 80
  editing 90
  examination 90
  file locations 151
  surrounding code 145
Source code editor 147, 211
Source file
  file locations 151
  locations on command line 278
Source lookup
  settings 157
Stabs debug format 162
Stack depth 112
Stack trace discarding 123
Stack traces 112
Stack traces (faq) 321
StackWalk 112
Index

Stall detection 121
Start application wizard 238
Start collecting data 261
Starting a program
  launch methods 238
  launching 238
  methods 237
Startup data
  settings 118
Startup thread acquisition 118
Statistics
  active objects 84
  code coverage 80
  contention 57, 65, 72, 74
  critical section 57, 65, 72, 74
  current locks 72
  locks 57, 65
  recursion 57, 65, 72, 74
  thread (history) 74
  wait time 57, 65, 72, 74
Status bar
  command description 50
  message area 50
  program information 50
  statistics 50
Status bar (editor) 211
Stop collecting data 261
Stopping your program 261
Stub
  as part of Thread Validator 15
Substitute drive
  settings 155
Summary tab
  data collection 108
  overview 10
Support 15
Suspended start mode 238
Suspended threads 205
svlCVStubService 289
svlTVExceptionReport (faq) 325
Symbol cache (faq) 318
Symbol environment variables 164
Symbol loading 159
Symbol loading messages 99
Symbol lookup 33
Symbol search path environment variables 33
Symbol search paths 99
Symbols
  caching 162
  lookup 162
  mapping 176
  not loading (faq) 327
  prefetching 164
  servers 164
Symbols and DbgHelp 99
Symbols and warnings
  settings 159
Symbols lookup
  settings 162
Symbols server
  settings 164
SymChk 164
Synchronization code coverage 80
Synchronization functions 119
Synchronization hooks 125
  hook reference 314
Synchronization location visits 126
Synchronization objects
  searching for 188
Syntax highlighting
  in the editor 211
System requirements 13

- T -

Tab size format 145
Tab visibility 46
Tab width 145
Tabs
  active objects 84
  analysis 90
  coverage 80
  current locks 72
  display windows 52
  locks 57
  objects 98
  overview 10
  per thread locks 65
  threads 74
Third party coverage 126
Third party DLL settings
  UI global hooks 169
Third party source files
  command line 278
  file locations 151
Thread
Thread
  coverage 80
  owning module 57, 65, 72, 74
  state 57, 65, 72, 74
  statistics 57, 65, 72, 74
Thread acquisition 118
Thread coverage
  auto merge 126
  caching 126
  data collection 126
  filters 126
  settings 126
  summary 53
Thread data (sampled) 74
Thread history 74
Thread sampling 74, 80
Thread stall detection 121
Thread status information 158
Thread status interval 111
Thread synchronization code coverage 80
Thread Validator
  contact 15
  design principles 11
  features 10
  getting started 25
  impact on program 11
  licensing 15
  purchasing 15
  quick start 27
  section overview 10
  stub and ui 15
  support 15
  what is it 10
  workflow 11
Threads 53
  callstacks for deadlocks 197
  contended 53
  fetching all callstacks 197
  filters 182
  id 182
  locked 53
  names 182
  re-entering locks 53
  related 205
  sleeping 205
  summary 53
  suspended 205
  total entered 53
  using locks 53
Threads tab 74
  overview 10
  user interface 74
Tips 262
Toolbar reference 48
Tools
  editor 211
  loaded modules 216
Tools menu 46
Trace message monitoring 121
Trace messages 90
  discarding 123
Tracking waitable handles 119, 126
Tutorials 25, 262
tvExample (application) 308
Types of object 98
- U -

UI global hooks
  managing 169
  settings 169
Unhooked lines 99
Unknown symbols (faq) 322
Unrecoverable errors 325
Unvisited locks 53
Update download location 224
Updating software 224
User
  account (NT services) 288
  permissions 18, 172
  privileges 18
User interface
  active objects tab 84
  analysis tab 90
  as part of Thread Validator 15
  command line 273
  coverage tab 80
  current locks tab 72
  locks tab 57
  mode 104
  mode for injection 246
  mode when launching 238
  mode when waiting for a program 251
  objects tab 98
  parts of the interface 33
  per thread locks tab 65
### User interface
- threads tab: 74
- visibility: 273
- workflow: 33

### User permissions warnings
- 172

### Version history
- 262

### Views (tab visibility)
- 46

### Visit counts
- 126

### Visited locks
- 53

### Visual Studio
- DbgHelp.dll version: 162
- supported compilers: 16

### Wait
- callstack collection: 110

### Wait monitoring
- 108

### Waitable handles
- tracking: 126

### Waitable handles in use
- 98

### Waiting for a program
- command line: 268
- startup mode: 251

### Waiting locks
- 90

### -waitName
- 268

### Waits
- average: 53
- maximum: 53
- summary: 53
- total: 53

### Warning dialogs
- global hooks: 169
- User permissions: 172

### Warnings
- .NET: 159
- debug information: 159
- symbol loading: 159

### Watermarks
- active objects: 84
- adding: 183
- analysis tab: 90
- dialog: 183
- first and last: 183

### Working directory
- command line: 268

### XML session export
- tags used: 234
- user interface: 232