Coverage Validator

by

Software Verify

Software Verify

Troubleshoot your software

Coverage Validator

Source code coverage analysis for Windows applications built using .Net, .Net Core, C#, VB.Net, C, C++, Delphi, Fortran 95 and Visual Basic 6.

---

by Software Verify Limited

Welcome to the Coverage Validator software tool. Coverage Validator is a source code coverage analysis software tool. Using Coverage Validator you can identify unvisited functions and unvisited lines in your source code. This information can be used to inform your testing program to ensure that you test all of your software.

Coverage Validator provides numerous features to allow you integrate Coverage Validator into your regression tests and unit tests. This allows you to monitor the progress of your software testing during your development programme.

We hope you will find this document useful.

Coverage Validator Help

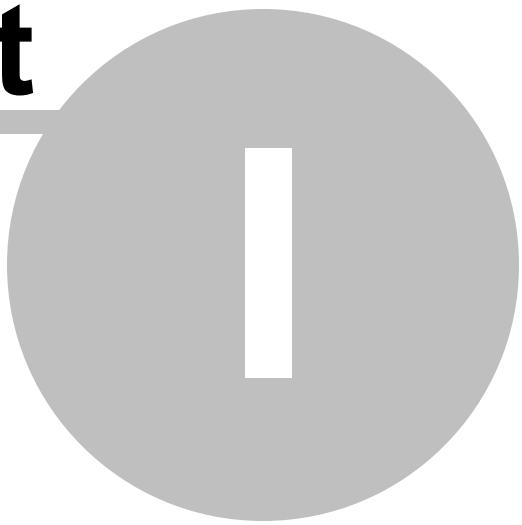
Printed: November 2024 in United Kingdom.

Table of Contents

Foreword

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

Part



1 Overview

Hi, welcome to the Coverage Validator help manual.

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

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

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

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

Read background information

The overview section covers things like:

- the capabilities of Coverage 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 merging sessions, or the command line interface.

1.1 Notation used in this help

› Instruction › steps
Menu action › steps

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

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

or

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

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

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

The second example directs you to open the Settings menu (from the menu bar in this case), and then choose the Edit 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 Coverage Validator

What is Coverage Validator?

Coverage Validator is an **automatic source code coverage analyzer** for Windows.

Coverage Validator works with versions of Windows from 10.0 through Windows XP, on x86 and x64 processors (and compatible).

What does Coverage Validator do?

Coverage Validator can find:

- which lines of your program have been executed
- visit counts for each line, function, file, directory and DLL
- visit percentages for each source file visited

The results are displayed in a summary dashboard and a variety of comprehensive but easily explorable hierarchical formats.

Source code editing is provided with colour coded lines so that you can see at a glance which lines were hooked or not hooked, and visited or unvisited.

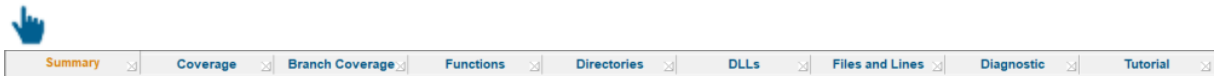
The Coverage overhead is very low and there is no need to recompile or relink the target program.

The only requirement is PDB files with debug information and/or MAP files with line number information.

Coverage Validator can also be used for unit testing and as part of a regression testing strategy used by Quality Assurance teams.

The main sections of Coverage Validator

The user interface is split via tabs into separate report sections (+Tutorials), each presenting or analysing coverage 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 summary of the code coverage for the whole application.
Coverage	A file by file summary of the number of lines visited, total visit count and percentage of the file that has been visited.
Branch Coverage	Statistics about branch coverage for each function containing branches.
Functions	Function lines visited, total visit count and percentage of each visited function.
Directories	Coverage information for each directory that contains source files
DLLs	Coverage information for each DLL that contains source files
Files and Lines	A file by file summary of the coverage details of visited files. Expandable to show individual lines and the corresponding source code.
Diagnostic	Lists diagnostic information collected by the stub, including lines that could not be hooked.

1.3 Why Coverage Validator?

Adapts to everyone's workflow

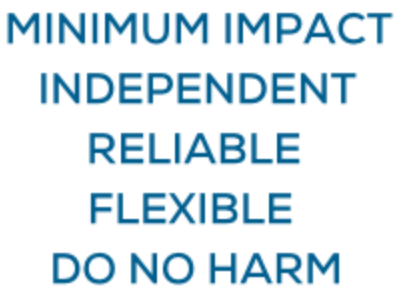
Coverage Validator allows you to find how much of your software is being executed by a particular test, using an intuitive colour-coded user interface.

If you want to edit the source code for a line that is not being visited, it's simple. Just double click on the code fragment shown and the appropriate source code file will be loaded into Coverage 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 interact with the collected data. You can also export 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 coverage reports for the software engineers.

Designed with principles

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



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

  Any hooks placed into the target program's code must not affect the registers or the condition code flags of that program. The program must behave in the same way when being inspected by Coverage 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 Coverage 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 Coverage 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**

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

  For example:

  If you are only interested in coverage 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**

  Coverage 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, Coverage Validator's user interface will continue to work.

In the unlikely event that the Coverage 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 Coverage 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

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

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

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

Registry Access Privileges

Coverage Validator requires read and write access to:

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

  This is used when working with services

If read and write access is *not* allowed:

- Coverage Validator will use default settings (thus any user selections will not apply)
- Error messages will be displayed when Coverage 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.

Operating System

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

At a minimum, Coverage Validator requires Windows XP or better.

1.5 Buying Coverage Validator and support

The best way to purchase Coverage 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 Coverage 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 Coverage Validator work?

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

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

The **stub** is typically injected into the target program and communicates with the Coverage 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.

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

Each line 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 line hook failure.

As your program executes, the hooks on each line record the visit counts for the line and communicates this to the user interface. The user interface calculates statistics based on the visit counts and provides a colour coded display for the user to inspect.

The stub can also be linked if required, so that it doesn't need to be injected into the program.

1.7 Supported Compilers

Coverage Validator will work with any portable executable (PE) file format and supports .Net, .Net Core, C#, VB.Net, C, C++, Delphi, Fortran 95 and Visual Basic.

Microsoft .Net, .Net Core

Both .Net and .Net Core technologies are supported as well all the native compilers listed below.

The following compilers are supported by Coverage Validator.

Microsoft http://www.microsoft.com

Coverage 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 Coverage Validator.

- **Microsoft Developer Studio 4.0**
- **Microsoft Developer Studio 5.0**
- **Microsoft Developer Studio 6.0**
- **Microsoft Visual Basic 6.0**
- **Microsoft Visual Studio 6.0 -** service pack 3 or later
- **Microsoft Visual Studio 7.0 / .net 2002**
- **Microsoft Visual Studio 7.1 / .net 2003**
- **Microsoft Visual Studio 8.0 / .net 2005**
- **Microsoft Visual Studio 9.0 / .net 2008**
- **Microsoft Visual Studio 10.0 / .net 2010**
- **Microsoft Visual Studio 11.0 / .net 2012**
- **Microsoft Visual Studio 12.0 / .net 2013**
- **Microsoft Visual Studio 14.0 / .net 2015**
- **Microsoft Visual Studio 15.0 / .net 2017**
- **etc...**

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

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

Intel http://www.intel.com

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

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

Metrowerks

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

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

Embarcadero https://www.embarcadero.com/

This includes compilers formerly produced by Borland.

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

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

MinGW http://www.mingw.org

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

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

➔ MinGW compiler in the *Getting Started* section.

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

- **QtCreator**

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

Salford Software http://www.salfordsoftware.co.uk

- **Salford Software Fortran 95**

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

Compaq

- **Compaq Visual Fortran 6.6**

The Compaq Visual Fortran product may be compatible with Coverage Validator. If you are using Compaq Visual Fortran and wish to use Coverage 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. In fact, the Borland C++, Borland Delphi, Metrowerks CodeWarrior, Salford Software's Fortran 95 compiler, and Intel Fortran support were all added at the request of customers.

1.8 User Permissions

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

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

Why do user privileges matter?

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

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

This is so that when Coverage 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`.

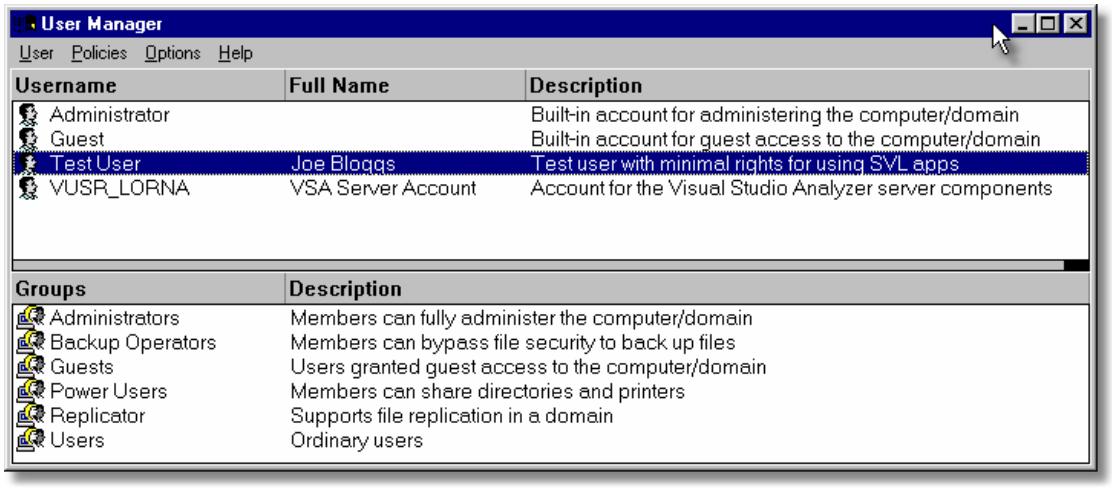
You can enable and disable various warnings using the User Permissions Warnings dialog.

User privileges

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

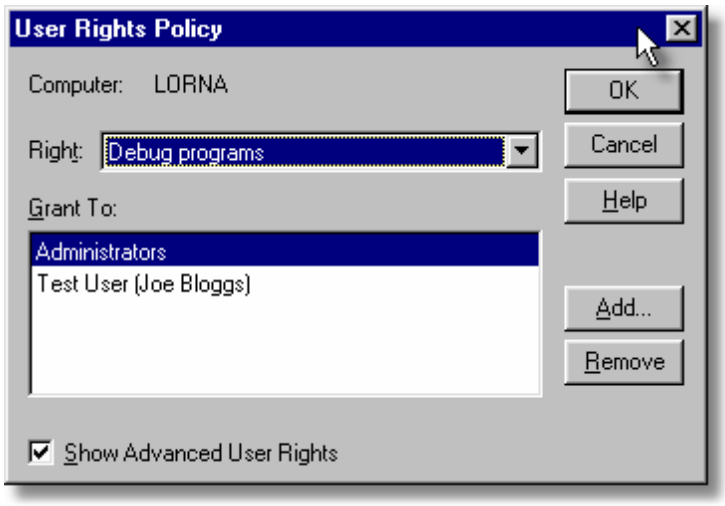
Debug Programs (SE_DEBUG_NAME)

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.

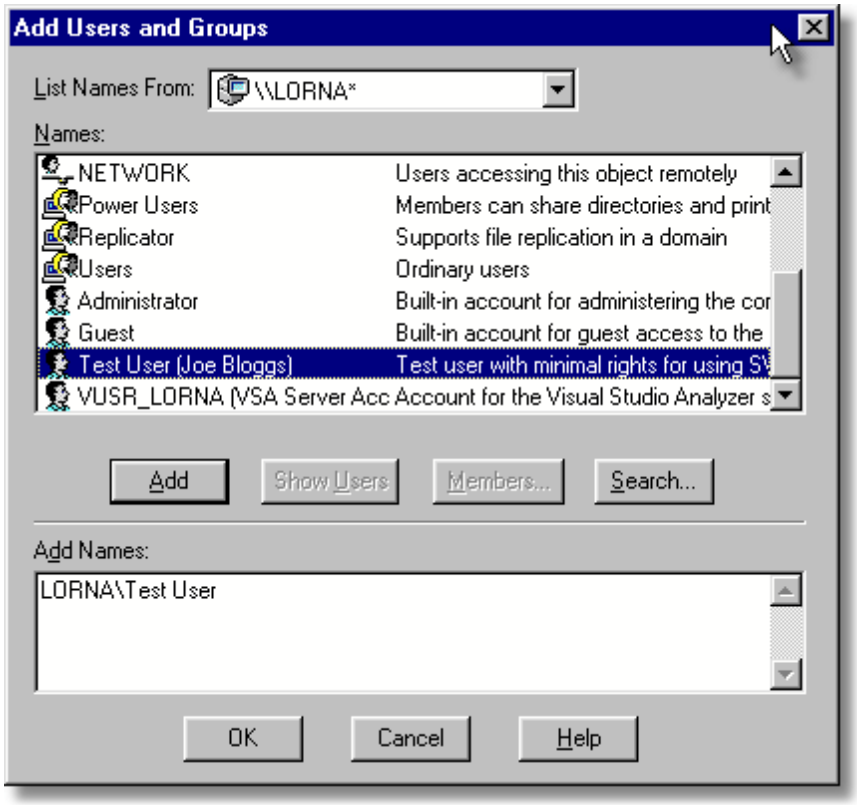


In the User Manager select the user - in this case "Test User".

Choose: **Policies** Menu › **User Rights** › check **Show Advanced User Rights** › select **Debug Programs** in the **Right** combo box



Click **Add...** › **Show Users**



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

Registry access privileges

Coverage Validator requires read and write access to:

- `HKEY_CURRENT_USER\Software\SoftwareVerification\CoverageValidator`

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

  This is used when working with services

If read and write access is *not* allowed:

- Coverage Validator will use default settings (thus any user selections will not apply)
- Error messages will be displayed when Coverage 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.

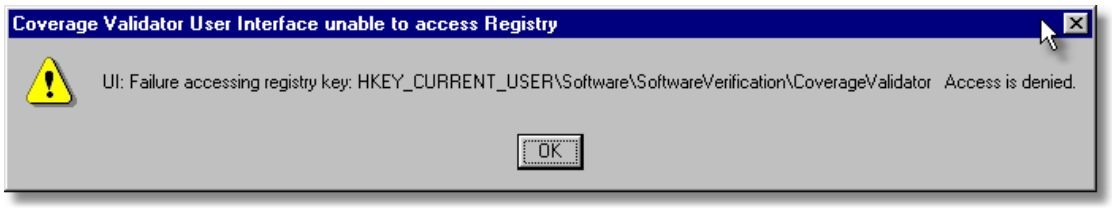
You can modify the registry access permissions using the **regedt32.exe** tool Security menu (or similar). Ask your administrator to modify your registry access permissions if you can't do this yourself.

➡ What's the difference between Regedit and Regedt32?

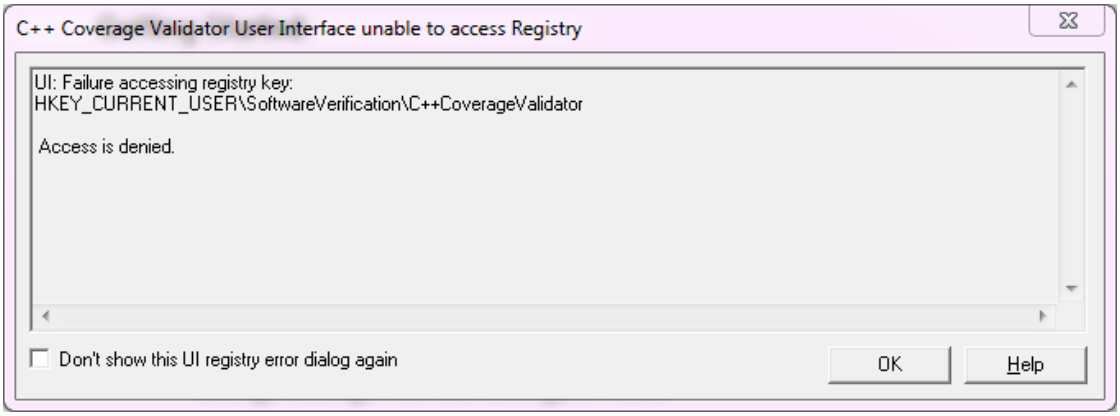
Error notifications

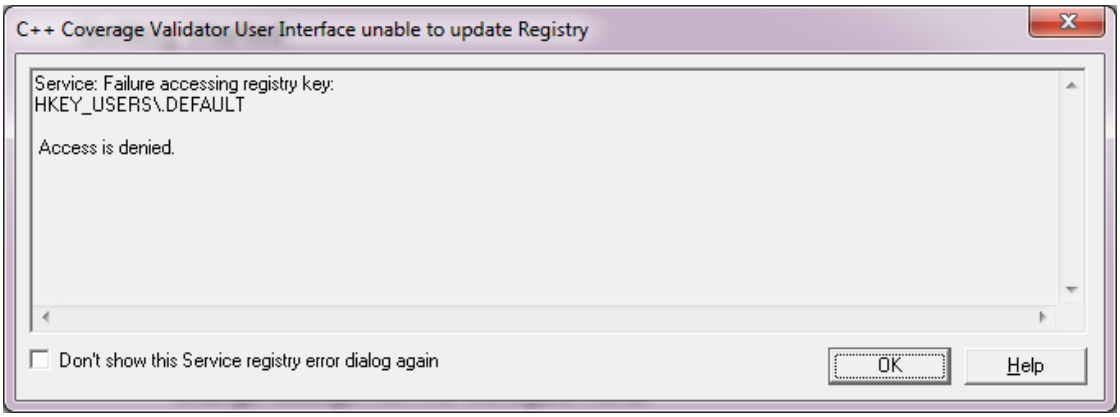
When Coverage Validator fails to gain access for read or write to the registry a message box is displayed indicating if the error is for the user interface (UI) or Services. The message indicates the name of the registry key that failed and the failure reason.

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



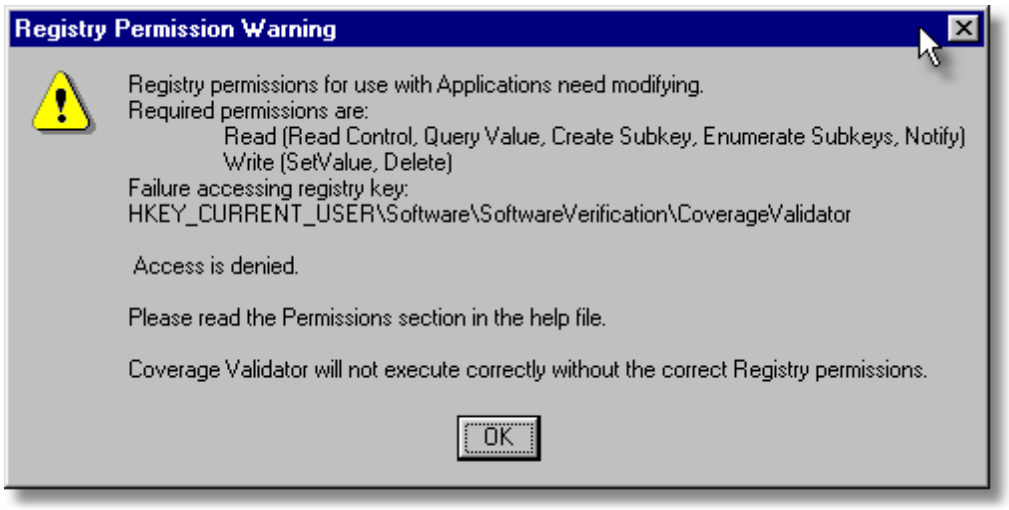
Message box like the following are displayed when Coverage Validator is not starting up or closing down. The messages differ in the registry key.





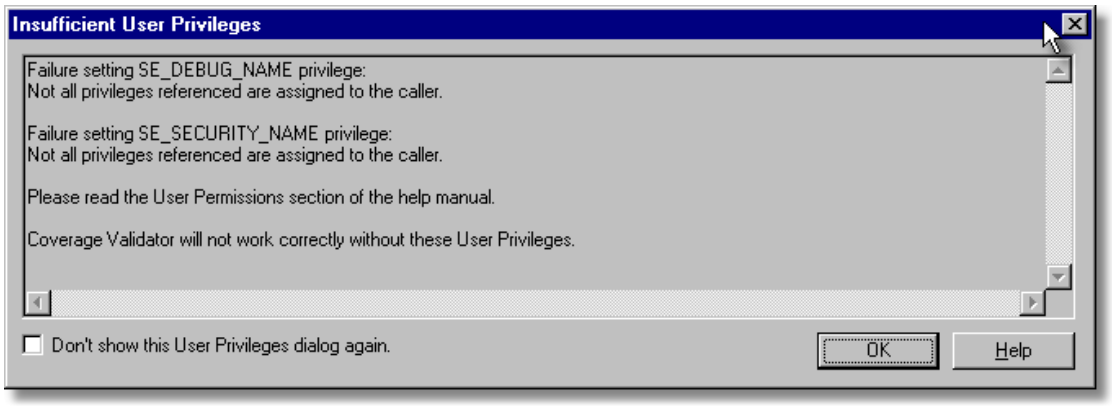
Detailed registry access error messages

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



Insufficient user privileges

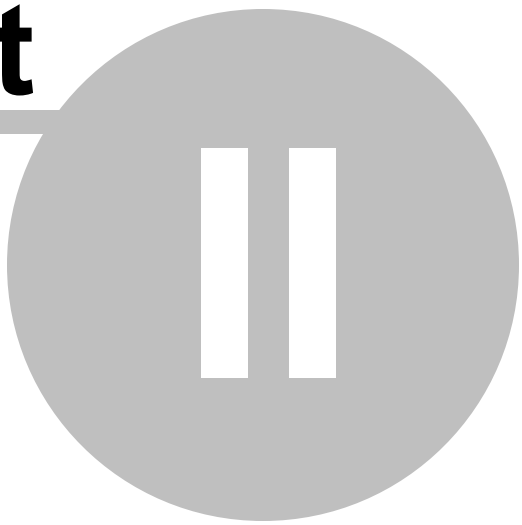
The following dialog is displayed if a user has insufficient privileges to use the software correctly.



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

➔ Creating Power User accounts for Windows XP.

Part



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?

If you have never used Coverage Validator before you have probably purchased Coverage Validator because you wish to analyse the code coverage of your application. As such, you may want to 'dive in' and start identifying your code coverage immediately.

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

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

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

2.1 Enabling Debugging

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

Detailed instructions are available for these IDEs / compilers:

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

Debug Information Formats

Thread Validator can understand debugging information in the following formats:

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

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

2.2 Quick Start

If you are:

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

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

Otherwise, we recommend reading these topics before starting:

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

Testing on the Example Program

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

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

Ensure you have debugging information

Your application needs to be compiled to produce debugging information and linked to make that debugging information. Details are available for enabling debugging information with Visual Studio, C++ Builder, Delphi, MinGW, and other compilers.

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

Launching

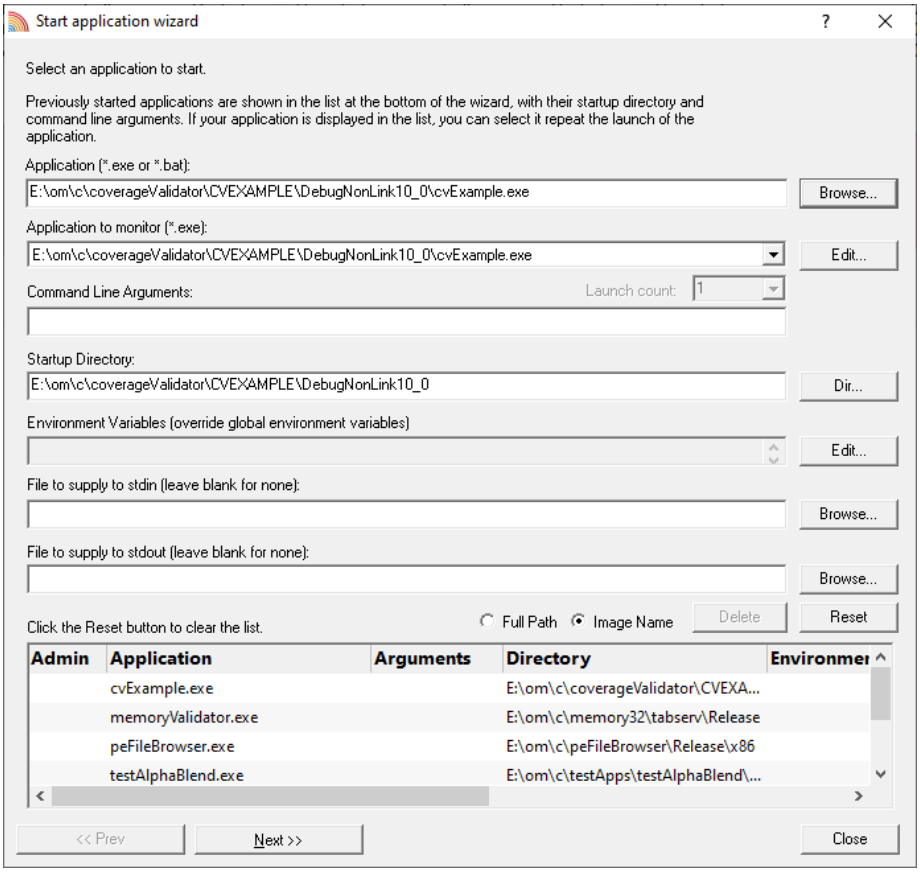
To start your program click on the launch icon on the session toolbar.

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

The Launch Wizard...

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

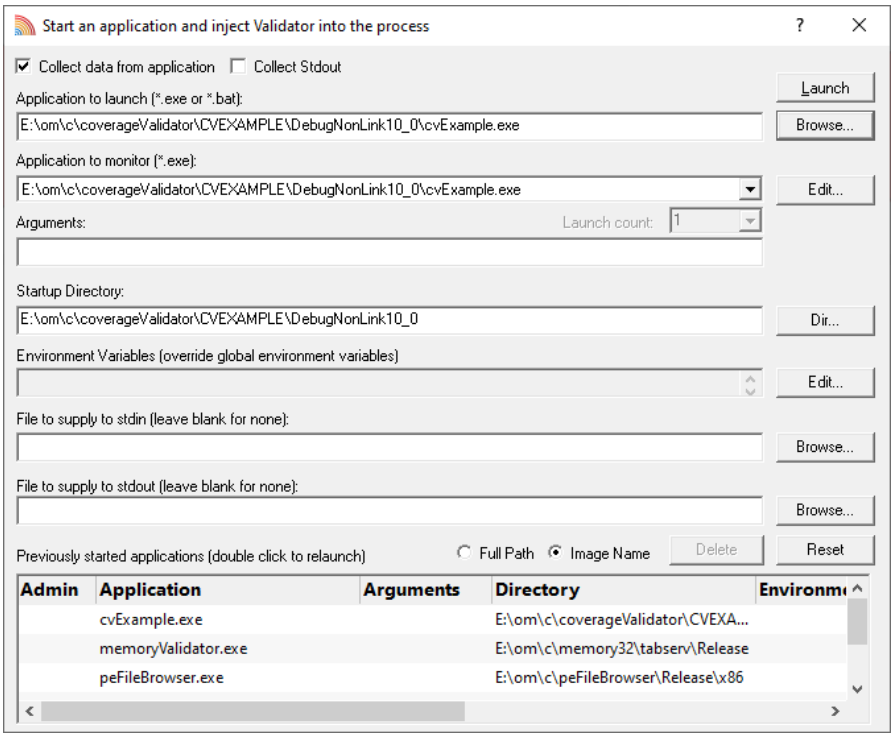
Click **Browse...** to choose a program to launch › **Next** › **Next** › **Next** › **Start Application...**



...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 › **Launch**



During launch

Coverage 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 Coverage Validator and data can be collected and viewed until the target program exits.

The Summary Tab and the Coverage Tab will update at intervals to show coverage so far. Other tabs need manually refreshing.

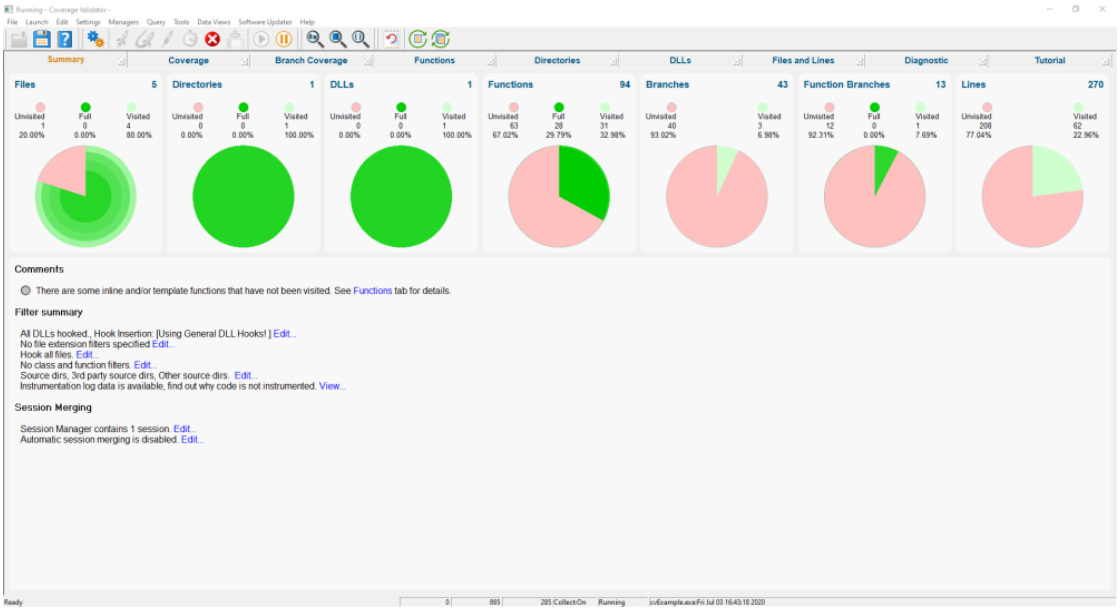
After exit - examining the output

When the target program exits, Coverage Validator closes the session. The data collection icons on the session toolbar are disabled, and the launch icons are enabled again.

The picture shown below shows an example of the Summary Tab displaying coverage at different granularities.

Some quick takeaways from this dashboard include:

- About 80% of the files were covered, but no individual file had 100% coverage (dial 1)
- A third of the functions were covered, most of those had 100% coverage (dial 4)
- Less than a quarter of all lines of source code were visited (dial 7)

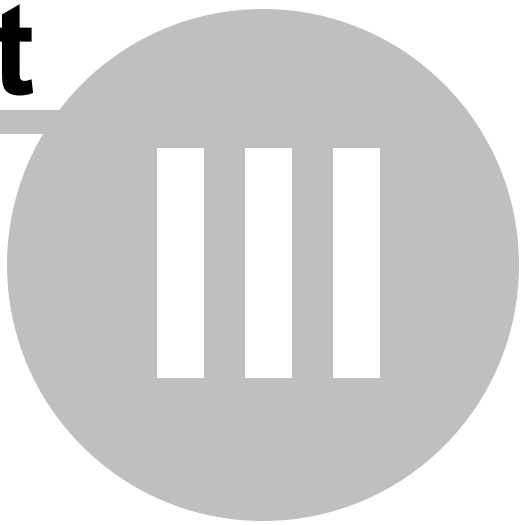


Ending the session

Even though the target program has exited, the session is still active and can be examined or saved until the session is closed via the **File** menu › **Close Session**.

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

Part



3 The User Interface

The part of Coverage Validator that you get to see is the user interface.

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 Coverage Validator.

Typical workflow

Typical usage of Coverage Validator is very simple:

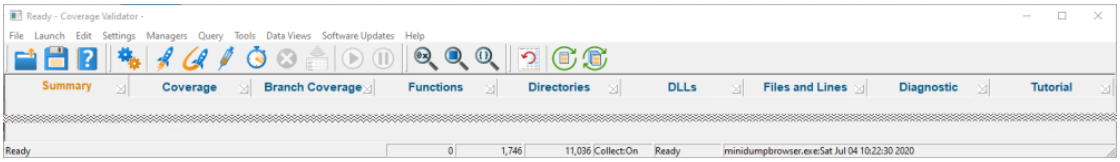
- Start the target program
- Collect and monitor the coverage data of the program
- Close the program
- Analyse the coverage - saving or exporting data if needed

However, there is much more to Coverage Validator than this simple workflow. For example, whilst your program is running, you can display data and gain insight into a bug you are looking at in the debugger. And of course you can determine the functionality needed for testing on order to achieve a higher percentage of code execution.

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.

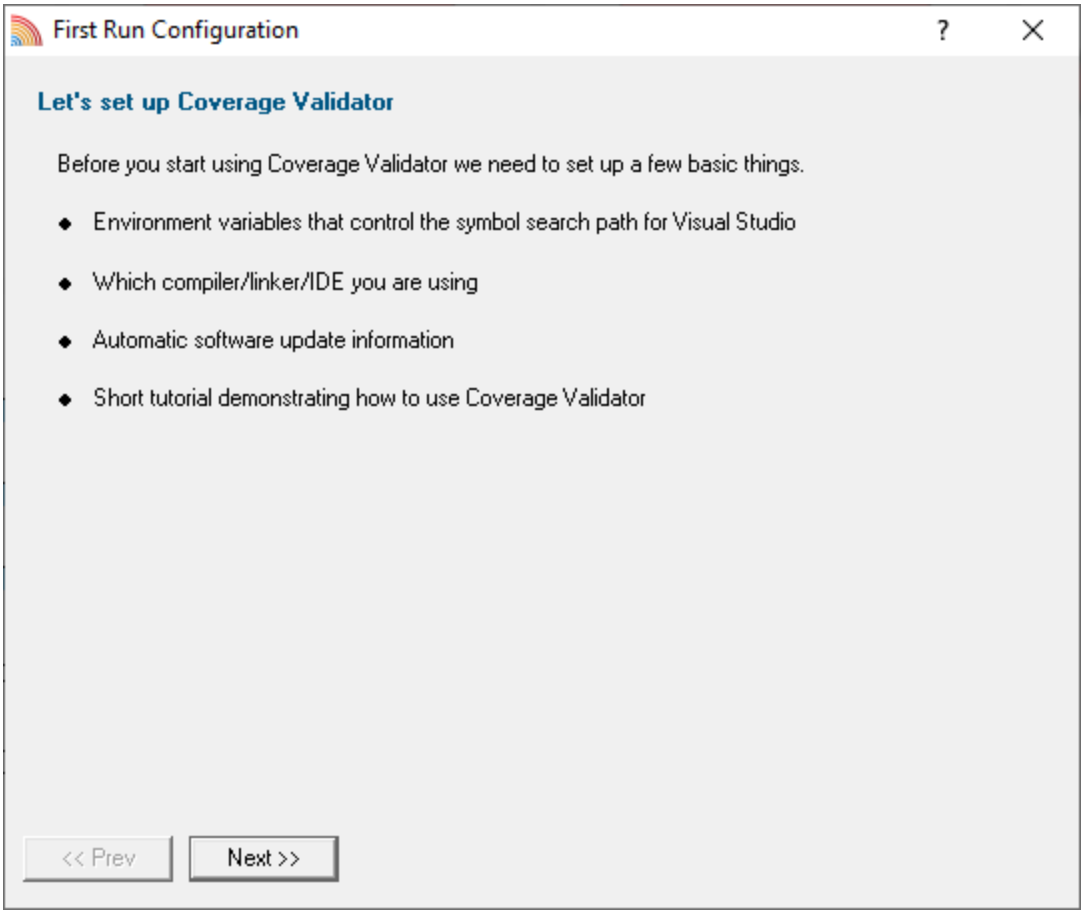


3.1 First run configuration

First run configuration

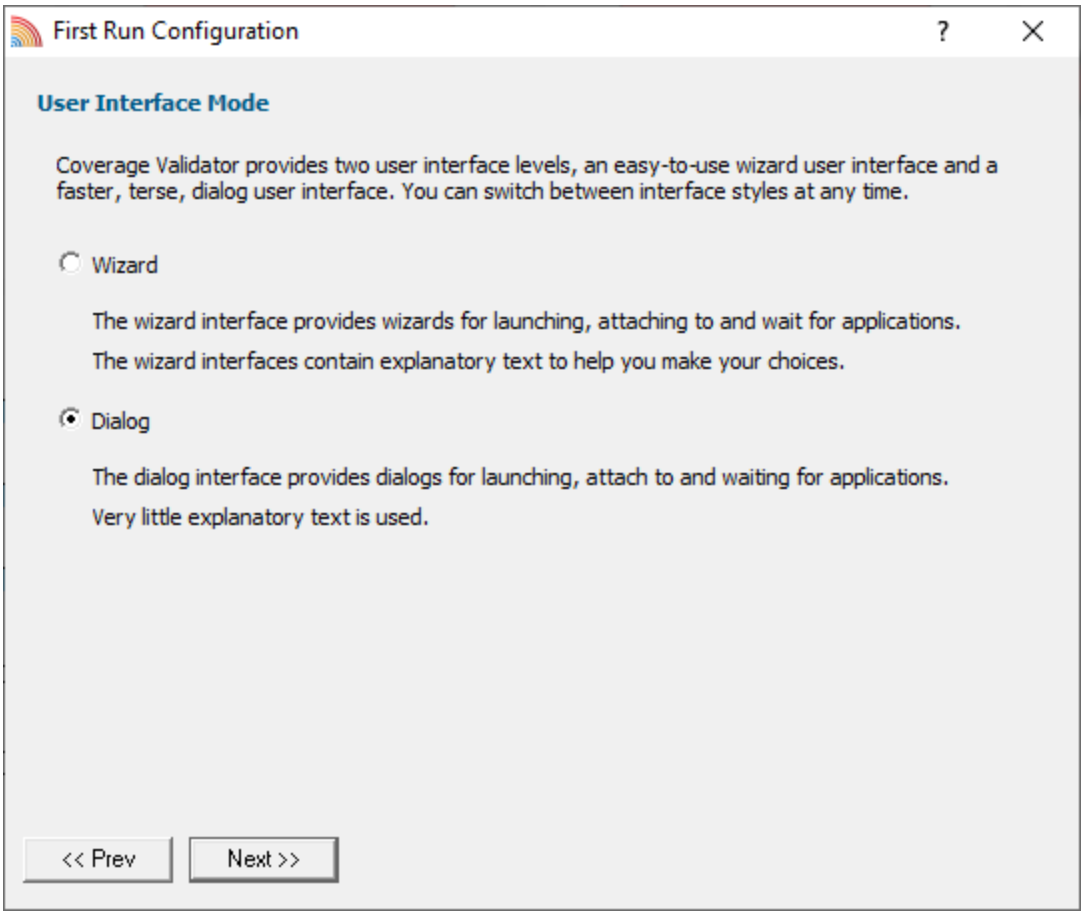
For new users of Coverage Validator, a configuration wizard appears the first time you run the application.

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



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.



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

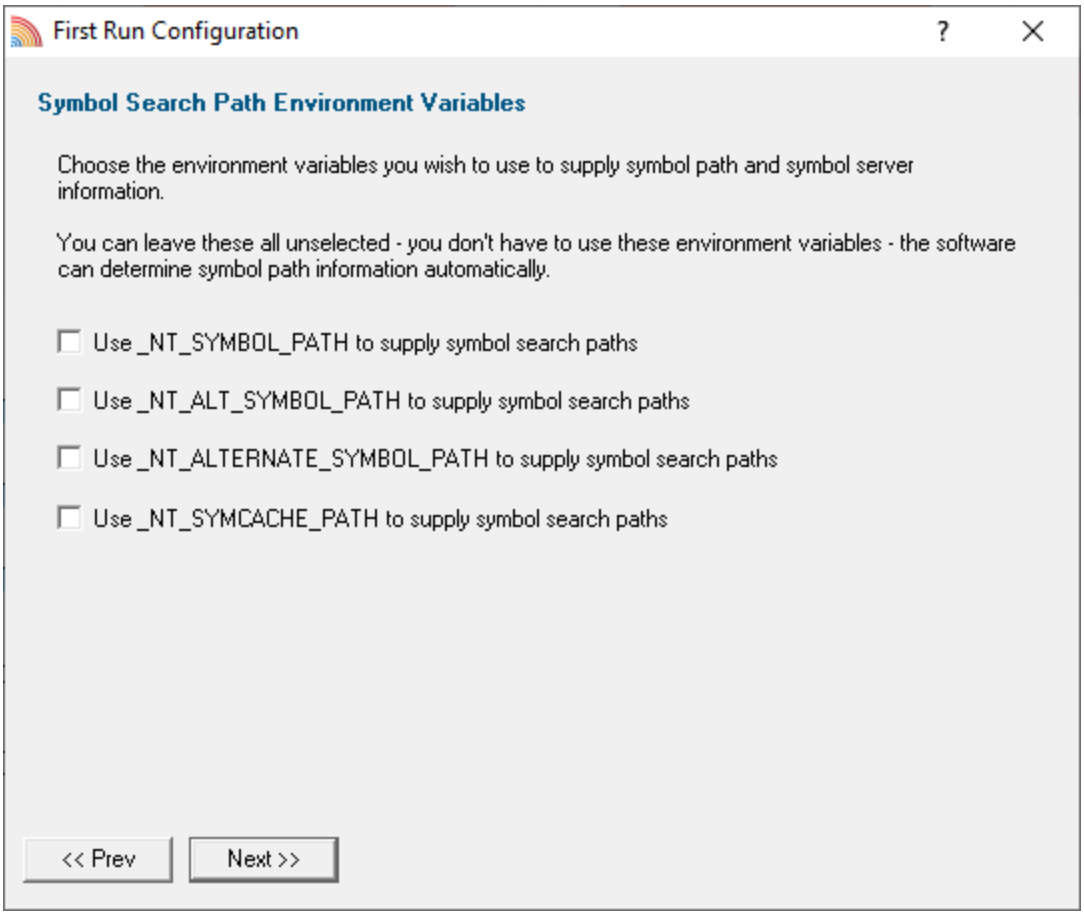
  Experienced users will find this mode quicker to use

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

Symbol search path environment variables

The next page of 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 Coverage Validator will try to automatically determine symbol path information.



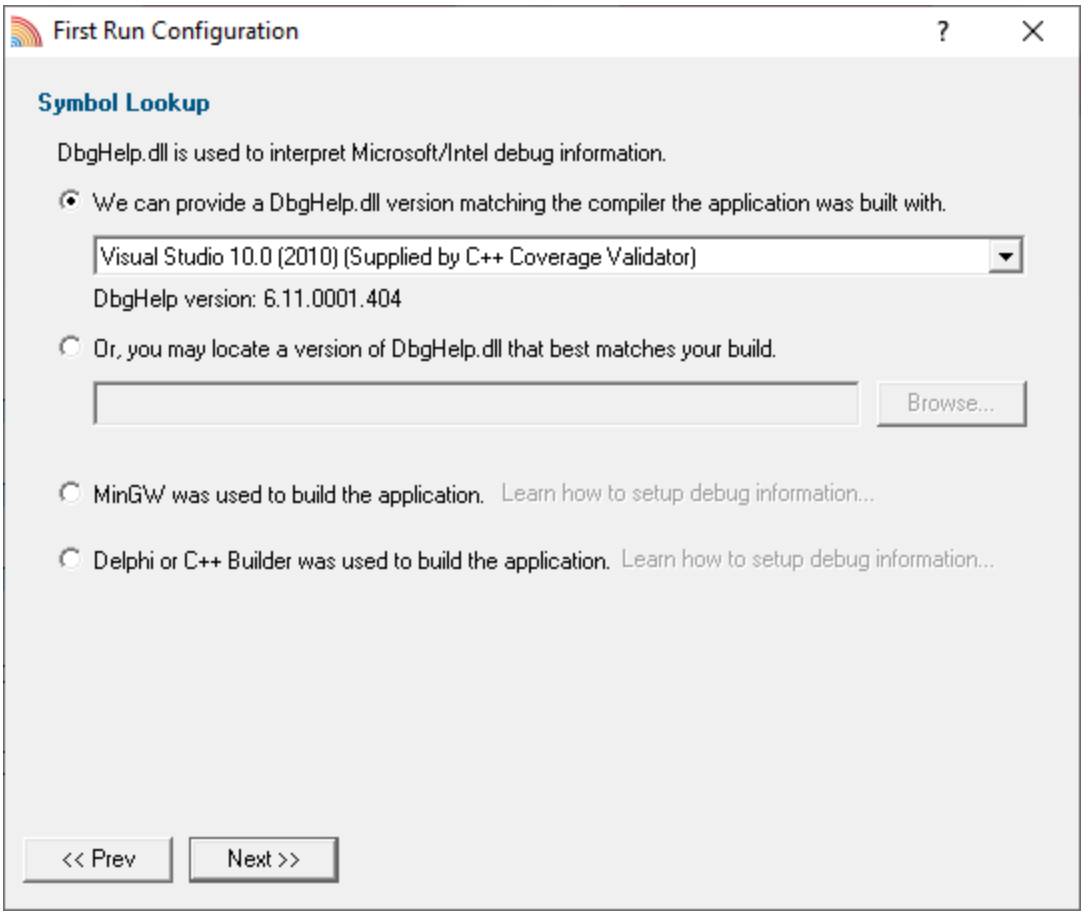
These environment settings can be changed at any time via the Configure Symbol Handling Environment Variables on the Symbol Server page of the Settings Dialog.

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.

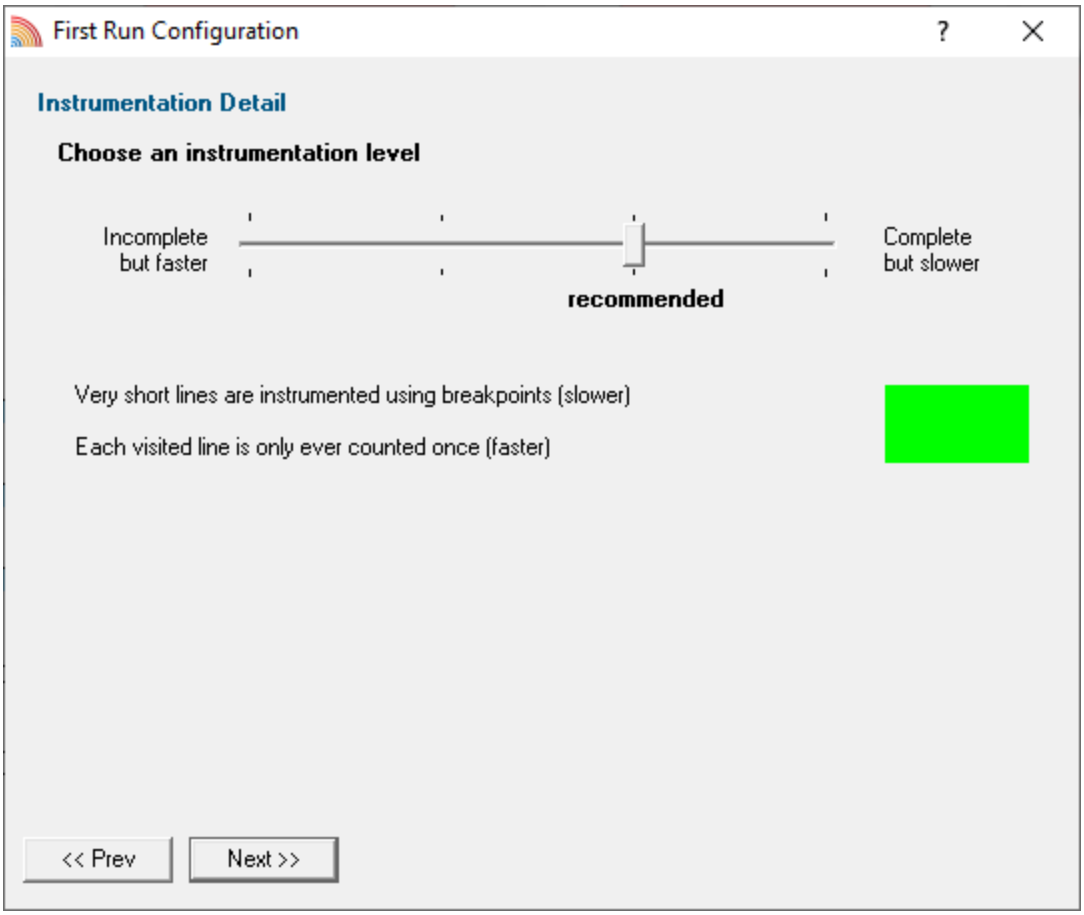
The default settings are shown below, although the Visual Studio version may vary.



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

Instrumentation details

The Instrumentation detail section of the wizard allows you to specify how lines are hooked and how many times a line visit is counted.



Two instrumentation settings affect how quickly the instrumented program will run:

- the hooking of very short lines

  Most lines can be monitored using a normal line hooking technology. These hooks are fast but cannot be used for all lines.

  Very short lines are hooked using breakpoint hooking technology which is very advanced but also incurs a serious performance penalty for each visit to that line.

- the counting of lines on every visit

  Counting lines just once is quicker than doing it on every visit.

The instrumentation level lets you balance the detail of collected visit counts for every line against speed of execution.

- **Incomplete but faster** › Short lines NOT counted. Lines counted once
- **Incomplete but slower** › Short lines NOT counted. Lines counted every visit.
- **Complete but faster** › Short lines counted. Lines counted once. (This is the recommended option)
- **Complete but slower** › Short lines counted. Lines counted every visit.

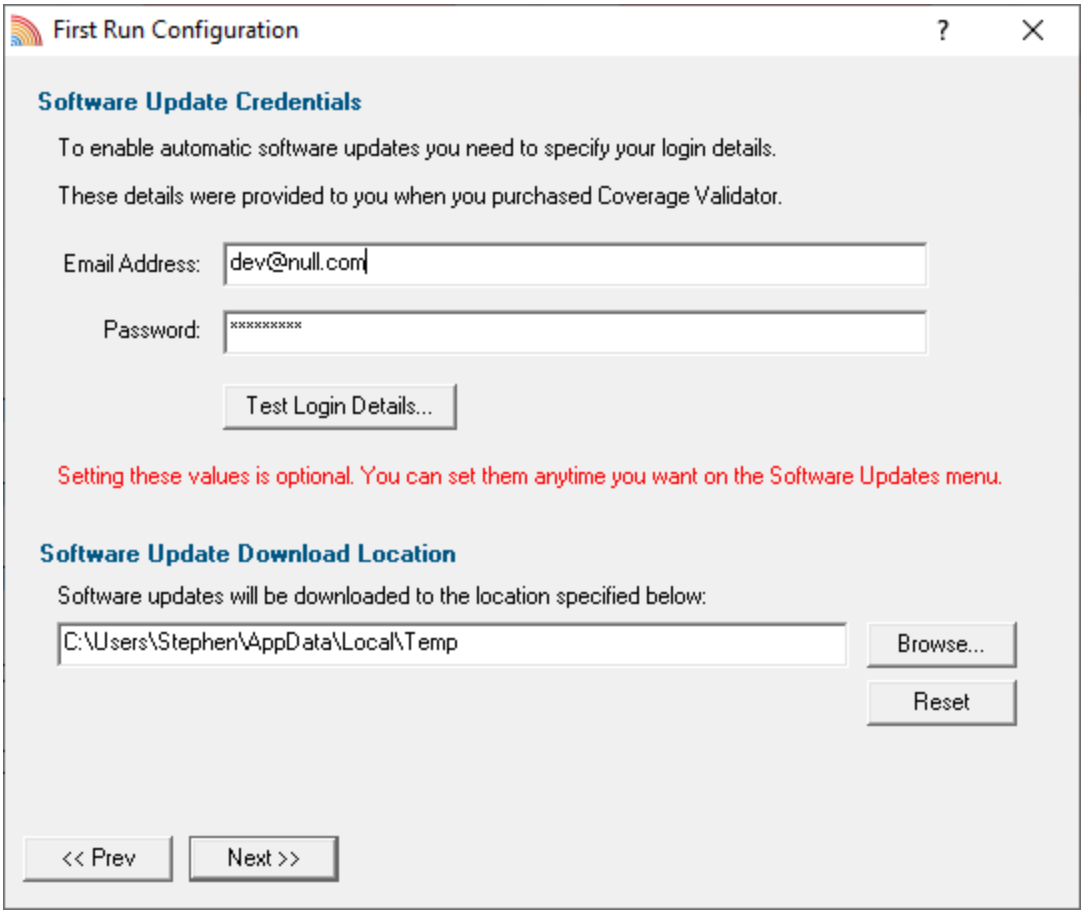
➜ See more about very short lines, and related caveats.

These instrumentation settings can be changed at any time on the Instrumentation Detail page of the Settings Dialog.

Software update credentials

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

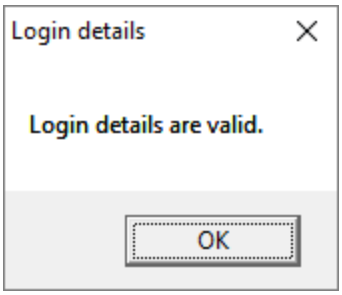
You can configure your software update credentials within the application and where updates are downloaded to.



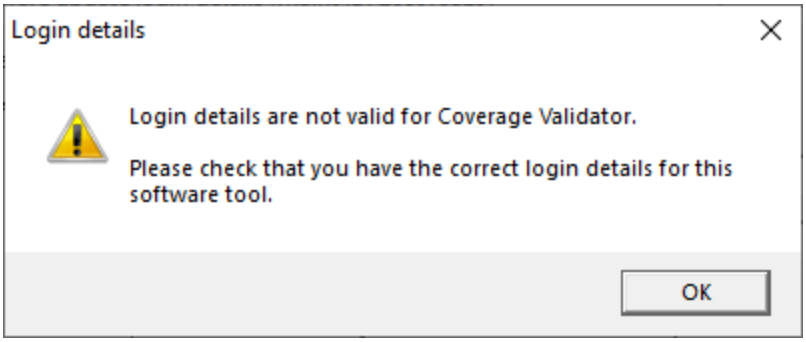
You can test the login details to ensure they are valid:

- **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.



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

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

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

Software update download location

It's important to be able to specify where software updates are downloaded to because of potential security risks that may arise from allowing the `TMP` directory to be executable.

We use the `TMP` directory as a default, but if you're not comfortable with that you can specify your preferred download directory. This allows you to set permissions for `TMP` to deny execute privileges if you wish.

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

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

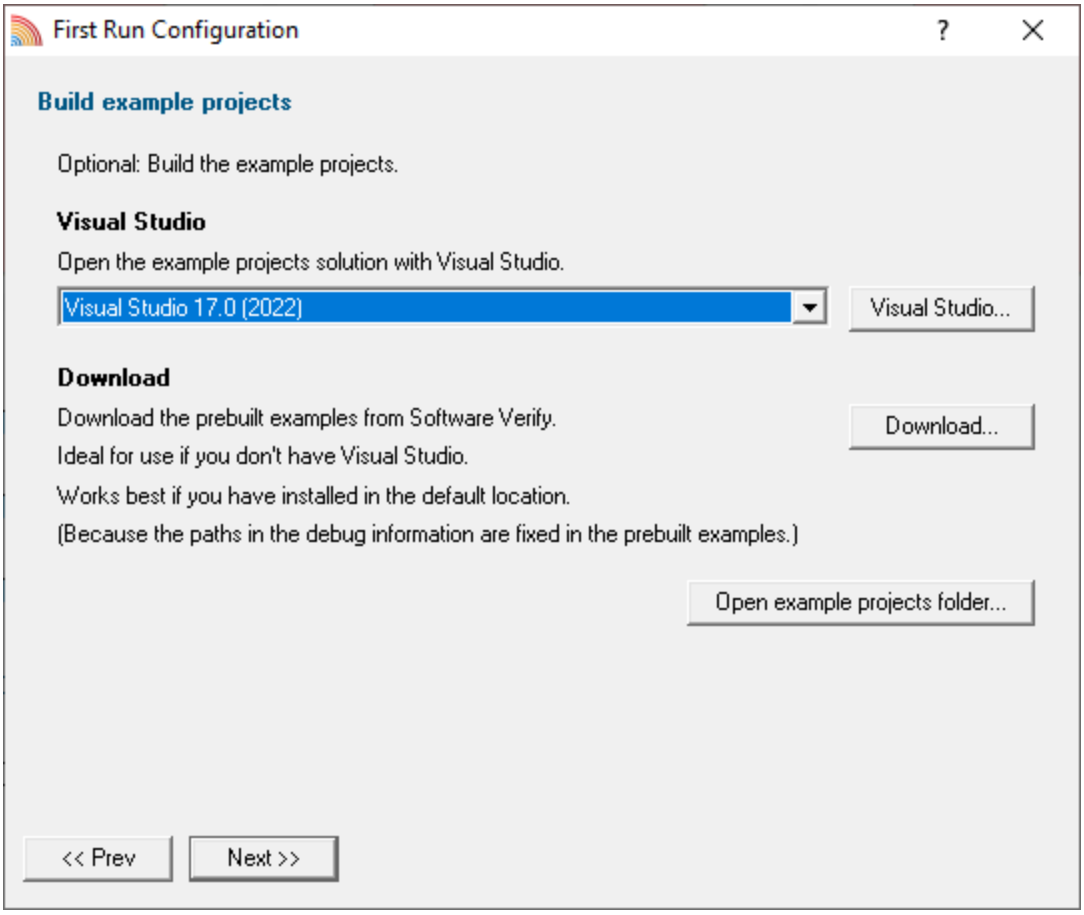
  The default location is `c:\users\[username]\AppData\Local\Temp`

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

Build example projects

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

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



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

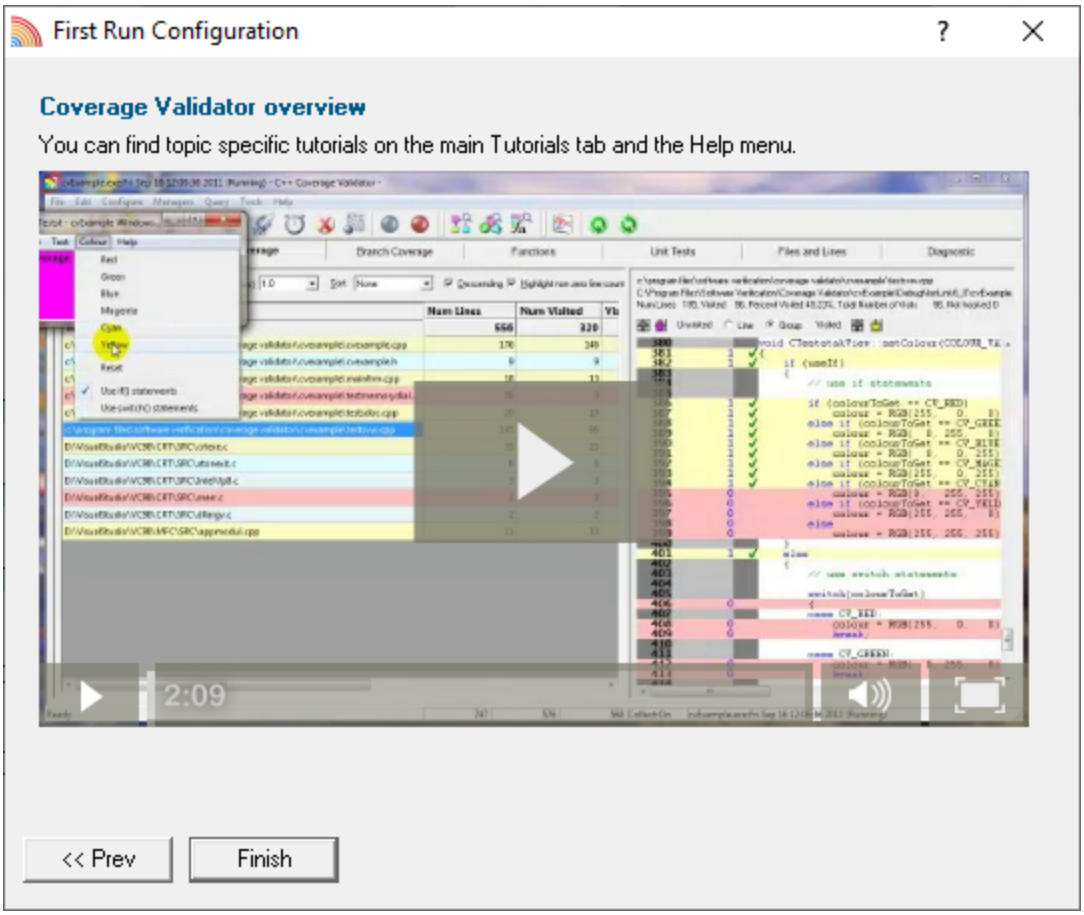
  If you choose this option and you have not installed Coverage Validator in the default location (assuming a 64 bit OS) the source file paths in the debug information will be incorrect - you will need to use the File Locations settings to inform Coverage Validator of the correct location(s).

- **Open example projects folder...** › opens the folder that contains all example projects

Video overview of application

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

The video has audio



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 Coverage Validator.

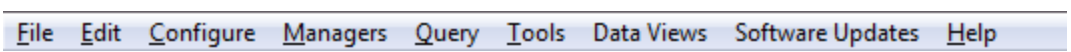
- **Finish** › closes the First Run Configuration dialog leaving the application ready to use

3.2 Menu Reference

The menus provide access to all the major features in Coverage 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.

Click in the picture below to jump to any menu's page:



3.2.1 File menu

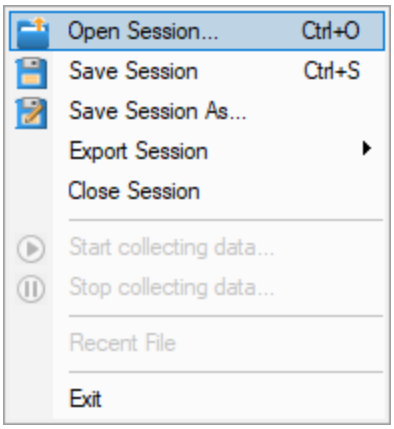
The **File** menu allows you to:

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

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

Near the bottom of the menu, a list of recently used file names 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.

3.2.2 Launch menu

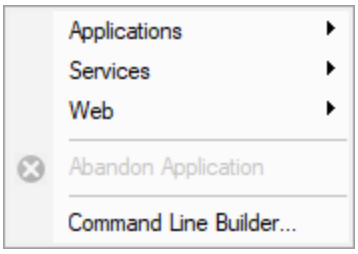
The **Launch** menu allows you to:

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

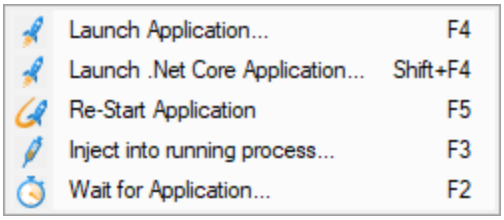
Most of these actions are also available via the standard or session toolbars.

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

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



Applications



Services

Web

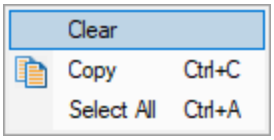
In addition to the function key short cuts shown above, you can redisplay the previously chosen launch dialog by using Ctrl + F4

3.2.3 Edit menu

Selections and the clipboard

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, or select all the items available.

Selected data is formatted into one line per row with a single space used to separate column data.



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 Coverage Tab might show:

File	% Visit...	Num Lines	Num Visited	Visit Count	DLL
Totals	17.91%	201	36	36	
nativeexample.cpp	57.89%	19	11	11	C:\Program Files (x86)\Software Verify\Cove
mainfrm.cpp	87.50%	8	7	7	C:\Program Files (x86)\Software Verify\Cove
testmemorydialog.cpp	0.00%	58	0	0	C:\Program Files (x86)\Software Verify\Cove
testsdoc.cpp	50.00%	6	3	3	C:\Program Files (x86)\Software Verify\Cove
testsvw.cpp	13.64%	110	15	15	C:\Program Files (x86)\Software Verify\Cove

This would result in the following being copied to clipboard:

```
File   % Visited     Num Lines     Num Visited   Visit Count   DLL
Totals 17.91% 201    36     36
nativeexample.cpp    57.89% 19     11     11     C:\Program Files (x86)\Software Verify\Cove
mainfrm.cpp   87.50% 8      7      7      C:\Program Files (x86)\Software Verify\Coverage Va
testmemorydialog.cpp 0.00%  58     0      0      C:\Program Files (x86)\Software Verify\Cove
testsdoc.cpp  50.00% 6      3      3      C:\Program Files (x86)\Software Verify\Coverage Va
testsvw.cpp   13.64% 110    15     15     C:\Program Files (x86)\Software Verify\Coverage Va
```

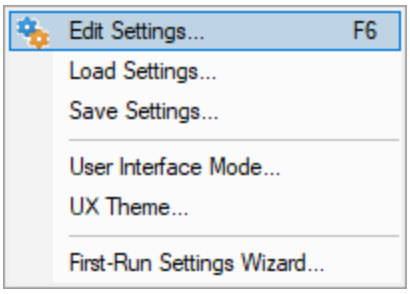
3.2.4 Settings menu

The **Settings** menu allows you to:

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

Global settings are also accessible via the session toolbar.

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

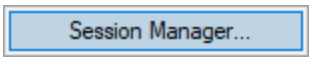


3.2.5 Managers menu

Managers

The **Managers** menu provides just one lonely (but powerful) tool to manage sessions including comparing and merging current or previous data.

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



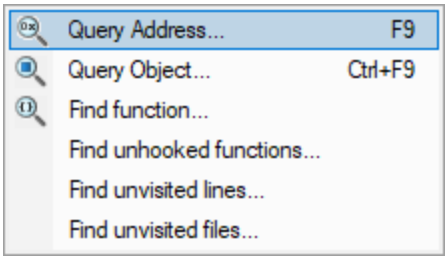
3.2.6 Query menu

Query

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

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

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



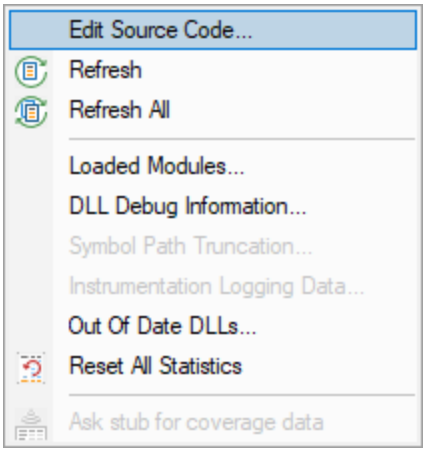
3.2.7 Tools menu

Tools

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
- A log of files, classes, functions, methods, or modules not instrumented, and reasons why not

Click on a menu item in the picture of the Tools Menu below to find out more:



3.2.8 Data Views menu

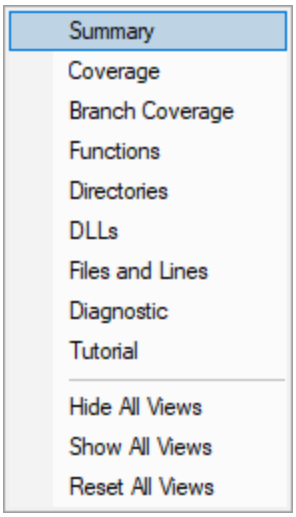
Data Views

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

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

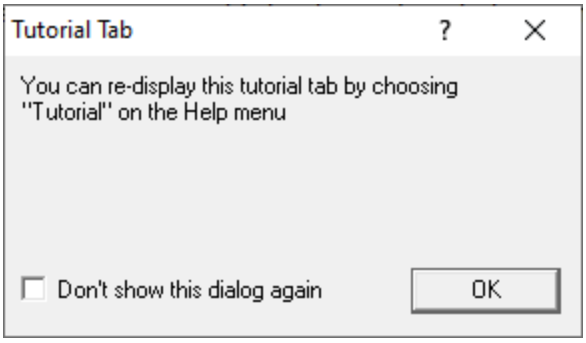
- **Hide All Views** › hides all tabs *except* the one that's currently visible
- **Show All Views** › shows *all* the listed tabs, and in their normal order
- **Reset All Views** › shows only the most *popular* tabs, so excludes the Unit Tests, and the Files and Lines tabs

  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.

You can choose not to keep seeing this reminder.



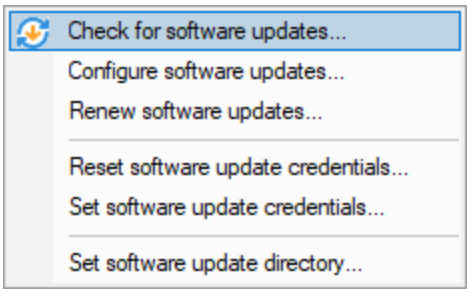
The Summary tab will always remain shown.

Hidden views are remembered between sessions.

3.2.9 Software Updates menu

Software Updates

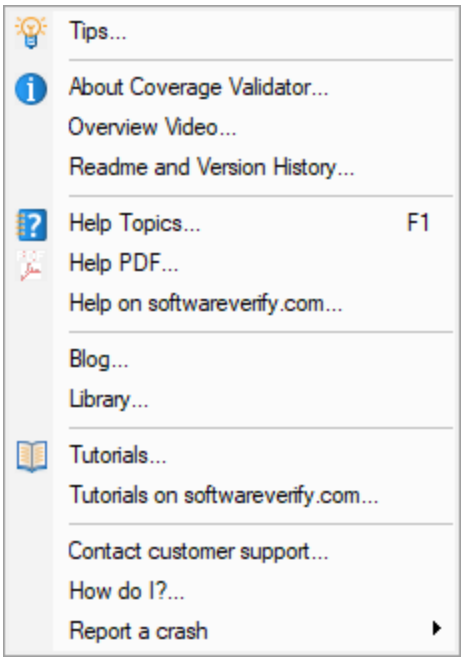
All six items in this menu are covered in the Software Updates topic.



3.2.10 Help menu

Help Menu

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 too!

3.3 Toolbar Reference

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

The items are listed in left to right order.

Click on any part of the pictures below to jump straight to the topic:

Standard toolbar

- Load session
- Save session
- Help

Session toolbar

- Settings
- Launch application using the launch chooser
- Relaunch the previously launched application
- Inject into application
- Wait for application to start
- Stop application
- Ask stub for coverage data
- Enable collecting data
- Disable collecting data

Query

- Query address
- Query object
- Find function

Tools

- Reset statistics
- 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
- data collection statistics
- program information

The message line

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

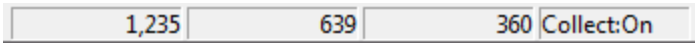
When you hover the mouse over a toolbar button or a menu item for a short time, a help message appears in the status bar describing the button's action.

Data collection statistics

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

This collection data has three counters and a collection status:

- Data items sent from stub that have been processed
- Symbols that have been processed
- Line coverage data entries
- Status indicating whether collection is currently on or off



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

The value increased

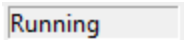
The value decreased

Coverage status

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

- **Ready**. Waiting to start a run, or a run has finished and is waiting for you to analyze the data.

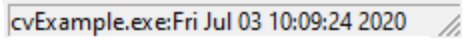
- **Starting**. Starting a run (hooks being installed etc).
- **Running**. Target executable is hooked and running.
- **Terminating**. Target executable has entered ExitProcess but has not yet finished executing.
- **Post Processing**. Target executable has finished executing. There is data that still needs to be processed.



Target program name

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

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



3.5 Keyboard Shortcuts

Keyboard shortcuts

The following shortcuts are available. Note the useful **F1** for contextual help.

Ctrl + A Select All

Ctrl + C Copy

Ctrl + F Find (file or function)

Ctrl + G Goto line (Source code views)

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 (Native / .Net)

Shift + F4 Start application (.Net Core)

F5 Restart application

F6 Monitor a Service

F7 Monitor IIS and ISAPI

F8 Monitor IIS and ASP.Net

F9 Monitor Web Development Server and ASP.Net

Ctrl + F4 Redisplay the previously chosen launch dialog.

Also

F7 and Shift + F7 navigate forwards and backwards through *unvisited* lines when source code has focus.

F8 and Shift + F8 navigate forwards and backwards through the *visited* lines of the source code.

3.6 Icons

Icons used in the main 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.

- Option enabled
- Option disabled
- Source code line indicator
- Source code
- Line has been visited

✕ Line could not be hooked

3.7 The main display

The tab windows

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

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

Typical usage might be to use the Summary or Coverage tab to monitor high level coverage in the target program, and then use another view to gain insights at a different granularity.

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

Hiding and showing tabs

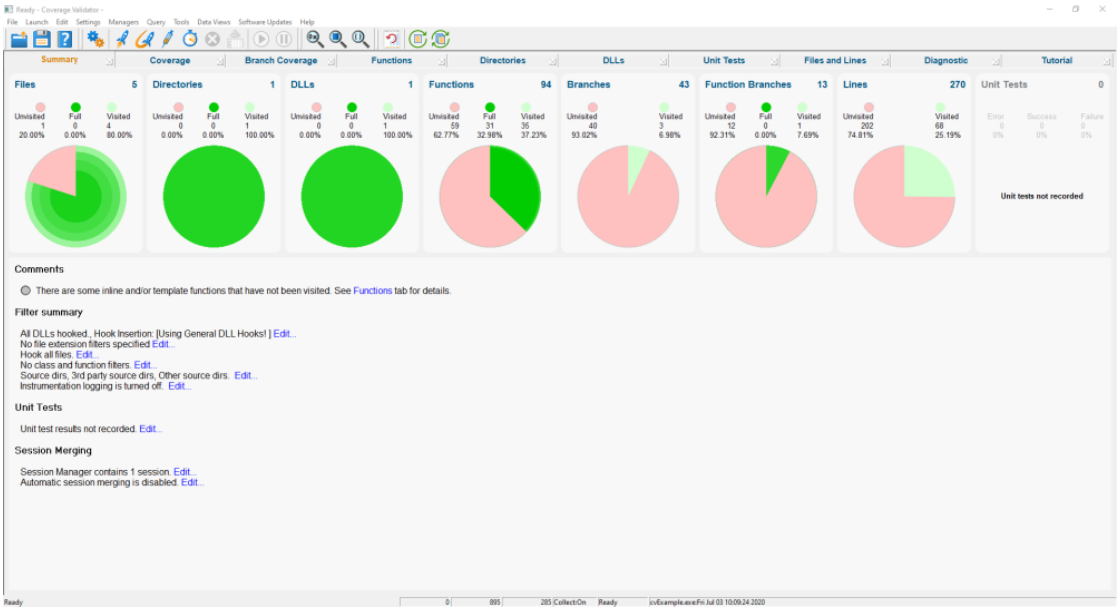
Each tabbed window can be closed by clicking the small [x] on the right hand side of the tab. The window can be redisplayed from the Data Views menu.

Icons

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

3.7.1 Summary

The **Summary** tab view displays a dashboard of high level information about the coverage of the current application.



The summary tab

The display shows various coverage statistics with percentages where appropriate.

Each dial summarises coverage of the different types of information found in the main tabs.

- Files (Coverage)
- Directories
- DLLs
- Functions
- Branches and Function Branches
- Lines

Clicking on the dials takes you to the corresponding main tab to explore in more detail.

Branches gets two dials, one for just branches (**Branches**) and one for functions that contain branches (**Function Branches**)

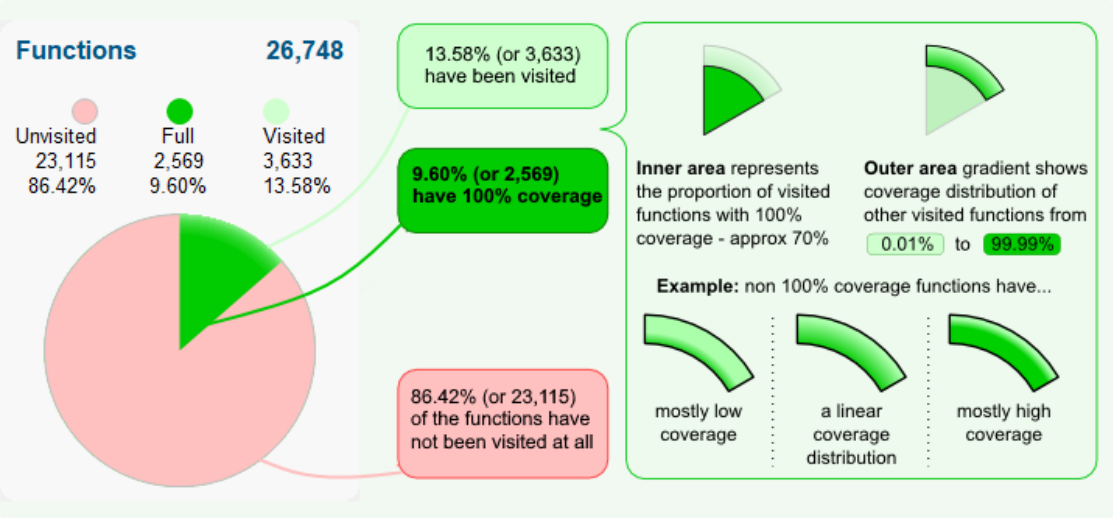
Understanding the dials

Each 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 data on a total of 26,748 known functions 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 filters, unit tests, 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.

Filter summary:

- DLLs Hooked — **Edit...** › Hooked DDLs settings
- File extension filters — **Edit...** › Hooked Source File Types settings
- File hooks — **Edit...** › Source Files Filter settings
- Class and function filters — **Edit...** › Class and Function Filter settings
- File location filters — **Edit...** › File Locations settings
- Instrumentation logging — **Edit...** › Instrumentation Logging settings (If logging is off)
  View... › Instrumentation Log dialog (If logging is on)

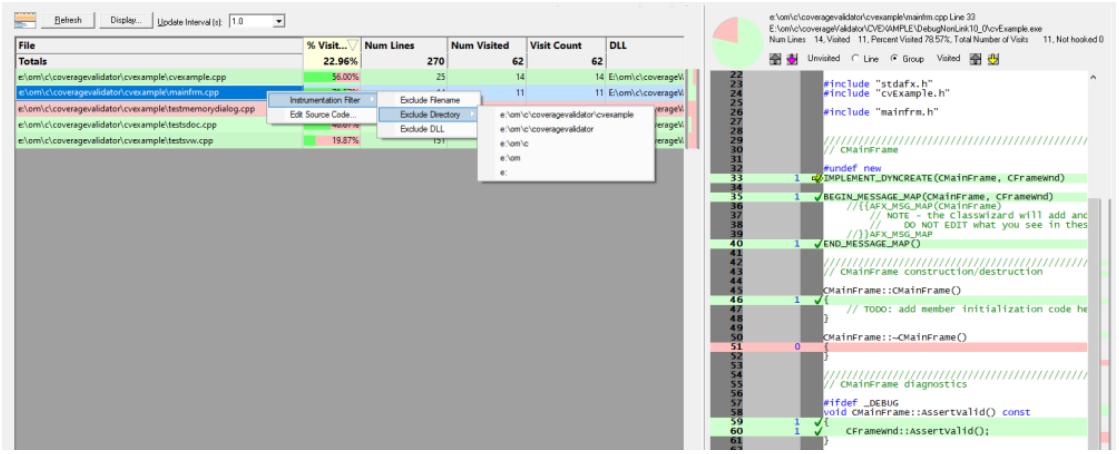
Session merging:

- Session manager status — **Edit...** › Session Chooser dialog
- Session merging status — **Edit...** › Auto Merge settings

3.7.2 Coverage

The **Coverage** tab view displays file coverage and gives a good overall picture of the coverage.

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



As with many of the tab views, the display is split into two resizable panes.

- the left side lists a summary of the file coverage data
- the right side shows source code for any file selected on the left

We'll cover the file list and its popup menu first, and then the source code view.

The file list

At the top of the list is a summary of the total current file coverage statistics.

File	% Visited	Num Lines	Num Visited	Visit Count	DLL
Totals	54.52%	332	181	63,062	

Each subsequent line in the list of files displays the following:

- the file name - either the full path or the short name
- the percentage of hookable lines that have been visited
- the number of hooked lines in the file, (unhooked lines in brackets)

This is not the same as the total number of lines in a file, which may include comments, declarations and white space.

Neither is it the same as the number of *hookable* lines since some lines or functions can be excluded from hooking.

The count also includes lines which should have been hooked but failed, for example because the instrumentation level was set low. The unhooked line count is shown in brackets.

- the number of *different* lines that have been visited
- the total visit count for the file

  This may be equal to the number of different lines visited unless you're counting each visit separately as seen below.

File	% Visited	Num Lines	Num Visited	Visit Count	DLL
Totals	**54.52%**	**332**	**181**	**63,062**	
testsvw.cpp	76.24%	181	138	31,267	C:\Program Files (x86)\Software V
mainfrm.cpp	73.68%	19	14	18,629	C:\Program Files (x86)\Software V
testsdoc.cpp	57.14%	21	12	13,086	C:\Program Files (x86)\Software V
cvexample.cpp	53.13%	32	17	80	C:\Program Files (x86)\Software V
testmemorydialog.cpp	0.00%	79	0	0	C:\Program Files (x86)\Software V

- the DLL in which the file was found

Files are ordered first by their directory and then by the files in those directories.

Line colours

Each line in the list is colour coded to indicate one of the following:

- the file has not been visited at all

testmemorydialog.cpp	0.00%	79	0	0	C:\Program Files (x

- the file has been visited to some extent

testsdoc.cpp	57.14%	21	12	12	C:\Program Files (x

- every hookable line in the file has been visited, i.e. 100% file coverage

testsvw.cpp	100.00%	178	178	178	C:\Program Files (x

- none of the lines in the file could be hooked

qstringlist.h	0.00%	2 (2 unhooked)	0	0	D:\dev\Gradie

Unhooked lines

The number of unhooked lines is shown in brackets in the **Num Lines** column.

Depending on what file or DLL hook filters are set up, not all the hookable lines in a header file may attempt to be hooked.

Only those that are actually used (e.g. via macros or inlines) in the program may actually be hooked.

By default, lines that could not be hooked don't stop the coverage from reaching 100%, but you can change this.

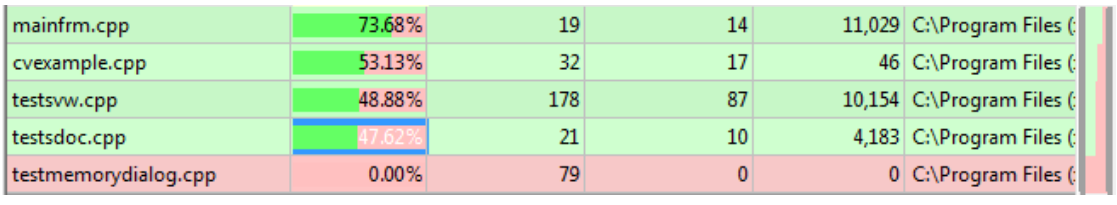
In this example below, the instrumentation level was set low (*'incomplete but faster'*) to force significant numbers of unhooked lines.

File	% Visit...	Num Lines	Num Visited	Visit Count	DLL
Totals	**41.16%**	**1,314 (194)**	**461**	**461**	
qhash.h	100.00%	64 (60 unhooked)	26	26	D:\dev\Gradients\
qscopedpointer.h	100.00%	4 (1 unhooked)	3	3	D:\dev\Gradients\
ui_gradients.h	100.00%	23 (2 unhooked)	21	21	D:\dev\Gradients\
qrc_gradients.cpp	100.00%	7 (6 unhooked)	1	1	D:\dev\Gradients\
main.cpp	100.00%	6 (1 unhooked)	5	5	D:\dev\Gradients\
qtmain_win.cpp	85.71%	8 (1 unhooked)	6	6	D:\dev\Gradients\
moc_gradients.cpp	71.43%	15 (8 unhooked)	5	5	D:\dev\Gradients\
qlist.h	40.00%	120 (25 unhooked)	38	38	D:\dev\Gradients\
gradients.cpp	37.98%	942 (39 unhooked)	343	343	D:\dev\Gradients\
qvector.h	17.81%	101 (28 unhooked)	13	13	D:\dev\Gradients\
qstringlist.h	0.00%	2 (2 unhooked)	0	0	D:\dev\Gradients\
qtextstream.h	0.00%	1 (1 unhooked)	0	0	D:\dev\Gradients\

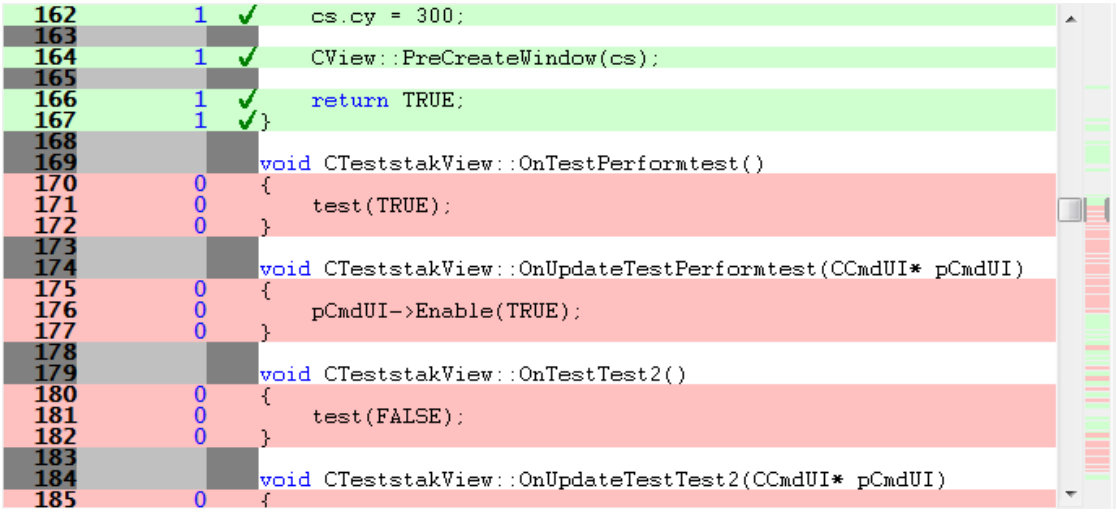
Scrollbar visualisations

To the right of each pane, beside the vertical scrollbars, you'll see a coloured area which represents the coverage.

For the file list, the visualisation shows the coverage of each file in the list - reflecting the progress meters in the second column.



For the source code view, the visualisation shows the coverage of each line in the file

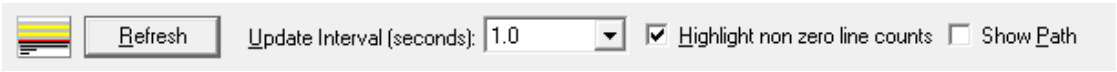


These visualisations provide a snapshot view of the whole list of files or the whole source file.

Irrespective of how much you can see in the file list or in the source code view, you can still get a global overview of the coverage distribution.

Coverage options

The **coverage** controls are shown below.



Window orientation

The horizontal or vertical orientation of the statistics and source code panes can be toggled with the orientation button.

Updating the display

- **Update Interval (seconds)** › 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.

  Only the Coverage view has automatic update intervals controlled in this way.

- **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.

Display settings

- **Highlight unvisited lines** › highlights rows for files and lines that have not been visited (on by default)

  If switched off, unvisited lines appear white - or whatever colour you've set as the *unselected* colour.

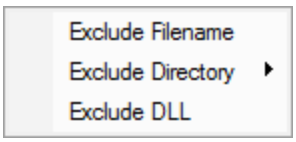
- **Show Path** › shows the short file name or the longer file path in the **File** column of the data

File list menu options

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

Menu options: Instrumentation Filter

The instrumentation filter lets you add filters at different levels of granularity:



- **Exclude Filename** › adds a new filter to the Source Files Filter settings, excluding it from the results of subsequent sessions
- **Exclude Directory** › excludes all files in the same directory as the selected file

  From the sub menu, choose the directory level at which you want to exclude files, right up to the drive specifier if you need to.

- **Exclude DLL** › excludes all files belonging to the same executable or DLL as the selected file

  This adds a filter to the Hooked DLLs settings.

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

Menu option: editing source code

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

The file source code view

Clicking on a file in the file list, shows that file in the source code view on the right.

The source code uses syntax highlighting by default, with the background colour of the line indicating if the line has been visited, is unvisited or could not be hooked.

Icons are displayed next to hooked and unhooked lines indicating visit and hook status and visit counts are shown in-line with the code.

Hovering over a line for a short period of time shows a tooltip with the number of visits to the line.

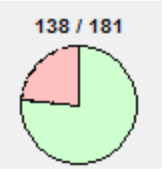
Contiguous groups of lines can be collapsed and hidden from view.

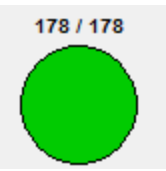
Source code file information

On the right hand panel, above the source code, you'll find some information about the source file

The details shown include the following:

- 'quick view' details for visited and unvisited lines, pink for unvisited, light green for partial visited, and dark green for 100% visited





- the source code filename and the executable or DLL to which it belongs

  c:\program files (x86)\software verification\c++ coverage validator\cvexample\testsvw.cpp
  C:\Program Files (x86)\Software Verification\C++ Coverage Validator\cvExample\DebugNonLinkANSI9_0\cvExample.exe

- the same file statistics as seen in the left hand panel

  Num Lines 181, Visited 138, Percent Visited 76.24%, Total Number of Visits 31665, Not hooked 0

Browsing visited and unvisited lines

-  › show the previous and next *unvisited* line of code

  F7 and Shift + F7 also navigate forwards and backwards when the source code has focus.

-  › show previous and next *visited* line of code

  The arrows are grey when disabled.

  F8 and Shift + F8 also navigate forwards and backwards through the visited lines.

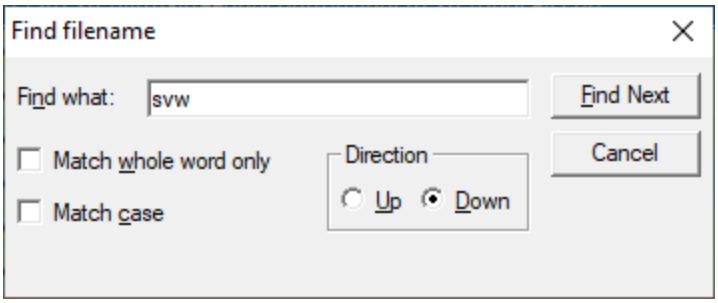
- **Line** › step by individual lines of code
- **Group** › step by groups of contiguous visited or unvisited lines

Keyboard access: Find and Goto

When the data view or the source code view has focus, some keyboard access is available to search for files and other text, or to navigate to numbered lines.

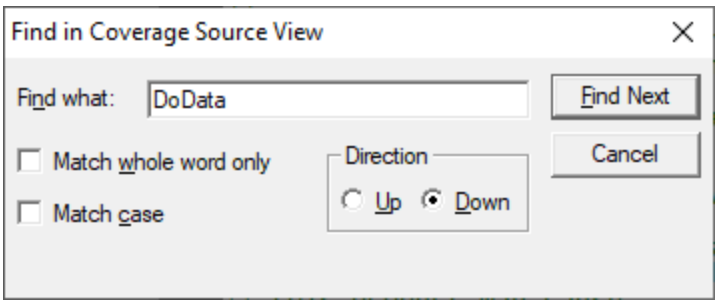
Find filename in data view

In the data view, Ctrl + F displays a dialog that will allow you to search by full or partial match for a filename.



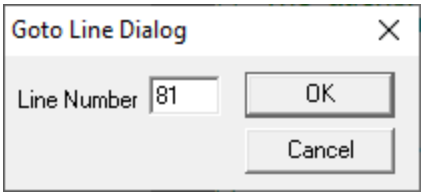
Find text in source view

In the source code view Ctrl + F lets you search by full or partial match anywhere in the file.



Goto line in source view

In the source code view, Ctrl + G displays a goto-line dialog.

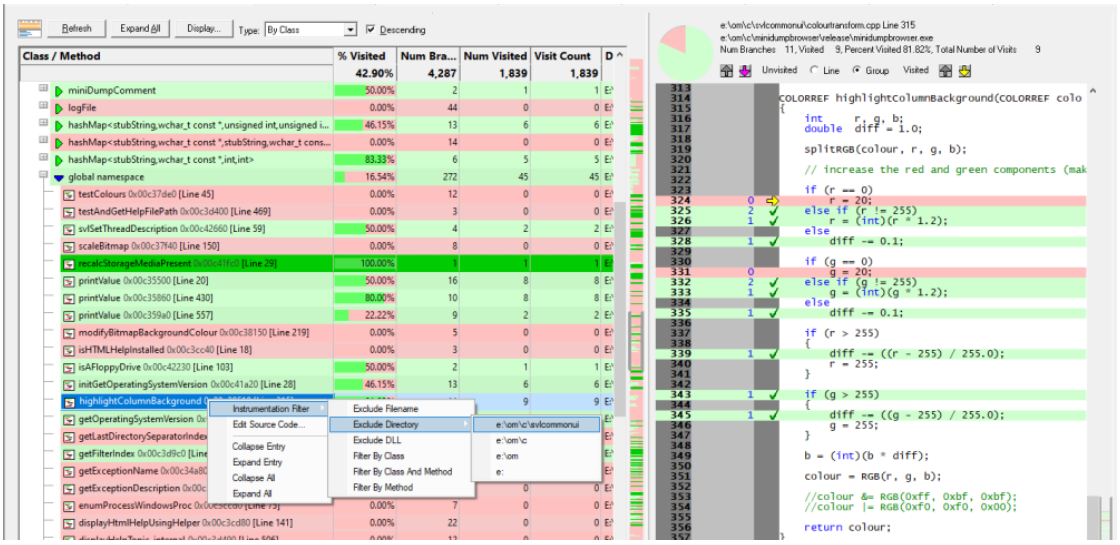


.

3.7.3 Branch Coverage

The **Branch Coverage** tab view displays branch coverage information for each function that contains branches.

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



Branches

The branch coverage focuses only on those areas of code which are conditionally executed.

This includes `if` / `else` statements and conditional loops such as `for` and `while`.

When viewing the source code, only the first line in each branch is marked. This is because any subsequent lines *in the same code block* must also be executed as part of the same branch.

As with many of the tab views, the display is split into two resizable panes.

- the left side lists a summary of the branch coverage data
- the right side shows source code for any method selected on the left

We'll cover the data view and its popup menu first, and then the source code view.

The data view

Class / Method	% Visited	Num Branches	Num Visited	Visit Count	DLL

Each line in the view displays the following:

- the file, class, method or function, depending on the Type of the view
- the percentage of hookable branches that have been visited in the function
- the number of hooked branches in the function

  This is not the same as the number of *hookable* branches since some lines or functions can be excluded from hooking.

- the number of *different* branches that have been visited in the function
- the total visit count for all the branches in the function

  This may be equal to the number of different branches visited unless you're counting each visit separately.

- the DLL in which the method was found

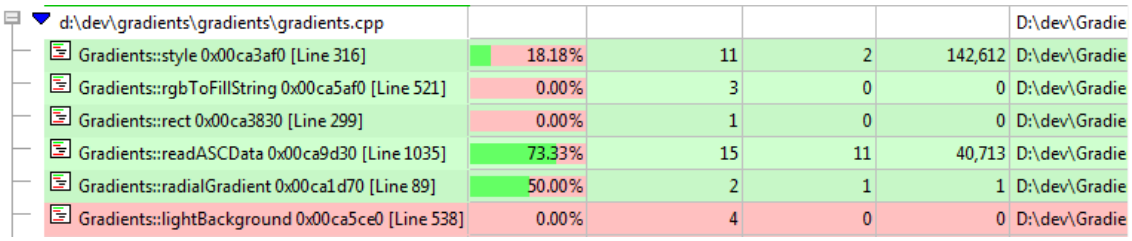
The Branch Coverage can show a hierarchical or list view of data according to the Type setting.

Hierarchical views

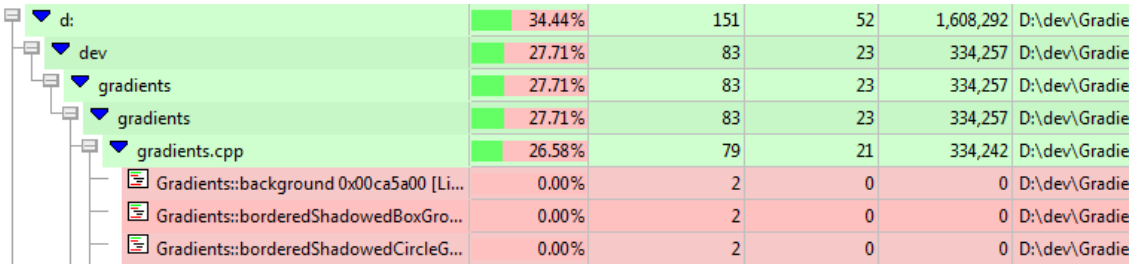
A hierarchical view is shown when the Type is set to one of the following:

- class / method
- file / function (the default)
- directory structure / file / function

Examples:



d:\dev\gradients\gradients\gradients.cpp					D:\dev\Gradie
Gradients::style 0x00ca3af0 [Line 316]	18.18%	11	2	142,612	D:\dev\Gradie
Gradients::rgbToFillString 0x00ca5af0 [Line 521]	0.00%	3	0	0	D:\dev\Gradie
Gradients::rect 0x00ca3830 [Line 299]	0.00%	1	0	0	D:\dev\Gradie
Gradients::readASCData 0x00ca9d30 [Line 1035]	73.33%	15	11	40,713	D:\dev\Gradie
Gradients::radialGradient 0x00ca1d70 [Line 89]	50.00%	2	1	1	D:\dev\Gradie
Gradients::lightBackground 0x00ca5ce0 [Line 538]	0.00%	4	0	0	D:\dev\Gradie



d:	34.44%	151	52	1,608,292	D:\dev\Gradie
dev	27.71%	83	23	334,257	D:\dev\Gradie
gradients	27.71%	83	23	334,257	D:\dev\Gradie
gradients	27.71%	83	23	334,257	D:\dev\Gradie
gradients.cpp	26.58%	79	21	334,242	D:\dev\Gradie
Gradients::background 0x00ca5a00 [Li...	0.00%	2	0	0	D:\dev\Gradie
Gradients::borderedShadowedBoxGro...	0.00%	2	0	0	D:\dev\Gradie
Gradients::borderedShadowedCircleG...	0.00%	2	0	0	D:\dev\Gradie

 For class and directory hierarchical views, the statistics against each directory or class is an aggregated sum of all the contained branches

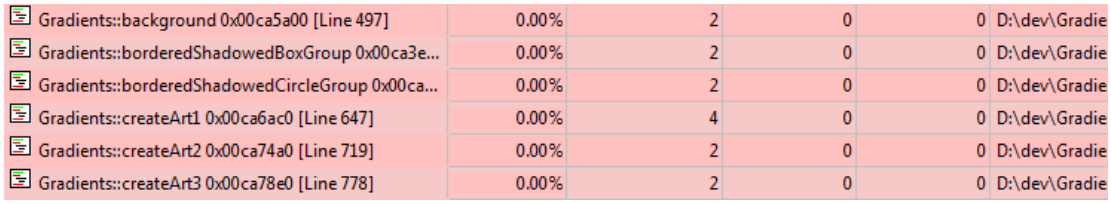
List views

A hierarchical view is shown when the Type is set to one of the following:

- method or function

- class or method (ordered by number of lines in file)

  Example:



Only the first column in the view can be used for sorting the data.

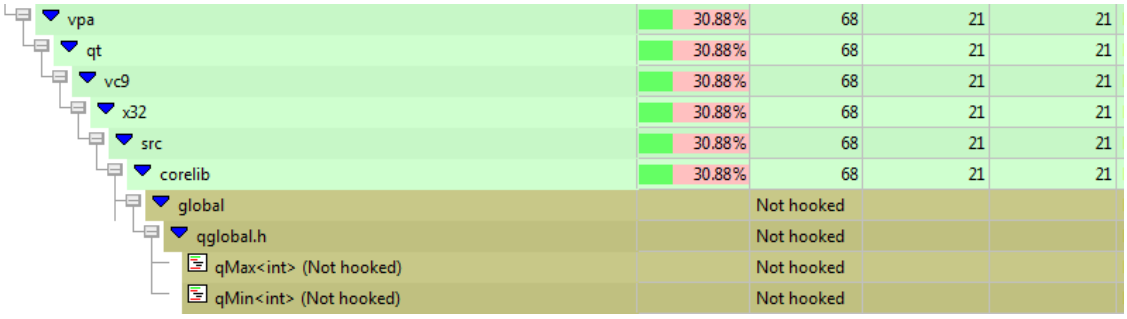
Unhooked lines

Depending on what file or DLL hook filters are set up, not all the hookable lines in a header file may attempt to be hooked.

Only those that are actually used (e.g. via macros or inlines) in the program may actually be hooked.

By default, lines that could not be hooked don't stop the coverage from reaching 100%, but you can change this.

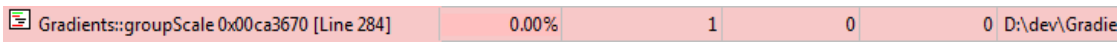
In this example below, the instrumentation level was set low (*'incomplete but faster'*) to force significant numbers of unhooked lines.



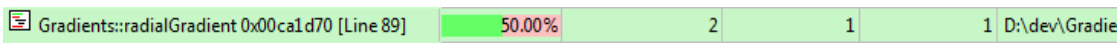
Line colours

Each line of data in any of the views is colour coded to indicate one of the following:

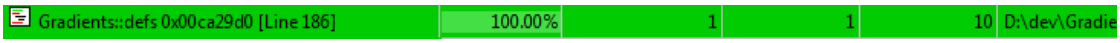
- the function has not had any branches visited at all



- some but not all branches in the function have been visited



- every hookable branch in the function has been visited, i.e. 100% file coverage



- no branches in the function could be hooked

Unhookable branches are likely when the instrumentation level is set low, for example *'incomplete but faster'*.

Scrollbar visualisations

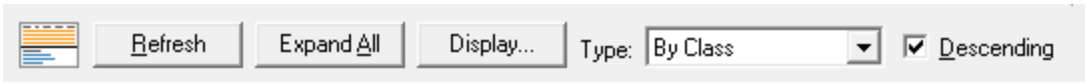
To the right of each pane, beside the vertical scrollbars, you'll see a coloured area which represents the coverage.

As with the scrollbar visualisation in the Coverage tab, the bar in the left hand pane shows an overall view of the coverage distribution across every function listed.

However, because branch coverage is broken down to the function level, selecting a function in the left hand pane will only highlight branches in the source code view *for that function*.

Branch coverage options

The **branch coverage** controls are shown below.



Window orientation

The horizontal or vertical orientation of the statistics and source code panes can be toggled with the orientation button.

Updating the display

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

View type

Changing the Type alters the view to one of the hierarchical or list views described earlier.

- **By Class** › shows a hierarchical C++ class / method view
- **By Function Name** › shows a list of all functions sorted by name
- **By Directory** › shows a hierarchical view of directory / file / function
- **By File** › shows a hierarchical file / function view
- **By Number of Lines** › shows a list of classes or methods ordered by number of lines in file

Each view shows the same overall data, just in different arrangements

Sort order

- **Descending** › reverses the direction of the data, sorted by the names in first column

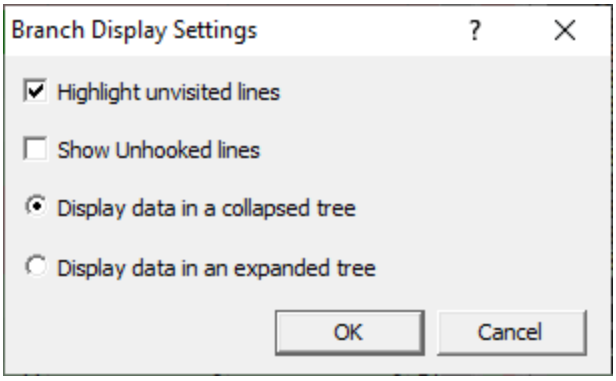
Expanding and collapsing the data

- **Expand All / Collapse All** › switches any hierarchical view between collapsed and expanded view
- **Collapsed** › sets the preferred state of the view *when the data is refreshed*

  This includes when you change some of the other display settings.

Display settings

- **Display...** › displays the branch display settings dialog



- **Highlight unvisited lines** › highlights rows for branches that have not been visited (on by default)

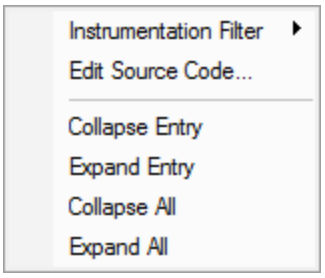
If switched off, unvisited lines appear white - or whatever colour you've set as the *unselected* colour.

- **Show unhooked functions** › display information about functions that could not be hooked (off by default)

  This means unhookable lines, for example lines that were too short, rather than lines or functions that are deliberately set not to be hooked.

File list menu options

The following popup menu is available over the data area to add filters, edit code, or expand and collapse the view.



Menu options: Instrumentation Filter

The instrumentation filter lets you add filters at different levels of granularity:

- **Exclude Filename** › adds a new filter to the Source Files Filter settings, excluding it from the results of subsequent sessions
- **Exclude Directory** › excludes all files in the same directory as the selected file

  From the sub menu, choose the directory level at which you want to exclude files, right up to the drive specifier if you need to.

- **Exclude DLL** › excludes all files belonging to the same executable or DLL as the selected file

  This adds a filter to the Hooked DLLs settings.

The other options all add a filter to the Class and function filter settings

- **Filter By Class** › excludes all functions in the selected class

  Example filter: `CTeststakDoc::`

- **Filter By Class And Method** › excludes only the selected class functions

  Example filter: `CTeststakDoc::OnNewDocument`

- **Filter By Method** › excludes only the selected method, whether in a class or the global namespace

  Example filter: `OnNewDocument`

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

Menu option: editing source code

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

Menu options: collapse / expand trace

- **Expand** or **Collapse Entry** › shows and hides the selected section in the view, the same as using the ⊞ or ⊟ buttons
- **Collapse All** › completely collapses all sections
- **Expand All** › expands all the sections

The branch source code view

Clicking on a function in the left hand panel, shows that file and function in the source code view on the right.

The source code uses syntax highlighting by default, with the background colour of the line indicating if the branch line has been visited, is unvisited or could not be hooked.

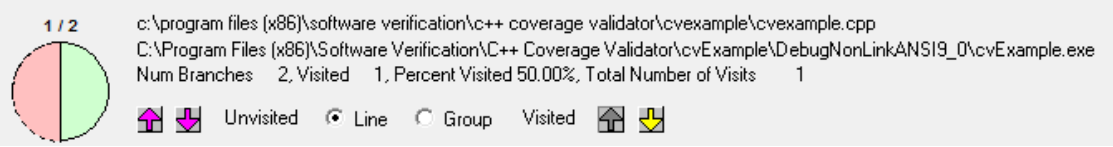
Icons are displayed next to hooked and unhooked branch lines indicating visit and hook status and visit counts are shown in-line with the code.

Hovering over a line for a short period of time shows a tooltip with the number of visits to the line.

Contiguous groups of lines can be collapsed and hidden from view.

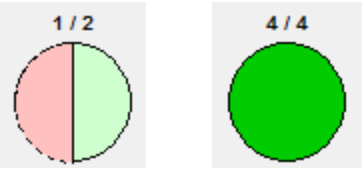
Source code branch information

On the right hand panel, above the source code, you'll find some information about branches in the selected function in the source file

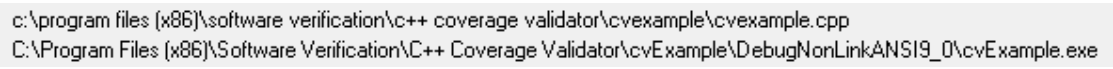


The details shown include the following:

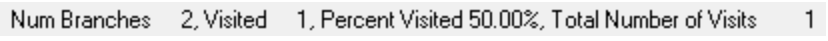
- 'quick view' details for visited and unvisited branches in the function, pink for unvisited, light green for partial visited, and dark green for 100% visited



- the source code filename and the executable or DLL to which it belongs



- the same branch statistics as seen in the left hand panel



Browsing visited and unvisited lines

-  show the previous and next *unvisited* branch in the function

F7 and Shift + F7 also navigate forwards and backwards when the source code has focus.

-  › show previous and next *visited* branch in the function

  The arrows are grey when disabled.

  F8 and Shift + F8 also navigate forwards and backwards through the visited lines.

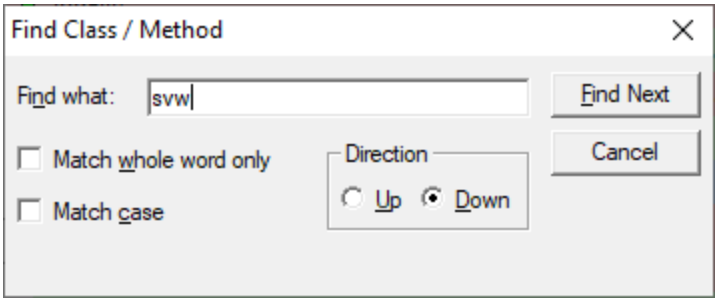
- **Line** › step by individual lines of code
- **Group** › step by groups of contiguous visited or unvisited lines

Keyboard access: Find and Goto

When the data view or the source code view has focus, some keyboard access is available to search for text, or to navigate to numbered lines.

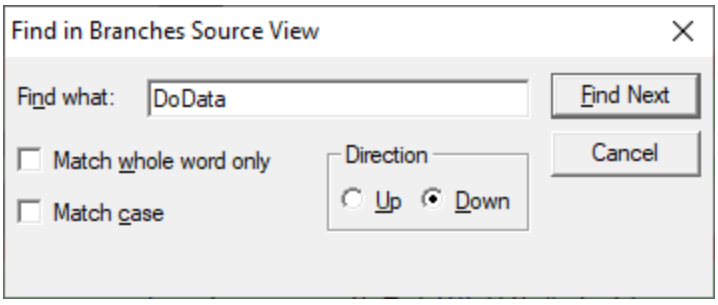
Find text in data view

In the data view, Ctrl + F displays a dialog that will allow you to search by full or partial match for text in the first column - the contents of which depends on the Type setting.



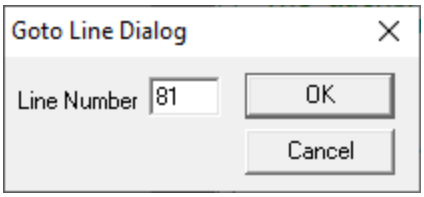
Find text in source view

In the source code view Ctrl + F lets you search by full or partial match anywhere in the file.



Goto line

In the source code view, Ctrl + G displays a goto-line dialog.

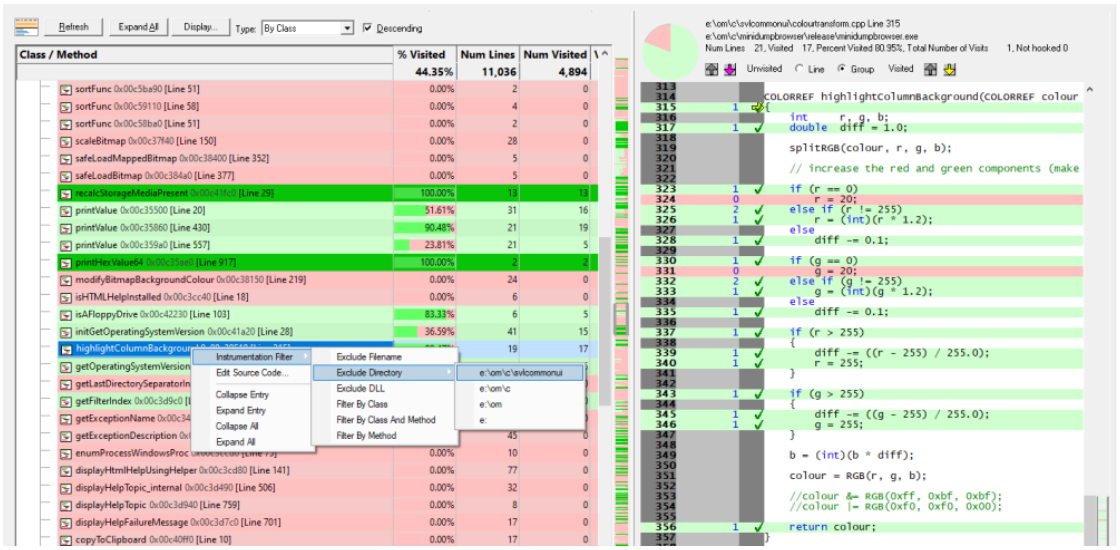


.

3.7.4 Functions

The **Functions** tab view displays function and line coverage information for each hookable function.

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



Functions

The function coverage view focuses on coverage at the function level. If you need to see an overview of coverage for a whole file, see the Coverage view.

When viewing the source code, only the lines in the selected function are highlighted.

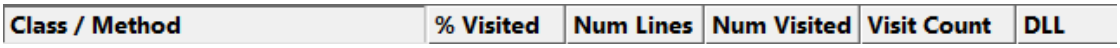
As with many of the tab views, the display is split into two resizable panes.

- the left side lists a summary of the function coverage data

- the right side shows source code for any function selected on the left

We'll cover the data view and its popup menu first, and then the source code view.

The data view



Each line in the view displays the following:

- the file, class, method or function, depending on the Type of the view
- the percentage of hookable lines that have been visited in the function
- the number of hooked lines in the function

  This is not the same as the number of *hookable* lines since some lines or functions can be excluded from hooking.
- the number of *different* lines that have been visited in the function
- the total visit count for the function

  This may be equal to the number of different lines visited unless you're counting each visit separately as seen in one of the hierarchical view examples below.

  The count also includes lines which should have been hooked but failed, for example because the instrumentation level was set low.
- the DLL in which the file was found

The Function coverage can show a hierarchical or list view of data according to the Type setting.

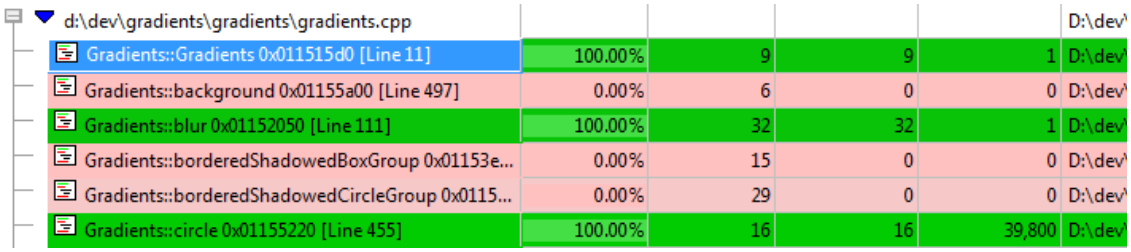
Hierarchical views

A hierarchical view is shown when the Type is set to one of the following:

- class / method
- file / function (the default)
- directory structure / file / function

  Examples:

  Data organised by file / function:



d:\dev\gradients\gradients\gradients.cpp					D:\dev\
Gradients::Gradients 0x011515d0 [Line 11]	100.00%	9	9	1	D:\dev\
Gradients::background 0x01155a00 [Line 497]	0.00%	6	0	0	D:\dev\
Gradients::blur 0x01152050 [Line 111]	100.00%	32	32	1	D:\dev\
Gradients::borderedShadowedBoxGroup 0x01153e...	0.00%	15	0	0	D:\dev\
Gradients::borderedShadowedCircleGroup 0x0115...	0.00%	29	0	0	D:\dev\
Gradients::circle 0x01155220 [Line 455]	100.00%	16	16	39,800	D:\dev\

Data organised by directory / file / function:

Ctrl

For class and directory hierarchical views, the statistics shown against each directory or class is an aggregated sum of all the contained functions

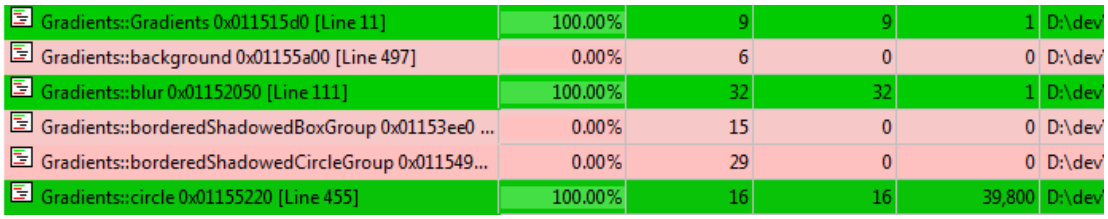
List views

A hierarchical view is shown when the Type is set to one of the following:

- method or function ordered by name
- class or method ordered by number of lines in file

Example:

Data organised by function name:



Gradients::Gradients 0x011515d0 [Line 11]	100.00%	9	9	1	D:\dev\
Gradients::background 0x01155a00 [Line 497]	0.00%	6	0	0	D:\dev\
Gradients::blur 0x01152050 [Line 111]	100.00%	32	32	1	D:\dev\
Gradients::borderedShadowedBoxGroup 0x01153ee0 ...	0.00%	15	0	0	D:\dev\
Gradients::borderedShadowedCircleGroup 0x011549...	0.00%	29	0	0	D:\dev\
Gradients::circle 0x01155220 [Line 455]	100.00%	16	16	39,800	D:\dev\

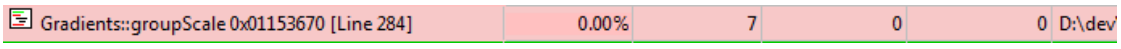
Function names are sorted alphabetically. Destructors (starting with ~) will likely be shown first or last depending on the sorting order.

Only the first column in the view can be used for sorting the data.

Line colours

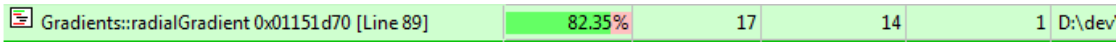
Each line of data in any of the views is colour coded to indicate one of the following:

- the function has not had any lines visited at all



Gradients::groupScale 0x01153670 [Line 284]	0.00%	7	0	0	D:\dev\

- some but not all lines in the function have been visited



- every hookable line in the function has been visited, i.e. 100% file coverage

- no lines in the function could be hooked

Unhookable lines are likely when the instrumentation level is set low, for example *'incomplete but faster'*.

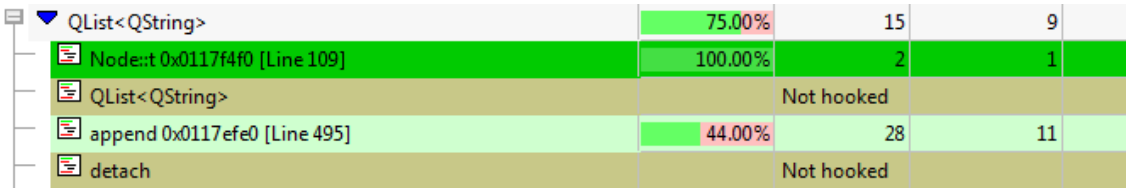
Unhooked lines

Depending on what file or DLLs hook filters are set up, not all the hookable lines in a header file may attempt to be hooked.

Only those that are actually used (e.g. via macros or inlines) in the program may actually be hooked.

By default, lines that could not be hooked don't stop the coverage from reaching 100%, but you can change this.

In this example below, the instrumentation level was set low (*'incomplete but faster'*) to force significant numbers of unhooked lines.



Scrollbar visualisations

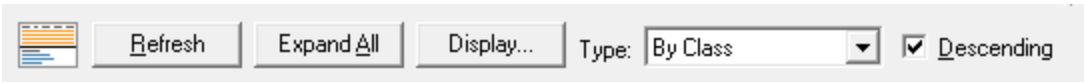
To the right of each pane, beside the vertical scrollbars, you'll see a coloured area which represents the coverage.

As with the scrollbar visualisation in the Coverage tab, the bar in the left hand pane shows an overall view of the coverage distribution across every function listed.

However, because function coverage is broken down to the function level, selecting a function in the left hand pane will only highlight lines in the source code view *for that function*.

Function coverage options

The **function coverage** controls are shown below.



Window orientation

The horizontal or vertical orientation of the statistics and source code panes can be toggled with the orientation button.

Updating the display

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

View type

Changing the Type alters the view to one of the hierarchical or list views described earlier.

- **By Class** › shows a hierarchical C++ class / method view
- **By Function Name** › shows a list of all functions sorted by name
- **By Directory** › shows a hierarchical view of directory / file / function
- **By File** › shows a hierarchical file / function view
- **By Number of Lines** › shows a list of classes or methods ordered by number of lines in file

Each view shows the same overall data, just in different arrangements

Sort order

- **Descending** › reverses the direction of the data, sorted by the names in first column

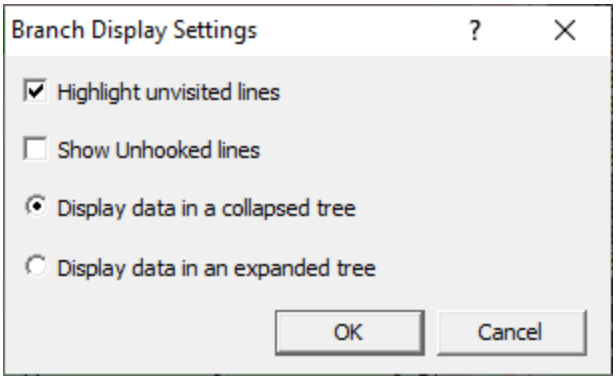
Expanding and collapsing the data

- **Expand All / Collapse All** › switches any hierarchical view between collapsed and expanded view
- **Collapsed** › sets the preferred state of the view *when the data is refreshed*

  This includes when you change some of the other display settings.

Display settings

- **Display...** › displays the branch display settings dialog



- **Highlight unvisited lines** › highlight rows for any functions that have not been visited (on by default)

  If switched off, unvisited lines appear white - or whatever colour you've set as the *unselected* colour.

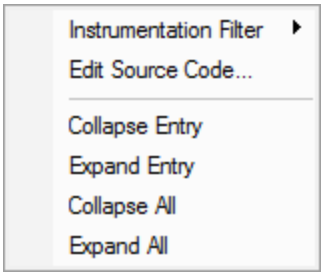
- **Show unhooked functions** › display information about functions that could not be hooked (off by default)

  This means unhookable lines, for example lines that were too short, rather than lines or functions that are deliberately set not to be hooked.

File list menu options

The popup menu is identical to that for the Branch Coverage view.

The following popup menu is available over the data area to add filters, edit code, or expand and collapse the view.



Menu options: Instrumentation Filter

The instrumentation filter lets you add filters at different levels of granularity:

- **Exclude Filename** › adds a new filter to the Source Files Filter settings, excluding it from the results of subsequent sessions
- **Exclude Directory** › excludes all files in the same directory as the selected file

  From the sub menu, choose the directory level at which you want to exclude files, right up to the drive specifier if you need to.

- **Exclude DLL** › excludes all files belonging to the same executable or DLL as the selected file

  This adds a filter to the Hooked DLLs settings.

The other options all add a filter to the Class and function filter settings

- **Filter By Class** › excludes all functions in the selected class

  Example filter: `CTeststakDoc::`
- **Filter By Class And Method** › excludes only the selected class functions

  Example filter: `CTeststakDoc::OnNewDocument`
- **Filter By Method** › excludes all functions in the selected class

  Example filter: `OnNewDocument`

Filters become effective at the start of the *next* session. Adding a filter during a session will show the relevant rows in grey so that you can see which items would be filtered, but the coverage results will continue to be included for the rest of the current session.

Menu option: editing source code

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

Menu options: collapse / expand trace

- **Expand** or **Collapse Entry** > shows and hides the selected section in the view, the same as using the or buttons
- **Collapse All** > completely collapses all sections
- **Expand All** > expands all the sections

The functions source code view

Clicking on a function in the left hand panel, shows that file and function in the source code view on the right.

The source code uses syntax highlighting by default, with the background colour of the line indicating if the function lines have been visited, are unvisited or could not be hooked.

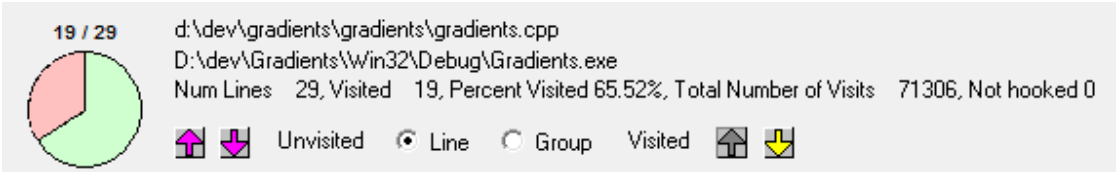
Icons are displayed next to hooked and unhooked function lines indicating visit and hook status and visit counts are shown in-line with the code.

Hovering over a line for a short period of time shows a tooltip with the number of visits to the line.

Contiguous groups of lines can be collapsed and hidden from view.

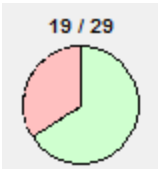
Source code function information

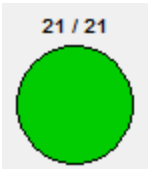
On the right hand panel, above the source code, you'll find some information about lines in the selected function in the source file



The details shown include the following:

- 'quick view' details for visited and unvisited lines in the function, pink for unvisited, light green for partial visited, and dark green for 100% visited





- the source code filename and the executable or DLL to which it belongs

  F1

- the same statistics as seen in the left hand panel

  Num Lines 29, Visited 19, Percent Visited 65.52%, Total Number of Visits 71306, Not hooked 0

Browsing visited and unvisited lines

-  › show the previous and next *unvisited* line in the function

  F7 and Shift + F7 also navigate forwards and backwards when the source code has focus.

-  › show previous and next *visited* line in the function

  The arrows are grey when disabled.

  F8 and Shift + F8 also navigate forwards and backwards through the visited lines.

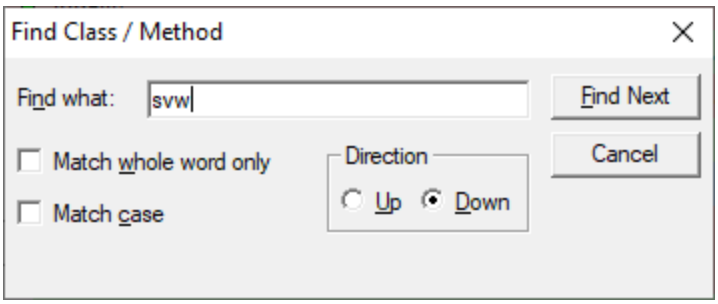
- **Line** › step by individual lines of code
- **Group** › step by groups of contiguous visited or unvisited lines

Keyboard access: Find and Goto

When the data view or the source code view has focus, some keyboard access is available to search for text, or to navigate to numbered lines.

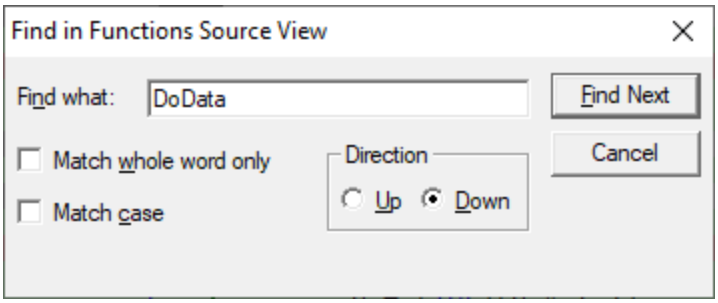
Find text in data view

In the data view, Ctrl + F displays a dialog that will allow you to search by full or partial match for text in the first column - the contents of which depends on the Type setting.



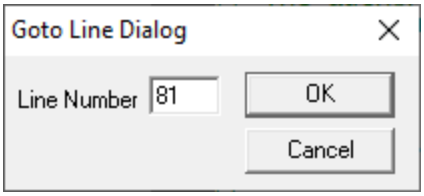
Find text in source view

In the source code view Ctrl + F lets you search by full or partial match anywhere in the file.



Goto line

In the source code view, Ctrl + G displays a goto-line dialog.

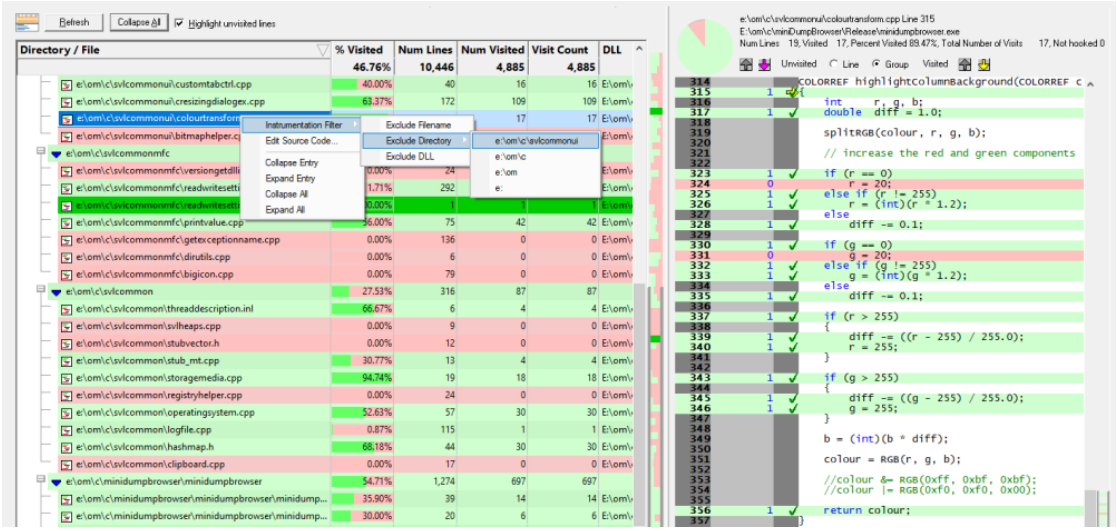


.

3.7.5 Directories

The **DLLs** tab displays coverage information for each DLL that contains hooked code.

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



The DLLs view is largely similar to the Coverage view except that the file list is structured as a hierarchy of DLLs and files rather than just a flat list of files.

As with many of the tab views, the display is split into two resizable panes.

- the left side lists a summary of the directory and file coverage data
- the right side shows source code for any file selected on the left

We'll cover the file list and its popup menu first, and then the source code view.

The directory and file list

Each line in the view displays the following:

- the directory or file name
- the percentage of hookable lines that have been visited
- the number of hooked lines in the directory or file

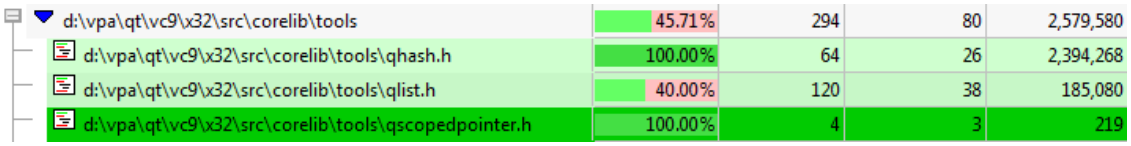
  This is not the same as the total number of lines in a file, which may include comments, declarations and white space.

  Neither is it the same as the number of *hookable* lines since some lines or functions can be excluded from hooking.

  The count also includes lines which should have been hooked but failed, for example because the instrumentation level was set low.

- the number of *different* lines that have been visited
- the total visit count for the file

This may be equal to the number of different lines visited unless you're counting each visit separately as seen below:



- the DLL in which the file was found

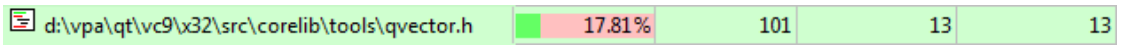
The statistics for each directory is an aggregated view of all the files within it.

Line colours

Each line in the list is colour coded to indicate one of the following:

- the file has not been visited at all

- the file has been visited to some extent



- every hookable line in the file has been visited, i.e. 100% file coverage

- none of the lines in the file could be hooked

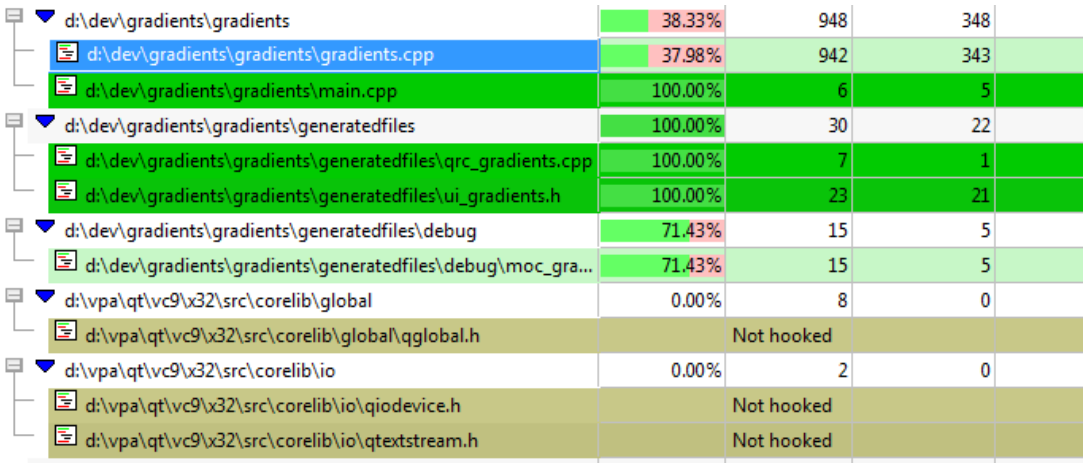
Unhooked lines

Depending on what file or DLLs hook filters are set up, not all the hookable lines in a header file may attempt to be hooked.

Only those that are actually used (e.g. via macros or inlines) in the program may actually be hooked.

By default, lines that could not be hooked don't stop the coverage from reaching 100%, but you can change this.

In this example below, the instrumentation level was set low (*'incomplete but faster'*) to force significant numbers of unhooked lines.

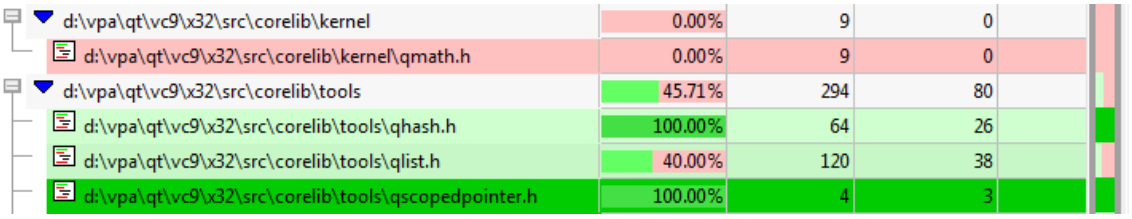


d:\dev\gradients\gradients	38.33%	948	348
d:\dev\gradients\gradients\gradients.cpp	37.98%	942	343
d:\dev\gradients\gradients\main.cpp	100.00%	6	5
d:\dev\gradients\gradients\generatedfiles	100.00%	30	22
d:\dev\gradients\gradients\generatedfiles\qrc_gradients.cpp	100.00%	7	1
d:\dev\gradients\gradients\generatedfiles\ui_gradients.h	100.00%	23	21
d:\dev\gradients\gradients\generatedfiles\debug	71.43%	15	5
d:\dev\gradients\gradients\generatedfiles\debug\moc_gra...	71.43%	15	5
d:\vpa\qt\vc9\x32\src\corelib\global	0.00%	8	0
d:\vpa\qt\vc9\x32\src\corelib\global\qglobal.h		Not hooked	
d:\vpa\qt\vc9\x32\src\corelib\io	0.00%	2	0
d:\vpa\qt\vc9\x32\src\corelib\io\qiodevice.h		Not hooked	
d:\vpa\qt\vc9\x32\src\corelib\io\qtextstream.h		Not hooked	

Scrollbar visualisations

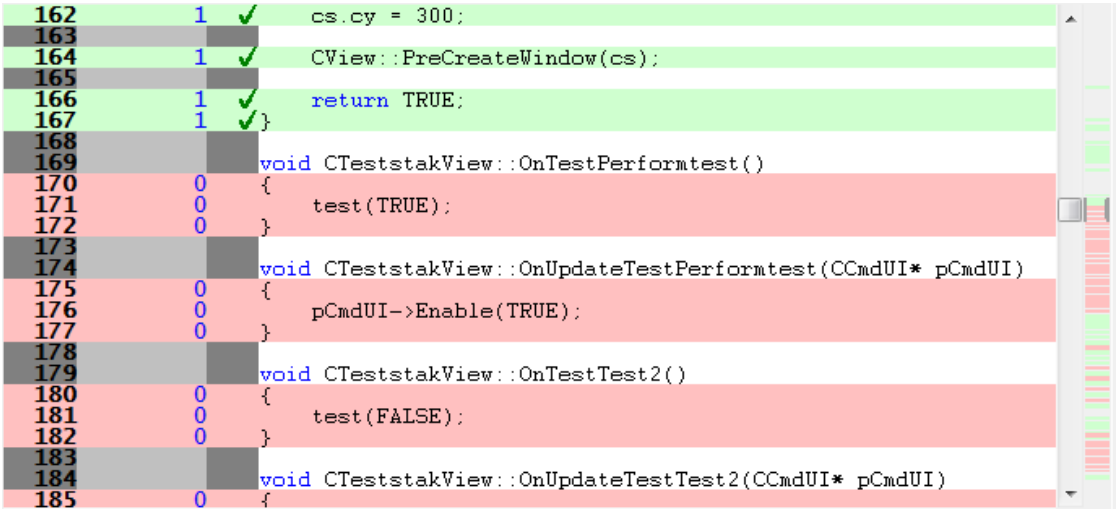
To the right of each pane, beside the vertical scrollbars, you'll see a coloured area which represents the coverage.

For the directory and file list, the visualisation shows the coverage of each item in the list - reflecting the progress meters in the second column.



d:\vpa\qt\vc9\x32\src\corelib\kernel	0.00%	9	0
d:\vpa\qt\vc9\x32\src\corelib\kernel\qmath.h	0.00%	9	0
d:\vpa\qt\vc9\x32\src\corelib\tools	45.71%	294	80
d:\vpa\qt\vc9\x32\src\corelib\tools\qhash.h	100.00%	64	26
d:\vpa\qt\vc9\x32\src\corelib\tools\qlist.h	40.00%	120	38
d:\vpa\qt\vc9\x32\src\corelib\tools\qscopedpointer.h	100.00%	4	3

For the source code view, the visualisation shows the coverage of each line in the file



```
162   1  ✓     cs.cy = 300;
163
164   1  ✓     CView::PreCreateWindow(cs);
165
166   1  ✓     return TRUE;
167   1  ✓}
168
169         void CTeststakView::OnTestPerformtest()
170   0     {
171   0         test(TRUE);
172   0     }
173
174         void CTeststakView::OnUpdateTestPerformtest(CCmdUI* pCmdUI)
175   0     {
176   0         pCmdUI->Enable(TRUE);
177   0     }
178
179         void CTeststakView::OnTestTest2()
180   0     {
181   0         test(FALSE);
182   0     }
183
184         void CTeststakView::OnUpdateTestTest2(CCmdUI* pCmdUI)
185   0     {
```

These visualisations provide a snapshot view of the whole list of files or the whole source file.

Irrespective of how much you can see in the directory and file list or in the source code view, you can still get a global overview of the coverage distribution.

Window orientation

The horizontal or vertical orientation of the statistics and source code panes can be toggled with the orientation button.

Updating the display

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

Sort order

- **Descending** › reverses the direction of the data, sorted by the names in first column

Expanding and collapsing the data

- **Expand All / Collapse All** › switches any hierarchical view between collapsed and expanded view

Display settings

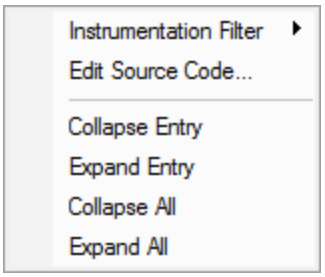
- **Highlight unvisited lines** › highlights rows for any functions that have not been visited (on by default)

  If switched off, unvisited lines appear white - or whatever colour you've set as the *unselected* colour.

File list menu options

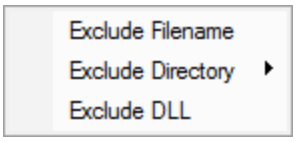
The popup menu is identical to that for the Branch Coverage and Functions view.

The following popup menu is available over the data area to add filters, edit code, or expand and collapse the view.



Menu options: Instrumentation Filter

The instrumentation filter lets you add filters at different levels of granularity:



- **Exclude Filename** > adds a new filter to the Source Files Filter settings, excluding it from the results of subsequent sessions
- **Exclude Directory** > excludes all files in the same directory as the selected file

  From the sub menu, choose the directory level at which you want to exclude files, right up to the drive specifier if you need to.

- **Exclude DLL** > excludes all files belonging to the same executable or DLL as the selected file

  This adds a filter to the Hooked DLLs settings.

Filters become effective at the start of the *next* session. Adding a filter during a session will show the relevant rows in grey so that you can see which items would be filtered, but the coverage results will continue to be included for the rest of the current session.

Menu option: editing source code

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

Menu options: collapse / expand trace

- **Expand** or **Collapse Entry** › shows and hides the selected section in the view, the same as using the ⊞ or ⊟ buttons
- **Collapse All** › completely collapses all sections
- **Expand All** › expands all the sections

The file source code view

Clicking on a file in the directory and file list, shows that file in the source code view on the right.

The source code uses syntax highlighting by default, with the background colour of the line indicating if the line has been visited, is unvisited or could not be hooked.

Icons are displayed next to hooked and unhooked lines indicating visit and hook status and visit counts are shown in-line with the code.

Hovering over a line for a short period of time shows a tooltip with the number of visits to the line.

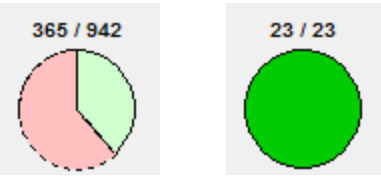
Contiguous groups of lines can be collapsed and hidden from view.

Source code file information

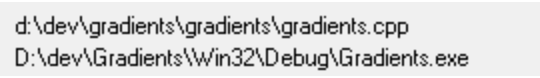
On the right hand panel, above the source code, you'll find some information about the source file

The details shown include the following:

- 'quick view' details for visited and unvisited lines, pink for unvisited, light green for partial visited, and dark green for 100% visited



- the source code filename and the executable or DLL to which it belongs



- the same statistics as seen in the left hand panel

  Num Lines 942, Visited 365, Percent Visited 38.75%, Total Number of Visits 3933220, Not hooked 0

Browsing visited and unvisited lines

-  › show the previous and next *unvisited* line of code in the file

  F7 and Shift + F7 also navigate forwards and backwards when the source code has focus.

-  › show previous and next *visited* line of code in the file

  The arrows are grey when disabled.

  F8 and Shift + F8 also navigate forwards and backwards through the visited lines.

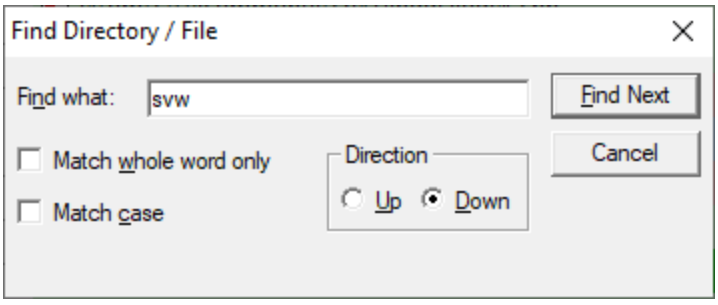
- **Line** › step by individual lines of code
- **Group** › step by groups of contiguous visited or unvisited lines

Keyboard access: Find and Goto

When the data view or the source code view has focus, some keyboard access is available to search for directories, or to navigate to numbered lines.

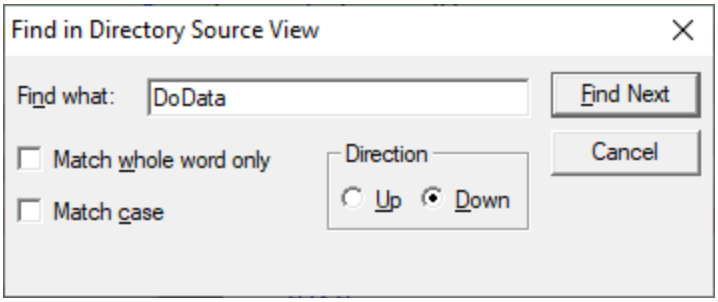
Find filename in data view

In the data view, Ctrl + F displays a dialog that will allow you to search by full or partial match for a directory path.



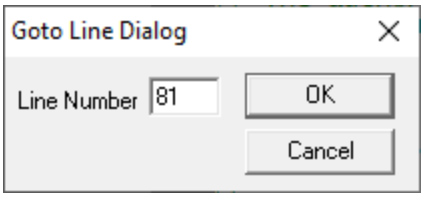
Find text in source view

In the source code view Ctrl + F lets you search by full or partial match anywhere in the file.



Goto line in source view

In the source code view, Ctrl + G displays a goto-line dialog.

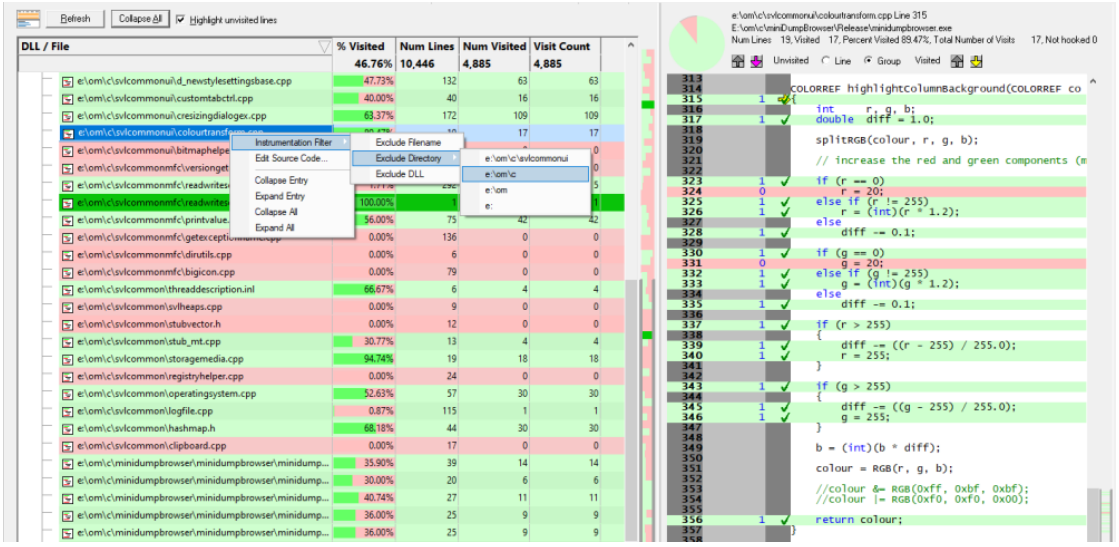


.

3.7.6 DLLs

The **DLLs** tab displays source file coverage information for each DLL that contains hooked functions.

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



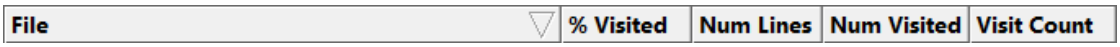
The DLLs view is largely similar to the Coverage view except that the file list is structured as a hierarchy of DLLs and files rather than just a flat list of files.

As with many of the tab views, the display is split into two resizable panes.

- the left side lists a summary of the DLL and file coverage data
- the right side shows source code for any file selected on the left

We'll cover the file list and its popup menu first, and then the source code view.

The DLL and file list



Each line in the view displays the following:

- the DLL or file name
- the percentage of hookable lines that have been visited
- the number of hooked lines in the DLL or file

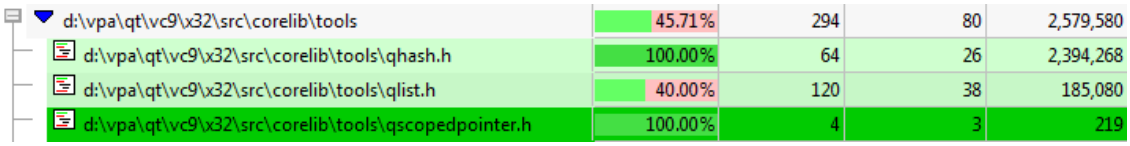
  This is not the same as the total number of lines in a file, which may include comments, declarations and white space.

  Neither is it the same as the number of *hookable* lines since some lines or functions can be excluded from hooking.

  The count also includes lines which should have been hooked but failed, for example because the instrumentation level was set low.

- the number of *different* lines that have been visited
- the total visit count for the file

  This may be equal to the number of different lines visited unless you're counting each visit separately as seen below:



The statistics for each DLL is an aggregated view of all the files within it.

Files in a DLL are ordered by their full paths, not by the directories in which they are found.

Line colours

Each line in the list is colour coded to indicate one of the following:

- the file has not been visited at all

d:\vpa\qt\vc9\x32\src\corelib\kernel\qmath.h	0.00%	9	0	0

- the file has been visited to some extent

d:\vpa\qt\vc9\x32\src\corelib\tools\qvector.h	17.81%	101	13	13

- every hookable line in the file has been visited, i.e. 100% file coverage

d:\dev\gradients\gradients\generatedfiles\qrc_...	100.00%	7	1	1

- none of the lines in the file could be hooked

d:\vpa\qt\vc9\x32\src\corelib\tools\qstring.h		Not hooked		

Unhooked lines

Depending on what file or DLLs hook filters are set up, not all the hookable lines in a header file may attempt to be hooked.

Only those that are actually used (e.g. via macros or inlines) in the program may actually be hooked.

 By default, lines that could not be hooked don't stop the coverage from reaching 100%, but you can change this.

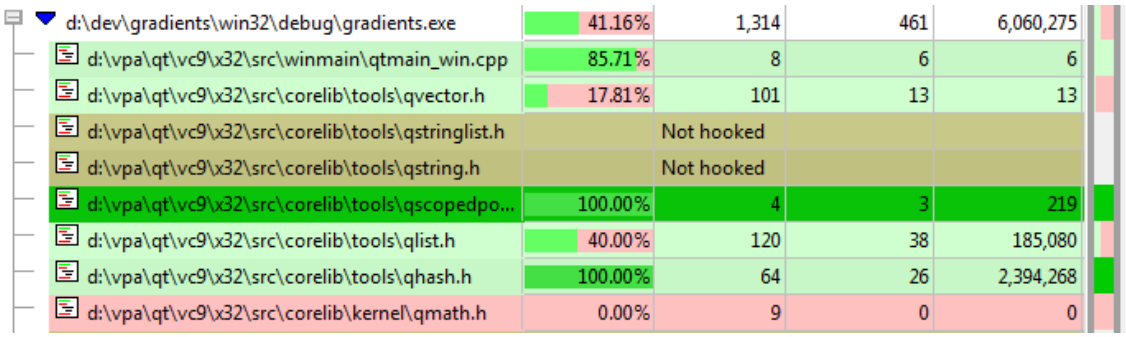
In this example below, the instrumentation level was set low (*'incomplete but faster'*) to force significant numbers of unhooked lines.

d:\dev\gradients\win32\debug\gradients.exe	41.16%	1,314	461	6,060,275
d:\vpa\qt\vc9\x32\src\winmain\qtmain_win.cpp	85.71%	8	6	6
d:\vpa\qt\vc9\x32\src\corelib\tools\qvector.h	17.81%	101	13	13
d:\vpa\qt\vc9\x32\src\corelib\tools\qstringlist.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\tools\qstring.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\tools\qscopedpo...	100.00%	4	3	219
d:\vpa\qt\vc9\x32\src\corelib\tools\qlist.h	40.00%	120	38	185,080
d:\vpa\qt\vc9\x32\src\corelib\tools\qhash.h	100.00%	64	26	2,394,268
d:\vpa\qt\vc9\x32\src\corelib\kernel\qmath.h	0.00%	9	0	0
d:\vpa\qt\vc9\x32\src\corelib\io\qtextstream.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\io\qiodevice.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\global\qglobal.h		Not hooked		

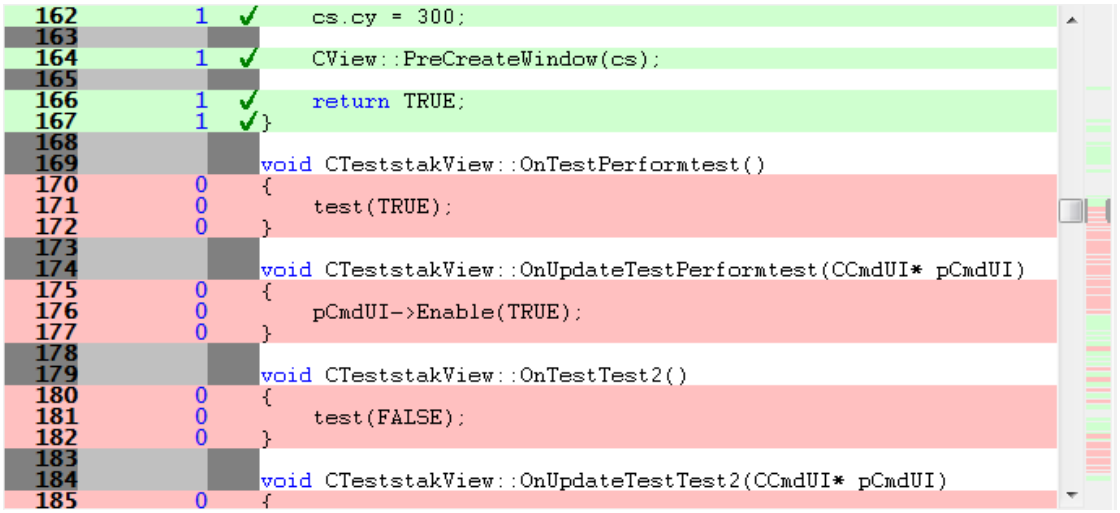
Scrollbar visualisations

To the right of each pane, beside the vertical scrollbars, you'll see a coloured area which represents the coverage.

For the DLL and file list, the visualisation shows the coverage of each item in the list - reflecting the progress meters in the second column.



For the source code view, the visualisation shows the coverage of each line in the file



These visualisations provide a snapshot view of the whole list of files or the whole source file.

Irrespective of how much you can see in the DLL and file list or in the source code view, you can still get a global overview of the coverage distribution.

Window orientation

The horizontal or vertical orientation of the statistics and source code panes can be toggled with the orientation button.

Updating the display

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

Sort order

- **Descending** › reverses the direction of the data, sorted by the names in first column

Expanding and collapsing the data

- **Expand All / Collapse All** › switches any hierarchical view from collapsed to expanded view

Display settings

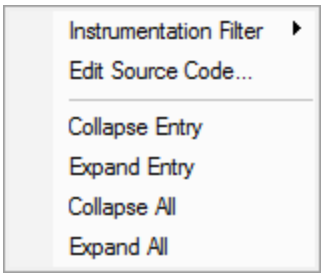
- **Highlight unvisited lines** › highlights rows for any functions that have not been visited (on by default)

  If switched off, unvisited lines appear white - or whatever colour you've set as the *unselected* colour.

File list menu options

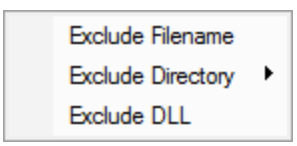
The popup menu is identical to that for the Branch Coverage and Functions view.

The following popup menu is available over the data area to add filters, edit code, or expand and collapse the view.



Menu options: Instrumentation Filter

The instrumentation filter lets you add filters at different levels of granularity:



- **Exclude Filename** › adds a new filter to the Source Files Filter settings, excluding it from the results of subsequent sessions
- **Exclude Directory** › excludes all files in the same directory as the selected file

  From the sub menu, choose the directory level at which you want to exclude files, right up to the drive specifier if you need to

- **Exclude DLL** › excludes all files belonging to the same executable or DLL as the selected file

  This adds a filter to the Hooked DLLs settings.

Filters become effective at the start of the *next* session. Adding a filter during a session will show the relevant rows in grey so that you can see which items would be filtered, but the coverage results will continue to be included for the rest of the current session.

Menu option: editing source code

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

Menu options: collapse / expand trace

- **Expand** or **Collapse Entry** › shows and hides the selected section in the view, the same as using the or buttons
- **Collapse All** › completely collapses all sections
- **Expand All** › expands all the sections

The file source code view

Clicking on a file in the DLL and file list, shows that file in the source code view on the right.

The source code uses syntax highlighting by default, with the background colour of the line indicating if the line has been visited, is unvisited or could not be hooked.

Icons are displayed next to hooked and unhooked lines indicating visit and hook status and visit counts are shown in-line with the code.

Hovering over a line for a short period of time shows a tooltip with the number of visits to the line.

Contiguous groups of lines can be collapsed and hidden from view.

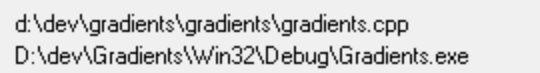
Source code file information

On the right hand panel, above the source code, you'll find some information about the source file

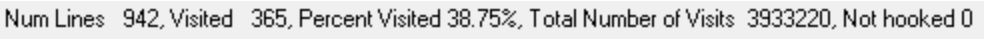
The details shown include the following:

- 'quick view' details for visited and unvisited lines, pink for unvisited, light green for partial visited, and dark green for 100% visited

- the source code filename and the executable or DLL to which it belongs



- the same statistics as seen in the left hand panel



Browsing visited and unvisited lines

-  › show the previous and next *unvisited* line of code in the file

  F7 and Shift + F7 also navigate forwards and backwards when the source code has focus.

-  › show previous and next *visited* line of code in the file

  The arrows are grey when disabled.

  F8 and Shift + F8 also navigate forwards and backwards through the visited lines.

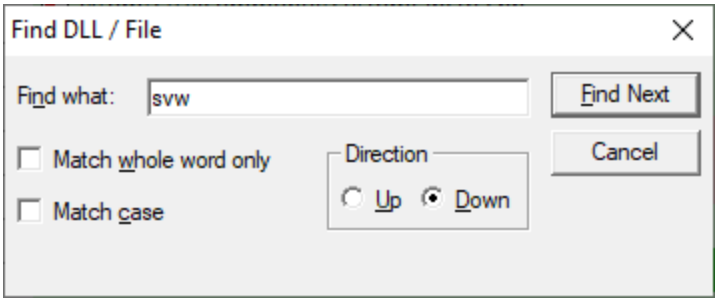
- **Line** › step by individual lines of code
- **Group** › step by groups of contiguous visited or unvisited lines

Keyboard access: Find and Goto

When the data view or the source code view has focus, some keyboard access is available to search for DLLs, source code, or to navigate to numbered lines.

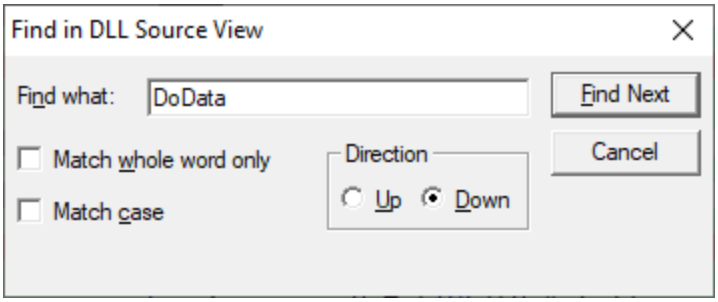
Find filename in data view

In the data view, Ctrl + F displays a dialog that will allow you to search by full or partial match for a DLL path



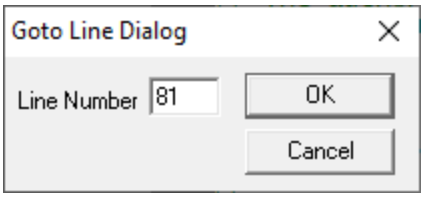
Find text in source view

In the source code view Ctrl + F lets you search by full or partial match anywhere in the file



Goto line in source view

In the source code view, Ctrl + G displays a goto-line dialog

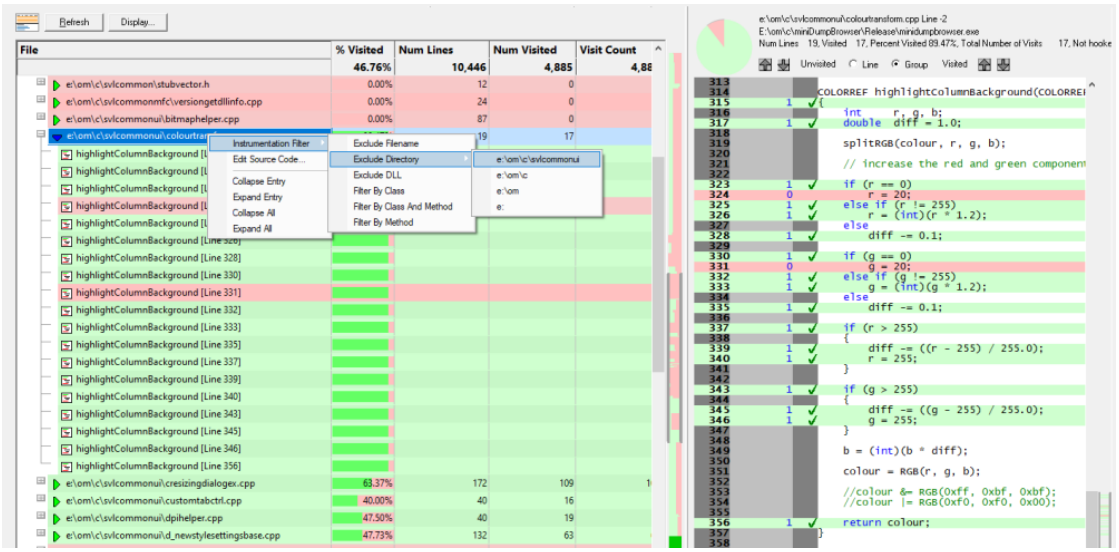


.

3.7.7 Files and lines

The **Files and lines** view displays coverage data for files and their lines.

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



The Files and Lines view is structured as a hierarchy of filenames and the hooked lines within those files.

As with many of the tab views, the display is split into two resizable panes.

- the left side lists a summary of the file and line coverage data
- the right side shows source code for any file or line selected on the left

We'll cover the file and line list and its popup menu first, and then the source code view.

The file and line list

Each line in the view displays the following:

- the filename or function line within the file
- for files, the percentage of hookable lines that have been visited, while line data echos the 'progress bar' of their file
- for files, the number of hooked lines in the file, (unhooked lines in brackets)

  This is not the same as the total number of lines in a file, which may include comments, declarations and white space.

  Neither is it the same as the number of *hookable* lines since some lines or functions can be excluded from hooking.

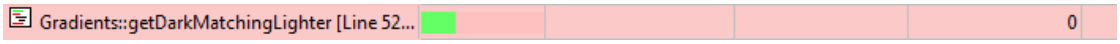
  The count also includes lines which should have been hooked but failed, for example because the instrumentation level was set low. The unhooked line count is shown in brackets.
- for files, the number of *different* lines that have been visited
- the total visit count for the file or line

  This may be equal to the number of different lines visited unless you're counting each visit separately:

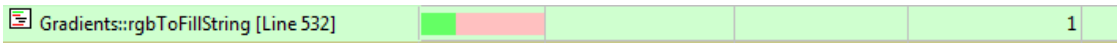
Line colours

Each line in the list is colour coded to indicate one of the following:

- the file or line has not been visited at all

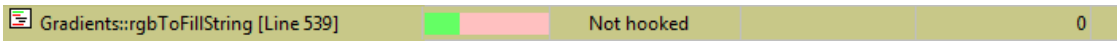


- the file has been visited to some extent (or the line has been visited)



- every hookable line in the file has been visited, i.e. 100% file coverage

- the line in the file could be hooked



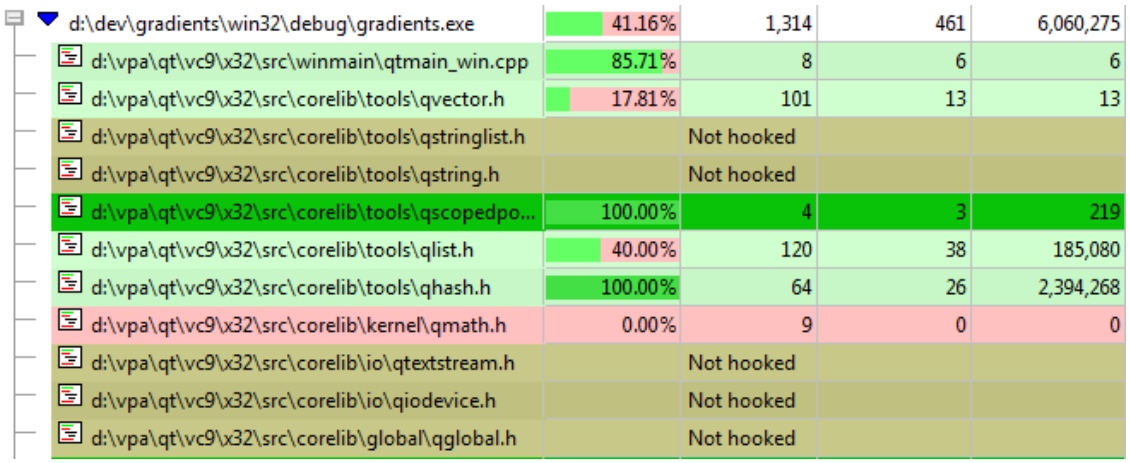
Unhooked lines

Depending on what file or DLLs hook filters are set up, not all the hookable lines in a header file may attempt to be hooked.

Only those that are actually used (e.g. via macros or inlines) in the program may actually be hooked.

By default, lines that could not be hooked don't stop the coverage from reaching 100%, but you can change this.

In this example below, the instrumentation level was set low (*'incomplete but faster'*) to force significant numbers of unhooked lines.

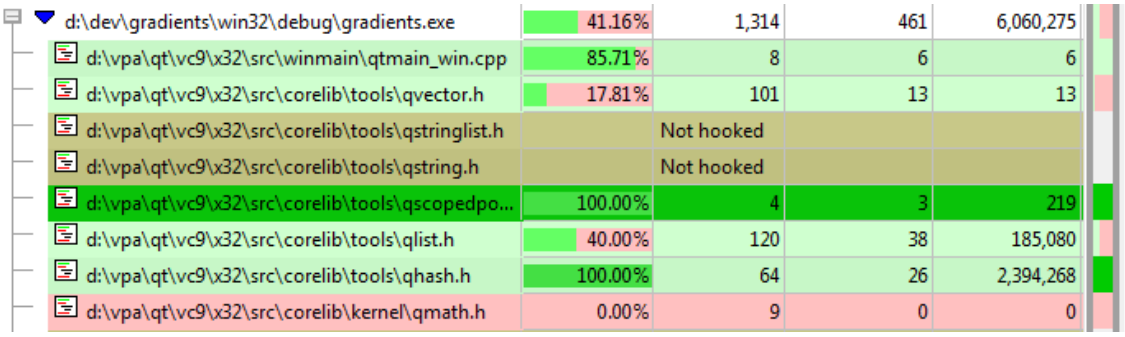


d:\dev\gradients\win32\debug\gradients.exe	41.16%	1,314	461	6,060,275
d:\vpa\qt\vc9\x32\src\winmain\qtmain_win.cpp	85.71%	8	6	6
d:\vpa\qt\vc9\x32\src\corelib\tools\qvector.h	17.81%	101	13	13
d:\vpa\qt\vc9\x32\src\corelib\tools\qstringlist.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\tools\qstring.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\tools\qscopedpo...	100.00%	4	3	219
d:\vpa\qt\vc9\x32\src\corelib\tools\qlist.h	40.00%	120	38	185,080
d:\vpa\qt\vc9\x32\src\corelib\tools\qhash.h	100.00%	64	26	2,394,268
d:\vpa\qt\vc9\x32\src\corelib\kernel\qmath.h	0.00%	9	0	0
d:\vpa\qt\vc9\x32\src\corelib\io\qtextstream.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\io\qiodevice.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\global\qglobal.h		Not hooked		

Scrollbar visualisations

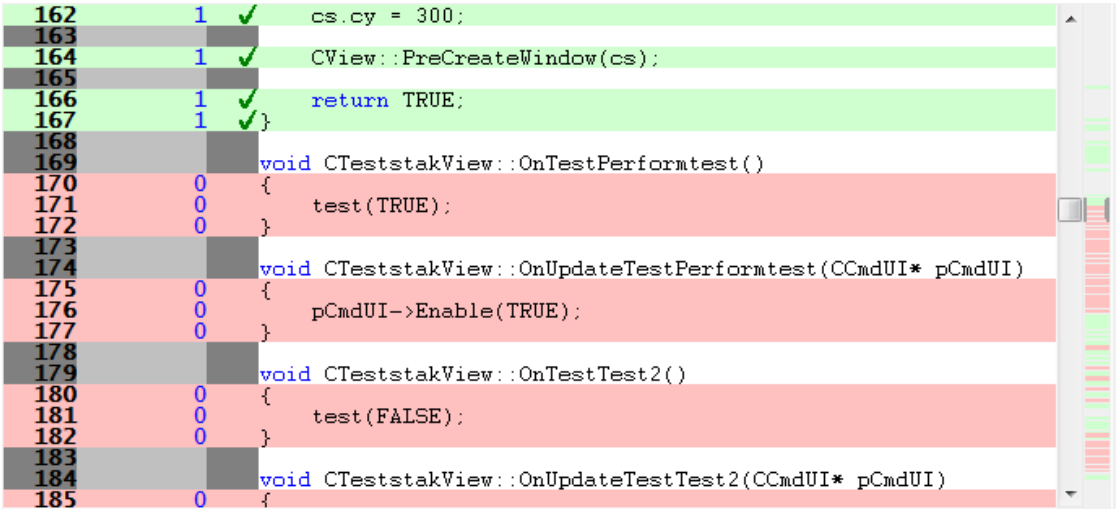
To the right of each pane, beside the vertical scrollbars, you'll see a coloured area which represents the coverage.

For the file and line list, the visualisation shows the coverage of each item in the list - reflecting the progress meters in the second column.



d:\dev\gradients\win32\debug\gradients.exe	41.16%	1,314	461	6,060,275
d:\vpa\qt\vc9\x32\src\winmain\qtmain_win.cpp	85.71%	8	6	6
d:\vpa\qt\vc9\x32\src\corelib\tools\qvector.h	17.81%	101	13	13
d:\vpa\qt\vc9\x32\src\corelib\tools\qstringlist.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\tools\qstring.h		Not hooked		
d:\vpa\qt\vc9\x32\src\corelib\tools\qscopedpo...	100.00%	4	3	219
d:\vpa\qt\vc9\x32\src\corelib\tools\qlist.h	40.00%	120	38	185,080
d:\vpa\qt\vc9\x32\src\corelib\tools\qhash.h	100.00%	64	26	2,394,268
d:\vpa\qt\vc9\x32\src\corelib\kernel\qmath.h	0.00%	9	0	0

For the source code view, the visualisation shows the coverage of each line in the file



These visualisations provide a snapshot view of the whole list of files or the whole source file.

Irrespective of how much you can see in the file and line list or in the source code view, you can still get a global overview of the coverage distribution.

Files and lines options

The **coverage** controls are shown below.

Window orientation

The horizontal or vertical orientation of the statistics and source code panes can be toggled with the orientation button.

Updating the display

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

Display settings

- **Highlight unvisited lines** › highlights rows for any lines that have not been visited (on by default)

  If switched off, unvisited lines appear white - or whatever colour you've set as the *unselected* colour.

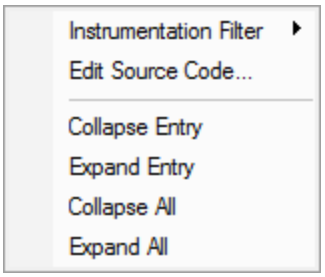
testmemorydialog.cpp	0.00%	79	0	0	C:\Program Files (x

- **Show Path** › for filenames, shows the short file name or the longer file path in the **File** column of the data
- **Show unhooked functions** › display information about lines that could not be hooked (off by default)

  This means unhookable lines, for example, lines that were too short, rather than lines or functions that are deliberately set not to be hooked.

Files and lines menu options

The following popup menu is available over the data area to add filters, edit code, or expand and collapse the view.



Menu options: Instrumentation Filter

The instrumentation filter lets you add filters at different levels of granularity:

- **Exclude Filename** › adds a new filter to the Source Files Filter settings, excluding it from the results of subsequent sessions
- **Exclude Directory** › excludes all files in the same directory as the selected file

  From the sub menu, choose the directory level at which you want to exclude files, right up to the drive specifier if you need to

- **Exclude DLL** › excludes all files belonging to the same executable or DLL as the selected file

  This adds a filter to the Hooked DLLs settings.

The other options all add a filter to the Class and function filter settings

- **Filter By Class** › excludes all functions in the selected class

  Example filter: `CTeststakDoc::`

- **Filter By Class And Method** › excludes only the selected class functions

  Example filter: `CTeststakDoc::OnNewDocument`

- **Filter By Method** › excludes all functions in the selected class

  Example filter: `OnNewDocument`

Filters become effective at the start of the *next* session. Adding a filter during a session will show the relevant rows in grey so that you can see which items would be filtered, but the coverage results will continue to be included for the rest of the current session.

Menu option: editing source code

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

Menu options: collapse / expand trace

- **Expand** or **Collapse Entry** › shows and hides the selected section in the view, the same as using the or buttons
- **Collapse All** › completely collapses all sections
- **Expand All** › expands all the sections

The file source code view

Clicking on a file in the file and line view shows that file in the source code view on the right, with the selected line highlighted.

The source code uses syntax highlighting by default, with the background colour of the line indicating if the line has been visited, is unvisited or could not be hooked.

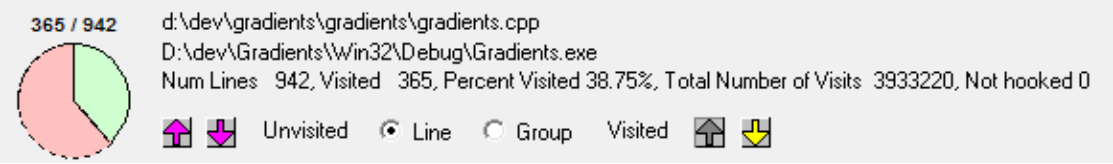
Icons are displayed next to hooked and unhooked lines indicating visit and hook status and visit counts are shown in-line with the code.

Hovering over a line for a short period of time shows a tooltip with the number of visits to the line.

Contiguous groups of lines can be collapsed and hidden from view.

Source code file information

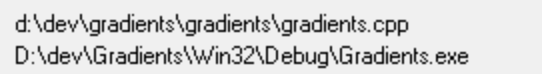
On the right hand panel, above the source code, you'll find some information about the source file



The details shown include the following:

- 'quick view' details for visited and unvisited lines, pink for unvisited, light green for partial visited, and dark green for 100% visited

- the source code filename and the executable or DLL to which it belongs



- the same statistics as seen for the file in the left hand panel

Browsing visited and unvisited lines

When a line has been selected in the files and lines view, the line is marked in the source code.

You can then use the following buttons to move to adjacent lines;

-  › show the previous and next *unvisited* line of code in the file

  F7 and Shift + F7 also navigate forwards and backwards when the source code has focus.

-  › show previous and next *visited* line of code in the file

  The arrows are grey when disabled, for example when a filename rather than a line is selected on the left.

  F8 and Shift + F8 also navigate forwards and backwards through the visited lines.

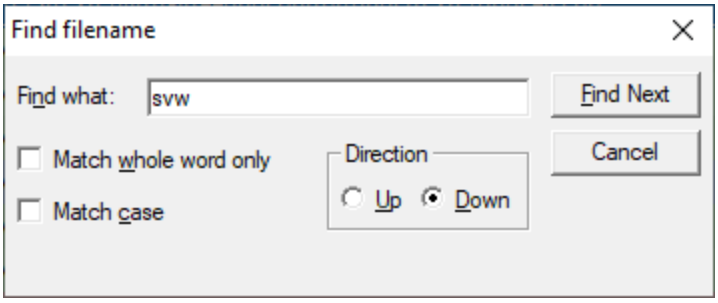
- **Line** › step by individual lines of code
- **Group** › step by groups of contiguous visited or unvisited lines

Keyboard access: Find and Goto

When the data view or the source code view has focus, some keyboard access is available to search for filenames, code or to navigate to numbered lines.

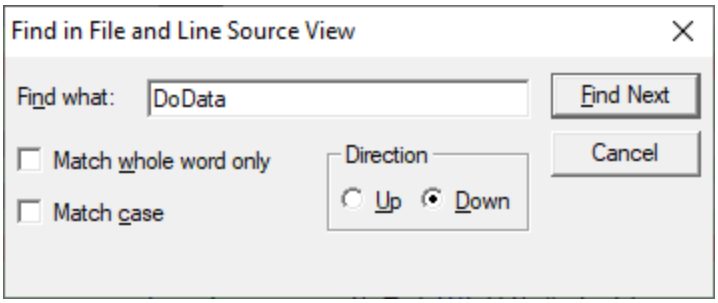
Find text in data view

In the data view, Ctrl + F displays a dialog that will allow you to search by full or partial match for text in the filenames in the first column.



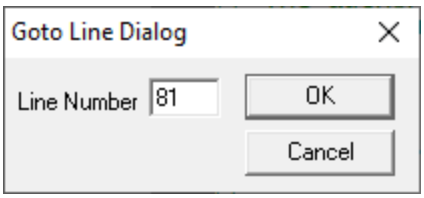
Find text in source view

In the source code view Ctrl + F lets you search by full or partial match anywhere in the file



Goto line

In the source code view, Ctrl + G displays a goto-line dialog



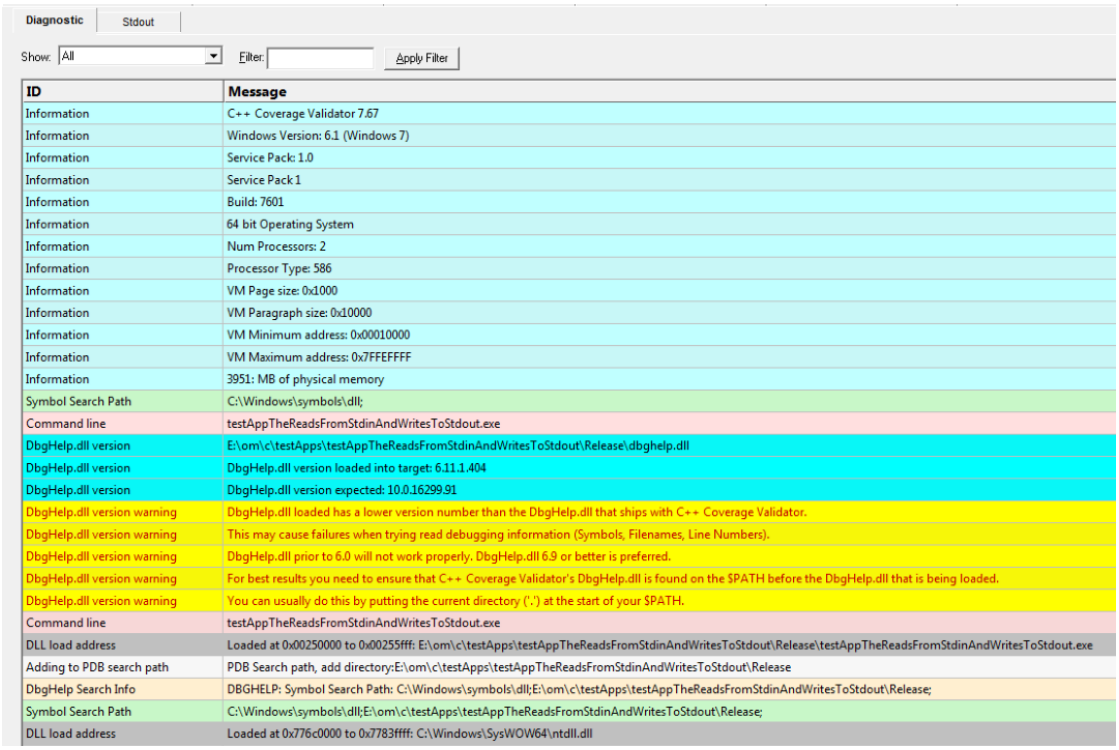
.

3.7.8 Diagnostic

The **Diagnostic** tab displays information collected by Coverage Validator about the target program.

There are two subtabs. One for Diagnostic information and one for displaying any data captured from stdout and stderr.

Diagnostic



Diagnostic information

When Coverage 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
- 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
- Other text message

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 Data Collection > Miscellaneous settings.

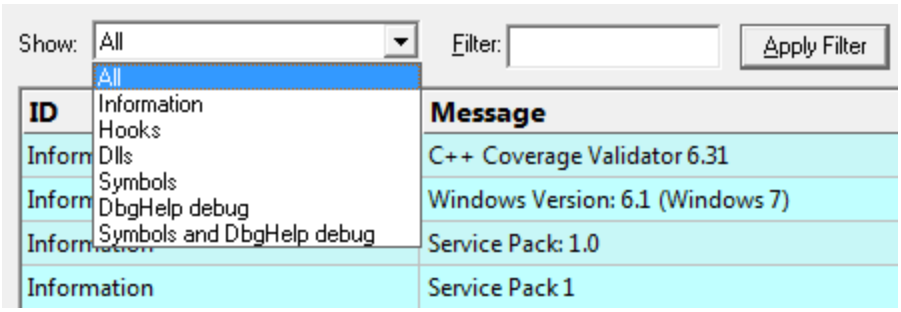
Unhooked lines

Messages indicating that particular source lines could not be hooked are common and are because there was not enough space to safely insert a hook for the particular line.

This is normal and is to be expected. The number of lines that cannot be hooked because of this will vary from program to program, and from debug mode to release mode based on what the program is doing and how the program is optimized.

Filtering diagnostic information

By default, all information is displayed, but you can filter the messages to show only one type:



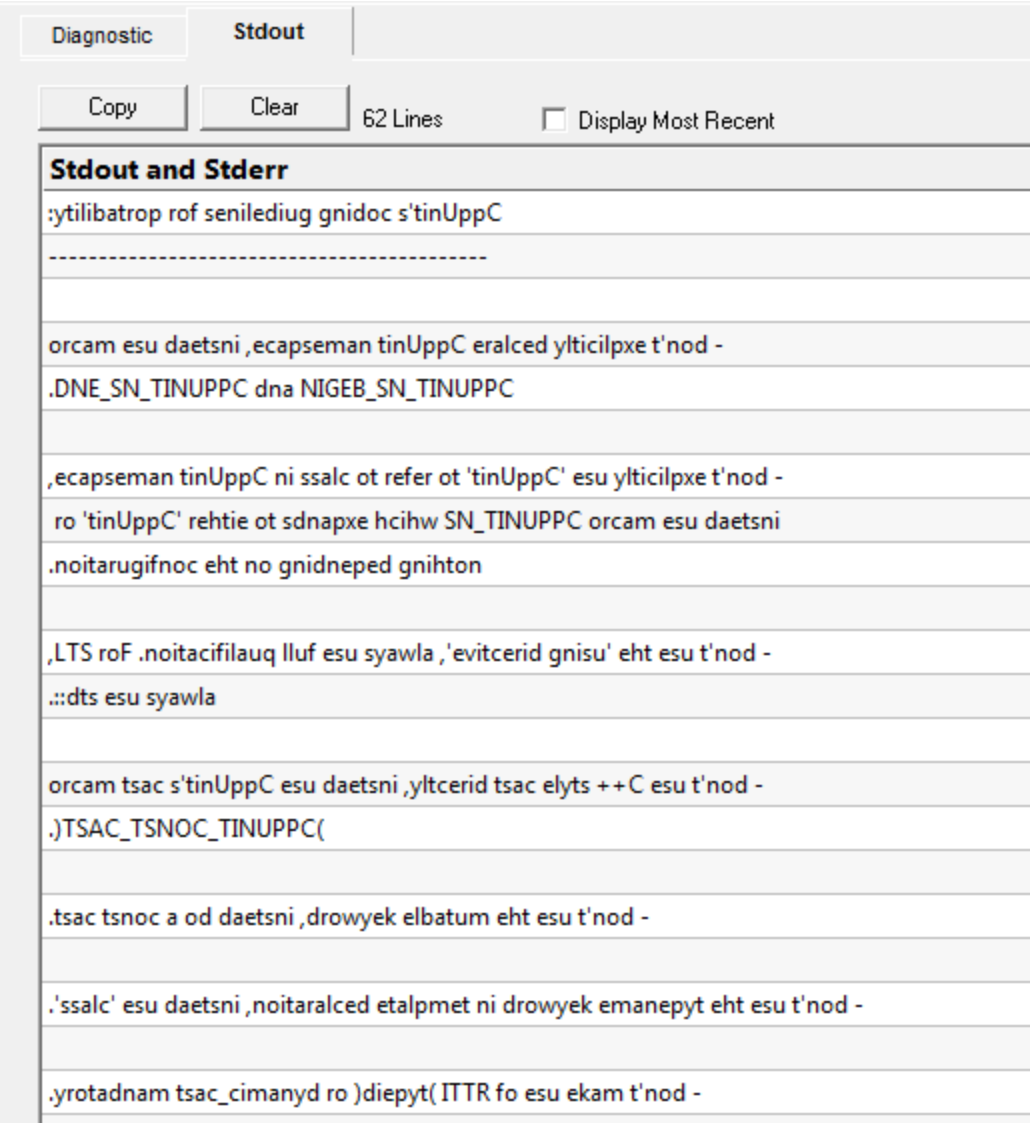
- **All** › the default
- **Information** › operating system and environment information, etc
- **Hooks** › hooking success and failure messages
- **DLLs** › DLL related information
- **Symbols** › symbol loading status messages
- **DbgHelp debug** › messages from DbgHelp.dll about the DLL symbol search 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 **Stdout** tab displays any data collected from stdout and stderr. ➜ The option to enable this data collection is specified on the launch dialog/wizard.

The above image shows some data collected from a program that reverses the characters in each line passed to it.

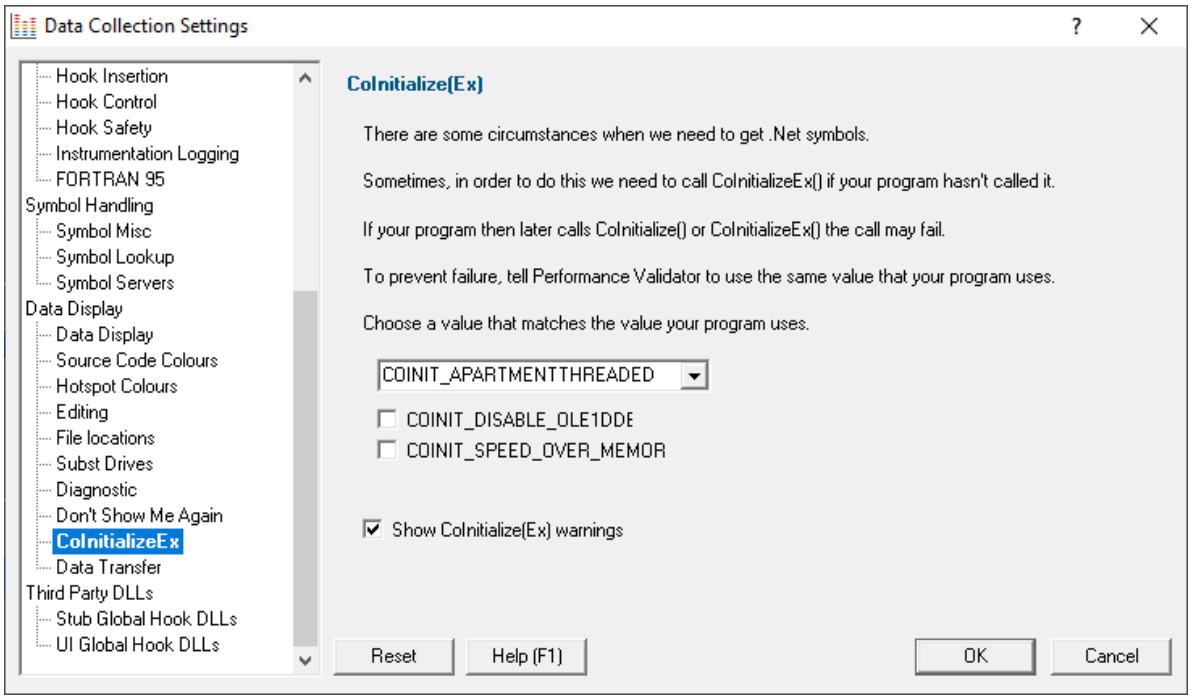
- **Copy** › copy all data from the display on to the clipboard. For large amounts of data this can be time consuming.
- **Clear** › clear the display of any captured data.
- **Display Most Recent** › the display will be scrolled to ensure the most recently captured data is displayed.

Environment Variables

Environment variables tab displays environment variables from Coverage Validator, environment variables from the program under test and environment variable substitution errors.

Choose which data you wish to view using the combo box at the top left of the tab.

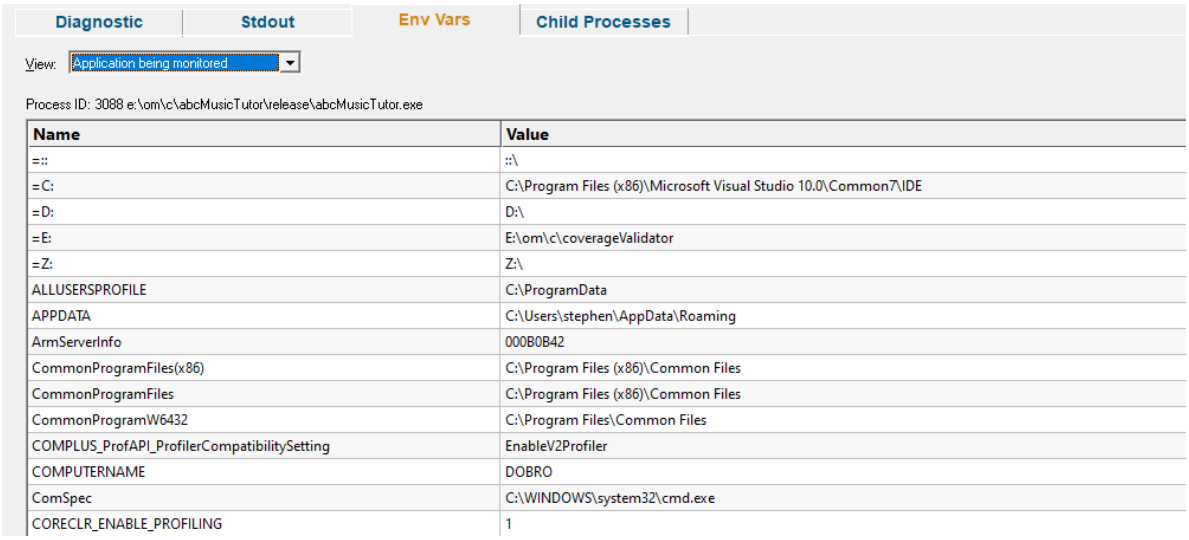
Coverage Validator environment variables



Target application environment variables

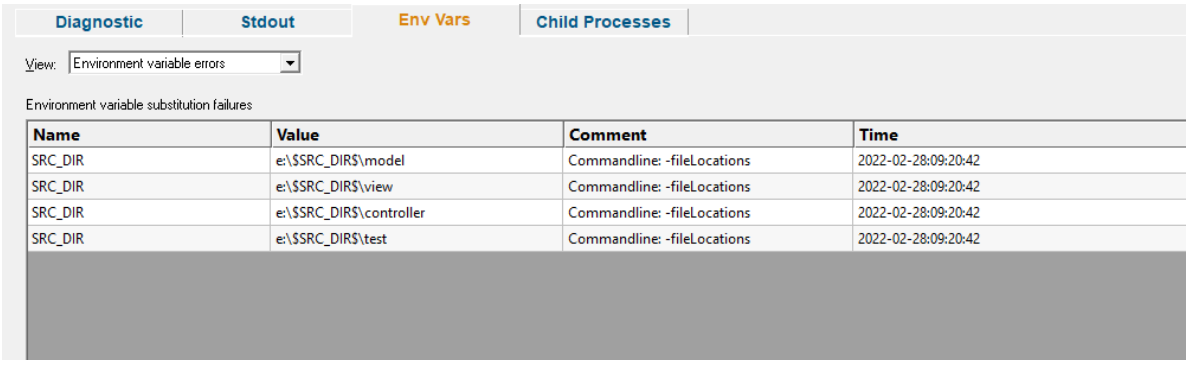
If you launched the target application from the Validator the target application's environment variables will be similar to those in the Validator, but with some additional env vars to control .Net profilers and and some other SVL_ prefixed env vars to communicate various data to Software Verify components that are loaded.

If you launched the target application as a standalone application, or service and used one of our APIs to connect to the Validator, the environment variables shown will reflect those in force at the time the application/service was started, and the account that application/service is running on.



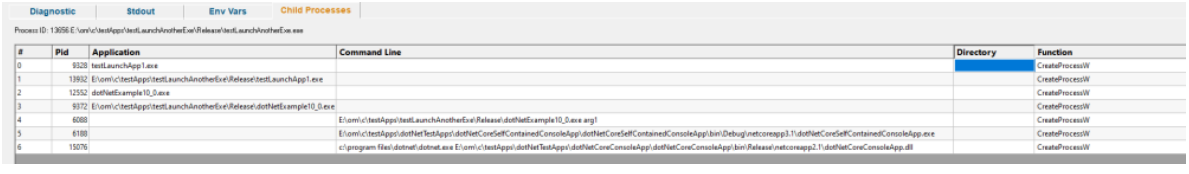
Environment variable errors

The environment variable errors display shows the name of the environment variable that could not be found, the string containing the environment variable, a comment indicating where the string came from (in this example, the command line), and a timestamp.

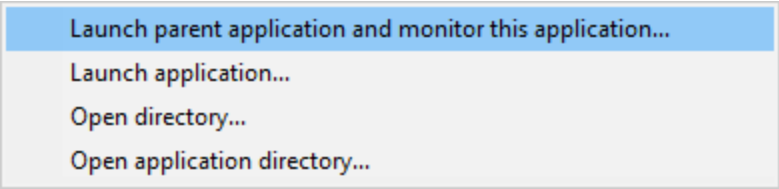


Child Processes

Information about child processes, and the appropriate launch parameters passed to CreateProcess (and related functions) are displayed on this tab.



A context menu is provided to allow you to perform some actions with the launched application data.

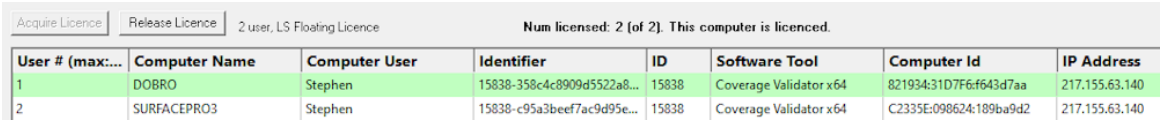


- **Launch parent application and monitor this application...** › the launch application dialog is displayed configured to launch the parent application and monitor this application
- **Launch application...** › the launch application dialog is displayed configured to launch and monitor this application
- **Open directory...** › Windows Explorer is launched to view the contents of the launch directory (the directory field is empty nothing will be shown)
- **Open application directory...** › Windows Explorer is launched to view the directory that contains the application (if the application specification has no path nothing will be shown)

3.7.9 Floating Licence

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

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



User # (max:...	Computer Name	Computer User	Identifier	ID	Software Tool	Computer Id	IP Address
1	DOBRO	Stephen	15838-358c4c8909d5522a8...	15838	Coverage Validator x64	821934:31D7F6:f643d7aa	217.155.63.140
2	SURFACEPRO3	Stephen	15838-c95a3beef7ac9d95e...	15838	Coverage Validator x64	C2335E:098624:189ba9d2	217.155.63.140

The screenshot above show two computers using the same 2 user floating licence, that has maintenance id 15838. 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.

An internet connection is required for floating licences to work. The licence server is managed and run by Software Verify.

Licence information

The information show in this display allows you to identify which of your colleagues are using the software and which versions of the software are in use.

- User
  The user id (1 to number of licensed users).
- Computer Name
  The name of the computer

- Computer User
  The login name of the user of the computer.

- Identifier
  The unique identifier for this licence, used on the licence server.

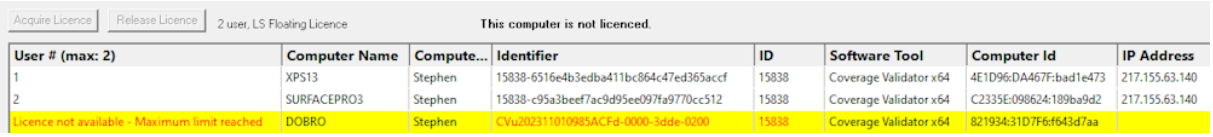
- ID
  The maintenance id for the software.

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

- Computer ID
  The unique id for this computer.

- IP Address
  This computer's IP address.

Unlicenced users



User # (max: 2)	Computer Name	Compute...	Identifier	ID	Software Tool	Computer Id	IP Address
1	XPS13	Stephen	15838-6516e4b3edba411bc864c47ed365accf	15838	Coverage Validator x64	4E1D96:DA467F:bad1e473	217.155.63.140
2	SURFACEPRO3	Stephen	15838-c95a3beef7ac9d95ee097fa9770cc512	15838	Coverage Validator x64	C2335E:098624:189ba9d2	217.155.63.140
Licence not available - Maximum limit reached	DOBRO	Stephen	CVu202311010985ACFd-0000-3dde-0200	15838	Coverage Validator x64	821934:31D7F6:f643d7aa	

If any additional users are trying to get a licence for the software, but there are not enough licences, they will also be shown in the display, but with red text on a yellow background.

Please note that on the machine of an unlicensed user the status information will be different.

The software checks to see if a licence has been released on a periodic basis, so that if a licence is released by another user, it can be acquired by the next waiting user.

Releasing a licence

If you have finished using a licence and wish to let a team mate use the software, you have two choices.

You can close Thread Validator, releasing the licence as it closes.

Or you can keep Thread Validator running by manually releasing the licence. Do this by clicking the **Release Licence** button.

Acquiring a licence

If you have released a licence you will need to actively reclaim a licence when you wish to use Coverage 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?

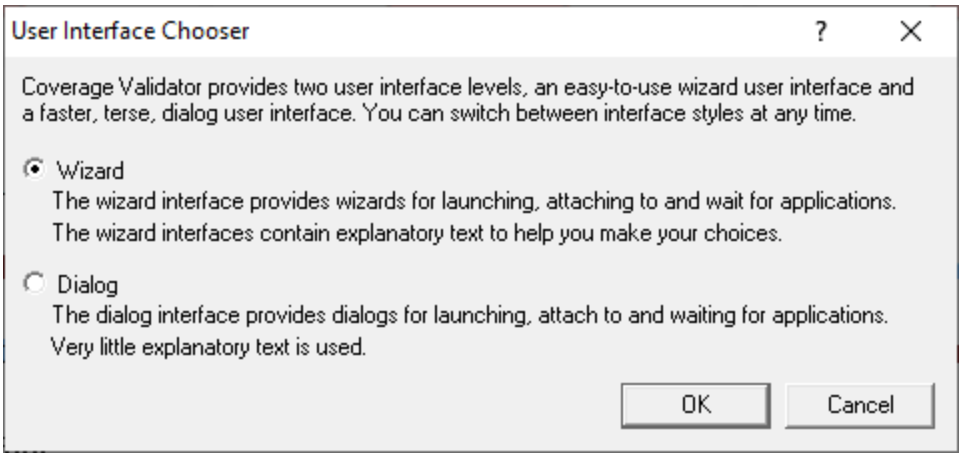
For some key tasks, there are two user interface modes controlling the way in which options are presented to you:

- **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 UX Theme

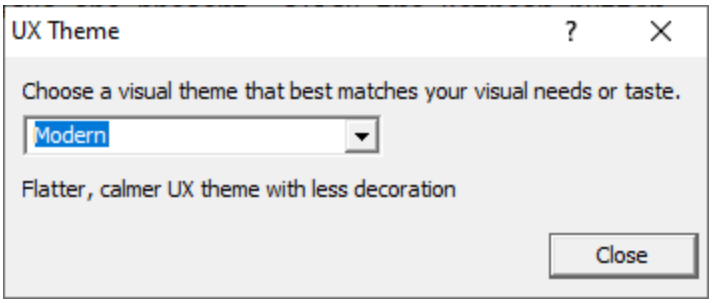
The user interface provides three UX themes.

- Modern. The look and feel of current Software Verify tools.
- Classic. The look and feel of previous Software Verify tools.
- High Contrast. A higher contrast version of the Modern theme.

Setting the UX theme

To set the UX theme

Settings menu › **UX Theme...** › shows the UX Theme chooser dialog



Changing the UX theme will update some of the colours that you can modify with the colour settings dialog.

3.10 Settings

Coverage 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 Global settings or Local settings.

Global and local settings

Global settings affect all data collected and its display throughout Coverage Validator.

Global settings are changed via the Data Collection Settings Dialog and the following 25 or so pages describe each available group of settings.

Local settings apply to controls and data displayed in each of the main display windows.

Local settings are found at the top of each relevant tab.

Other settings

There are a few more settings not included in the global settings dialog such as:

- User interface mode
- Session settings

3.10.1 Data Collection Settings

The Data Collection Settings dialog allows you to control all the global settings in Coverage Validator that affect the way data is collected and displayed. There are also local display options on most of the main tabs.

This page has a warning about use of the **Reset** button.

Opening the settings dialog

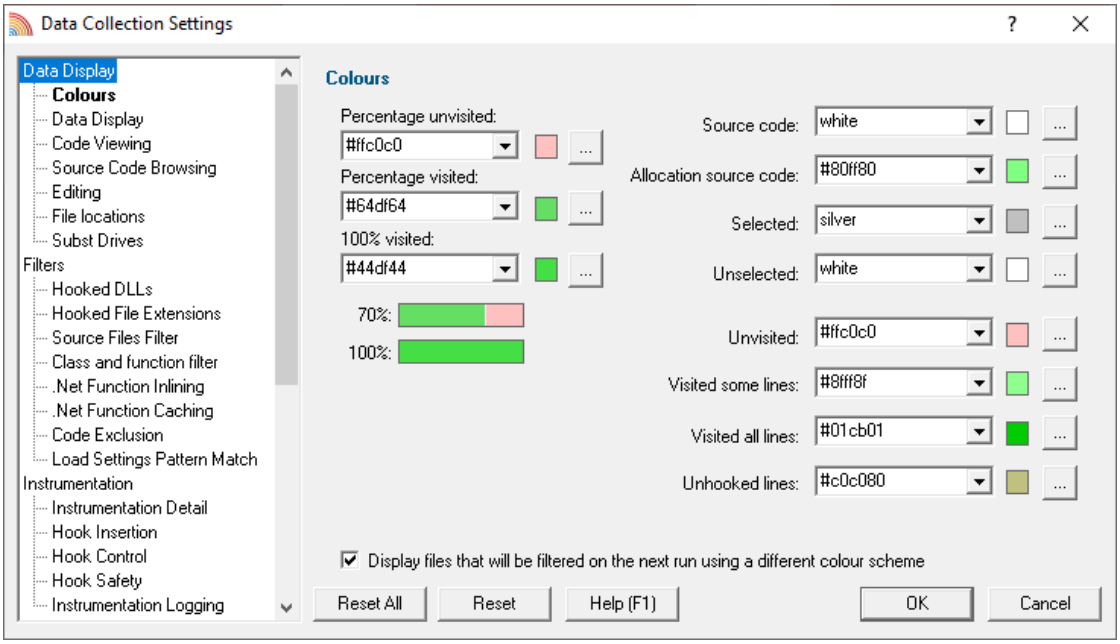
To view the settings dialog, choose **Settings** menu > **Edit Settings...**

Or use the option on the Session Toolbar:

Using the settings dialog

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

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

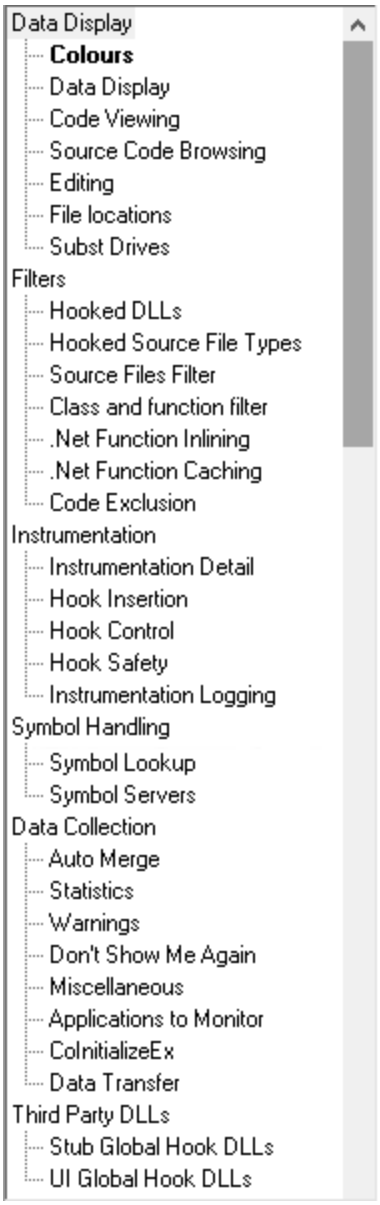


After selecting a topic, you can also use the cursor up and down arrow keys to change the selected item.

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, Coverage 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 **Reset All** and **Reset** buttons near the bottom left of the dialog which you can use to reset all global settings back to their default values.

The **Reset All** button resets **all global settings** in Coverage Validator, not just the settings visible on the current tab of the dialog.

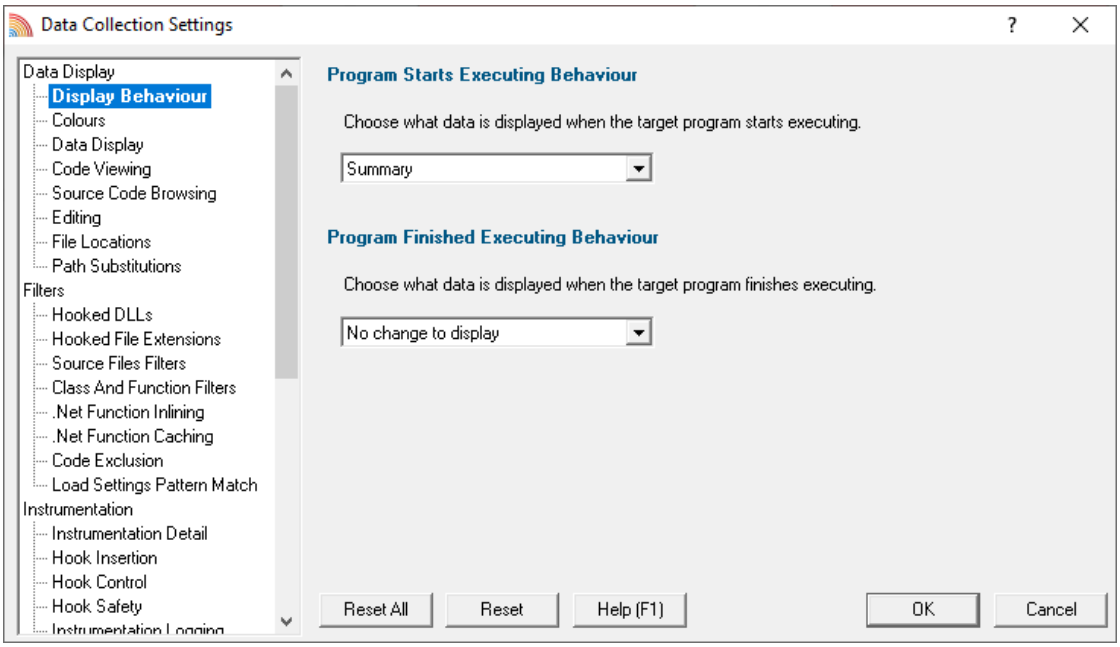
The **Reset** button resets just the settings visible on the current tab of the dialog.

3.10.1.1 Data Display

3.10.1.1.1 Display Behaviour

The **Display Behaviour** tab allows you control how which displays are shown when a program starts executing and when a program finishes executing.

The default options are shown below:



Coverage Validator can change the current display to any of the following displays.

- **Summary** › the main display
- **Coverage** › code coverage
- **Branch Coverage** › branch code coverage

- **Functions** › function code coverage
- **Directories : Types** › directory code coverage
- **DLLs : Sizes** › dll code coverage
- **Files and Lines : Locations** › code coverage for files and lines
- **Diagnostic : Diagnostic** › diagnostic information
- **Diagnostic : Stdout** › text collected from stdout
- **Diagnostic : Environment Variables** › environment variables from the program under test
- **Diagnostic : Child Processes** › processes launched by the program under test

Program Starts Executing Behaviour

When Coverage Validator starts monitoring the behaviour of an application the current display can automatically be switched to any of the displays listed above.

There is also the option not to change the display.

Program Finished Executing Behaviour

When Coverage Validator has finished processing all the information from the target application the current display can automatically be switched to any of the displays listed above.

There is also the option not to change the display.

The type of display that may interesting for collected data depends on the type of program that was executed. Native, .Net or Mixed mode. To accommodate this we provide one setting for each of the three program types.

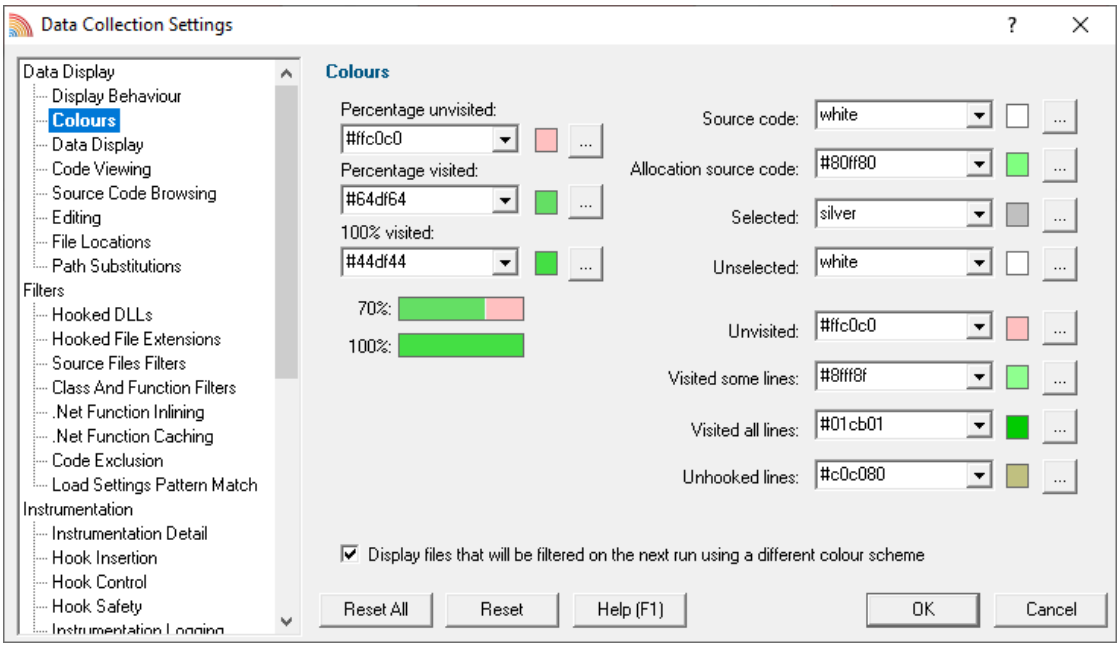
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.1.2 Colours

The **Colours** tab lets you choose the colours used to display statistics and source code.

The default colours are shown below:



The colours on the right are used for highlighting data in the Summary Tab; other tab data displays, and the source code.

The three colours on the left are used for percentage bars in the data displays.

Highlighting data that won't be shown on the next run

You can use other settings, such as the Source Files Filter, to exclude files or directories of source code from being analyzed.

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, you can opt to highlight the data that will excluded. The *Selected* colour (light grey in the above example) will be used for this.

- **Display files that will be filtered on the next run...** › If checked, this will highlight filtered data (the default)

In the example below, the Microsoft Visual Studio folder has just been added to the Source Files Filter, removing two of the files that were displayed:

File	% Visit...	Num Lines	Num Visited
Totals	**25.28%**	**352**	**89**
c:\program files (x86)\software verification\c++ coverage validator\cvex...	84.21%	19	16
c:\program files (x86)\microsoft visual studio 9.0\vc\atlmfc\include\atld...	66.67%	3	2
c:\program files (x86)\software verification\c++ coverage validator\cvex...	63.64%	33	21
c:\program files (x86)\software verification\c++ coverage validator\cvex...	57.14%	21	12
c:\program files (x86)\software verification\c++ coverage validator\cvex...	19.59%	194	38
c:\program files (x86)\microsoft visual studio 9.0\vc\atlmfc\include\afx.inl	0.00%	3	0
c:\program files (x86)\software verification\c++ coverage validator\cvex...	0.00%	79	0

Changing display colours

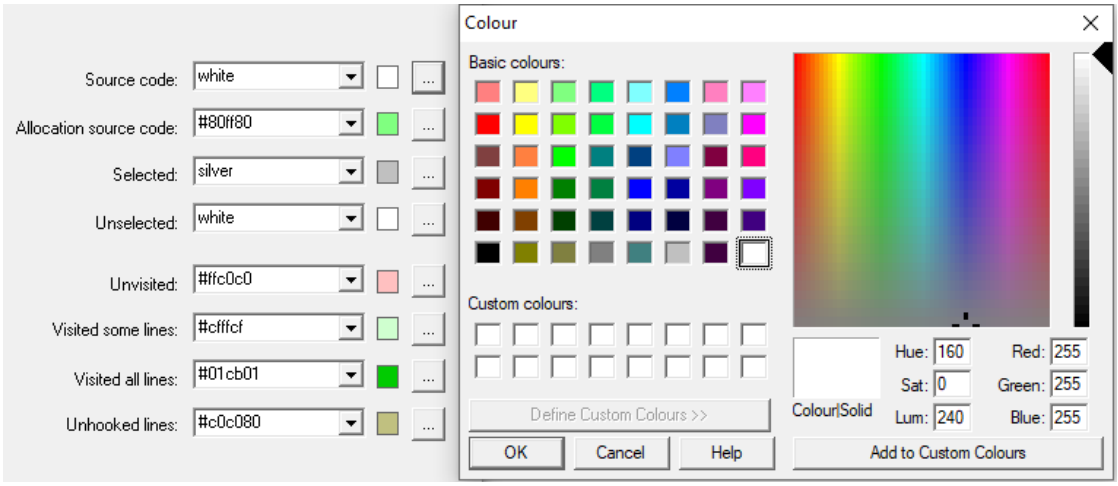
For each colour you can choose a predefined colour or make your own:

- Use the drop-down list 

› pick one of 16 predefined colours below

- Click the 

button › edit the colour using the standard colour dialog:



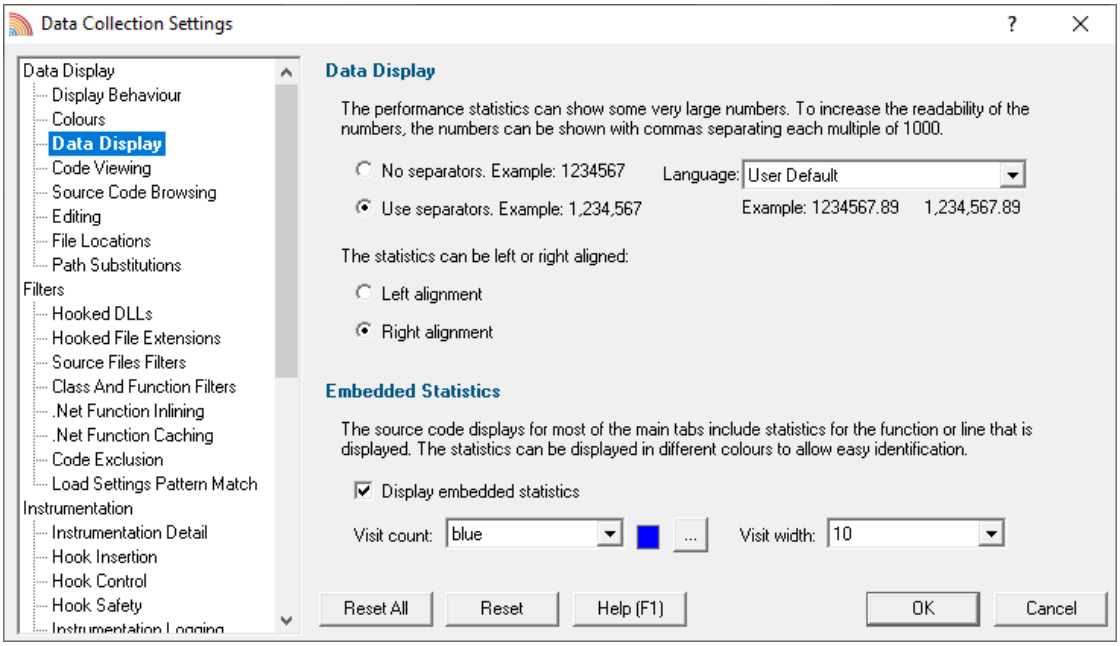
➡ See the Data Display Settings for changing the colour of the visit count shown in the source code..

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

Reset - Resets the settings on the current page.

3.10.1.1.3 Data Display

The **Data Display** page allows you to specify how data is shown on all the data 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 Coverage 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.

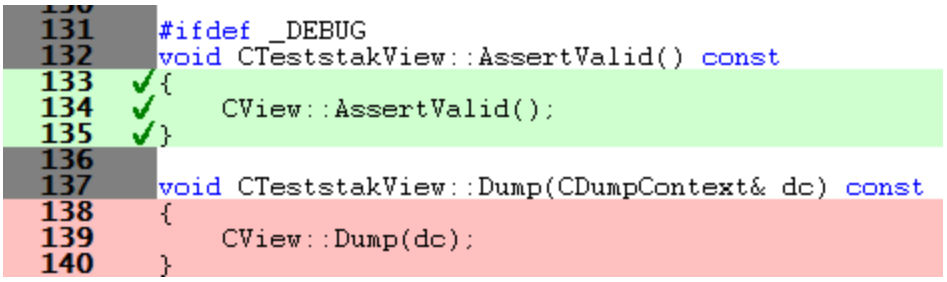
- **Left alignment / Right alignment** › Choose preferred alignment

Embedding visit counts in the source code

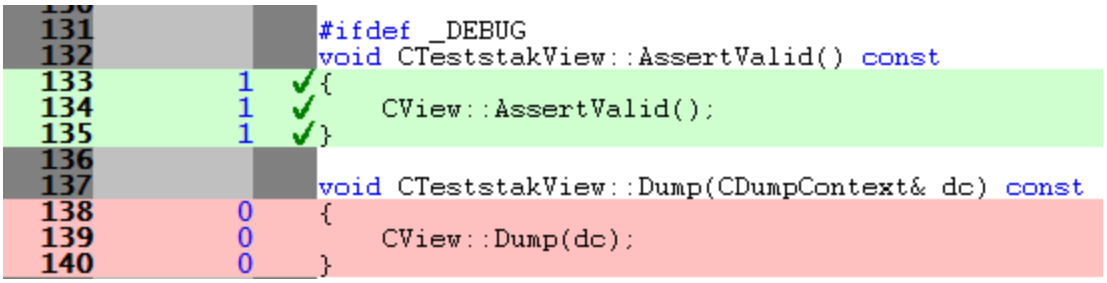
The source code displayed on most of the main tabs can show a colour-coded visit count in-lined to the left of the source code.

You can choose a colour to display these statistics to distinguish them from the code and line numbers.

Without embedding statistics:



With embedded statistics:



- **Display embedded statistics** › Choose whether to include the visit count

  If displayed (the default), you can choose a preferred text colour, and a width for the column:

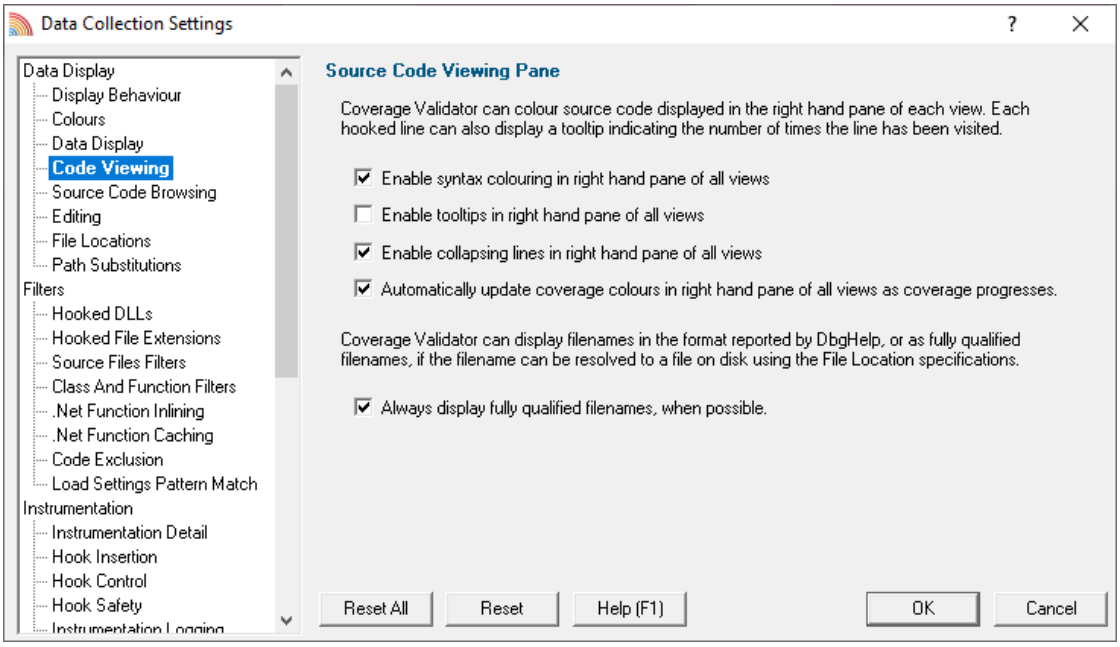
  - **Visit count** › Set your preferred colour
  - **Visit width** › Change the column width

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

Reset - Resets the settings on the current page.

3.10.1.1.4 Code Viewing

The **Code Viewing** tab allows you to configure some options when viewing the source code pane.



Source Code Viewing Pane

The source code pane is displayed on the right hand side of most of the main tabs:

- Coverage
- Branch Coverage
- Functions
- Directories
- DLLs
- Unit Tests
- Files and Lines

Syntax highlighting

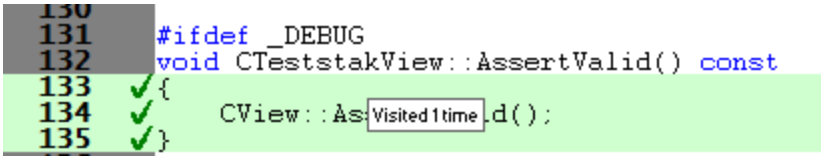
Usually the source code shows a colour coded syntax:

- **Enable syntax colouring...** > Untick if you don't want colour coded syntax

Visit count tooltips

Tooltips showing the visit count may be useful if you've disabled the display of the visit count in the source code.

- **Enable tooltips...** › Tick if you want tooltips like the one below



Collapsing lines

You may want to temporarily collapse sections of source code you're not interested in.

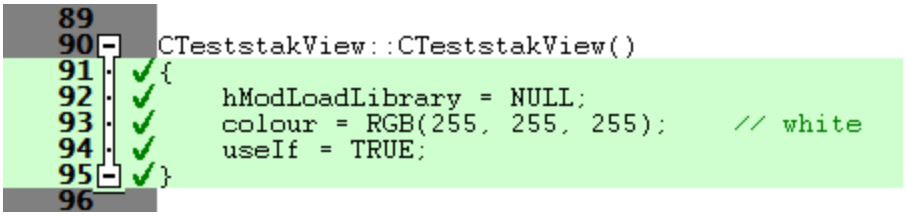
- **Enable collapsing lines...** › Check this to enable collapsing lines (see below)

To create a collapsible region:

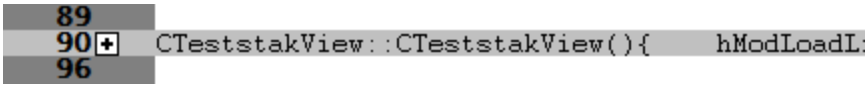
Left click drag (up or down) in the dark grey margin immediately to the left of the source code. Release the mouse at the end of the region you want to collapse.

Page up/down and mouse-wheel scrolling works during the drag if you want to select more than you can see.

A 'collapse line' will appear as below:



To collapse the region, *left* click anywhere on the collapse line, and the region will reduce to a single grey highlighted line:



To expand the region again, *left* click the ⊞ symbol.

To remove the collapsible status, *right* click anywhere on the collapse line.

A collapsible region that spans right across another region will take precedent and the smaller regions will be removed.

Creating a region that partially overlaps another will merge them into one.

Auto update

Viewing the source code while your program is still running may cause the view to update as you browse.

Updates happen once a second by default, but you can change this interval in the local settings of each tab view.

Reasons you might want to also disable the auto-update could be:

- You want to keep an eye on the statistics, but still browse the source code without it updating every second
- You may find that very large files are slow to update regularly.

To disable the auto update:

- **Automatically update coverage colours...** › Uncheck this to prevent the source code auto updating

Without auto-update, you'll need to use the manual refresh button found on each view.

Fully qualified file names

If the display of file names in the data is of a shortened format (as reported by DbgHelp), Coverage Validator can determine the fully qualified file name based on any source file locations you may have provided.

- **Always display fully qualified filenames...** › Check to show complete file name paths where possible

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

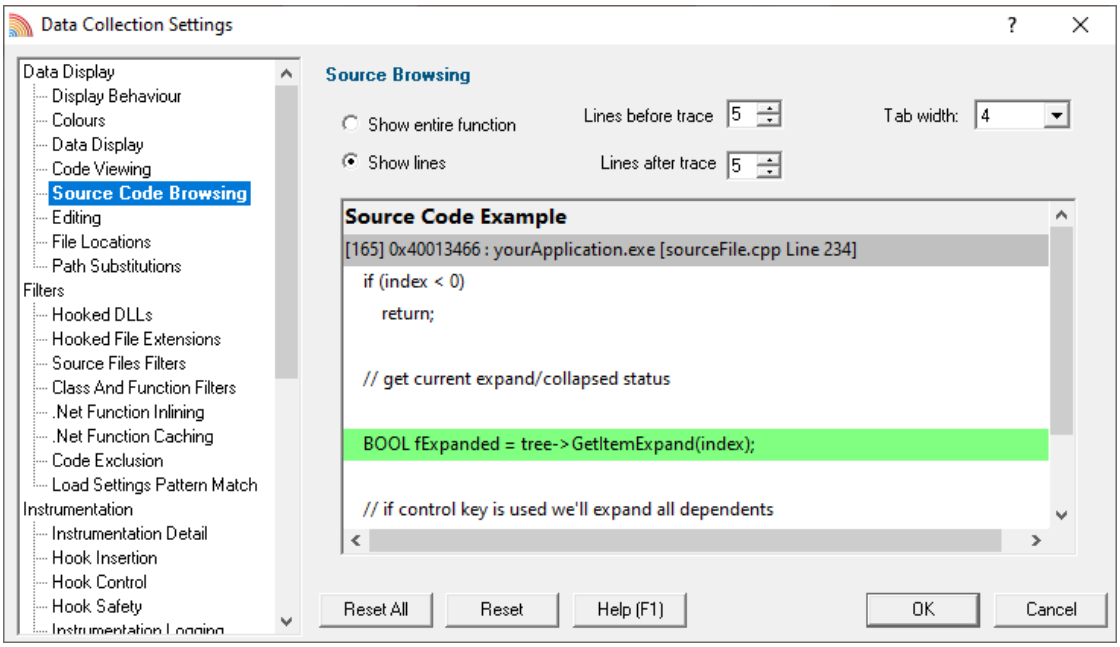
Reset - Resets the settings on the current page.

3.10.1.1.5 Source Browsing

There a few areas in Coverage Validator where you can view snippets of source code, such as in the Query tool results.

The **Source Browsing** tab allows you control how much source code is displayed and the indentation.

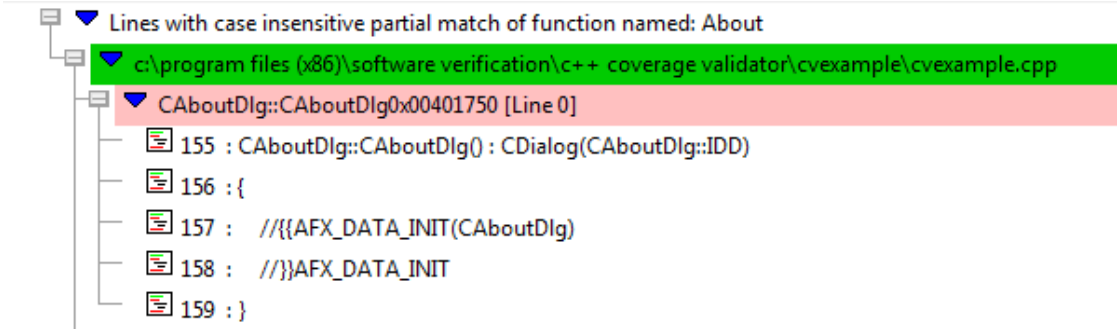
The default options are shown below:



Source browsing

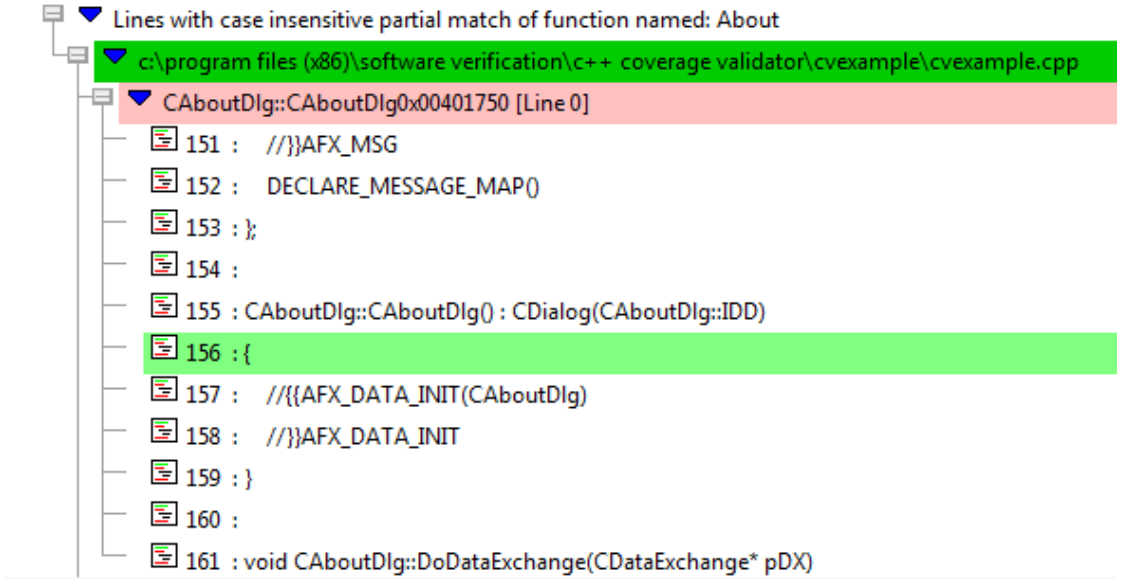
When viewing sections of source code, you can choose to see the whole function or a few lines either side of the line of interest.

- **Show entire function** › shows the whole function source as below:



- **Show lines** › shows a given number of lines before and after the point of interest:
  - **Lines before trace** › number of lines *before*, from 0 to 100
  - **Lines after trace** › number of lines *after*, from 0 to 100

  The default is to show 5 lines above and below, as in this example:



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 display 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 control the tab size

- **Tab width** › set the tab size between 1 and 16 characters

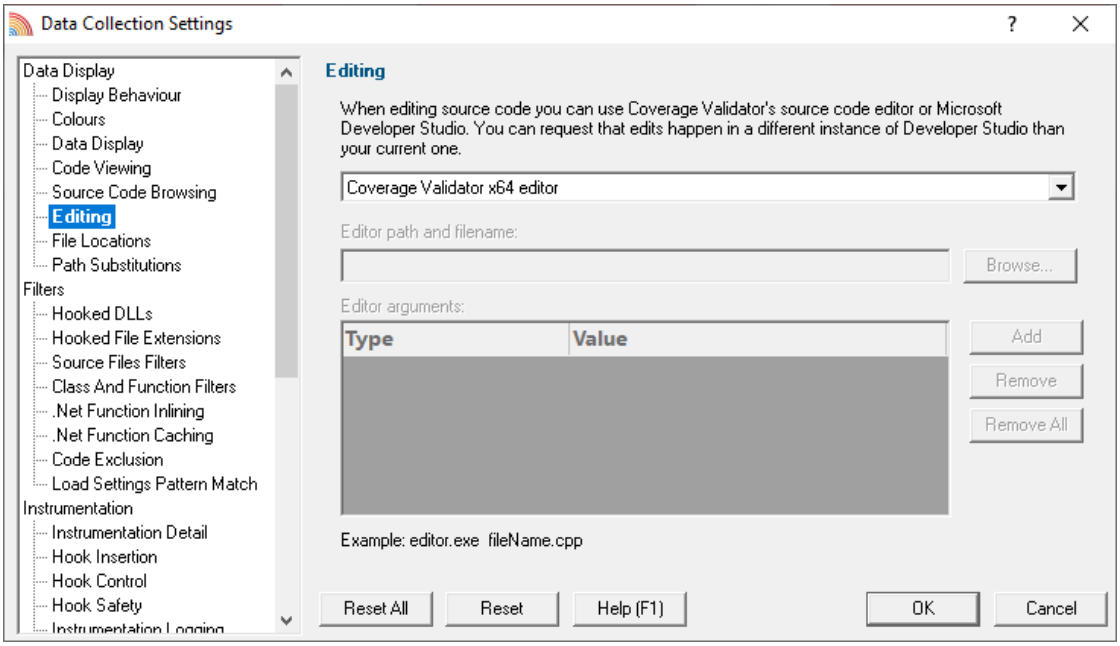
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.1.6 Editing

The **Editing** tab allows you to configure which editor Coverage Validator will use for editing source code.

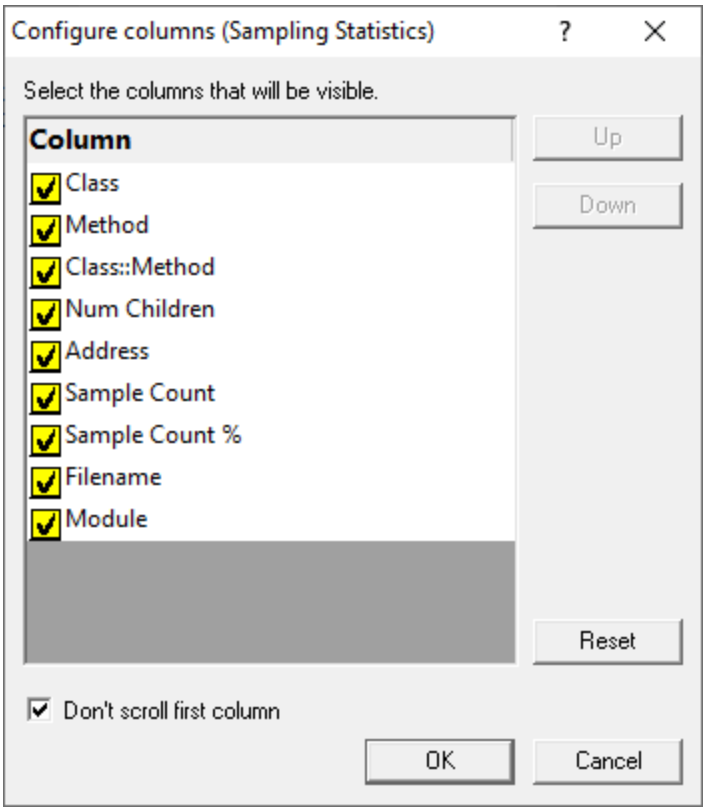
The default settings are shown below:



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:



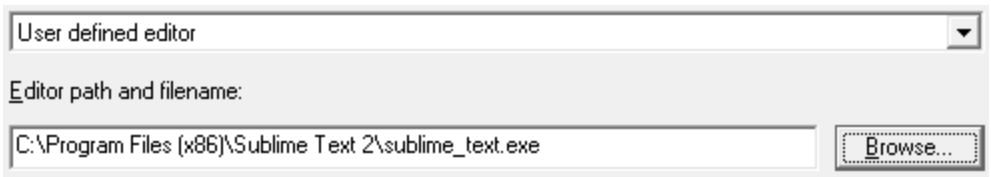
When choosing one of the editors listed, you can request a currently open instance (e.g. the same one you are using to develop your application), or to open a new instance.

➜ SCiTE is included in the list of editors, but there are many text editors that can be used for source code on windows. Wikipedia has a comparison of editors including their programming feature support

Editing with your preferred editor

We've all got our favourite editors! To use yours:

- Select **User defined editor** from the list of options › enables the fields below
- Enter the **Editor path and filename** or just **Browse** › choose the executable for your preferred editor



Now when you want to edit source code, that editor will be opened, but typically you'll need to specify some command line arguments with which to start the editor.

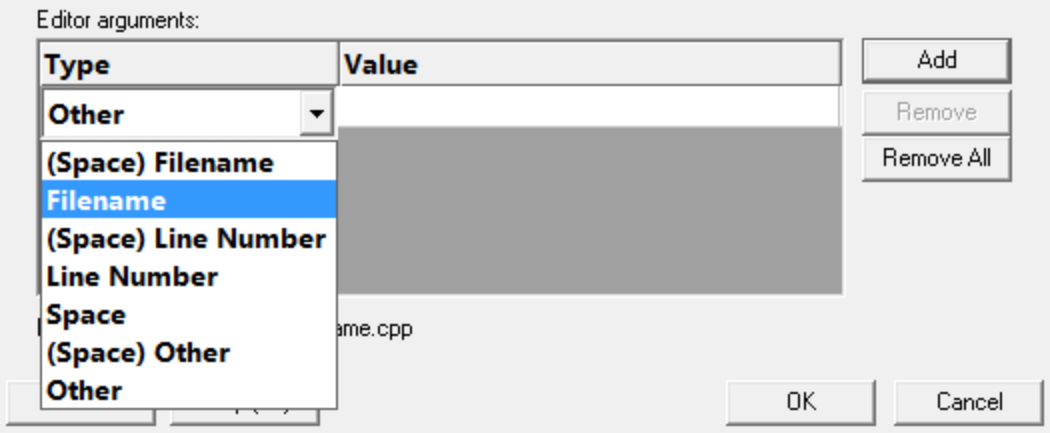
Starting your preferred editor with command line arguments

By default, just the file name is passed as a command line argument to the editor.

Depending on the editor, you may need to tailor the arguments, for example if you want the file scrolled to a particular line.

The arguments can be specified by adding them to the table provided, one at a time and in the order required

- **Add** › adds a row to the **Editor arguments** table › select an argument **Type** from the following options



The possible arguments include:

- **(Space) Filename** › appends a space followed by the filename
- **Filename** › appends just the filename

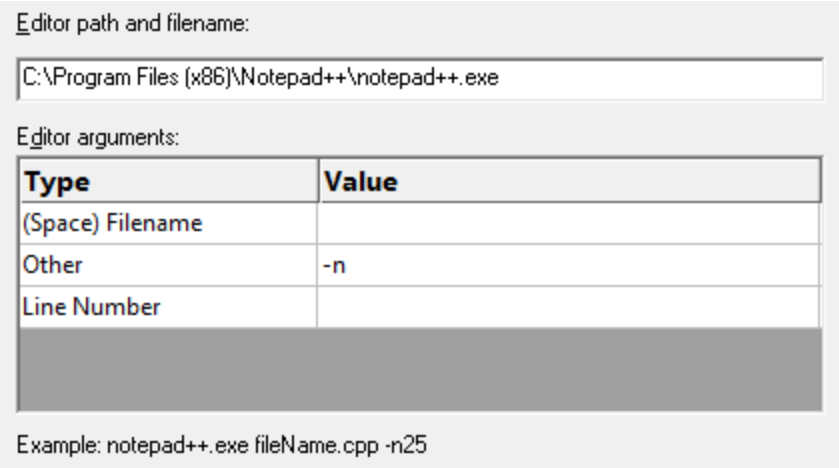
- **(Space) Line Number** › a space followed by the line number
- **Line Number** › just the line number

- **Space** › a space

- **(Space) Other** › a space followed by the text typed in the **Value** column of the list
- **Other** › just the text typed in the **Value** column of the list

Only the *Other* options need an entry in the Value column.
You will need to press **Return** after entering the value otherwise the entry won't get recognized.

The example below configures NotePad++ to edit a file at the required line using the -n switch



As you modify the arguments an example command line is shown below the list.

Managing the command line arguments

Edit a **Type** or **Value** by double clicking the entry. The usual controls apply for removing list items:

- **Remove** › removes selected arguments in the list
- **Remove All** › removes all arguments, clearing the list

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

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

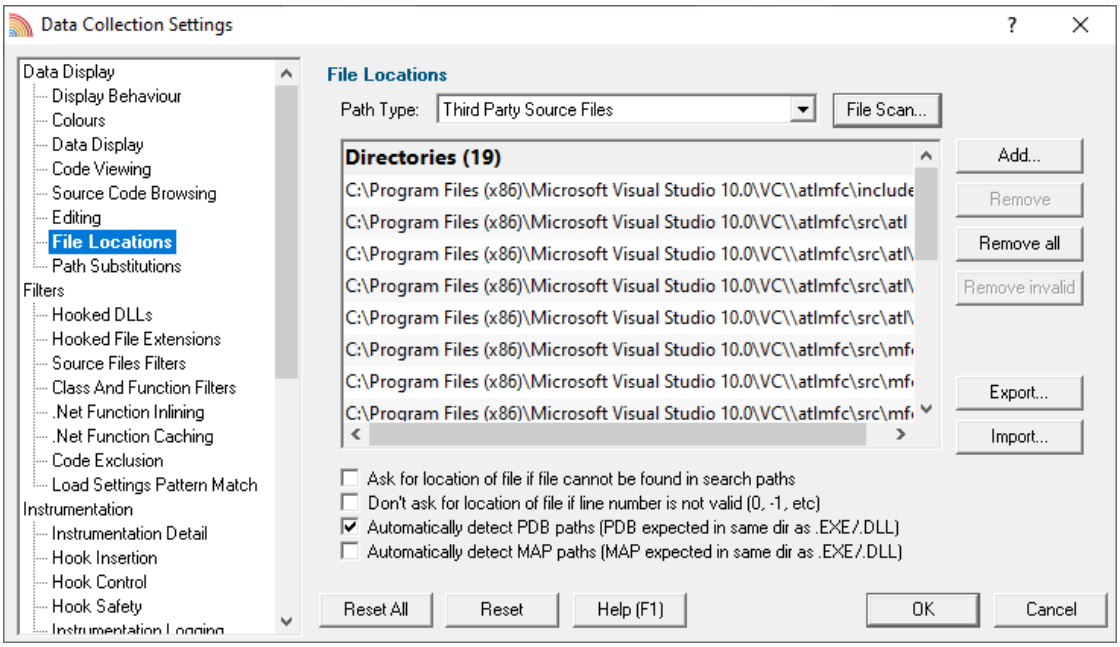
Reset - Resets the settings on the current page.

3.10.1.1.7 File Locations

The **File Locations** tab allows you to specify which directories Coverage 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:



File locations

Sometimes the information Coverage Validator has access to consists of the file name, but not the directory.

When this happens Coverage Validator scans a set of directories that it knows about in order to find the file.

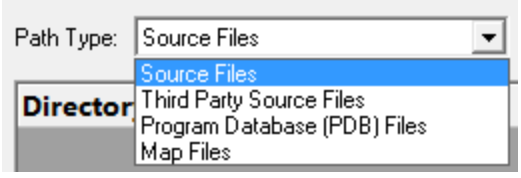
The options below allow you to specify those directories that should be searched for source files, PDB files and MAP files.

If a file can't be found, you'll get prompted for a location, but you can control this below as well.

Setting directories for a path type

There are four 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 if you don't want to, or if you just don't have them. Nor do you have to give directories for *all* the path types.

Prompting for file locations

Whenever a file still cannot be found, then the default action is for a dialog to ask you where it is.

To avoid frequent user interruption, it is recommended that the directories for source code files (yours and third party) are specified, enabling Coverage Validator to automatically load source code for browsing.

If however, you don't want to be prompted for locations, you can disable that too.

- **Ask for location of file...** › untick to stop prompting for file locations

Even when prompting is switched on, it can still happen that the line in question is invalid anyway, e.g. line number 0 or -1.

The default is not to prompt for invalid lines, but if you want to know when that happens, just switch that behaviour off.

- **Don't ask for location of file if line number is not valid...** › untick to be prompted for invalid lines anyway

PDB (program database) file paths

Normally PDB search paths are automatically generated, based on the same directories that .exe and .dll files are found in:

- **Automatically detect PDB paths** › automatically detect PDB locations (the default)

However, it is recommended that you specify paths for PDB (program database) files, especially if your build environment dictates that PDB files are kept in different directories to their binaries.

If you don't automatically generate PDB paths and you don't specify any paths for PDBs, the search path will be defined as the current directory plus any paths found in the following environment variables:

- _NT_SYMBOL_PATH
- _NT_ALTERNATE_SYMBOL_PATH
- SYSTEMROOT

MAP file paths

It's recommended that you specify paths for Map files if your build environment means they are kept in different directories to their binaries.

If you don't specify any paths for Map files, then search paths are automatically generated, based on the same directories that .exe and .dll files are found in.

Manually adding path type directories

Once you have chosen your path type you can modify the list of files for each path type in the following ways:

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

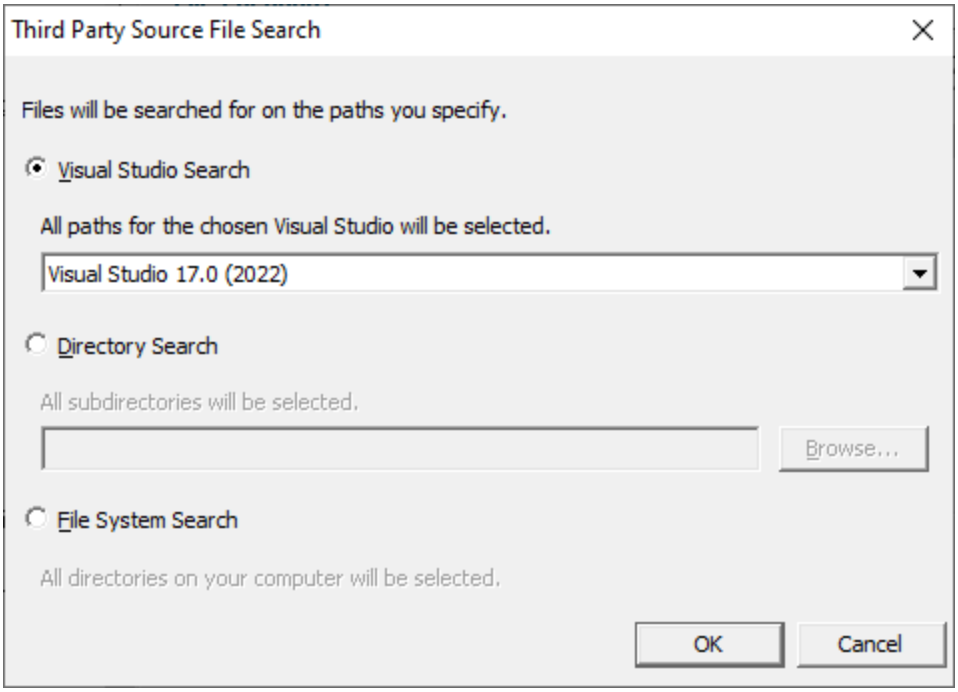
Edit a directory path by double clicking the entry. The usual controls apply for removing list items:

- **Remove** › removes selected items from the list
- **Remove all** › clears the list
- **Remove invalid** › removes all items that are not valid directories from the list

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

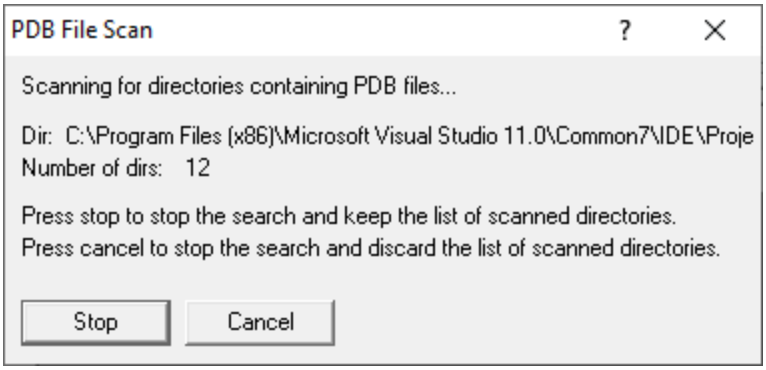
Scanning for directories to add

The **File Scan...** button displays the File Search dialog to provide three ways of specifying the files to scan.

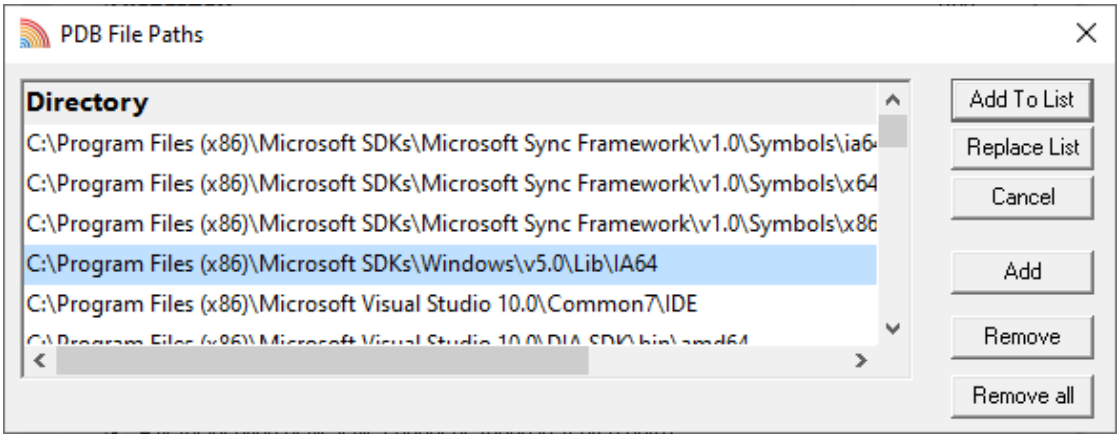


- **Visual Studio Search** › choose the version of Visual Studio › **OK** › starts a scan for directories related to that version of Visual Studio
- **Directory Search** › **Browse...** displays a directory browser › **navigate** to a location you want to scan within › **OK** › starts a scan for directories
- **File System Search** › **OK** › starts a scan of all drives for directories containing files

All options will bring up a **File Scan** dialog indicating number of relevant directories found, and giving you a chance to **Stop** or **Cancel** the scan at any time:



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



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 *.cvxfl file › **Open** › loads the hooking rule and the list of modules

 The exported file can be used with the -fileLocations command line option.

Export file format

The file format is plain text with one folder listed per line. Sections are denoted by a line containing `[Files]` (for source code files), `[Third]` (for third party source code files), `[PDB]` etc.

Example:

```
[Files]
c:\work\project1\
[Third]
d:\VisualStudio\VC98\Include
[PDB]
c:\work\project3\debug
c:\work\project3\release
[MAP]
c:\work\project3\debug
c:\work\project3\release
```

Checking directory scanning order

To see the order in which the *DbgHelp.dll* process checks directories to find symbols, see the diagnostic tab, showing *DbgHelp debug* in the drop-down.

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

Currently, the four checkbox items at the bottom of this page are **not** reset as part of the global settings.

Reset - Resets the settings on the current page.

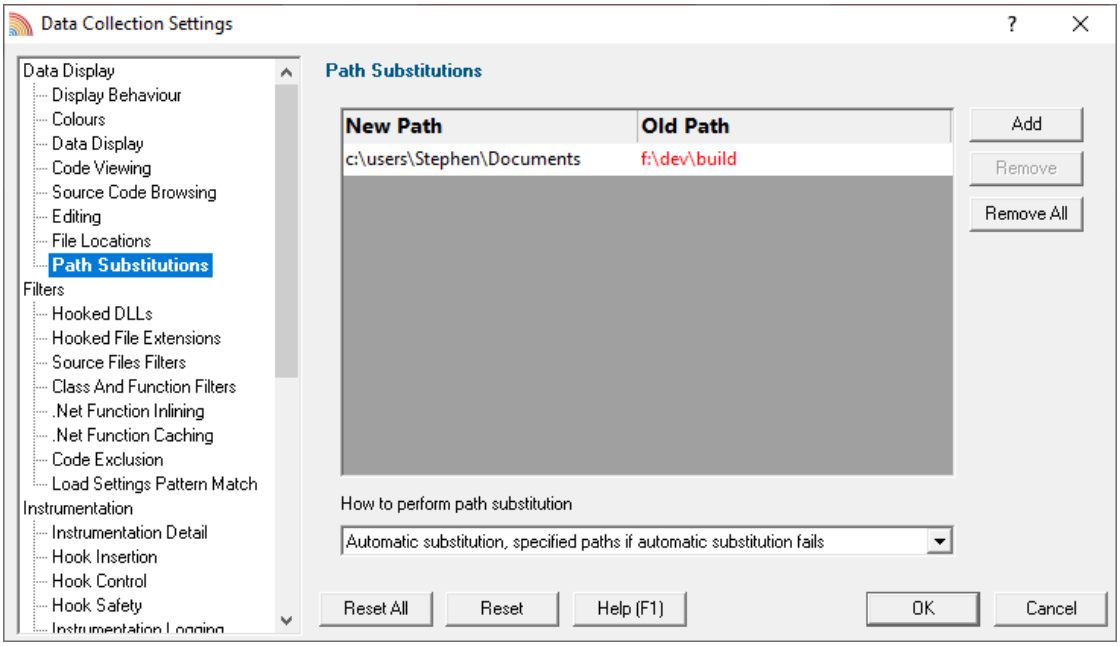
Currently, the four checkbox items at the bottom of this page are **not** reset.

3.10.1.1.8 Path Substitutions

The **Path Substitutions** tab allows you to specify how file paths can be modified to account for copying a build from a build machine to a development machine or test machine.

File paths need to be modified otherwise the debug information will point to source code in locations that don't exist on the new machine.

There are no substitutions set up by default:



Path Substitutions

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

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

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

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

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

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

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

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

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

Example: Changed disk drive

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

Example: Project copied to a new location

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

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

Path Substitution Method

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

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

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

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

IMPORTANT

File Substitution is for updating the source code paths in the debugging information as the code coverage information is being collected.

It is not for fixing up paths because you accidentally collected coverage using one set of paths on computerA and a different set of paths on computerB and then want to fix those paths (for example when merging collected data).

The correct solution for this scenario is to fix the paths on both machines to match each other. You are in control of this scenario, ***therefore you can set it up correctly***.

It is not the same as the build machine vs dev machine environment where the build directory may change with each rolling build and the dev machine may have multiple branches on it, each with a different path.

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

Reset - Resets the settings on the current page.

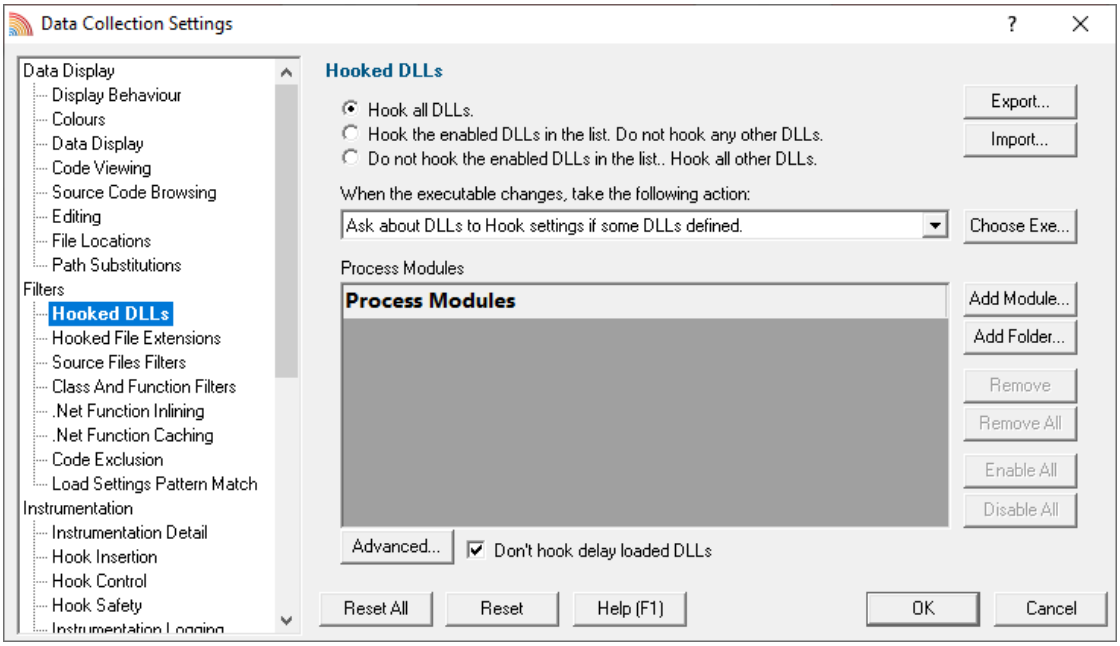
3.10.1.2 Filters

3.10.1.2.1 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, Coverage Validator will try to hook all DLLs and .EXEs used by your application, but you can choose to list only those which should be included or excluded:

- **Hook all DLLs** › hook everything - ignoring the settings in the list

- **Hook the enabled DLLs in the list** › hook only the ticked modules listed
- **Do not hook the enabled DLLs in the list** › *ignore* all the ticked modules in the list, and hook everything else

Populating the process modules list

The process modules list should specify the following items to be included or excluded from hooking in the target application

- DLLs
- .EXEs
- folders *containing* DLLs and .EXEs

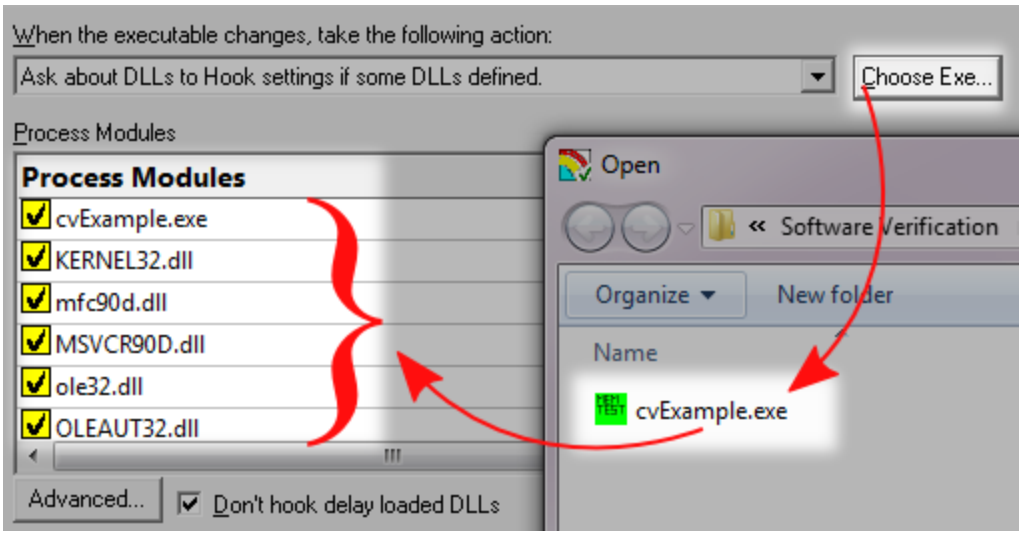
Initially the list is empty as the default option is to hook all DLLs and ignore the list. You can add modules to the list by:

- automatically adding modules on which your application is dependent
- manually adding modules or folders
- editing modules or folders already in the list

Automatic module addition

You can automatically populate the list with all the dependent modules for your application:

- **Choose Exe...** › navigate to your application and click **Open** › all the process modules appear in the list



Manual module addition

You can also manually add one or more modules or a folder to the list.

- **Add Module** › navigate to the DLL or EXE and click **Open** › all the selected items are added
- **Add Folder** › navigate to the folder and click **OK** › the folder is added to the list

Manual addition might be useful for example if you use `LoadLibrary()` to load a DLL rather than linking it, as this would not be picked up automatically by the *Choose Exe...* method.

By default, all the modules are ticked in the yellow checkboxes.

Any DLLs in the list override the DLL Hook Insertion settings on the Hook Insertion tab.

 Note that ticked modules or folders are either **in**cluded or **ex**cluded depending on the hooking rule above

Altering existing module names

Although you can't add blank entries to the list and edit them, you *can* edit existing items in the list by double clicking on an entry:

- enter only the module name, not the path
- you can use wild-cards like MFC*.dll, but only for DLLs, not folders

Managing the process modules list

The usual controls apply for removing or changing the enabled state of items in the list:

- **Remove** › removes selected items in the list
- **Remove All** › removes all items, clearing the list
- **Enable All** › ticks all items in the list for applying to the hooking rule
- **Disable All** › unticks all items in the list, meaning they won't apply to the hooking rule

Alternatively, press 

to delete selected items, and 

+ 

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 *.cvx file › **Open** › loads the hooking rule and the list of modules

The exported file can also be used with the -dllHookFile command line option.

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 coverage results, so you can opt to be warned about the DLLs being hooked whenever the target application changes between sessions (using the dialog below).

The choices in the drop down list are only applicable when the application changes:

- **Ask about DLLs to Hook settings if some DLLs defined**

  You'll only be asked about the settings if you defined some DLLs in the list *and* if the hooking rule is not set to *hook all DLLs*

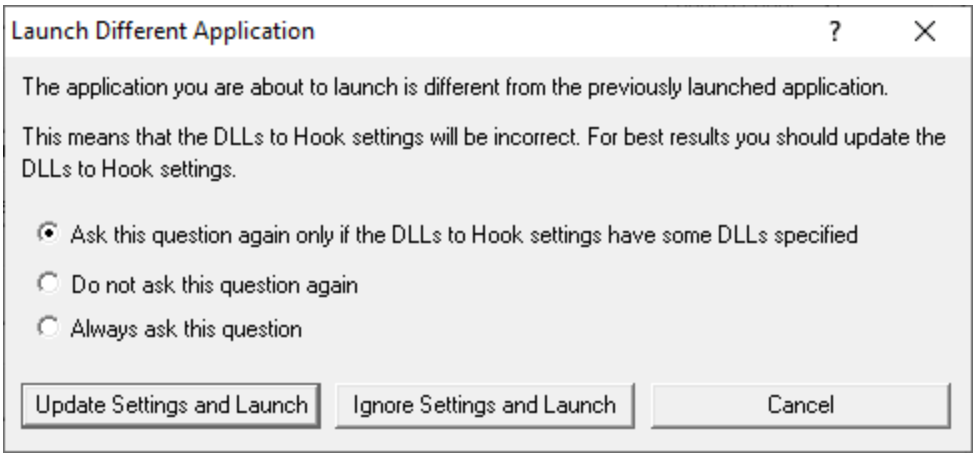
- **Always ask about DLLs to Hook settings**

  You'll always be asked about the settings - *whatever* the other settings are.

- **Never ask about DLLs to Hook settings**

The 'Launch Different Application' dialog

When being asked about the hooked DLL settings, you'll see the following dialog:



You can update the settings; ignore them and launch anyway, or just cancel the launch:

- **Update Settings and Launch** › **edit** the settings › click **OK** › the application will be launched
- **Ignore Settings and Launch** › the application will be launched without updating the settings
- **Cancel** › won't launch the application

To change when you are asked this question, just choose the appropriate option in the dialog.

Advanced options

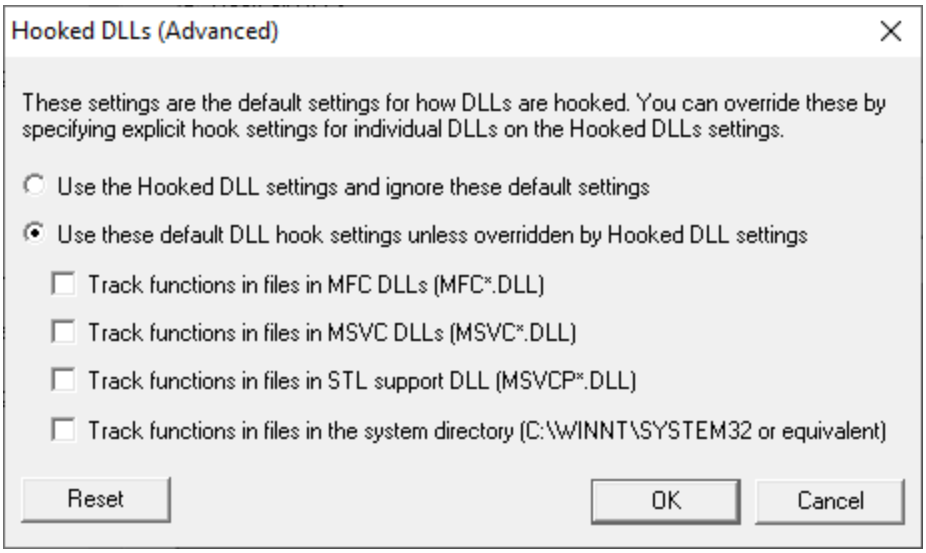
There are a few system DLLs which are generally not much use when hooked for code coverage.

The source code for these DLLs is unlikely to be available, and even if it is, your application is unlikely to be able to influence the code coverage.

Consequently, by default these system files are not hooked - unless you override that by specifically including them in the process modules list above.

With the advanced options below, you can change this behaviour if you need to, letting you specify which DLLs will be processed according to the rules for Source Code Line Hook Insertion.

- **Advanced** › shows the Hooked DLLs (Advanced) dialog for function and line profiling



In the Hooked DLLs (Advanced) dialog:

- **Use the Hooked DLL settings...** › if selected then *only* the general settings apply (i.e. those on the Hooked DLLs tab)
- **Use these default DLL settings...** › if selected then the checkboxes control the default behaviour for the relevant DLLs

  The default is *not* to track functions in files in the following:

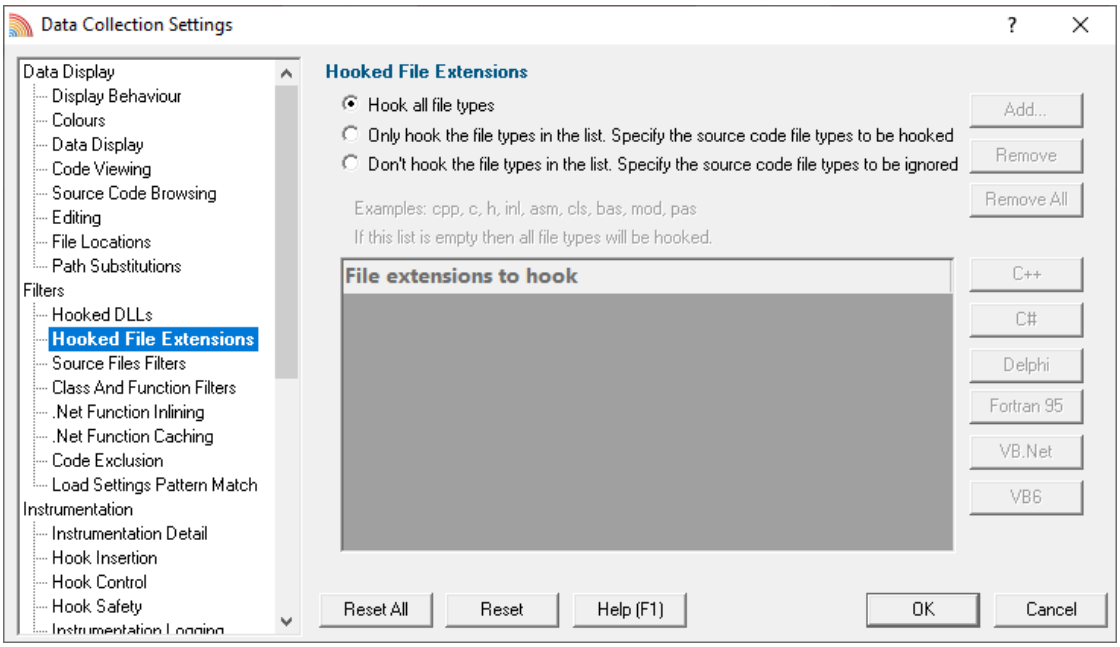
  - **MFC DLLs...** › check to cover `MFC*.DLL` or leave unchecked to ignore
  - **MSVC DLLs...** › check to cover `MSVC*.DLL` or leave unchecked to ignore
  - **STL support DLL...** › check to cover `MSVCP*.DLL` or leave unchecked to ignore
  - **the system directory...** › check to cover anything in `C:\WINDOWS\SYSTEM32` or leave unchecked to ignore them
- **Reset** › resets the settings in the dialog

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

Reset - Resets the settings on the current page.

3.10.1.2.2 Hooked File Extensions

The **Hooked File Extensions** page lets you specify which source code file types should be hooked or not.



Which file types to hook - the hooking rule

By default, Coverage Validator will try to hook all file types used by your application, but you can choose to list only those which should be included or excluded:

- **Hook all file types** › hook everything - ignoring the settings in the list
- **Only hook the file types in the list** › hook only the listed file types
- **Don't hook the file types in the list** › *ignore* all the listed file types, and hook everything else

File extensions to hook

By default, all source code files are hooked, but you can change this by specifying only those file extensions which *should* be hooked.

For example, you may want to include C++ source and header files but exclude C source files with the .c extension:

```
cpp
h
```

or

```
cpp
cxx
hpp
hxx
h
```

The usual controls apply for adding, removing or changing items in a list:

- **Add...** › adds a new row to the list › enter the extension you want to allow
- **Remove** › removes selected items in the list
- **Remove All** › removes all items, clearing the list
- **C++** › adds the file extensions for C++
- **C#** › adds the file extensions for C#
- **Delphi** › adds the file extensions for Delphi
- **Fortran 95** › adds the file extensions for Fortran 95
- **VB.Net** › adds the file extensions for the VB.Net
- **VB6** › adds the file extensions for the Visual Basic 6

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

Clear the list to hook all source file types again.

 Do not include the dot in the extension.

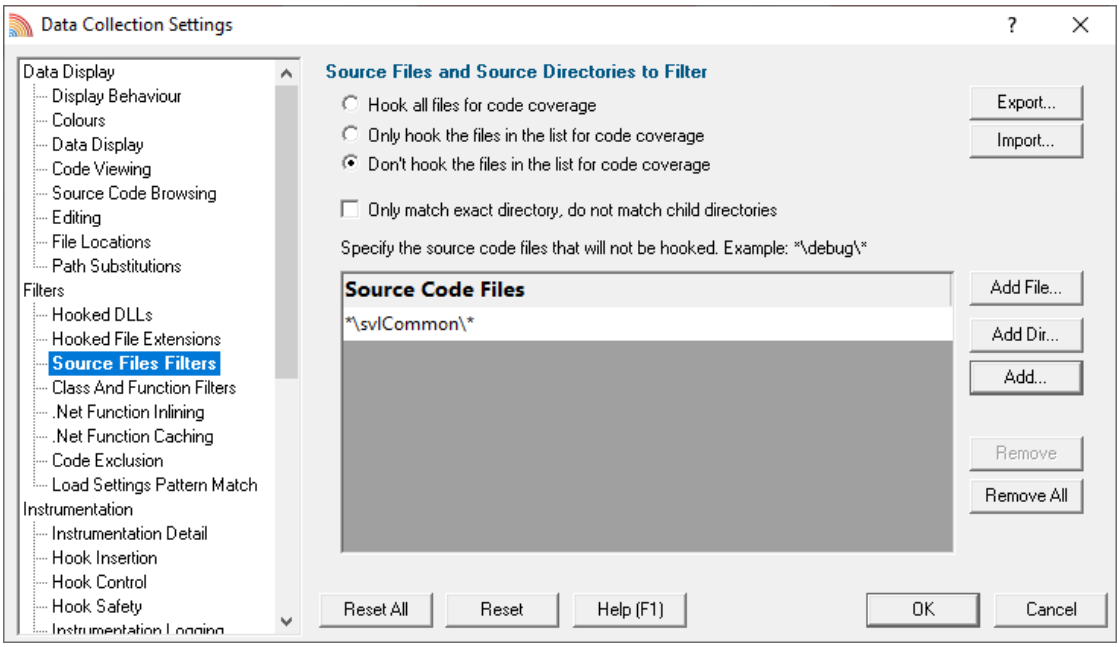
The filters take effect on the next session. If you're in the middle of a session and existing views show data that will be excluded in the next session, then that data will be highlighted according to the colours settings

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

Reset - Resets the settings on the current page.

3.10.1.2.3 Source Files Filters

The **Source Files Filter** page allows you to specify which source files should not be hooked.



Listing files to exclude or include

By default, Coverage Validator will try to hook all source files used by your application, but you can choose to list only those which should be included or excluded

- **Hook all files...** › hook all files
- **Don't hook the files in the list...** › hook everything *except* the files in the list
- **Only hook the files in the list...** › hook *only* the source files listed

In the list you can include files or directories. If using directories you may want only that specific directory, or everything underneath it (the default):

- **Only match exact directory...** › check this so as *not* to recurse into child directories

Managing the list of files

Add files or directories on disk:

- **Add File...** › navigate to the source files › select one or more files › click **Open** › all the selected items are added
- **Add Dir...** › navigate to the folder › select it and click **OK** › the folder is added to the list

Or manually enter a file:

- **Add...** › a new row is added to the list › Type the file path › press **Return** › the file is added to the list

Remove items as normal:

- **Remove** › removes selected items in the list
- **Remove All** › removes all items, clearing the list

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 source files can be quite complicated to set up, 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 filtered source files to the file
- **Import...** › **navigate** to an existing *.cvxft file › **Open** › loads the filtered source files

The exported file can be used with the -sourceFileFilterHookFile command line option.

Wildcards

All file and directory specifications can contain the * wildcard.

Some examples will help:

Consider a project with three source directories:

```
e:\dev\srcMain\
e:\dev\srcCommon\
e:\dev\srcCustom\
```

Possible filters could be:

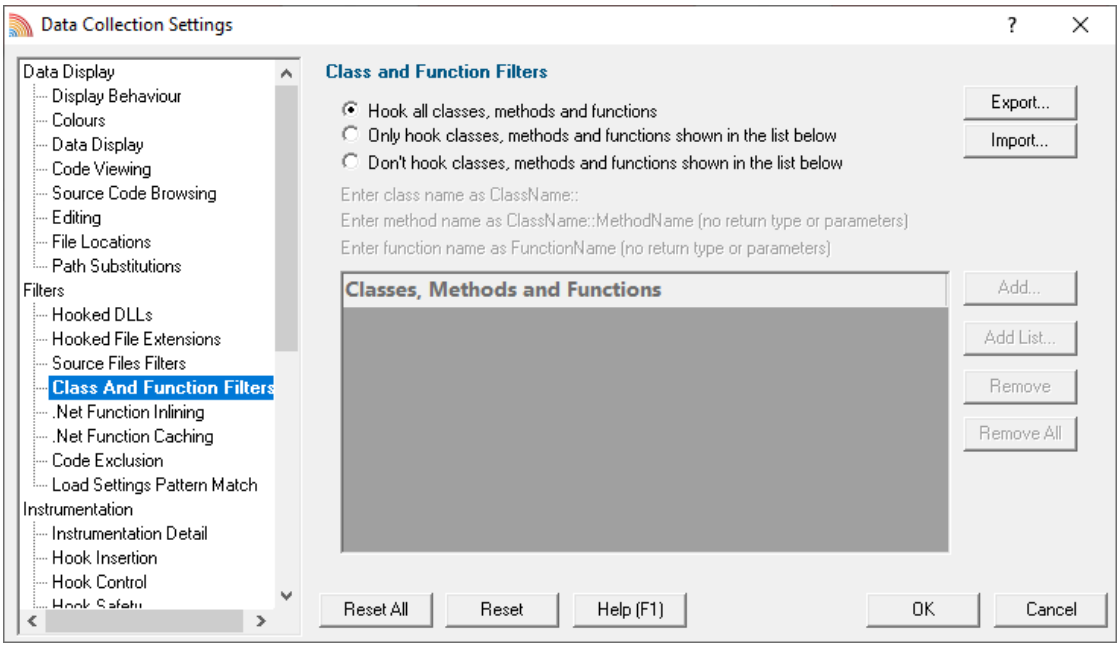
```
e:\dev\src*\
*\src*\
*\srcC*\*
```

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

Reset - Resets the settings on the current page.

3.10.1.2.4 Class and Function Filters

The **Class and function filter** page allows you to set specific classes, methods and functions to be included in, or excluded from, the hooking process.



These filters apply only to the DLLs and files that you have not already excluded via other filters.

Listing what to exclude or include

By default, Coverage Validator will try to hook all classes, methods and functions in the hooked files, but you can choose to list only those which should be included or excluded.

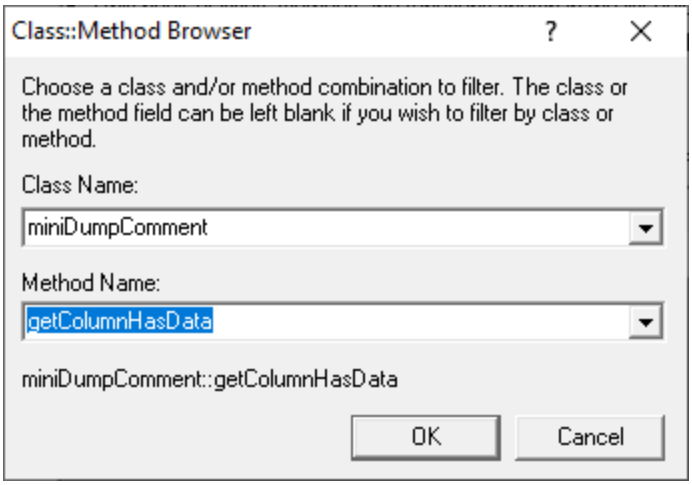
- **Hook all...** › hook everything (the default)
- **Only hook ...** › hook *only* the classes, methods and functions listed
- **Don't hook ...** › hook everything *except* the classes, methods and functions listed

Managing the list

Choose from known methods:

- **Add...** › shows the **Class::Method Browser** dialog below

  Type or choose a class and/or method from the pre-populated drop-down list of all functions:

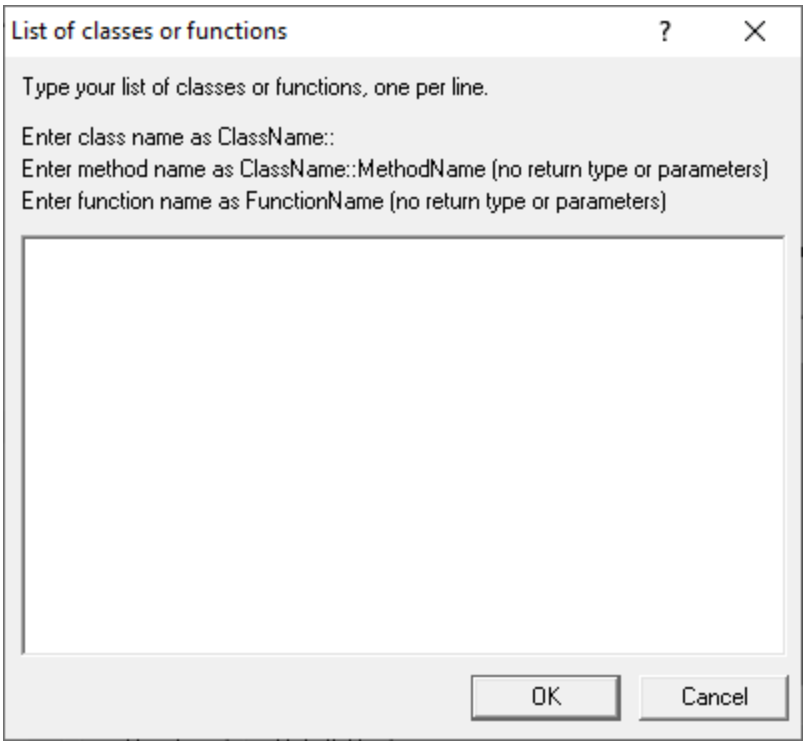


When typing an entry, you'll see a message at the bottom of the dialog if the symbol doesn't exist:

Choose from known methods:

- **Add List...** > shows the **List of classes or functions** dialog below

  Enter any number of plain class::method or function names or just paste them in.



The list will not be validated against known functions. Do not include return types, brackets, parameters or other details.

Remove items as normal:

- **Remove** › removes selected items in the list
- **Remove All** › removes all items, clearing the list

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

Exporting and importing

Since the list can be quite complicated to set up, 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 filtered list to the file
- **Import...** › **navigate** to an existing *.cvxc file › **Open** › loads the filtered list

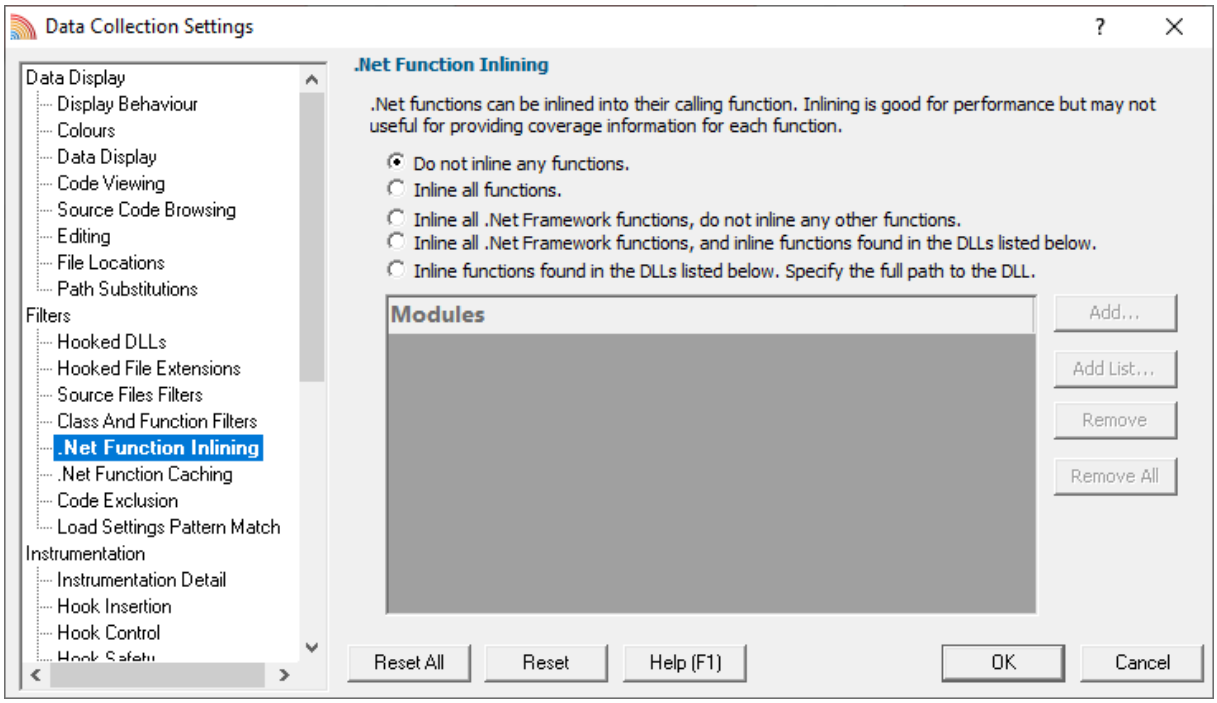
The exported file can be used with the -classAndFunctionFile command line option.

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

Reset - Resets the settings on the current page.

3.10.1.2.5 .Net Function Inlining

The **.Net Function Inlining** tab allows you to configure how .Net inlines functions.



Inlining of .Net functions can affect your code coverage results.

- Do not inline any functions.
- Inline all functions.
- Inline only .Net Framework functions.
- Inline only .Net Framework functions, and functions in specific DLLs.
- Inline only functions in specific DLLs.

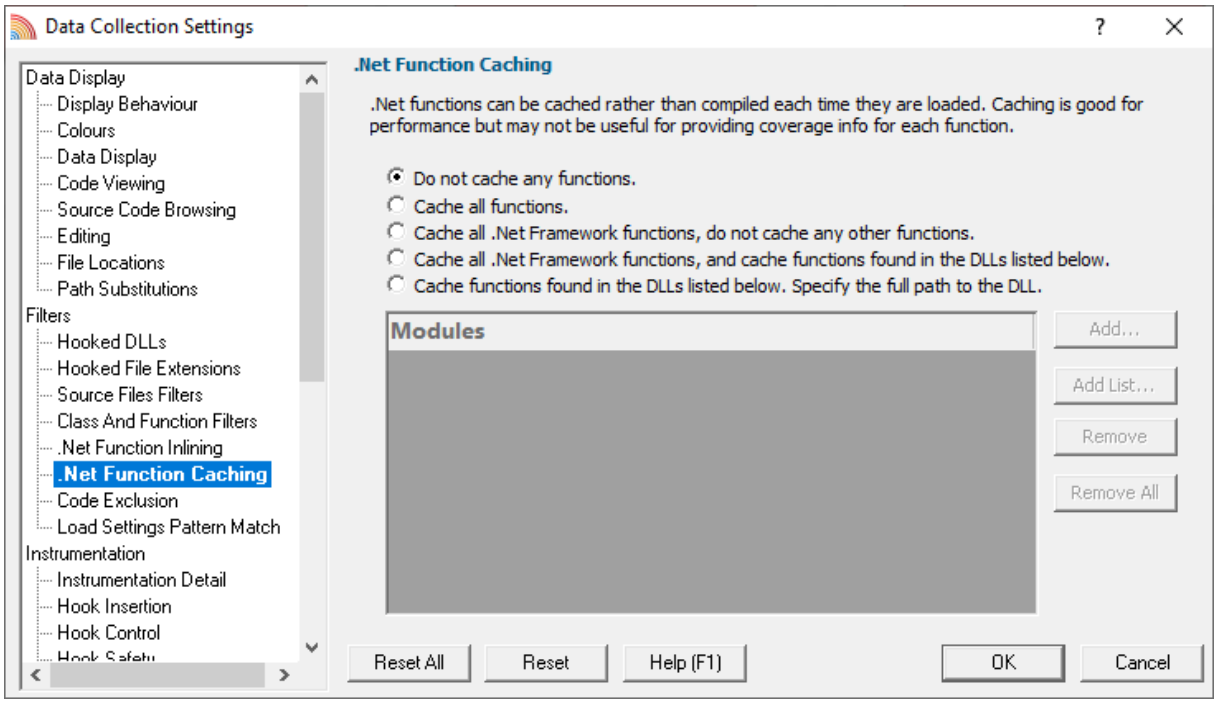
We recommend that you don't inline any functions. This is the default option.

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

Reset - Resets the settings on the current page.

3.10.1.2.6 .Net Function Caching

The **.Net Function Caching** tab allows you to configure how .Net caches functions.



Caching of .Net functions can affect your code coverage results.

- Do not cache any functions.
- Cache all functions.
- Cache only .Net Framework functions.
- Cache only .Net Framework functions, and functions in specific DLLs.
- Cache only functions in specific DLLs.

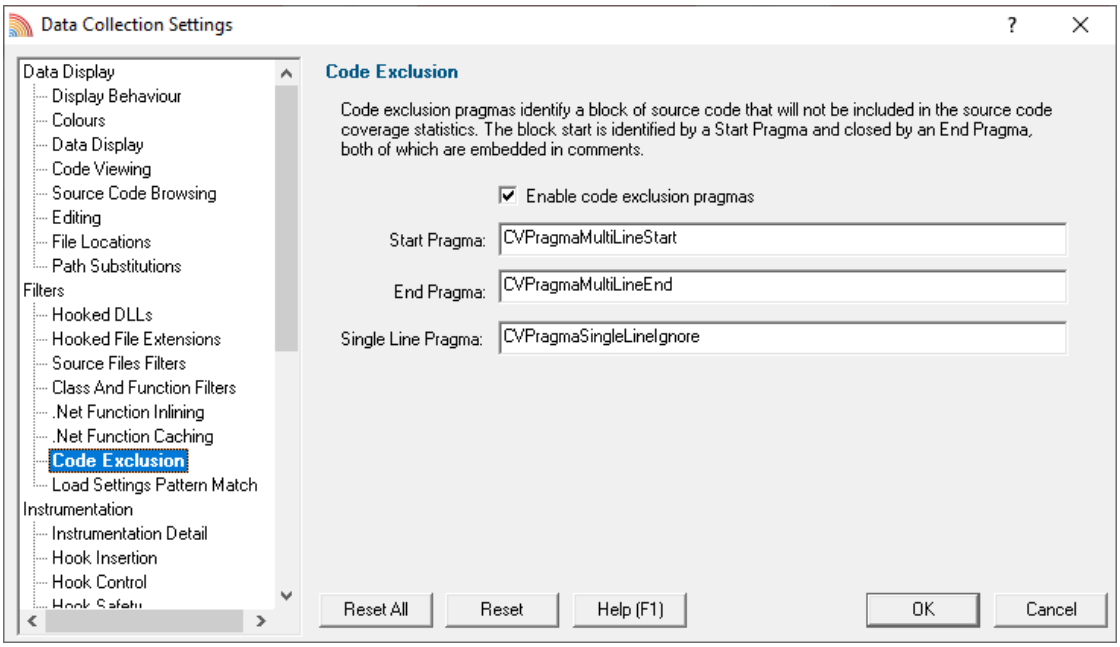
We recommend that you don't cache any functions. This is the default option.

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

Reset - Resets the settings on the current page.

3.10.1.2.7 Code Exclusion

The **Code Exclusion** tab allows you to configure blocks of code to be removed from code coverage, using comments in the source code.



Code exclusion

In general the code coverage filters work on discreet elements like file, class and function names.

Code exclusion lets you further control sections of a source file to be excluded by using pragmas (keyword directives) to mark the a region of code.

The pragmas can surround complete classes, functions or just single lines of code.

Using code exclusion

The reason for providing code exclusion is to allow code coverage tests to be setup knowing that a file can reach 100% coverage even if certain lines are not processed.

Such lines may not be processed because they are on a code path that will only be taken during error conditions. See the examples below.

- **Enable code exclusion pragmas** › check the box to turn the feature on

Multi-line code exclusion

- **Start Pragma** › enter the keyword that you will use in the code to mark the *start* of a block to exclude

  E.g. `CVPragmaMultiLineStart`

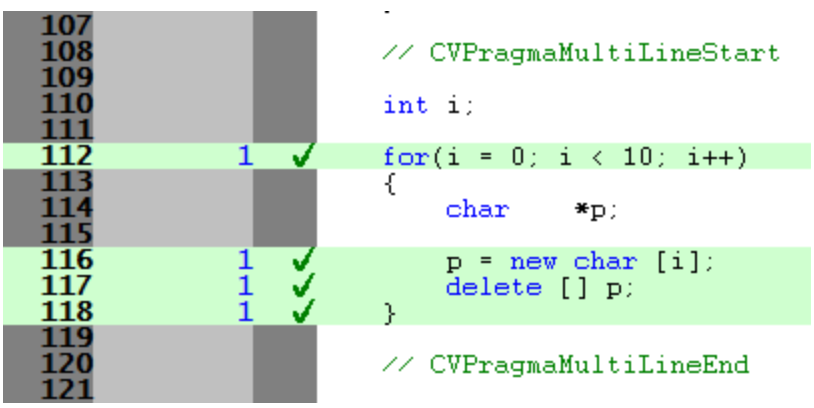
- **End Pragma** › enter the keyword that you will use in the code to mark the *end* of the excluded block

E.g. `CVPragmaMultiLineEnd`

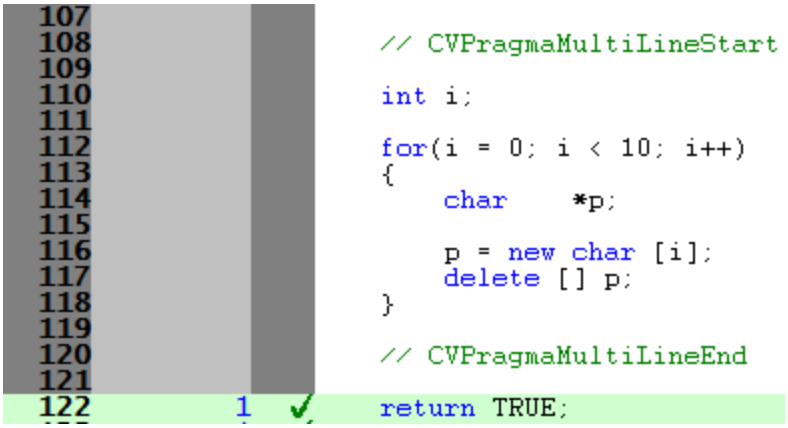
Now you can exclude a section of code by enclosing code with comments containing these pragmas

Example:

Without code exclusion switched on, the `for` loop below would count towards coverage:



After turning code exclusion on, the pragmas shown here force the loop to be excluded from coverage statistics:



Single line code exclusion

- **Single Line Pragma** › enter the keyword that you will use in the code

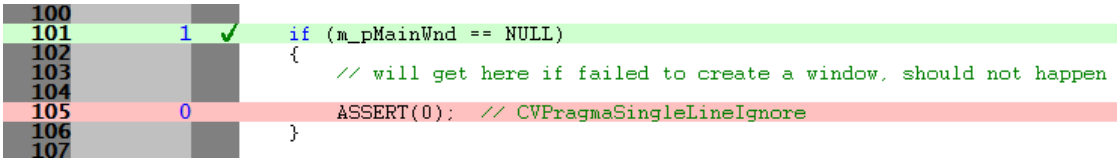
E.g. `CVPragmaSingleLineIgnore`

Now you can exclude a single line of code by adding a comment to the line containing the pragma `CVPragmaSingleLineIgnore`

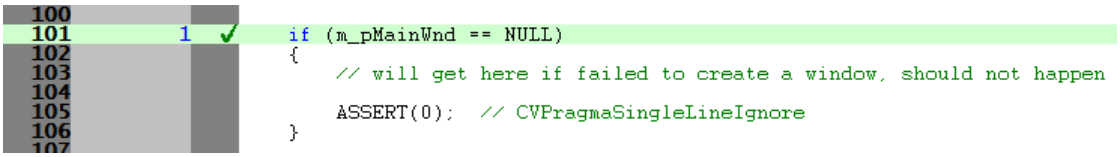
Example:

The `assert` statement in the code below only happens in an error situation, and probably an unrecoverable one at that.

Without using code exclusion, the assertion would count towards incomplete coverage.



After turning code exclusion on and setting up the pragma blow, the assertion line is ignored in the coverage



Instrumentation logging

The logging of DLLs, source files, classes, methods and functions that are not instrumented can help you understand the reason why part of your code isn't getting the coverage information you expect.

Once enabled, and a session has started, you can view a list of items that have not been instrumented via the Tools menu.

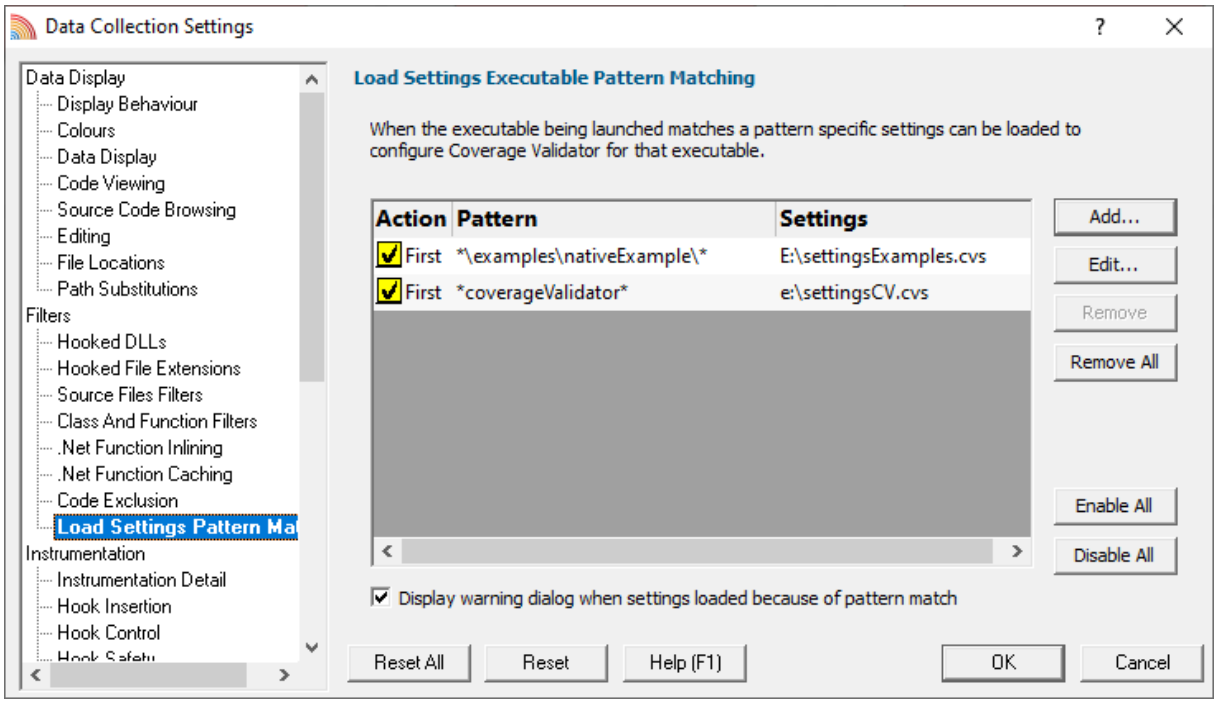
- **Enable instrumentation logging** › check to enable logging once the next session starts

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

Reset - Resets the settings on the current page.

3.10.1.2.8 Load Settings Pattern Match

The **Load Settings Pattern Match** tab allows you to configure loading of different settings depending on the executable being launched (or relaunched).



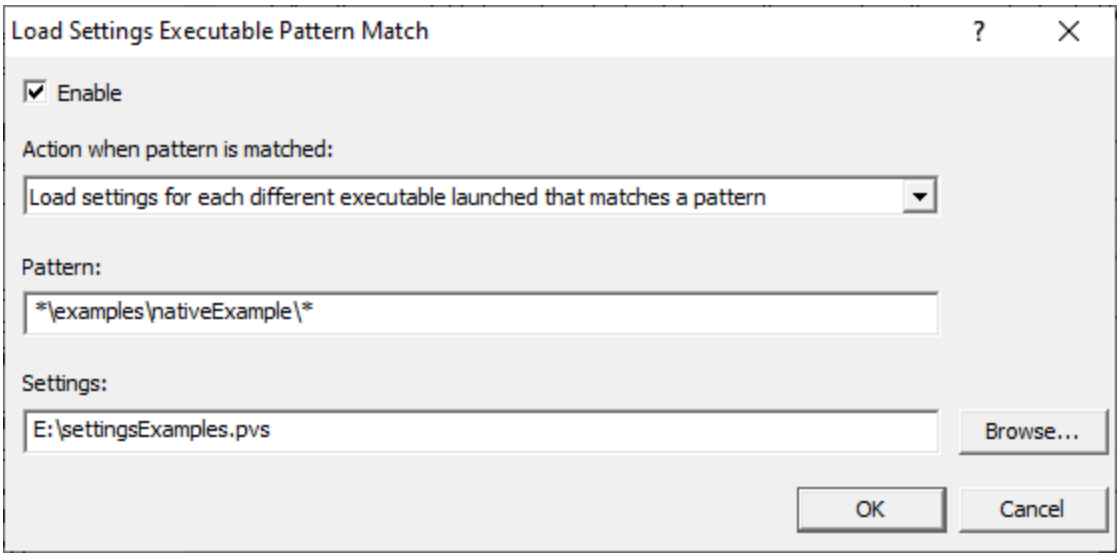
The grid shows one pattern match per line.

The buttons alongside allow you to Add, Edit and Remove patterns that you have created. You can also enable and disable them all.

- **Add...** › display the pattern match dialog to create a pattern to match.
- **Edit...** › display the pattern match dialog to edit the selected pattern.
- **Remove...** › delete the selected pattern.
- **Remove All...** › delete all selected patterns.
- **Enable All...** › enable all patterns.
- **Disable All...** › disable all patterns.

Pattern Match Dialog

The pattern match dialog allows you to create and edit pattern matches.



- **Enable** › enable or disable this pattern.
- **Action** › how to evaluate if this pattern is matched.
  - **Load settings for first executable...** › the first executable that matches a pattern causes the settings to be loaded.
  - **Load settings for each executable...** › each executable that matches a pattern causes the settings to be loaded provided it is not the same executable that previously loaded the settings.
  - **Load settings for every executable...** › every executable that matches a pattern causes the settings to be loaded.

  When a different pattern matches the first/each status is reset.
- **Pattern** › a text pattern, including the * wildcard, to match an executable path.

  Examples:
  ```
  *\examples\nativeExample\*
  c:\tests\*
  e:\dev\myProject\release\*.exe
  ```
- **Settings** › the full path to the settings you want to load if the action and pattern match an executable.

How does the pattern matching work?

It is probably easiest to demonstrate how pattern matching works with some examples.

Let's assume we have two patterns:

`*\examples\nativeExample\*` that will load `e:\settingsExamples.cvs`
`*coverageValidator*` that will load `e:\settingsCV.cvs`.

We'll cover each of the possible action criteria for a sequence of application launches, showing which settings are loaded and why.

- **Load settings for first executable...** › the first executable that matches a pattern causes the settings to be loaded.

Launched application	Settings loaded	Reason
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.cvs	1st application, new pattern
e:\examples\nativeExample\release\nativeExample.exe		repeat application, same pattern
e:\examples\nativeExample\debug\nativeExample.exe		2nd application, same pattern
c:\program files (x86)\software verify\coverage validator x86\coverageValidator.exe	e:\settingsCV.cvs	1st application, new pattern
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.cvs	1st application, new pattern

- **Load settings for each executable...** › each executable that matches a pattern causes the settings to be loaded provided it is not the same executable that previously loaded the settings.

Launched application	Settings loaded	Reason
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.cvs	new application, new pattern
e:\examples\nativeExample\release\nativeExample.exe		repeat application, same pattern
e:\examples\nativeExample\debug\nativeExample.exe	e:\settingsExamples.cvs	new application, same pattern
c:\program files (x86)\software verify\coverage validator x86\coverageValidator.exe	e:\settingsCV.cvs	new application, new pattern
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.cvs	new application, new pattern

- **Load settings for every executable...** › every executable that matches a pattern causes the settings to be loaded.

Launched application	Settings loaded	Reason
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.cvs	Every application
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.cvs	Every application
e:\examples\nativeExample\debug\nativeExample.exe	e:\settingsExamples.cvs	Every application
c:\program files (x86)\software verify\coverage validator x86\coverageValidator.exe	e:\settingsCV.cvs	Every application
e:\examples\nativeExample\release\nativeExample.exe	e:\settingsExamples.c	Every application

vs

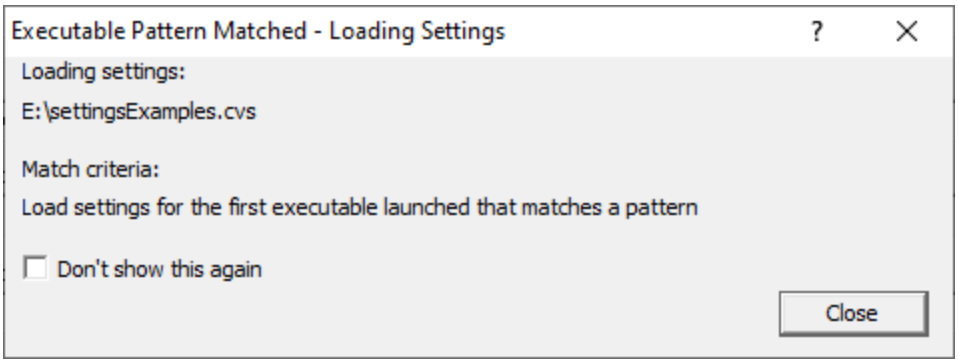
Warning

When a pattern is matched and the action criteria are satisfied the specified settings will be loaded.

A warning can be displayed at this point to remind you that the settings are being changed.

- **Display warning dialog...** > the warning dialog will be displayed when the pattern match criteria are met.

The warning dialog looks like this:



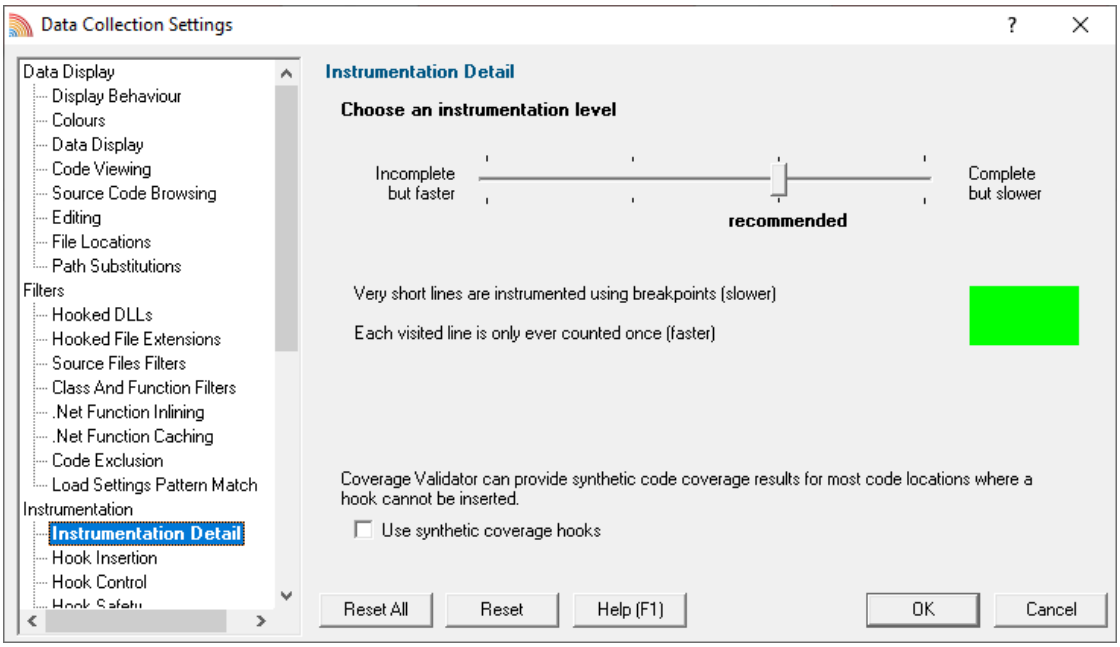
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.3 Instrumentation

3.10.1.3.1 Instrumentation Detail

The **Instrumentation Detail** page allows you to specify how lines are hooked and how many times a line visit is counted.



Instrumentation detail versus speed of execution

Two instrumentation settings affect how quickly the instrumented program will run:

- the hooking of very short lines

  Most lines can be monitored using a normal line hooking technology. These hooks are fast but cannot be used for all lines.

  Very short lines are hooked using breakpoint hooking technology which is very advanced but also incurs a serious performance penalty for each visit to that line.

- the counting of lines on every visit

  Counting lines just once is quicker than doing it on every visit.

Instrumentation Level

The instrumentation level lets you balance the detail of collected visit counts for every line against speed of execution.

- **Incomplete but faster** › Short lines NOT counted. Lines counted once
- **Incomplete but slower** › Short lines NOT counted. Lines counted every visit.
- **Complete but faster** › Short lines counted. Lines counted once.

- **Complete but slower** › Short lines counted. Non-breakpoint lines counted every visit. Breakpoint lines counted once.

More about very short lines

For instruction sequences that are too short (less than 5 bytes) to be hooked, Coverage Validator has an additional method of hooking instructions.

Breakpoint instruction and operating system exception handler routines intercept individual breakpoint instructions and direct the exception handler to the original code for the line.

Due to the exception handling overhead for each instruction hooked this way this is a slower execution method than normal line hooking.

It is effective however, and allows Coverage Validator to hook most lines in most applications with no failures.

The only instructions that *can't* be hooked are breakpoints and the various x86 LOOP instructions, which are avoided by modern compilers.

Caveats to hooking very short lines

There are two caveats to using the Breakpoint Hooks option.

You can't use them at the same time as:

- Running your application in the debugger (or attaching the debugger to your application) at the same time as using Coverage Validator to monitor the application.

  The debugger will see the breakpoint exception *before* our hook on the exception handling mechanism.

- Using MAP files for release builds.

  The information in the MAP file does not correctly identify the start of individual lines of code.

  This is because the MAP file is produced before the linker optimizer re-arranges the code for the specified optimization flags.

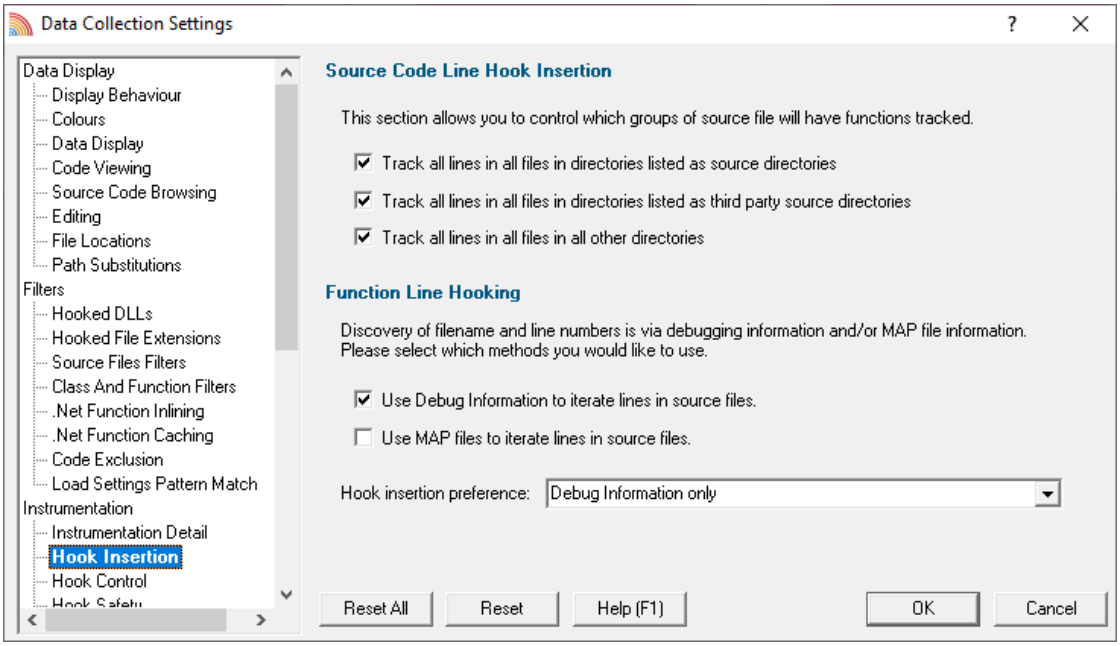
  Because it is not possible to differentiate between debug build MAP files and release build MAP files, Coverage Validator will not use Breakpoint Hooks with MAP files.

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

Reset - Resets the settings on the current page.

3.10.1.3.2 Hook Insertion

The **Hook Insertion** page allows you to specify which groups of source files are hooked in the target program.



These hook settings will not take effect during a session that is already running.

Source Code Line Hook Insertion

The File Locations settings let you set several lists of files including

- your own source files
- third party source files

and then of course there's implicitly:

- 'everything else' that you *didn't* specify.

The default behaviour is to hook everything referenced in a PDB or MAP file that is in one of these three categories.

You can change whether to include or exclude any of these sets.

Untick the boxes to stop tracking all lines in all files in...

- **directories listed as *source* directories**
- **directories listed as *third party source* directories**

- ***all other* directories** (i.e. not any directory matching the two options above)

Function Line Hooking

To hook each source code line Coverage Validator has to find the start address of each line and then ensure hooking the line will not result in corruption of the code relating to other parts of the program.

The data to determine the location of each source code line is found in PDB files and MAP files that contain line number information.

You can choose whether to use *only* PDB files, *only* MAP files, or to set a preference for one source over the other.

- **Use PDB files...** > Allow use of PDB files for iterating files
- **Use MAP files...** > Allow use of MAP files for iterating files

If setting PDB *and* MAP files to be used, you can set the preferred use in the drop-down list:

Hook insertion preference

- **PDB files only** > Ignores MAP files (same as unchecking MAP file usage)
- **MAP files only** > Ignores PDB files (same as unchecking PDB file usage)
- **Use MAP when no PDB** > PDB files are used in preference to MAP files.

  MAP files are only used when the required PDB file cannot be found
- **Use PDB when no MAP** > MAP files are used in preference to PDB files.

  PDB files are only used when the required MAP file cannot be found, or the MAP file does not contain line number information

 **Warning:** You should be aware that MAP files may not include all line information. Consider using the **Use MAP when no PDB** option instead.

Map file not recognised?

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.

Coverage Validator will not recognise such a MAP as valid, but rebuilding the application will resolve this.

When MAP files are enabled, the **MAP file dates must match DLL** check box is displayed.

Normally you should have this check box enabled, so that the MAP file dates are compared with the DLL dates.

But there can be circumstances where you know the MAP file is valid, but the dates don't match, which prevents instrumentation. In that case deselect **MAP file dates must match DLL**.

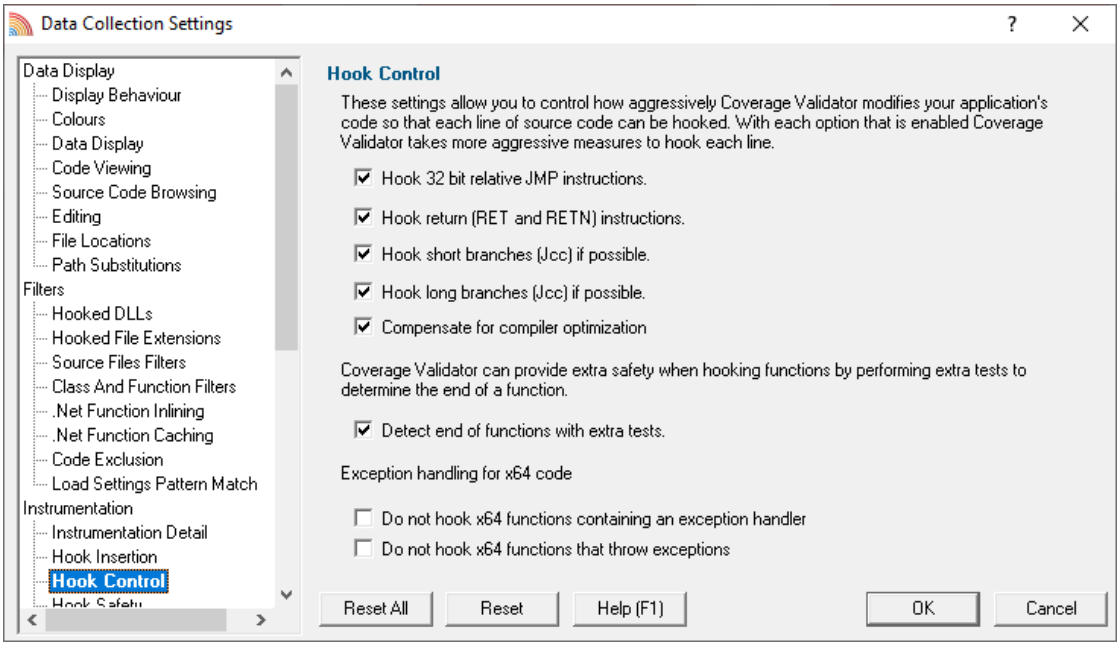
*We strongly recommend that you keep **MAP file dates must match DLL** selected and rebuild your application to make the MAP file dates match the DLLs.*

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

Reset - Resets the settings on the current page.

3.10.1.3.3 Hook Control

The **Hook Control** page allows you to specify how some of the more 'awkward' lines of code are hooked.



Most of these options are fairly advanced. They are also mostly enabled by default, but as usual, we provide the ability to control the settings if needed.

Hook Control

Some lines of code are trickier to hook than others. By default Coverage Validator is quite zealous in performing the safety checks and enabling hooking of these lines.

If necessary, you can choose *not* to enable these hooks.

The options are:

- **Hook 32 bit relative JMP instructions** › hooks lines with `JMP` instructions in them
- **Hook return (RET and RETN) instructions** › hooks lines with `RET` or `RETN` instructions in them
- **Hook short branches (Jcc) if possible** › hooks lines with short branches in them

  (`Jcc` is a conditional jump)
- **Hook long branches (Jcc) if possible** › hooks lines with long branches in them

Line ordering due to compiler optimisation

When the compiler optimizes things it can move lines around relative to the original source code.

During instrumenting extra checks are added so that hooks can get correctly recognized/improved automatically.

- **Compensate for compiler optimisation** › untick to switch off these extra checks

Extra care at function ends

An additional option makes Coverage Validator more cautious when detecting the end of a function, to avoid overwriting the code for any function that follows.

- **Detect end of functions with extra tests** › hook lines with JMP instructions in them

Filling in the gaps

Finally Coverage Validator can synthesize coverage results for lines that simply could not be hooked.

Not *all* lines can have their coverage results synthesized, only those in the same execution path.

Using this option will *not* introduce errors into your coverage tests.

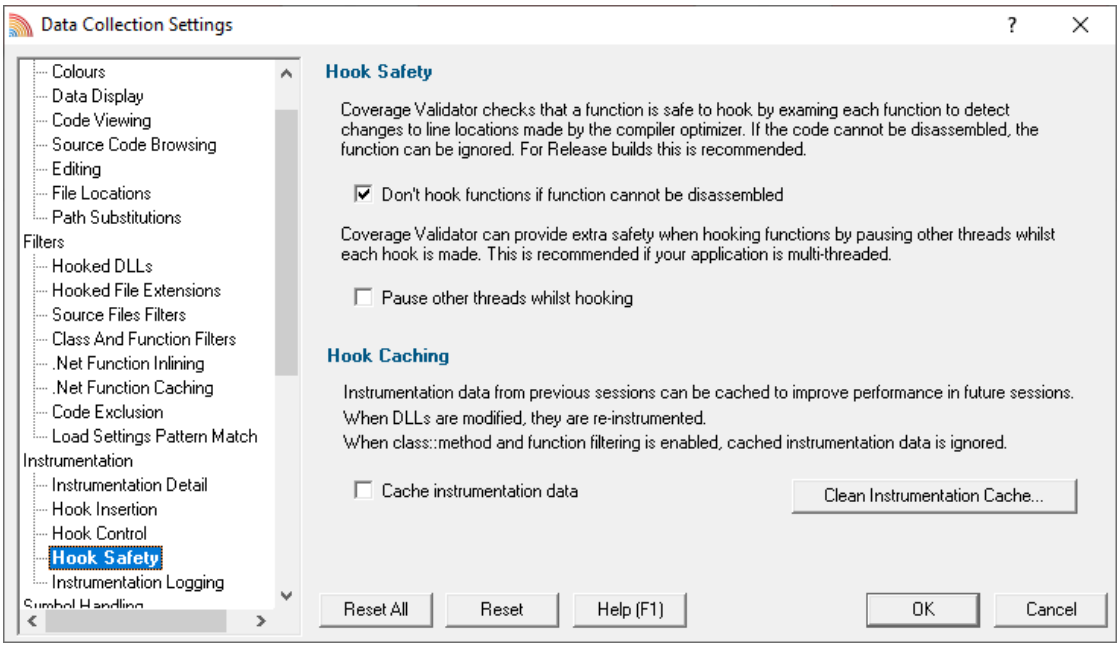
- **Use synthetic coverage hooks** › synthesize coverage results - not on by default.

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

Reset - Resets the settings on the current page.

3.10.1.3.4 Hook Safety

The **Hook Safety** page allows you to specify which safety features are enabled during hooking.



Hook safety with Release Mode software

When hooking functions, it is important that the information being used is verified against the actual object code being modified to insert the hooks.

This is especially important in Release builds where the code optimizer may have rearranged the object code so that it does not match the information in the PDB and MAP files.

To ensure this, the default behaviour is to disassemble each function and check it prior to hooking it. If the function can't be disassembled, it won't get hooked.

You can switch off disassembling of functions and hook them anyway:

- **Don't hook functions if function cannot be disassembled** › uncheck to hook the functions anyway

  It's recommended to keep this option enabled for Release builds

Hook safety with multi-threaded applications

When working with multi-threaded applications, it is possible for the thread hooking the application to be modifying code that is executing in a different thread.

If you think this might be affecting you, you can pause the other threads while hooking..

- **Pause other threads whilst hooking** › check to pause other threads

Hook caching

When hook caching is enabled, the first time a module (DLL or EXE) is instrumented, various information about the module is stored in a cache file.

Every subsequent time the module is instrumented, the cached data is used to instrument the module, rather than inspecting the module again.

This can provide quite significant performance improvements.

- **Cache instrumentation data** › check to use the instrumentation cache

   If the module is recompiled/relinked, the cache data will be discarded and recalculated.

About cache files

Cached instrumentation files are stored in the same directory as the module to which they refer.

They have the same name as the module, with the **.svlCV_coverage** extension.

For example, instrumentation data for:

```
c:\winnt\system32\msvcrtd.dll
```

is stored as:

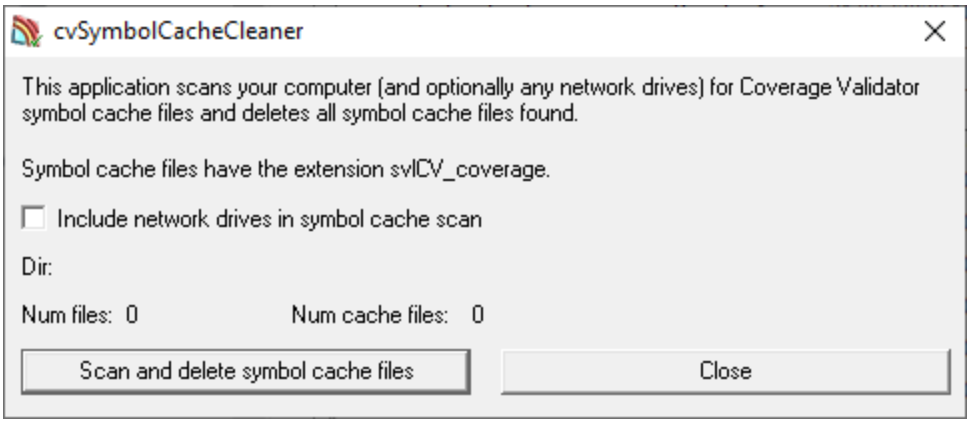
```
c:\winnt\system32\msvcrtd.dll.svlCV_coverage
```

Cleaning up the cache files

You may not want to keep cache files lying around on your system amongst your code, so Coverage Validator provides a utility to automatically clean up those files.

Files with the `.svlCV_coverage` extension are searched for on your drive, and deleted if found.

- **Clean instrumentation cache** › shows the Symbol Cache Cleaner utility dialog



- **Include network drives** › tick this to clean up networked drives
- **Scan and delete symbol cache files** › starts the scan

The dialog shows a count of the number of files scanned and the total number of cache files found

- **Close** › cancels the process and closes the dialog

You can continue to use Coverage Validator as normal while the scan takes place. If starting a new session that caches instrumentation, be aware that cache files may be recreated after the scan has passed!

Cached files and the Class and Function Filters

If the Class and Function Filter has been set up to only include or exclude specified classes or functions, then the Cached Instrumentation data will be ignored.

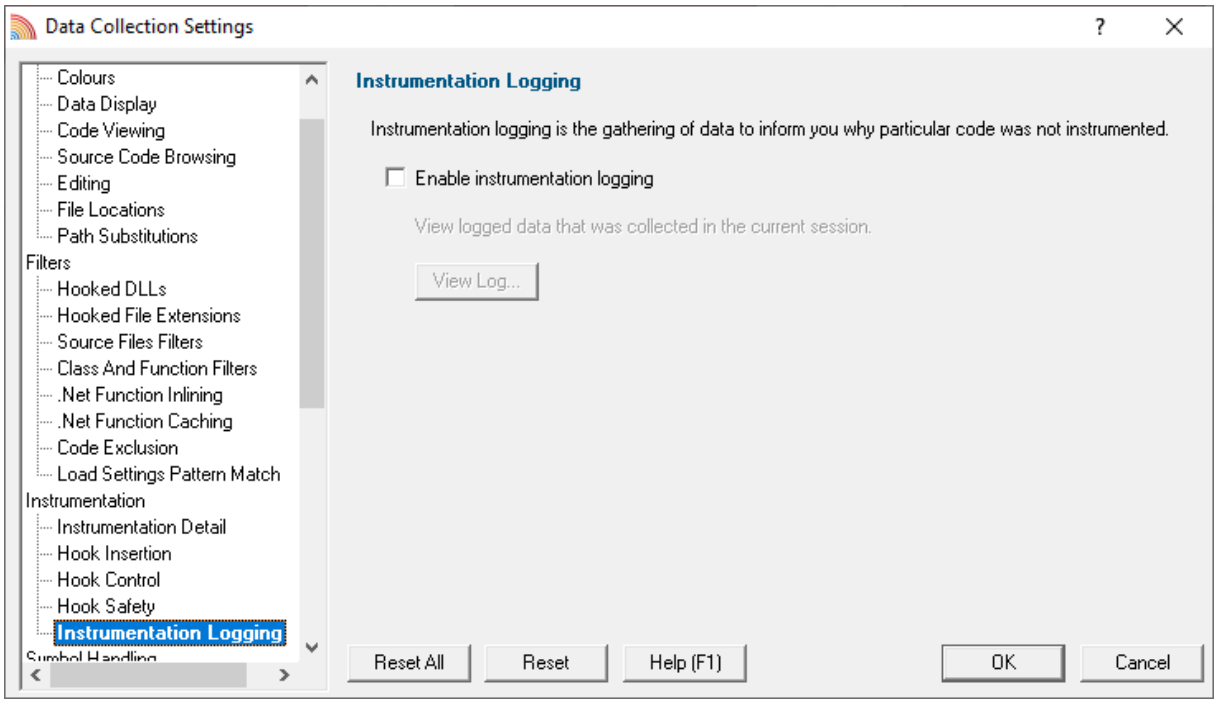
Instead, functions will be hooked according to the Class and Function Filter.

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

Reset - Resets the settings on the current page.

3.10.1.3.5 Instrumentation Logging

The **Instrumentation Logging** tab allows you to control Coverage Validator's internal logging.



If you enable instrumentation logging a log file will be created during instrumentation that indicates why each DLL, file and function was or was not instrumented according to the various settings and filters.

The instrumentation log can be useful to identify the reasons why a particular file or function or class is or is not be instrumented.

- **View Log...** › to view the instrumentation log. You can also view the log from the Tools menu.

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

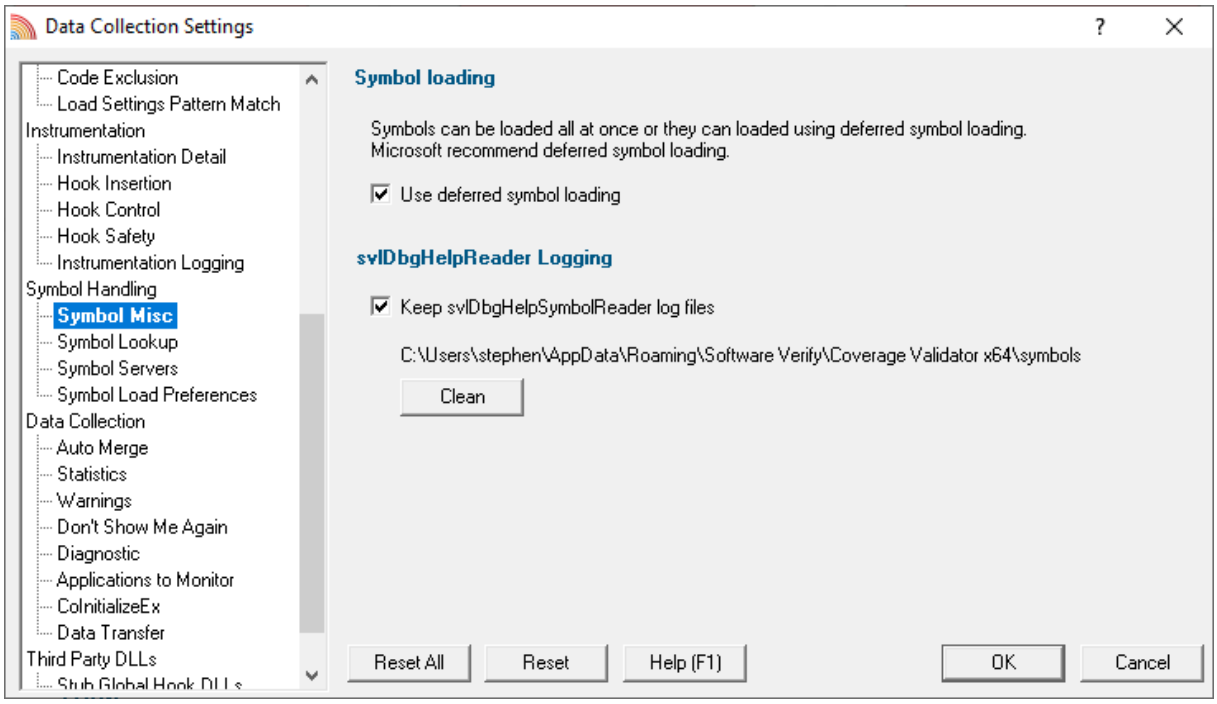
Reset - Resets the settings on the current page.

3.10.1.4 Symbol Handling

Enter topic text here.

3.10.1.4.1 Symbol Misc

The **Symbol Misc** tab allows you to set miscellaneous symbol settings.



Immediate or deferred symbol loading

When converting program addresses to symbol names, you can choose immediate symbol loading, or defer loading until each symbol is needed.

- **Use deferred symbol loading** › uses deferred symbol loading rather than 'all at once' (on by default)

  Microsoft® recommend deferred symbol loading, claiming it is the fastest option. We give you the choice.

Symbol Reader Logging

Symbols are fetched from symbol servers using a helper process svlDbgHelpSymbolReader.exe. We log the command line and behaviour of this helper tool. This is displayed on the diagnostic tab.

If you wish the log files can be kept for later analysis. By default this option is turned off.

- **Keep svlDbgHelpSymbolReader log files** › keep the log files after Coverage Validator has finished processing them

The path to the directory containing the log files is shown.

- **Clean** › delete all svlDbgHelpSymbolReader log files

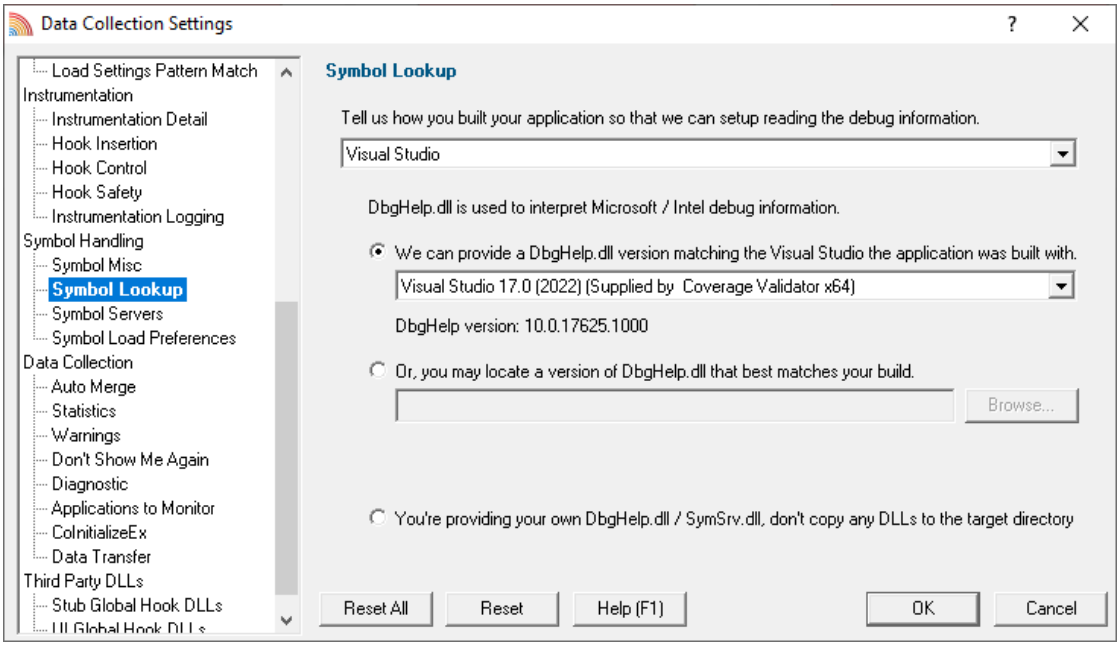
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.4.2 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.



Compiler / IDE Choice

Use the first combo box to choose which compiler / IDE you used to build your software.

Coverage Validator will use the appropriate methods to read your symbols.

The choices are:

- Visual Studio
- Visual Basic 6
- Delphi or C++ Builder
- MingW
- Rust
- Dev C++
- Metrowerks CodeWarrior

- Salford Fortran 95
- Other

Symbol lookup for Microsoft / Intel compilers

- **We can provide a Dbghelp.dll** › choose one of Coverage Validator's known good DbgHelp.dll's based on the version of Visual Studio you are using

  Coverage Validator fetches symbols for your application using an appropriate symbol handler for the type of debugging information you have.

  For Microsoft Visual Studio users each version of Visual Studio provides different debugging formats which are readable by the appropriate DbgHelp.dll supplied by Visual Studio. A given version of DbgHelp.dll is usually able to read earlier formats of Microsoft debugging information but is not able to read a future format. For example Visual Studio 2005 (version 8) can read Visual Studio 6 debug information but cannot read Visual Studio 2008 debug information.

  Visual Studio 6.0 doesn't supply a DbgHelp.dll so we have provided one for use with Visual Studio 6.0.

  Visual Studio 10 is unusual in that the DbgHelp.dll (6.12) supplied by Visual Studio cannot read the debug information created by Visual Studio. To solve this problem we have supplied DbgHelp.dll (6.11) as an alternative.

  Coverage Validator will choose the appropriate (most recent) version of Visual Studio automatically. You can override Coverage Validator's choice by choosing the Visual Studio version from the **Visual Studio** combo box.

Specify your own DbgHelp.dll

- **Or, you may locate a version of DbgHelp.dll** › specify your own DbgHelp.dll to use with Coverage Validator

  If you wish to explicitly specify which DbgHelp.dll to use choose the **Or, you may locate a version of DbgHelp.dll** option enter the path in the **DbgHelp.dll** edit field or use the **Browse...** button to select the dbgHelp.dll.

  Note that the directory that contains DbgHelp.dll ***should also contain symsrv.dll*** if you wish to use symbol servers with Coverage Validator.

Don't update DbgHelp.dll

- **You're providing your own DbgHelp.dll** › use the DbgHelp.dll that ships with your application

  If your application needs to use a specific version of DbgHelp.dll that you're already providing with your application you should choose the **You're providing your own DbgHelp.dll** option to prevent Coverage Validator from overwriting your DbgHelp.dll.

Note that the directory that contains DbgHelp.dll ***should also contain symsrv.dll*** if you wish to use symbol servers with Coverage Validator.

Visual Studio DbgHelp.dll version compatibility

For Microsoft Visual Studio users, each VS version provides different debugging formats which are readable by the appropriate DbgHelp.dll supplied with Visual Studio.

These handlers are usually backwards compatible, but not forwards compatible. For example Visual Studio 2005 (version 8) can read Visual Studio 6 debug information but cannot read Visual Studio 2008 debug information.

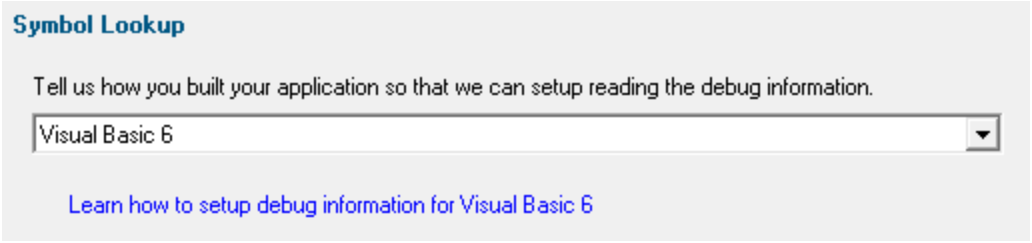
Visual Studio 6.0 doesn't supply a DbgHelp.dll so we have provided one for use with Visual Studio 6.0.

Visual Studio 10 is unusual in that the DbgHelp.dll (6.12) supplied by Visual Studio cannot read the debug information created by Visual Studio! To solve this problem we supply version 6.11 as an alternative.

To see the order in which the *DbgHelp.dll* process checks directories to find symbols, see the diagnostic tab with the filter set to *DbgHelp debug*.

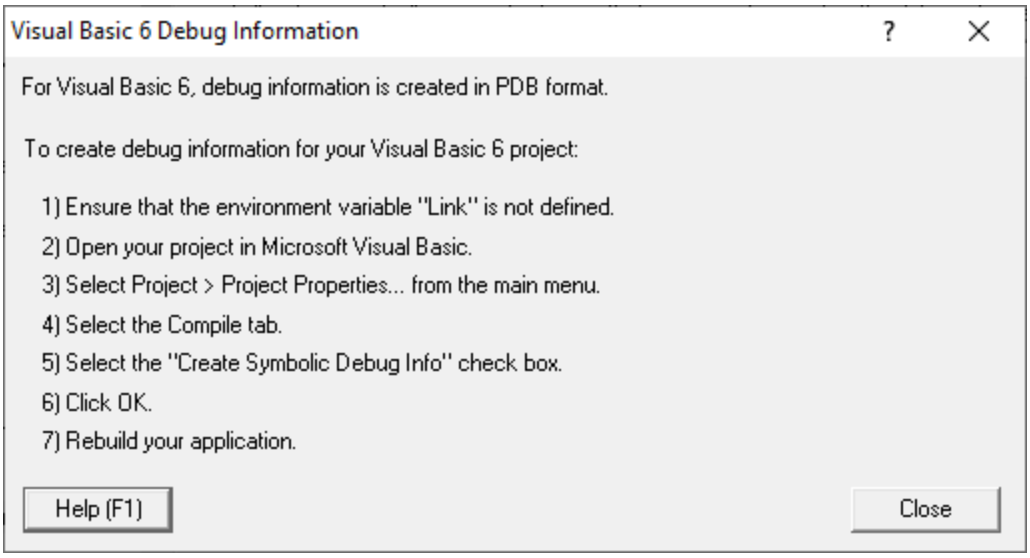
Symbol lookup for other compilers

If you are using another compiler click the link to see information about configuring debug information for that compiler.



After selecting the compiler, clicking the link will show a dialog box containing information relevant to the selected compiler.

For example:



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

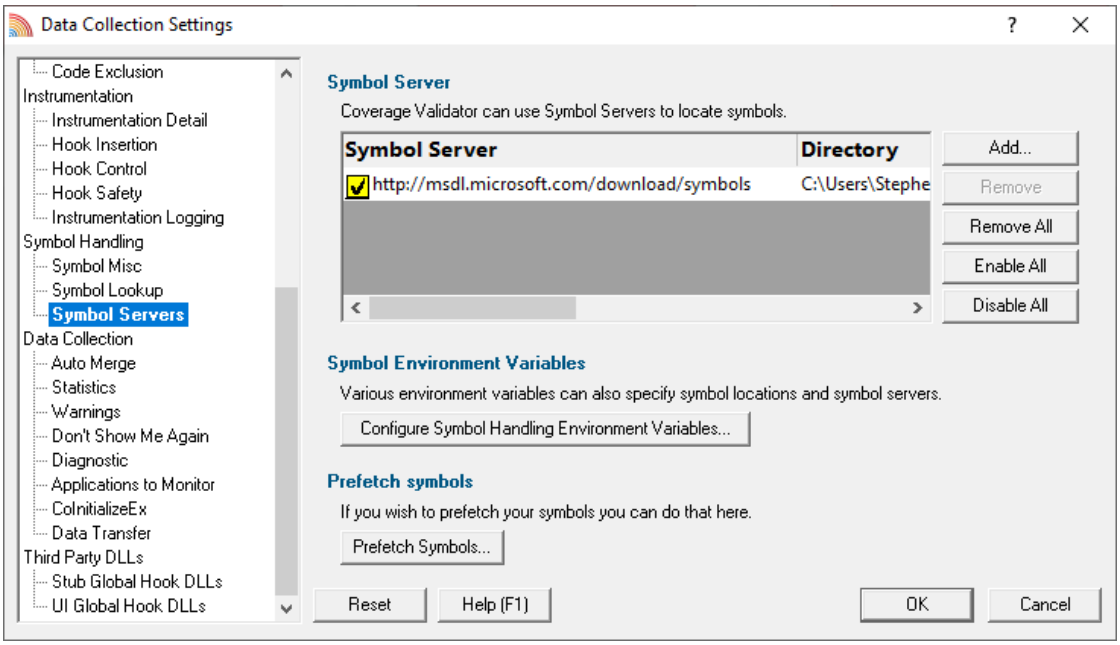
Reset - Resets the settings on the current page.

3.10.1.4.3 Symbol Servers

The **Symbol Servers** tab allows you to specify Symbol Servers to retrieve symbols used in your application.

You do not *need* to specify symbol servers if you do not wish to, and Coverage 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.

Symbol Server Errors

Any symbol server entry shown in red indicates there is a problem with parts of the definition of that symbol server.

In the image shown above the symbol server at http://127.0.0.42:8000 cannot be reached. It is either offline or does not exist.

Managing symbol servers

- **Add...** › displays the symbol server dialog described below

- **Remove** › remove selected symbol server(s) in the list
- **Remove All** › remove all symbol servers

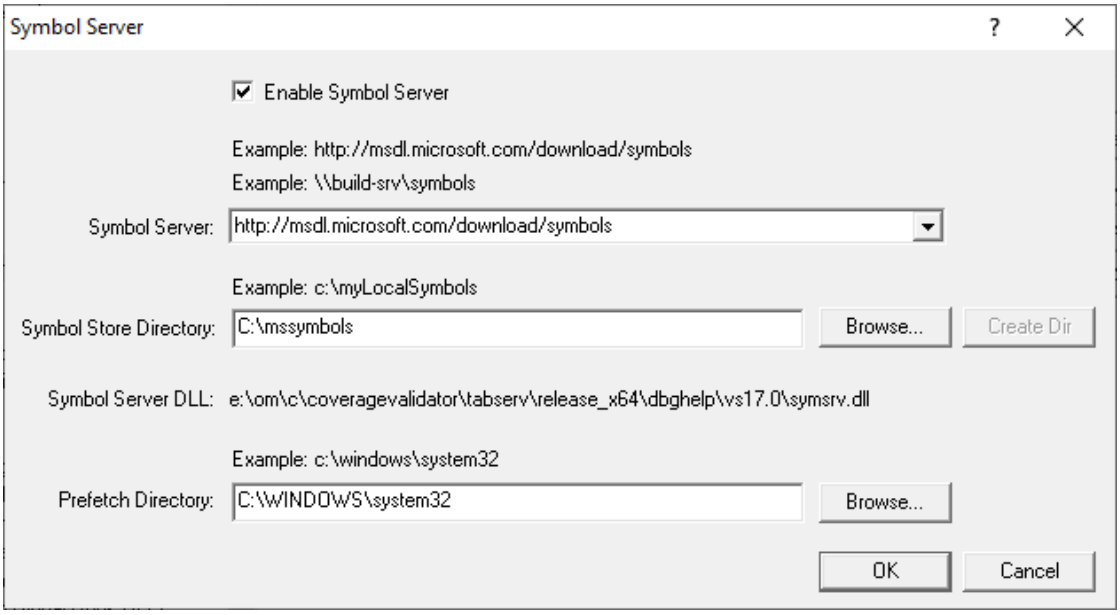
- **Enable All** › enables all symbol servers in the list
- **Disable All** › disables all symbol servers

> You can also enable or disable an item in the list via the yellow check box at the left of each row.

To edit the details for a symbol server, just double click the entry in the list to show the symbol server dialog again.

Symbol server dialog

The dialog initially appears pre-populated with some default values and allows you to set up or edit the definition of a symbol server. Some of the default values can be changed.



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

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

> OK button not enabled?The OK button will only be enabled when the following entries have a valid value: - Symbol Server DLL names a dll present in the Memory Validator install directory. - Symbol Store Directory has been specified and exists. - Symbol Server URL has been specified (this value will not be checked for correctness).

- **Symbol Server** › select a predefined public symbol server or enter the URL of the symbol server you wish to use - the Microsoft server is initially set as the default

- **Symbol Store Directory** › enter or **Browse** to set the directory that will contain local copies of the downloaded symbols
    - **Create Dir** › creates a directory if you entered a directory name that does not exist yet

The **Symbol Server DLL** is set based on the Symbol Lookup settings you have chosen.

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

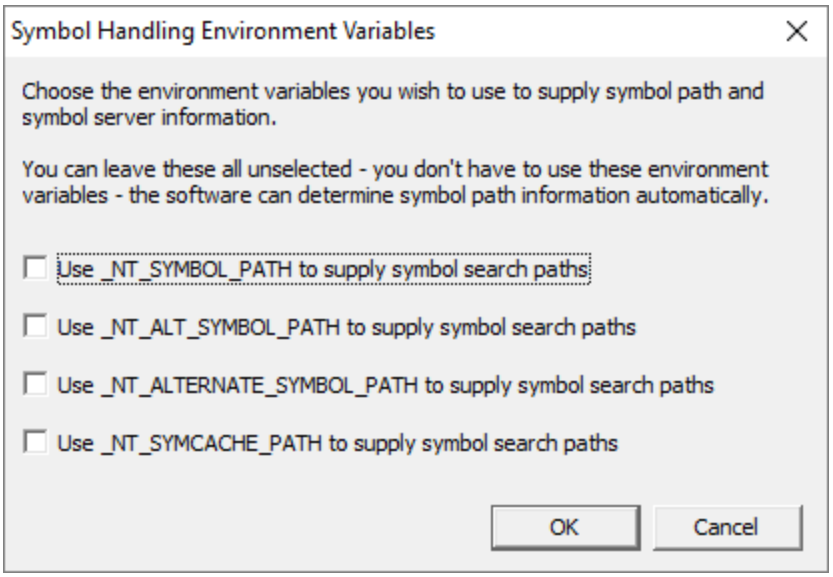
- **Prefetch Directory** › specify the directory to scan for symbols

Environment variables related to symbols

If you wish, you can set some environment variables to supply symbol paths.

- **Configure Symbol Handling Environment Variables** › opens the dialog below

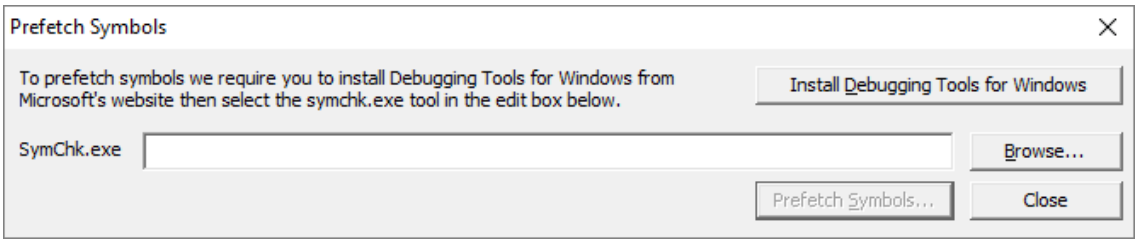
    Check the desired options - if any.



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



Prerequisites for pre-fetching symbols

The pre-fetching of symbols requires the installation of Microsoft's Debugging Tools.

You may already have Debugging Tools if you've previously installed the Windows Driver Kit (DDK or WDK) or the Windows SDK.

- **Install Debugging Tools for Windows** › opens a web page (as above) to download and install the x86 or x64 Debugging Tools for Windows

After installing the Debugging Tools, you must specify the location of SymChk.exe from the installed area.

- **SymChk.exe** › enter or **Browse** to SymChk.exe location

  A typical path might be `C:\WinDDK\7600.16385.1\Debuggers\symchk.exe`

Getting the symbols

Note that prefetching symbols may consume a large amount of disk space and download bandwidth.

You should ensure that you have at least 2 or 3Gb of disk free space, because of the total size of the download packages.

- **Prefetch Symbols...** › runs SymChk.exe to get all the symbols

  The symbols for each symbol server are stored in the associated symbol store directory.

If no symbol servers are specified in the symbol server settings above, you'll see a warning dialog and no symbols will be fetched.

Command line pre-fetching of symbols with the SymChk utility

The section on Pre-fetching symbols above is a convenient alternative to manually using the SymChk,exe utility.

To avoid delays when using symbol servers, you can pre-fetch symbols using the SymChk.exe command line tool that is part of Microsoft's Debugging Tools.

You may want to add the folder of the Debugging Tools for Windows package to the PATH environment variable on your system so that you can access this tool easily from any command prompt.

Example:

To use SymChk.exe to download symbol files for all of the components in the `c:\windows\System32` folder, you might use the command:

```
symchk.exe /r c:\windows\system32 /s SRV*c:\symbols\*http://msdl.microsoft.com/download/sym
```

where

`/r c:\windows\system32` finds all symbols for files in that folder *and any sub-folders*

`/s SRV*c:\symbols*http://msdl.microsoft.com/download/symbols` specifies the symbol path to use for symbol resolution.

In this case, `c:\symbols` is the local folder where the symbols will be copied from the symbol server.

➔ To obtain more information about the command-line options for SymChk.exe, type `symchk /?` at a command prompt.

Other options include the ability to specify the name or the process ID (PID) of an executable file that is running.

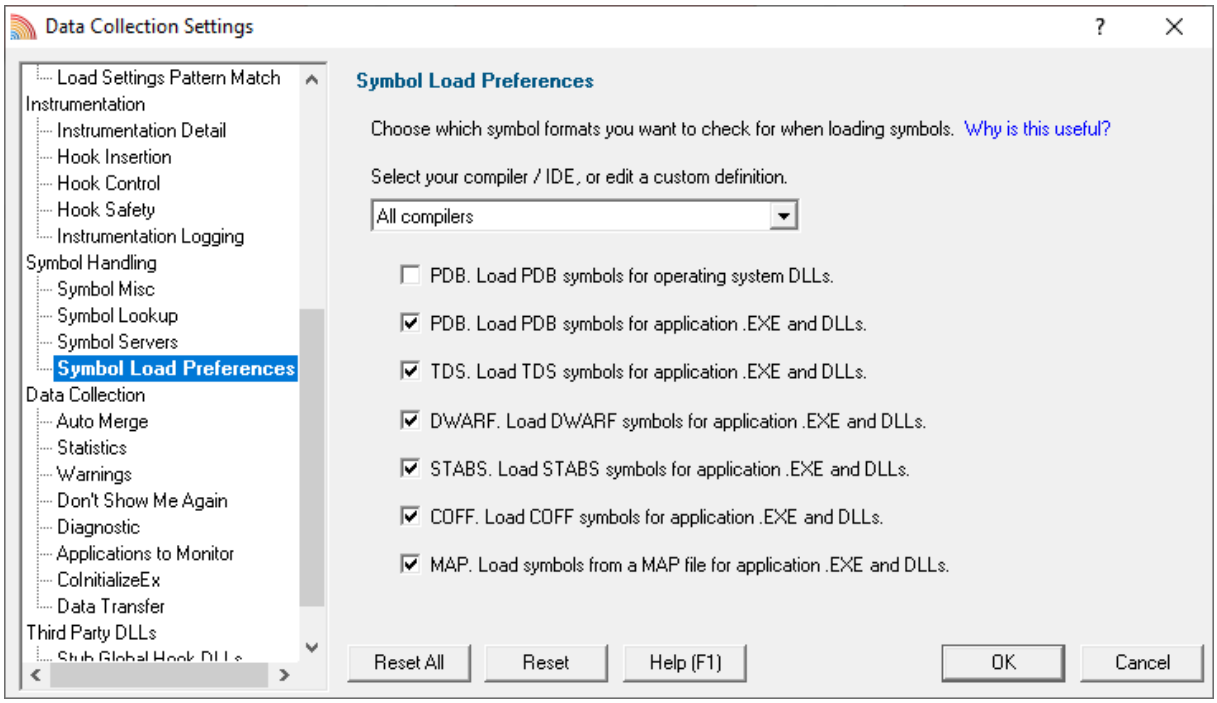
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.4.4 Symbol Load Preferences

The **Symbol Load Preferences** tab allows you to configure which debug information types are looked for and which are ignored.

This can save some time fetching symbols each time a DLL is loaded.



- **Select your compiler / IDE...** › choose a preset definition for a compiler / IDE, or edit one of four custom symbol load preferences

The present definitions are:

- **I don't know which compilers** › choose this if you don't know which compilers were used to build the software
- **All compilers** › choose this to let Coverage Validator fetch the symbols
- **Visual Studio** › choose this if you're only using Visual Studio
- **Visual Basic 6** › choose this if you're only using Visual Basic 6
- **Delphi** › choose this if you're only using Delphi
- **C++ Builder** › choose this if you're using C++ Builder on 32 bit Windows
- **C++ Builder 32 bit** › choose this if you're using C++ Builder to build 32 bit applications
- **C++ Builder 64 bit** › choose this if you're using C++ Builder to build 64 bit applications
- **MingW / gcc / g++ / QtCreator / Dev C++** › choose this if you're using MingW / gcc / g++
- **QtCreator / Dev C++** › choose this if you're using QtCreator
- **Dev C++** › choose this if you're using Dev C++
- **Rust** › choose this if you're using Rust
- **Salford Fortran 95** › choose this if you're using Salford Fortran 95
- **Custom 1** › choose this to edit a definition you can reuse
- **Custom 2** › choose this to edit a definition you can reuse
- **Custom 3** › choose this to edit a definition you can reuse
- **Custom 4** › choose this to edit a definition you can reuse

Editing a definition

Once a definition has been selected the appropriate check boxes next to each debug information type are populated.

You can edit these selections, for example to include or exclude PDB debug information for operating system DLLs, or allow Coverage Validator to search for COFF debug information, whatever is optimal for the way you are working.

Custom definitions

Only the custom definitions will be remembered if they are edited.

The four custom definitions will be remembered, so the next time you choose them you'll get the definition you edited. If you choose one of the preset definitions and edit it, you'll use the edited definition, but if you then change to a different preset (or a custom definition) and then back to the original preset you'll get the preset definition, not your edited version of the preset definition.

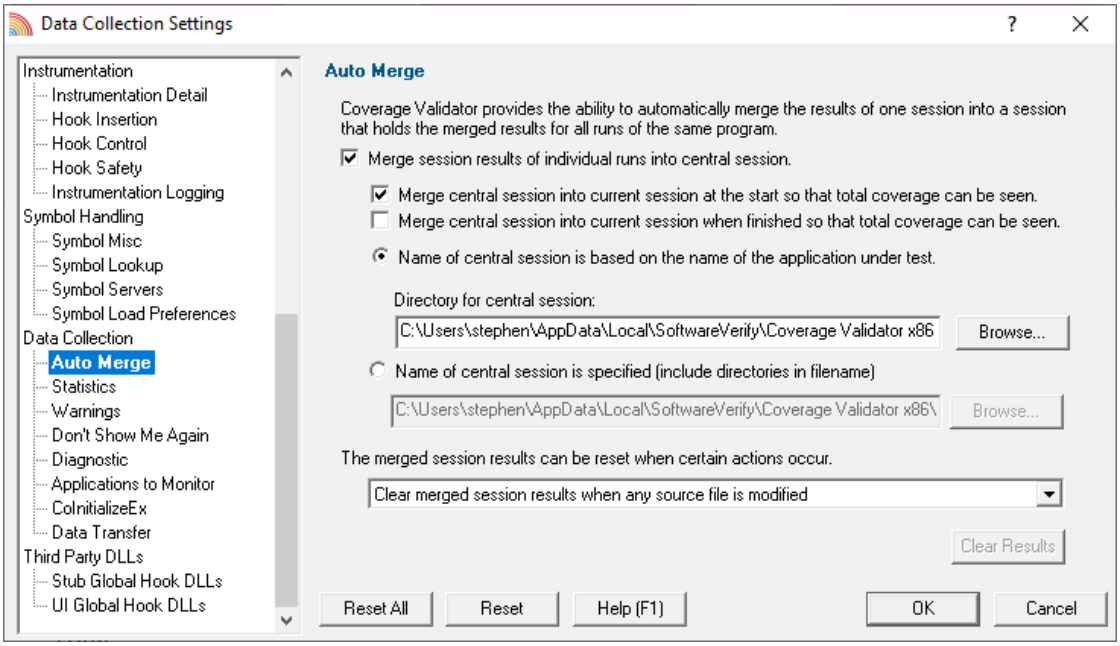
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.5 Data Collection

3.10.1.5.1 Auto Merge

The **Auto Merge** page allows you to specify how the automatic coverage merging functions works.



Coverage auto merging

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.

The automatic merging works by merging the results of each individual 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. A **TestThis.exe** session would get saved in **TestThis.cvm** in the same directory as Coverage Validator resides.

- **Merge session results of individual runs into central session** › switches the merging feature on (note - the default is off)

Merging data into the current session

When session merging is enabled, you can specify that the central session is merged into the current session.

This can be done at the *start* of a session or when a session is *finished*.

For interactive testing you will typically want to merge the central session into the current session at the start of the current session:

- **Merge central session into current session at start...** › merges data into the current session as it starts

For non-interactive testing, say unit testing where you are only interested in the merged results at the end of all the tests, then merging into the current session when it finishes may be more suitable:

- **Merge central session into current session when finished...** › merges data into the current session as it ends

Central session file

Depending on how many applications you are performing coverage on, you may want your 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 Coverage Validator, but you can change this:

  Directory for auto-merge session › saves the auto-merge session in the specified directory (enter or **Browse...** to a directory)

  For example, if you run the application `nativeExample.exe`, and specify a central session directory of `e:\coverageResults`, the central session will be saved in a file named `e:\coverageResults\nativeExample.cvm`.

- **Name of central session is specified** › save the auto-merge session in a filename and path of your choice (enter or **Browse...** to a file)

Auto-merge session reset

The auto-merge results can be automatically cleared by certain triggers, or not cleared at all.

When performing 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 coverage data from the modified software with existing coverage data.

The four triggers for clearing the merged session results are:

- Clear ***all*** merged session results when ***any*** **source file is modified** (the default)
- Clear ***only*** merged session results for **source files** that have been ***edited***
- When the **application** under test changes
- **No clearing** of merged session results occurs under any circumstance

There is also a manual option to clear the central session results at any time (a session needs to be recorded/loaded if the central session name is based on the application being tested).

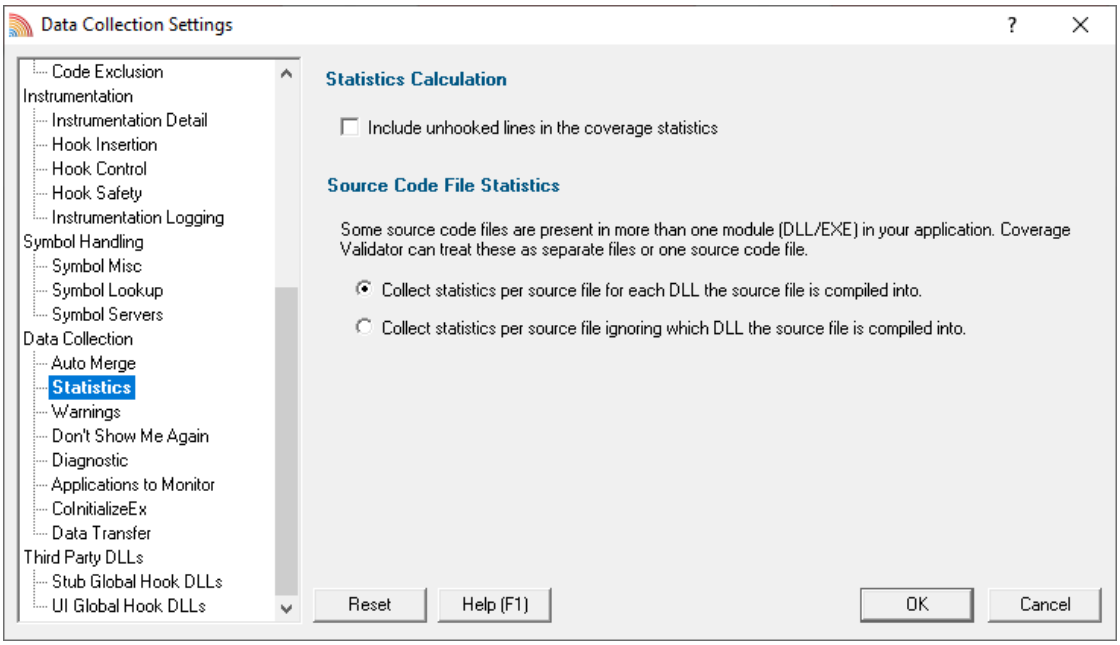
- **Clear Results** › deletes the saved central session results

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

Reset - Resets the settings on the current page.

3.10.1.5.2 Statistics

The **Statistics** page allows you to control how statistics are calculated by Coverage Validator for unhooked lines and for multiple inclusions of the same file.



Statistics Calculation

By default, lines which could not be hooked do not get included in coverage statistics.

Note that this does not affect DLLs, files and lines which get filtered out *before* instrumentation.

- **Include unhooked lines in the coverage statistics** > include the number of *unhooked* lines in the coverage statistics

Multiple source code file inclusion

Depending on how you build your software it's possible that some of your source files are present in more than one module (DLL / EXE) in your application.

For some coverage uses, you may prefer to treat these multiple source file inclusions as independent files or as the same file.

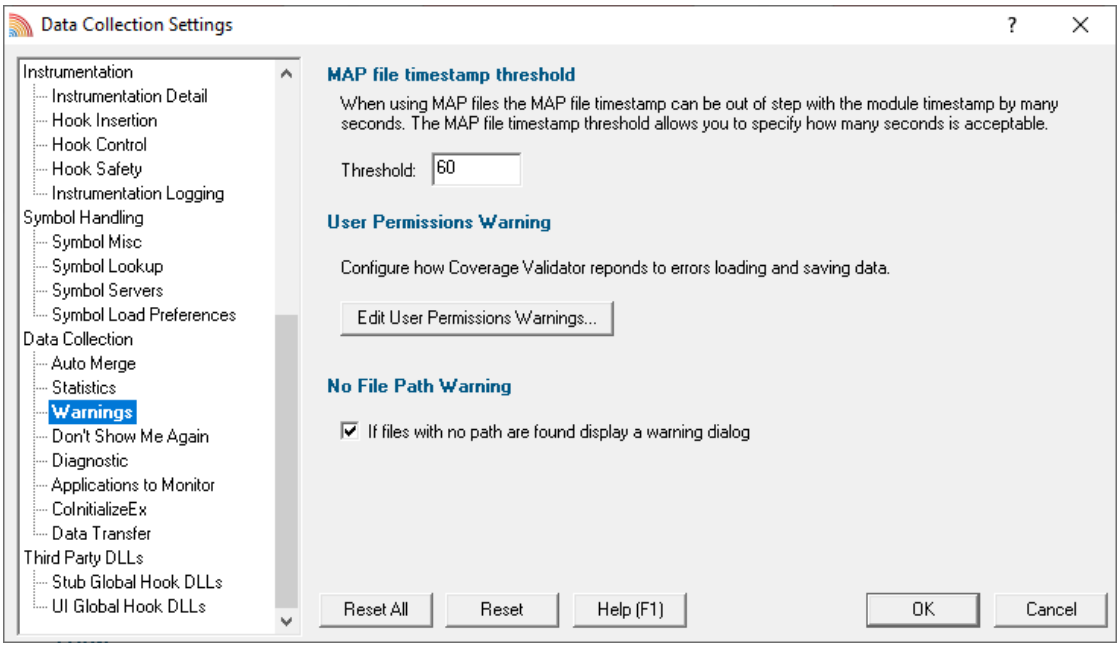
- **Collect statistics per source file for each DLL...** > calculate coverage separately for each inclusion
- **Collect statistics per source file ignoring which DLL...** > calculate coverage for a single instance

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

Reset - Resets the settings on the current page.

3.10.1.5.3 Warning

The **Warning** page allows you to control some of the warnings that Coverage Validator displays.



MAP file timestamp threshold

When working with MAP files the timestamp of a module and its MAP file often differ by many seconds.

You can set the maximum acceptable difference between the two timestamps before the MAP file is considered invalid.

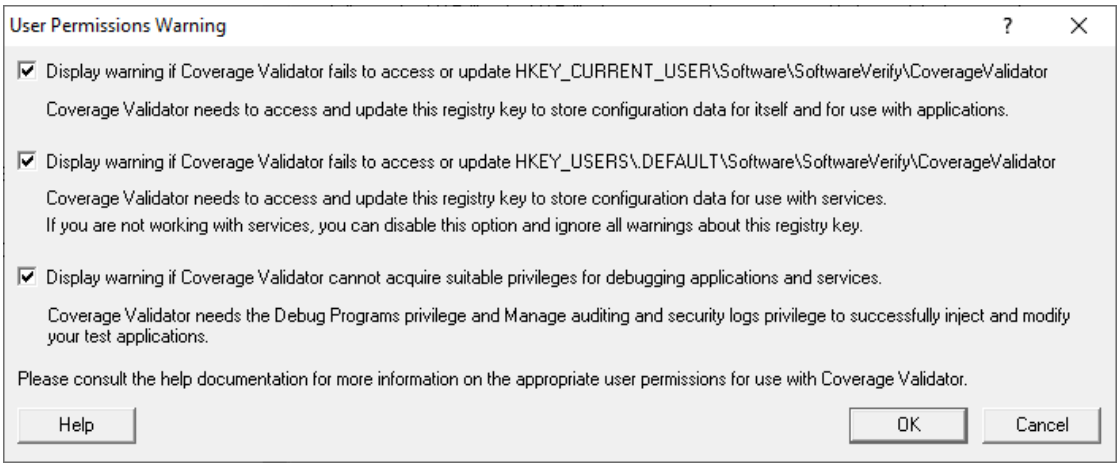
- **Threshold** › enter the maximum acceptable difference (the default is 60 seconds)

User permissions warnings

You may see warning dialogs when Coverage 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:

- **Edit User Permissions Warnings...** › shows the User Permissions Warnings dialog below



The Help button displays the User Permissions help topic.

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

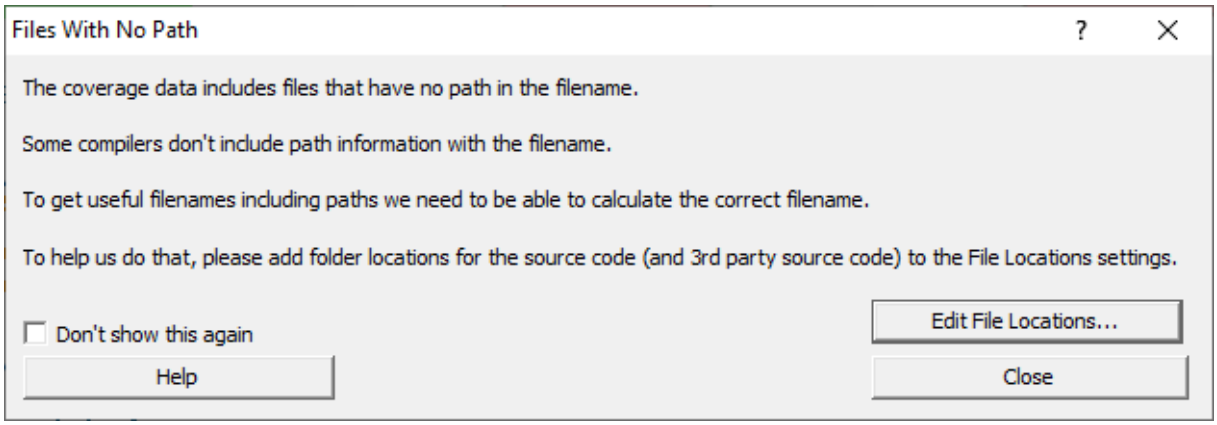
No file path warnings

When Coverage Validator collects code coverage data and finds filenames without paths the No File Path Warning dialog can be displayed.

Without knowing the file path Coverage Validator can't inspect the file for source code exclusion pragmas, and also can't display the source code in any of the displays.

The solution is to declare where source code (and 3rd party source) is located using the File Locations settings.

- **If files with no path are found...** › when enabled the No File Path Warning dialog will be displayed when files without a path are found in code coverage data



- **Edit File Locations...** › displays the File Locations settings dialog set to "Source Code"

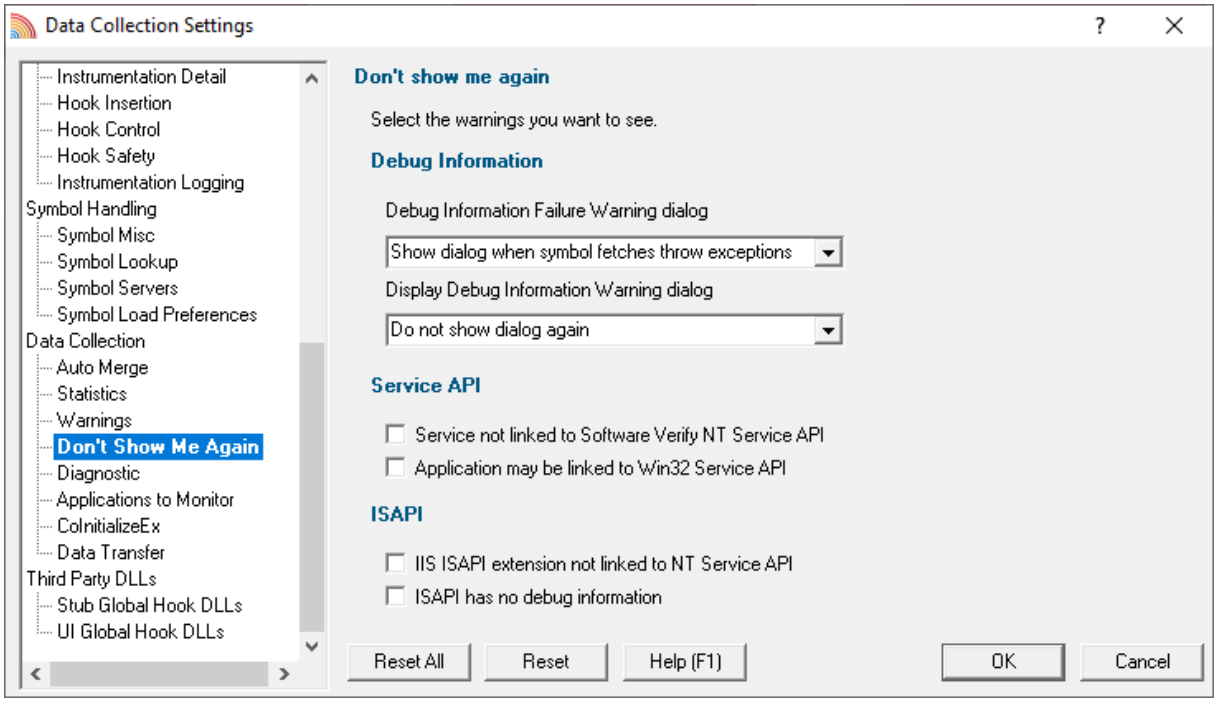
- **Don't show this again...** > turns off the display of this dialog. To re-enable it, select the **If files with no path are found...** check box on the Warnings settings.

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

Reset - Resets the settings on the current page.

3.10.1.5.4 Don't Show Me Again

The **Don't Show Me Again** page allows you to control warnings that Coverage Validator displays.



Debug Information

Debug Information Failure Warning

When there is a failure collecting debug symbols a warning can be displayed. The options are:
- Always show dialog
- Never show dialog
- Show dialog when symbol fetches throw exceptions

Display Debug Information Warning

When no debug information is available for at least one module a warning can be displayed. The options are:

- Always show dialog
- Never show dialog
- Show dialog when debug information is missing

Services API

- **Service not linked to Software Verify NT Service API** › warning will be shown if you try to monitor a service not linked to the Software Verify NT Service API. (on by default)

  When trying to monitor a service Coverage Validator can detect if the service is not linked to the NT Service API and display a warning.

  It is possible to use the service API without linking to it (use GetProcAddress() to lookup the functions and call them) - in this case you would want to turn this warning off.

- **Application may be linked to Win32 Service API** › warning will be shown if you try to start an application that appears to be a service - it uses Win32 Service APIs. (on by default)

ISAPI

NT Service API
When trying to monitor ISAPI extensions Coverage Validator can detect if the ISAPI is not linked to the NT Service API and display a warning.

It is possible to use the service API without linking to it (use GetProcAddress() to lookup the functions and call them) - in this case you would want to turn this warning off.

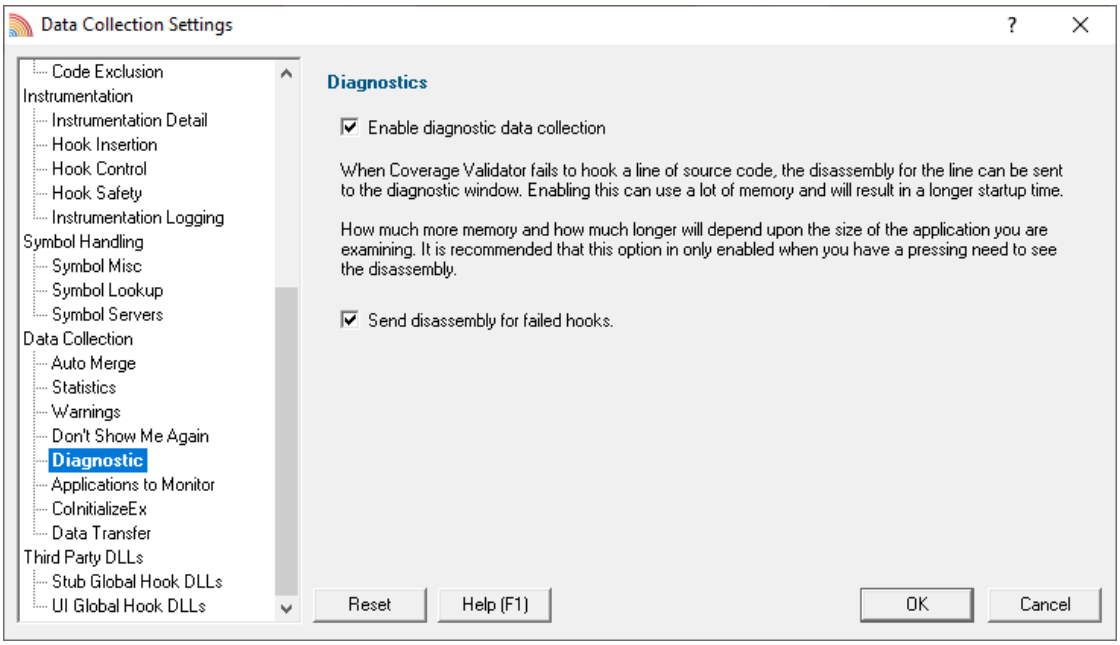
Debug Information
Coverage Validator can warn if the ISAPI has no debug information. There may be cases where you don't want to see this warning.

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

Reset - Resets the settings on the current page.

3.10.1.5.5 Diagnostic

The **Diagnostic** page controls diagnostic display.



Diagnostics

Collection: A lot of diagnostic information is collected and displayed on the diagnostic tab when attaching to a target program.

Some of this information is *always* sent to Coverage 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)

Disassembly: When hooking source code lines, some lines cannot be hooked due to the object code that corresponds to the source code location.

- **Send disassembly for failed hooks** › shows the disassembly for lines that cannot be hooked (enabled by default)

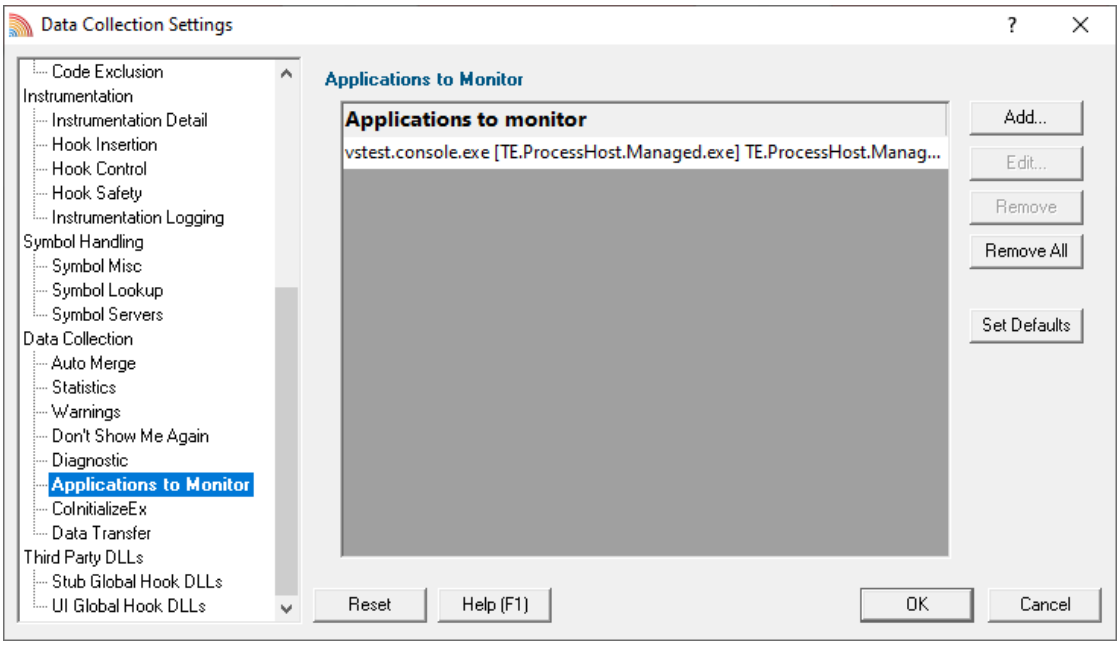
  This *can* increase startup time and memory usage if used very frequently.

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

Reset - Resets the settings on the current page.

3.10.1.5.6 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 are interested in.

Your program may launch child applications and it may be one of *those* that you want to monitor with Coverage Validator.

An example might be for unit testing where a test program spawns one or more child applications, or it might launch the same application multiple times.

The applications to monitor

The main list of **Applications to monitor** shows programs you may want to launch and the child applications they subsequently start - i.e. the you may be interested in monitoring.

Once a definition has been added, you can then use the Application to Monitor setting on the Launch Dialog or wizard to choose which of these child applications you actually want to monitor in a given session.

Managing the applications to monitor

The list contains a set of definitions - each one being for a different launch program.

For each launch program you can set the child applications you might want to monitor later.

An application is defined by its type (native and .Net, or .Net Core), the application executable name, and for .Net Core applications an additional application DLL that is used to identify the application.

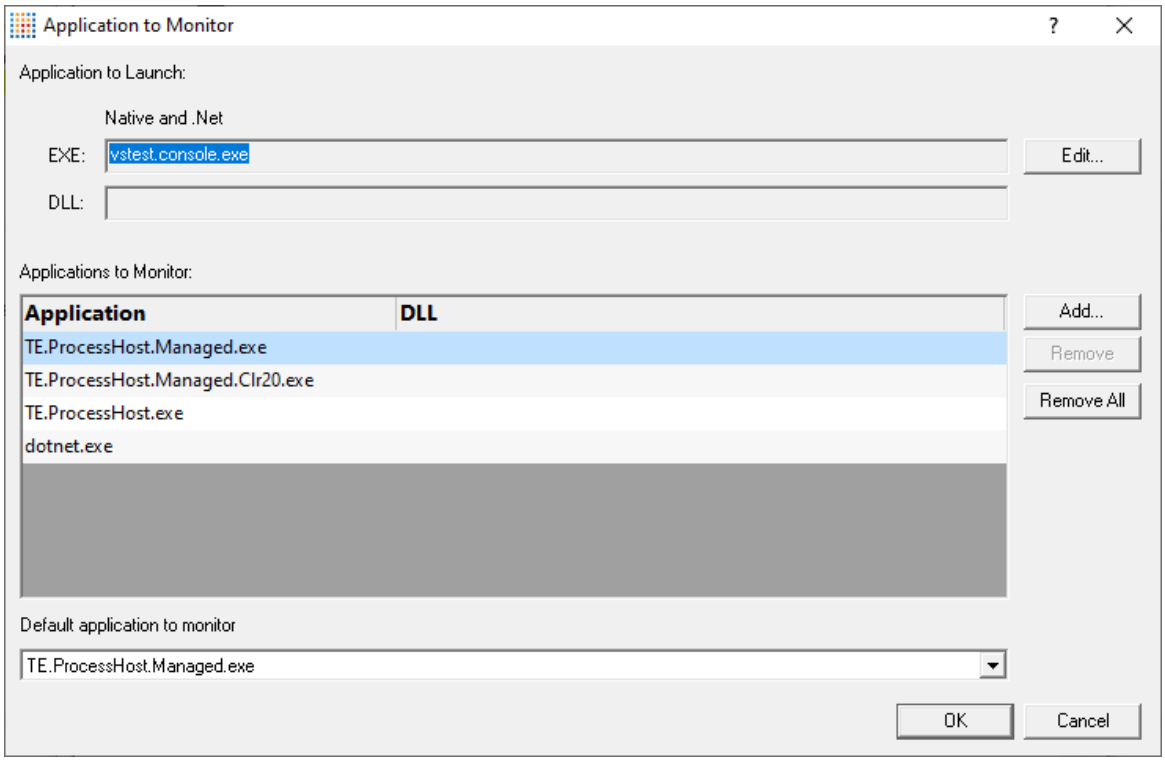
- **Add** › add a new module definition using the **Application to Monitor** dialog below
- **Edit** › modify a selected definition in the list, using the **Application to Monitor** dialog again
- **Remove** › removes any selected definitions in the list
- **Remove All** › clears the list
- **Set Defaults** › reset the list of known applications to those as configured with a new install of Coverage Validator

  The defaults are currently setup for Microsoft's Visual Test software `vstest.console.exe`.

The Application to Monitor dialog

The **Application to Monitor** dialog lets you define or edit a launch program and it's child applications.

The values you specify here are the ones used on the launch dialog and launch wizard to customize which application actually gets monitored.



- **Application to Launch** › **Edit...** to select the initial starting application that will be *launching* the applications you want to monitor

  Any executable names found in the selected program will automatically be displayed in the list of **Applications to Monitor**.

If you don't wish to use these automatic names you can **Remove** them.

- **Add** › add an additional application that you know will be started by the launch program

  Child applications that you add are used *without* the path.

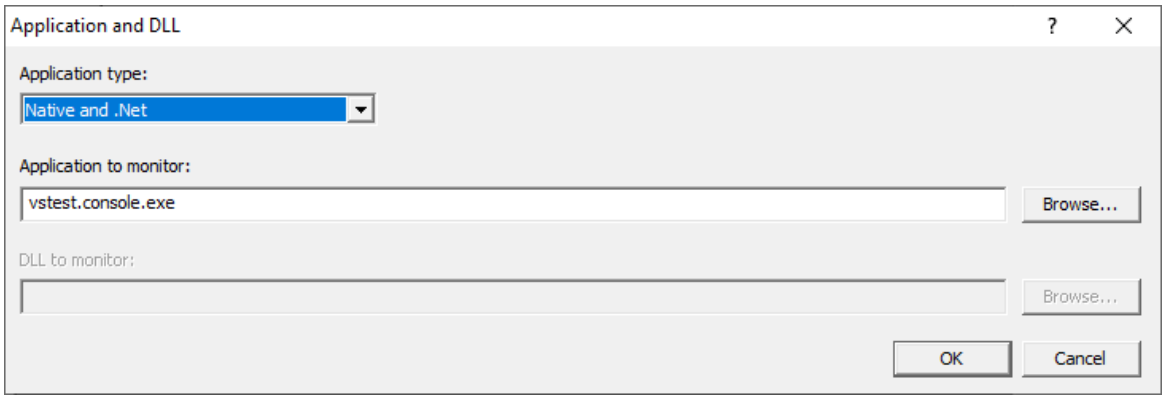
  Excluding the path gives more scope for matching launched application names if they are launched with a different path.

- **Remove** › removes any selected applications in the list
- **Remove All** › clears the list
- **Default application to monitor** › choose the appropriate item to be the default item

  The default application will be selected on the launch dialog (or wizard) whenever the *start* program is specified as the one at the top of this dialog.

The Application and DLL dialog

The **Application and DLL** dialog lets you define or edit a launch program and a launch DLL.



- **Application type** › choose the type of application
  - Native and .Net
  - .Net Core (Framework Dependent)
  - .Net Core (Self Contained)
- **Application to monitor** › edit or **Browse...** the application EXE to monitor.

  This can be an executable name or the full path to the executable. For example **test.exe** or **c:\unitTests\test.exe**.

  For native applications this is the application executable.

  For .Net Framework applications this is the application executable.

For .Net Core Framework-dependent applications this is most likely going to be **c:\program files\dotnet\dotnet.exe**.

For .Net Core Self-contained applications this is the application executable.

- **DLL to monitor** › edit or **Browse...** the application DLL to monitor. This field is only needed for .Net Core applications.

  This can be an executable name or the full path to the executable. For example **test.dll** or **c:\unitTests\test.dll**.

  For native applications this is not used.

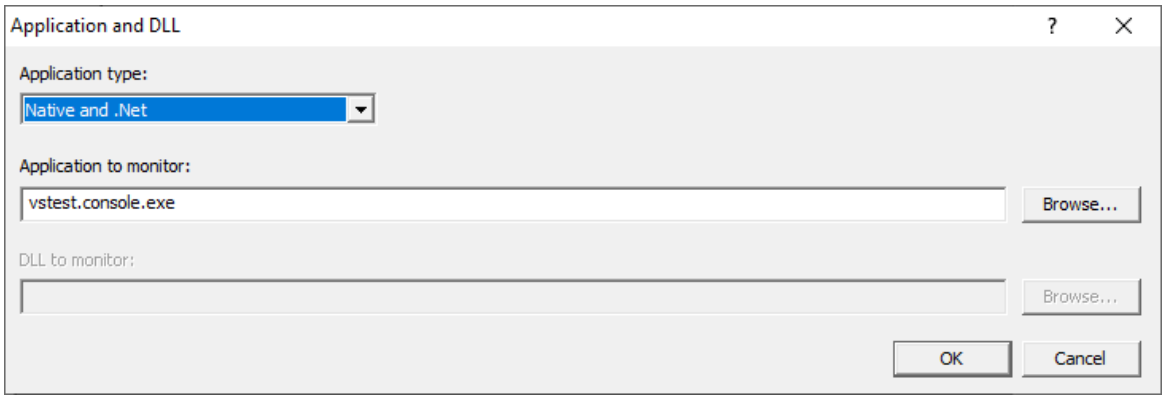
  For .Net Framework applications this is not used.

  For .Net Core Framework-dependent applications this is the application dll. (the name of the dll that you would pass to dotnet.exe on the command line).

  For .Net Core Self-contained applications this is the dll that has the same name as the application executable. (for theApp.exe, the dll name is theApp.dll).

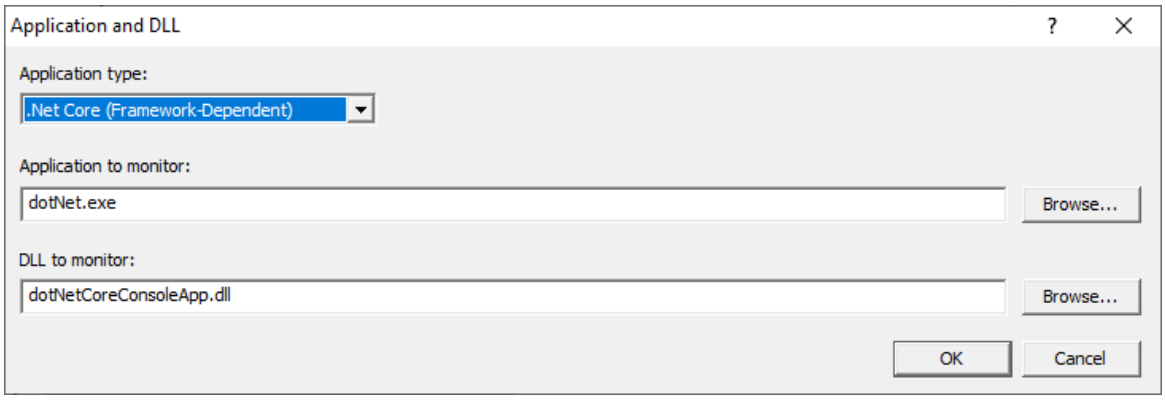
Example Dialogs

Native

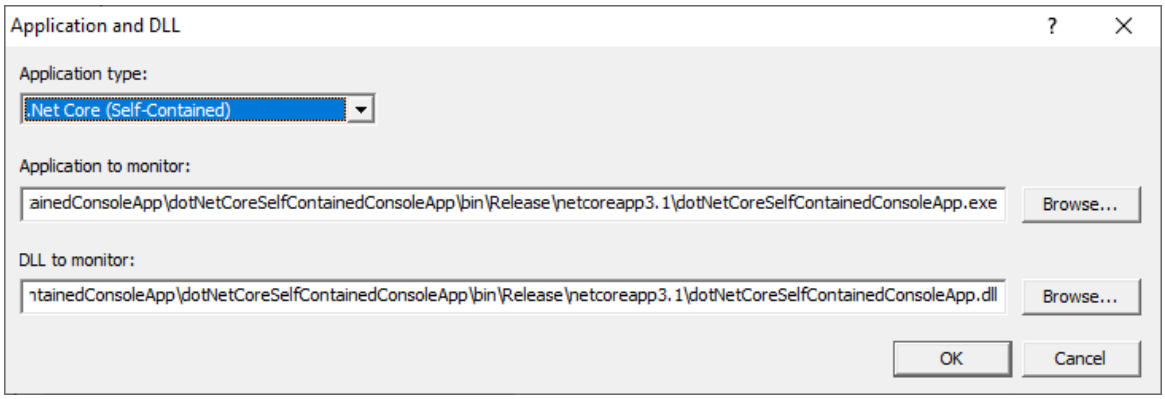


.Net

.Net Core (Framework-dependent)



.Net Core (Self-contained)



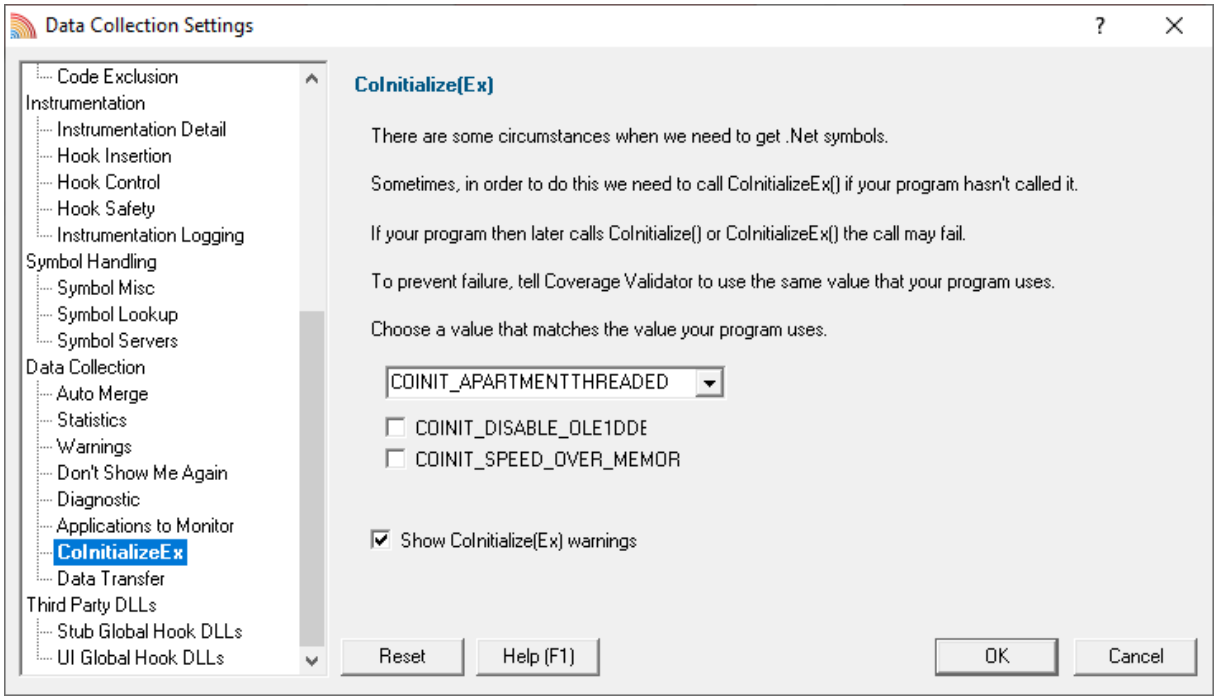
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.5.7 CoInitializeEx

The **CoInitializeEx** tab allows you to set the default behaviour used to initialize COM if Coverage Validator needs to initialize COM to acquire symbols for .Net modules.

The default settings are shown below:



CoInitializeEx

In some situations the Validator needs to get .Net symbols and to do that COM needs to be initialized. This normally isn't a problem, but if your program also performs COM initialization and the sequence of events results in your COM initialization coming after the Validator's COM initialisation rather than getting the expected `ERROR_SUCCESS` return code you'll get either `ERROR_INVALID_FUNCTION` or `RPC_E_CHANGED_MODE`.

If you get `ERROR_INVALID_FUNCTION` this is OK, this just means you've called CoInitialize() or CoInitializeEx() multiple times with the same flags. Your code needs to handle `ERROR_INVALID_FUNCTION` as **not an error**.

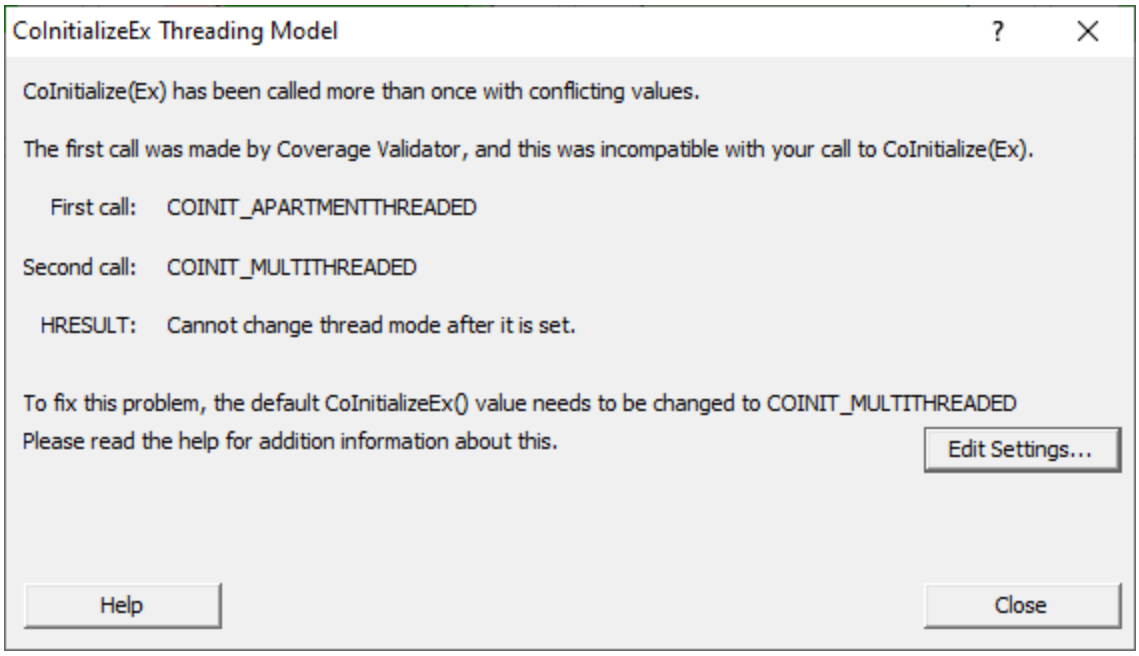
If you get `RPC_E_CHANGED_MODE` this means you need to change the Validator's default value to the same value your program is using. That's what this dialog allows you to do.

If you also wish to disable OLE DDE or favour speed rather than memory use we've provided appropriate options for you to select to add those flags to the threading mode.

See the Microsoft documentation for additional information on the behaviour of CoInitialize() and CoInitializeEx().

Runtime detection of CoInitializeEx conflict

When the above scenario happens, that the Validator has initialized COM before your code initializes COM and your call returns RPC_E_CHANGED_MODE, we display a dialog to warn you about this failure and provide you with the option of editing the default value for subsequent runs of your application.



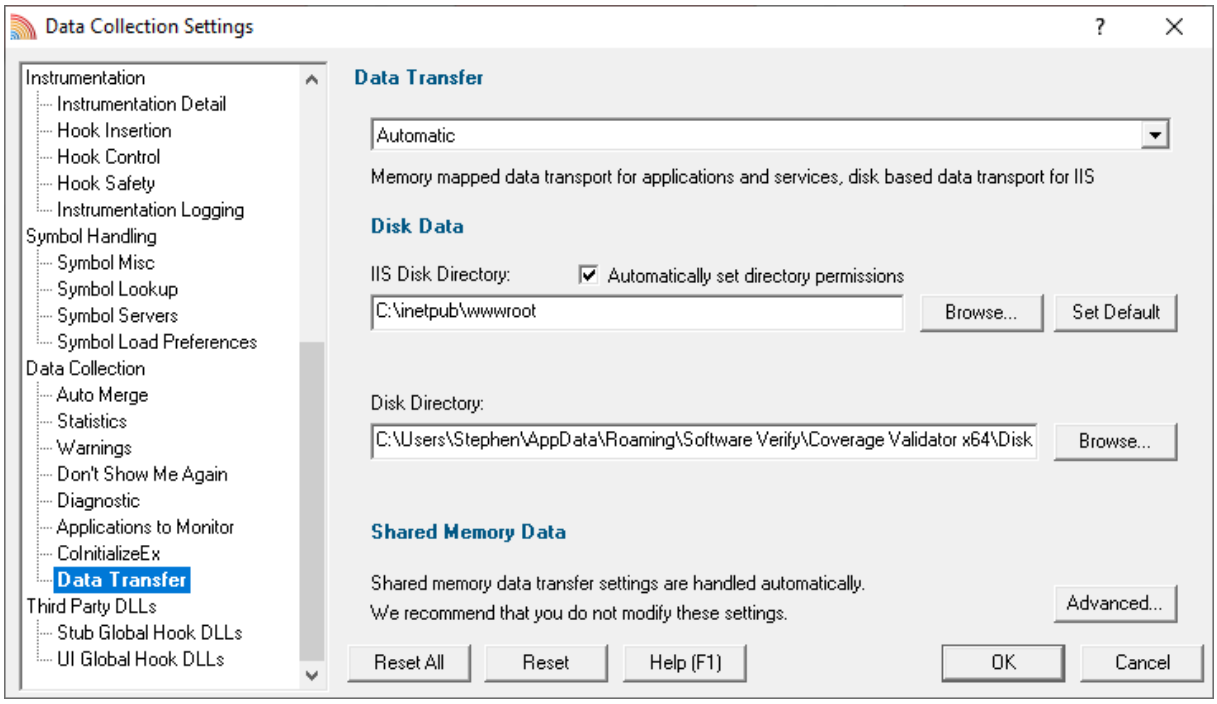
- **Edit Settings...** > opens the CoInitializeEx dialog shown above

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

Reset - Resets the settings on the current page.

3.10.1.5.8 Data Transfer

The **Data Transfer** tab allows you to specify the overall behaviour of data transfer between your application and Coverage Validator.



Data Transport

Choose the type of data transport you wish to use.

- **Automatic**. Applications and services use shared memory to transfer data. IIS uses disk based data transfer.
- **Disk**. Applications, services and IIS use disk based data transfer.
- **High Volume**. Data transfer has no data throttling applied to it. This mode is for use with applications that generate very high volumes of data rapidly. They typically exceed the buffering capabilities of Coverage Validator when working with shared memory. The High Volume setting uses a data transport that doesn't have a data-throttling requirement allowing the high volume application to continue without waiting.

Automatic

Under most circumstances data transfer between Coverage Validator and the target program (desktop, service, etc) is via shared memory. This is handled automatically.

Disk Data

Some applications and services don't allow shared memory access. For these occasions we use a file based data transfer, where the files are stored in a directory of your choice.

We provide two options for this, one for most applications and services, and one for Internet Information Server, as this operates in a very restricted environment.

Both options are configured automatically, but you can override either by typing the path to a suitable directory or using the Microsoft directory browser.

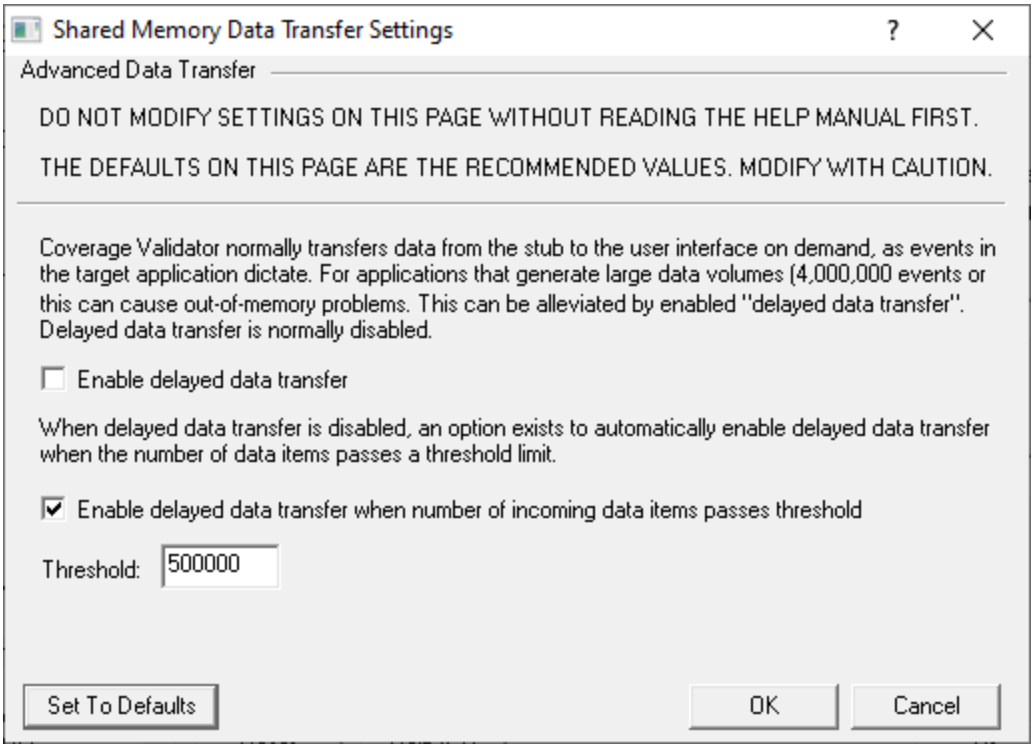
The ISS path you enter will be determined by the settings you have configured for IIS using the Internet Information Services Manager tool. We won't discuss that here because if you're using IIS we assume you already know how to configure IIS correctly.

Advanced Shared Memory

Shared memory data transfer can also be configured **but we strongly recommend that you leave these settings alone**.

The **Data Transfer Helper** is a separate application supplied in the installation directory.

- **Advanced...** › opens the data transfer settings dialog.



Here be dragons!

Caution: Modifying the settings on this page and using the data transfer helper application can prevent Coverage Validator from working correctly.

- **Set To Defaults** › if you have modified the settings, this resets them

➡ See also the Reset to default buttons on the data transfer helper application below

If in doubt, don't modify these settings. If you promise to be careful, read on!

Delayed data transfer

Delayed data transfer is the process of throttling data rates in the stub so that the slower user interface can keep up with processing the data received.

In the stub, as an event occurs, data is queued and then sent to the user interface.

In the user interface, data from the stub is received and queued again for processing.

Any delay is usually in the slower user interface, but still not a problem for most applications.

However, some data intensive applications can generate so much data that the user interface gets swamped and can't process it all before running out of memory.

Temporarily limiting the data rate in the *stub* allows the user interface to stabilize the data processing.

Managing data rates

We recommend the default settings as shown above:

- disable delay data transfer for most applications
- enable *automatic* delay data transfer at a threshold of between 100,000 and 1,000,000 data items

If delayed data transfer is enabled all the time, the automatic options don't apply.

If you have more than 1GB RAM, you can raise these thresholds.

Data transfer helper application

A separate data transfer helper application is supplied in the installation directory.

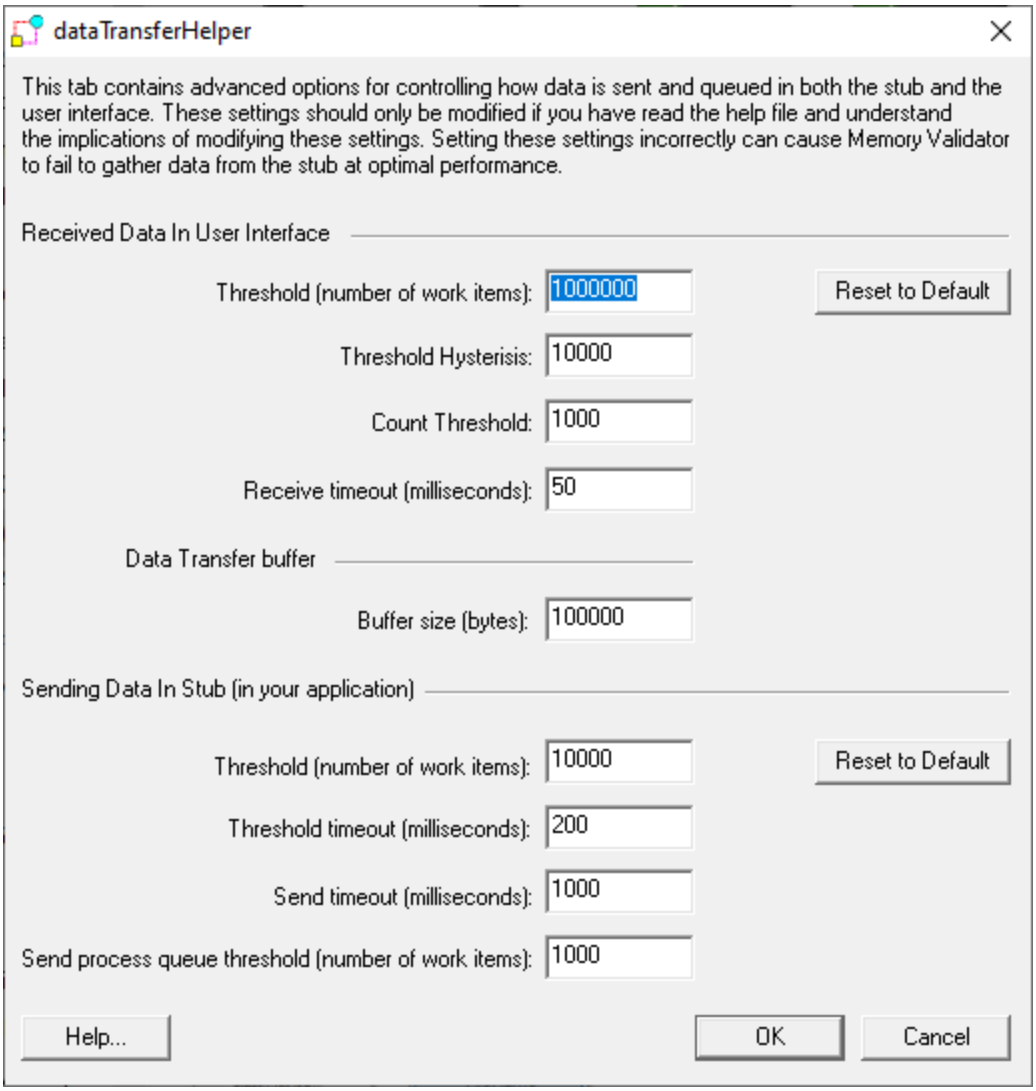
The helper application can be used to modify low level settings that apply when delay data transfer is activated as above.

 The helper should be used with care. We already warned of dragons above, but here we are, warning you again!

An HTML help page for this application is available by clicking the **Help** button on the helper application.

You can also find the help page directly as `dataTransferHelp.html`.

Please do take a moment to read the help before use.



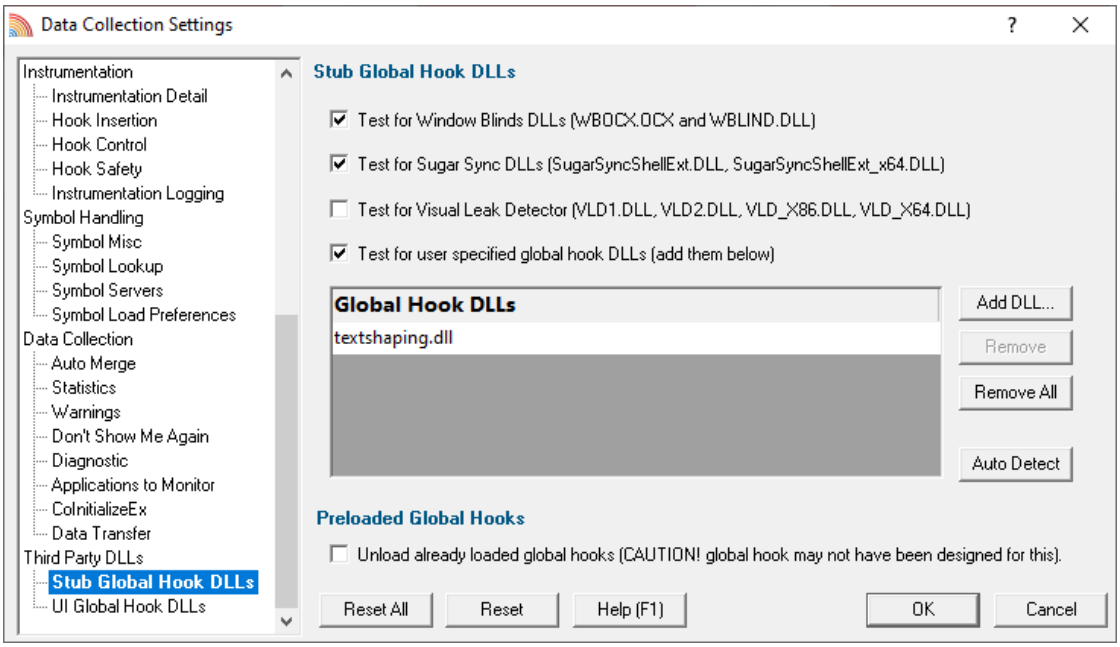
Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.6 Third Party DLLs

3.10.1.6.1 Stub Global Hook DLLs

The **Stub Global Hook DLLs** tab allows you to detect and specify global hook DLLs that may not be wanted in the stub process.



About global hook DLLS

Some third party products such as storage devices and video cards are supplied with software to help integrate the hardware device into the computer desktop environment.

An example is the Iomega® Zip® drive. This uses a global hook via the IMGHOOK.DLL which allows the *browse for files* and *browse for folders* interfaces to correctly display all the storage devices on the computer, including the zip drive and any special options for the drive.

Some global (or *system*) hook DLLs can interfere with the correct operation of Coverage 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 Coverage Validator attached to the process.

Managing global hook DLLs

- **Test for Window Blinds...** › test for Window Blinds DLLs loading into your application, and prevent them from loading

  Window Blinds DLLs WBOCX.OCX and WBLIND.DLL are not compatible with Memory Validator.

- **Test for Sugar Sync...** › test for Sugar Sync DLLs loading into your application, and prevent them from loading

Sugar Sync DLLs SugarSyncShellExt.dll and SugarSyncShellExt_x64.dll are not compatible with Memory Validator.

- **Test for Visual Leak Detector...** › test for Visual Leak Detector DLLs loading into your application, and prevent them from loading

  Visual Leak Detector is not compatible with Memory Validator. If you are linked to Visual Leak Detector you'll need to create a build without Visual Leak Detector to use with Memory Validator.

- **Test for user specified...** › test for user specified DLLs loading into your application, and prevent them from loading
- **Add DLL...** › browse and select one or more DLLs › **Open** › adds the chosen DLLs to the **Global Hook DLLs** list
- **Remove** › removes any selected DLL from the list
- **Remove All** › removes all DLLs from the list

Auto detecting global hook DLLs

Coverage 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

Additionally, you can request unloading of any of the listed global hook DLLs that are detected as already loaded into the target process when Coverage Validator attaches to it:

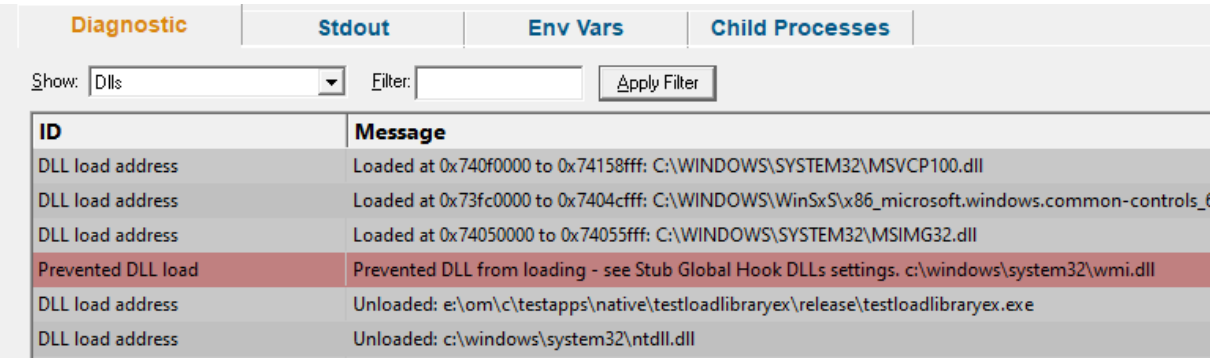
- **Unload already loaded global hooks** › when ticked, forces an unload of any listed global hook DLLs

   Use this with caution, as not all global hook DLLs may have been designed or intended for this!

Viewing the diagnostic information

If a DLL is prevented from loading because of these settings, or is allowed to load because of these settings, there will be an entry on the Diagnostic tab.

To view this data, go to the Diagnostic tab, select the Diagnostic sub tab, then set the Show combo box to "Dlls". All information about DLLs will be shown. Scroll through the list looking for "Prevented DLL load" in the left hand column. The right hand column will indicate if a DLL was prevented from loading, or allowed to load. The DLL name will also be shown.

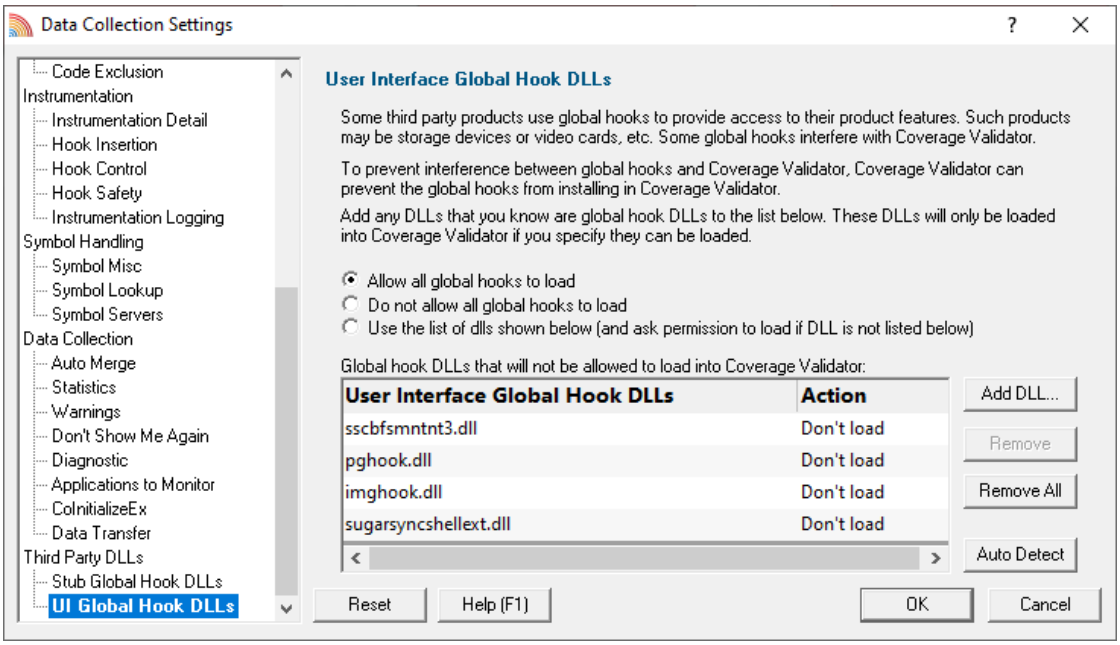


Reset All - Resets **all** global settings, not just those on the current page.

Reset - Resets the settings on the current page.

3.10.1.6.2 User Interface Global Hook DLLs

The **User Interface Global Hook DLLs** tab allows you to detect and specify global hook DLLs that may not be wanted in the Coverage Validator user interface process



About global hook DLLS

See the similar topic on stub global hooks to read about global hook DLLs

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 Coverage Validator
- **Do not allow any global hooks to load** › prevent any global hook DLLs from loading (the default)
- **Use the list of dlls shown** › provide per-DLL control over which DLLs load or don't load via the **User Interface Global Hook DLLs** list

  Any global hook DLLs not listed will result in the user being asked for permission to load a DLL via the Global Hook Warning Dialog below

Managing user interface global hook DLLs

- **Add DLL...** › browse and select one or more DLLs › **Open** › adds the chosen DLLs to the **Global Hook DLLs** list

  Having added a DLL to the list, you can change whether the DLL is allowed to load or not, by double clicking in the second column and changing the value: **Load** or **Don't load**
- **Remove** › removes any selected DLL from the list
- **Remove All** › removes all DLLs from the list

Auto detecting global hook DLLs

Coverage 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:

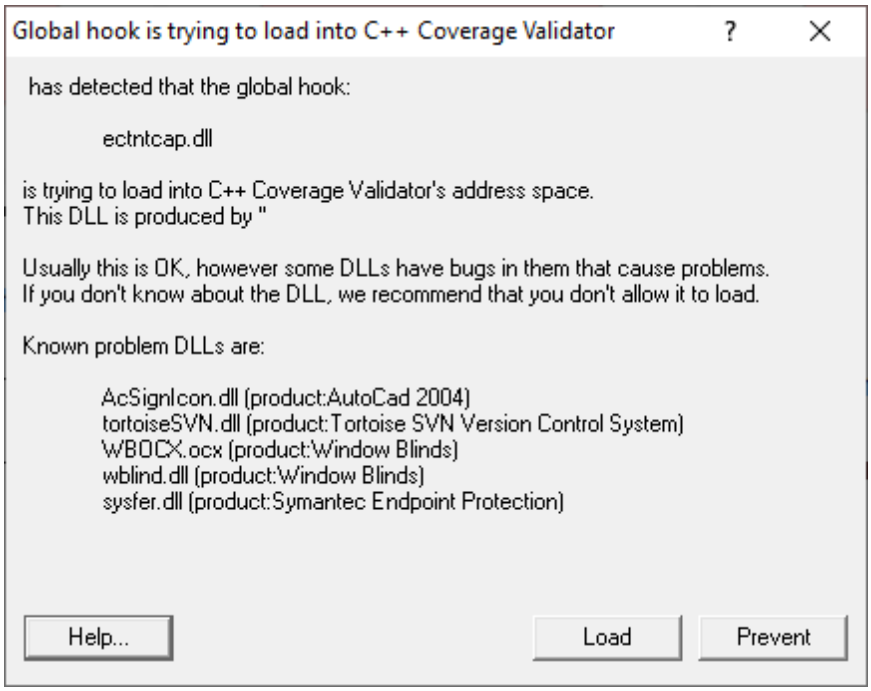
- **Auto Detect** › automatically detect DLLs which may be global hook DLLs, *adding* them to the **Global Hook DLLs** list

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.



- **Help** › displays this help page
- **Yes** › lets the DLL load
- **No** › blocks the DLL

Your response is automatically recorded in the **Global Hook DLLs** list, so that you won't be asked again.

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

Reset - Resets the settings on the current page.

3.10.2 Loading and saving settings

Saving and loading settings files

Coverage 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

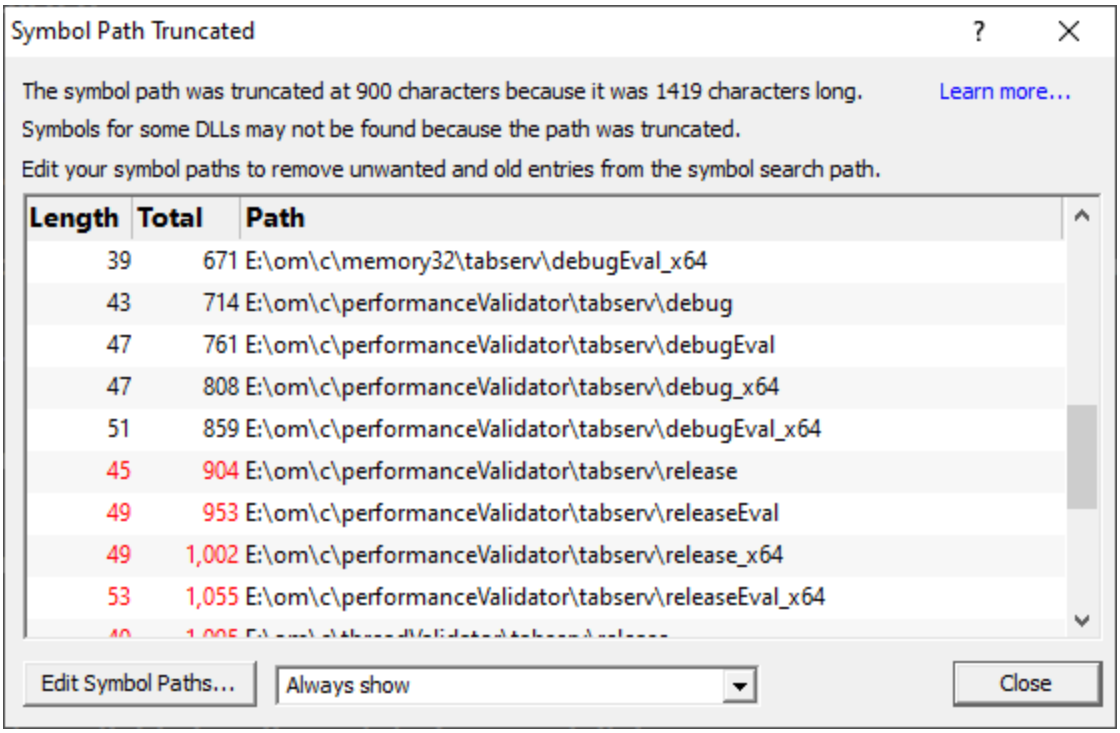
Loading settings via command line option

Different settings files can be used in the command line interface by using the -settings command line option and the required file path to load the settings when Coverage Validator starts up.

3.10.3 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.



- **Edit Symbol Paths...** › shows the file locations dialog so that you can edit the paths used for subsequent runs of the program.

You can choose when this dialog is displayed.

- **Always show** › The dialog is always shown when the symbol path is too long.
- **Show when path changes** › The dialog is shown when the symbol path is too long, but only if the symbol path is different than last time this warning was shown.
- **Never show** › The dialog is never shown.

Whether this dialog is displayed or not there is always a warning message written to the diagnostic window when the symbol path is truncated.

The display lists each path with it's length (including the unshown ';' path separator) and the total length so far so that you can see which paths exceed the truncation point (length and total displayed in red).

Any paths that don't exist on this computer are displayed in red.

Why is this dialog displayed?

You may see a Symbol Path Truncated warning dialog in some rare circumstances.

This dialog is displayed when the symbol path that has been calculated to pass to DbgHelp.dll to load Microsoft debugging symbols (found in .PDB files) is too long.

If the symbol path has been truncated because it is too long it is possible this may mean that some symbol searches will fail, resulting in failure to load some symbols. We display this dialog so that you are aware that the symbol path is too long and would benefit from editing to make the symbol path shorter.

Passing a symbol path that is too long to DbgHelp.dll will cause the program being tested to end with an EXCEPTION_INVALID_CRUNTIME_PARAMETER C runtime error. This happens because internally DbgHelp.dll is using a fixed length array to format a string. To prevent this fatal termination of the test program we limit the length of the path passed to DbgHelp.dll.

Typically if a path that is long enough to cause this problem is passed to DbgHelp it's because the number of paths in the calculated path contain paths not relevant to finding symbols for the test program. We use the Symbol Path Truncated warning dialog to show you the calculated paths so that you can work out which paths to delete.

The calculated symbol paths come from several places:

- File locations PDB paths
- Symbol server symbol storage directories
- Symbol handling environment variables

Fixing the symbol path

For this example, we are testing the program **E:\om\c\3RD_SRC\cdplayer\Release\cdplayer.exe**

In the image shown above you can see that seven paths exceed the truncation limit, one of the 7 paths doesn't exist.

To work out what to do we need to do several actions:

1. Looking at the environment variable settings shows that none of the environment variables are being used. We do not need to consider the content of these environment variables.

2. Examining the symbol servers shows that **C:\Users\Admin** is a local symbol storage location. We should keep this path.

3. We should delete the path that doesn't exist: **E:\om\c\testApps\testStdinStdoutRedirectEx\Release**. We do this using the file locations dialog by clicking **Edit Symbol Paths...** then click **Delete invalid**.

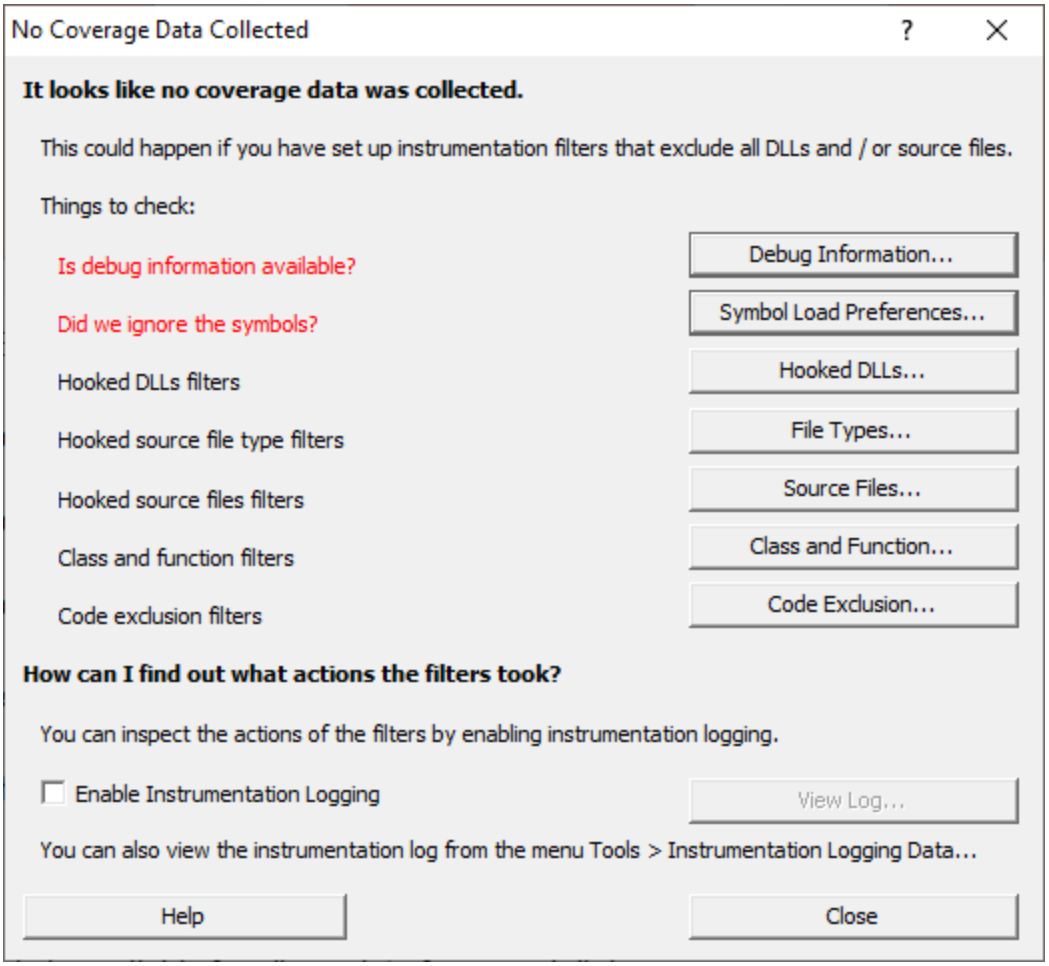
4. Examining the paths in the file locations dialog we can identify any paths not relevant to the program we are testing. In this case the following paths are not relevant and can be deleted.

E:\om\c\testApps\testStdinStdoutRedirect\Release
E:\om\c\testApps\testAppTheReadsFromStdinAndWritesToStdout\Release
E:\om\c\testApps\testSimpleMemoryLeak\Release
e:\om\c\3rd_src\cppunit-1.12.1\examples\cppunittest\release
e:\om\c\3rd_src\cppunit-1.12.1\examples\cppunittest\releasedll

3.10.4 No Coverage Data Collected Warning

The No Coverage Data Collected warning dialog

The no coverage data collected warning dialog is displayed to warn you no coverage data has been collected.



- **Debug Information...** > shows the DLL Debug Information dialog so that you can inspect which DLLs have debug information and understand why debug information may have failed to load.
- **Did we ignore...** > shows the Symbol Load Preference dialog so that you can inspect which symbol types you are interested in and which you are ignoring. Sometimes you'll fail to load symbol information because you have your symbol load preferences incorrectly configured. In in doubt set them to "All Compilers".
- **Hooked DLLs...** > shows the Hooked DLLs dialog so that you can choose which DLLs are processed for code coverage.
- **File Types...** > shows the Source File Types dialog so that you can choose which source file types (file extensions) are processed for code coverage.
- **Source Files...** > shows the Source Files dialog so that you can choose which source files are processed for code coverage.
- **Class and Function...** > shows the Class and Functions dialog so that you can choose which functions, classes and class methods are processed for code coverage.
- **Code Exclusion...** > shows the Code Exclusion dialog so that you can edit the code exclusion parameters that affect code coverage processing.

Why is this dialog displayed?

You will see a No Coverage Data Collected warning dialog in whenever no code coverage data is collected.

There are two main reasons for not getting any code coverage.

- No debug information.
- Instrumentation filters excluding too much data.

No Debug Information

There are many reasons that debug information may fail to load. This is discussed in detail in the DLL Debug Information topic.

Instrumentation Filters

Instrumentation filters are useful for focusing code coverage on the parts of your program that you are interested in.

However it's also possible to create combinations of filters than when used together exclude all code coverage locations from the result set. This results in no code coverage data.

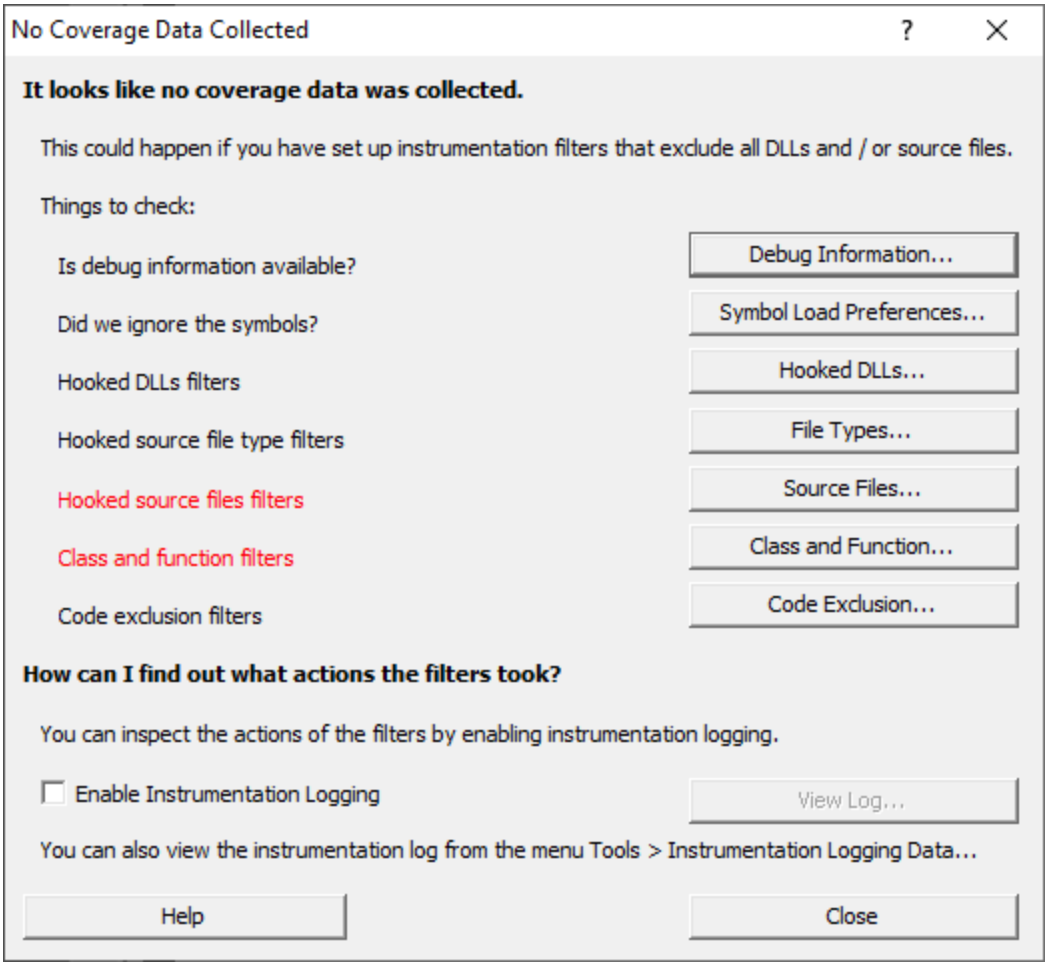
The five instrumentation filter groups are Hooked DLLs, Source File Types, Source Files, Class and Function filters and Code Exclusion pragmas.

You will need to examine the various filters to identify which filters need editing to prevent all code locations from being excluded from a code coverage report.

Items coloured in red are items that Coverage Validator thinks may be causing the problem. In the above image Is debug information available? indicates that the Debug Information should be inspected to ensure that the debug information has been found, and if not, why wasn't it found?

The Instrumentation Log

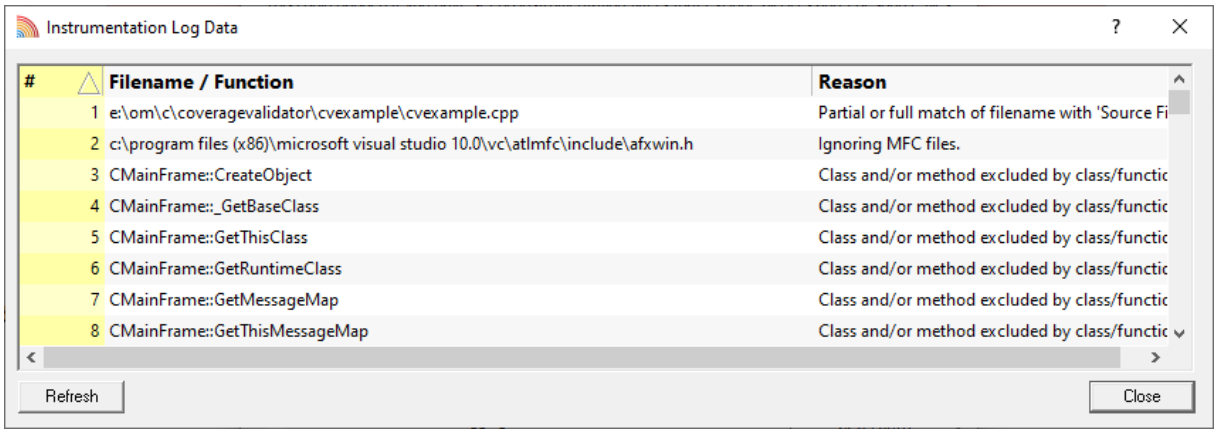
Sometimes it's hard to understand the behaviour of the filters by just looking at the filters.



It would be useful if you could see a log of the behaviour of the filters.

To do that select the **Enable Instrumentation Logging** check box and re-run your code coverage test.

When there is an instrumentation log to view you can view it using **View Log...** on this dialog or **Tools** menu > **Instrumentation Logging Data...**



In the above image we can see:

- a filename match
- some MFC files are ignored
- some class and function filter matches
- some C runtime files are ignored
- in the area of the log we can't see are also entries describing DLLs that have been ignored (Microsoft and C runtime DLLs)

The filename match and the class and filter matches are of interest to us. We'll need to change these to ensure we get some code coverage results.

3.11 Managers

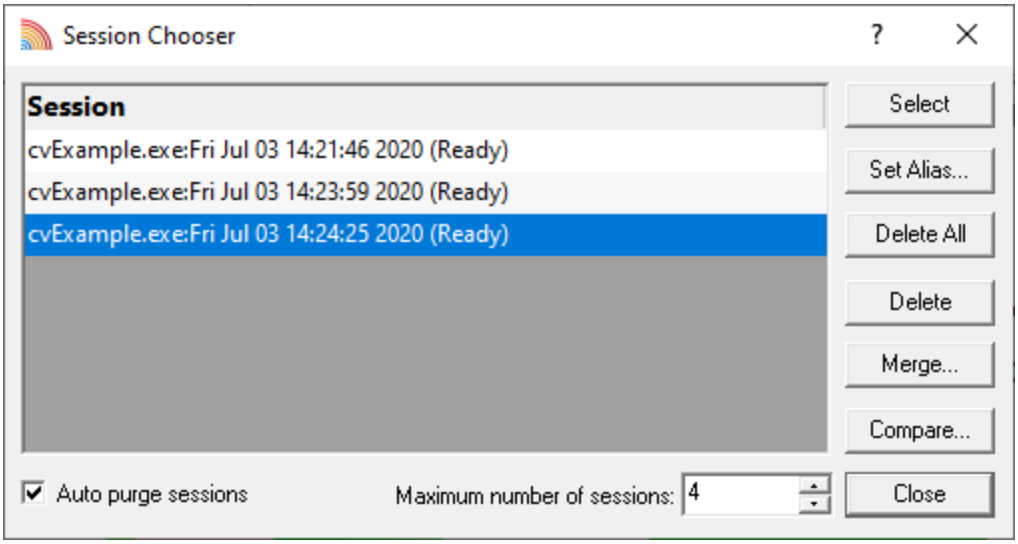
3.11.1 Session Manager

Managing multiple sessions

Coverage Validator can manage multiple sessions at once.

As well as the actively running session, open sessions may include those run since Coverage Validator started, or reloaded sessions that had been saved earlier.

Managers menu › **Session Manager...** › shows the **Session Chooser** dialog below, highlighting the current session



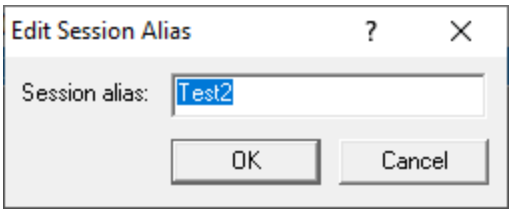
Each time a session is started or loaded it is added to this list, using the name of the executable program and the date and time the session started.

Managing the sessions

- **Select** › makes the selected entry the *current* session, i.e. the one for which data will be displayed

  Some tab views may update immediately, others may need a manual refresh

- **Set Alias...** › opens the **Edit Session Alias** dialog so you can give the session a more useful name



- **Delete** › removes the selected session

  You can't delete a session that is actively collecting data.

- **Delete All** › removes all the loaded sessions

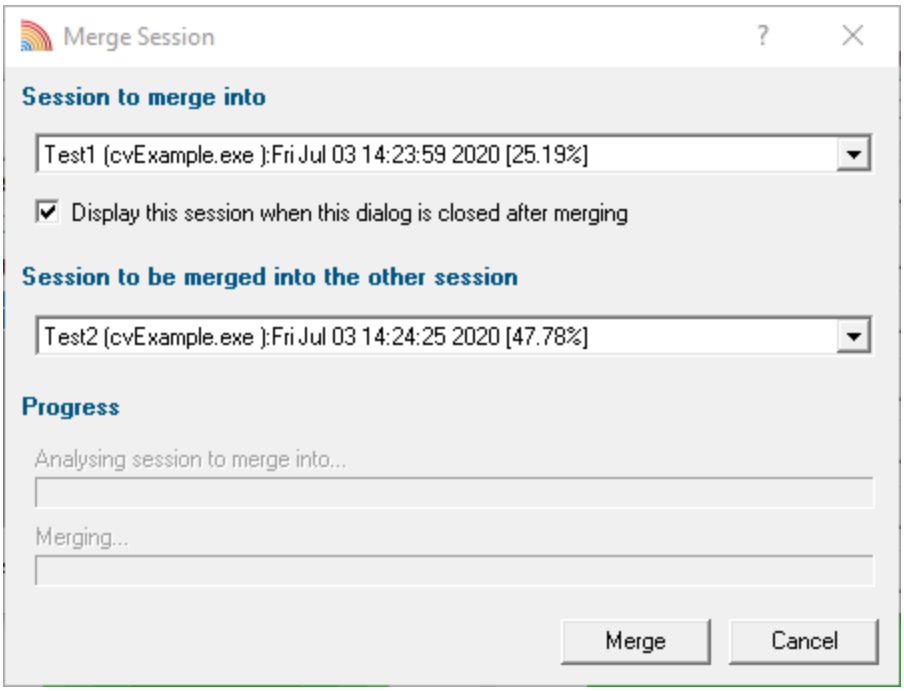
  If one of the session is actively collecting data, this will be disabled.

- **Close** › closes the dialog (as opposed to closing any selected sessions!)

Merging sessions

When two different sessions are loaded they can be merged into one.

- **Merge...** › shows the **Merge Session** dialog



Choose the two sessions that you want to merge.

- **Session to merge into** › pick the final destination session for all merged data

Optionally use the resulting statistics to update the tab views (the default)

- **Display this session...** › uncheck to *not* update the views

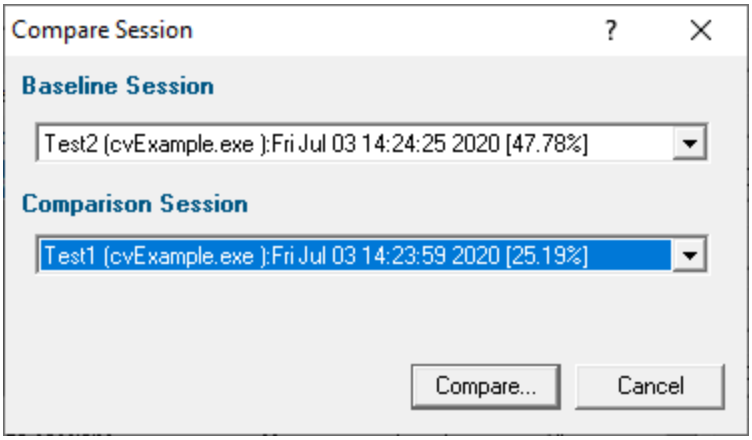
- **Session to be merged into the other session** › pick the 'other' session
- **Merge** › starts the merge, displaying progress as it goes

Note that the merged session data is *not saved to disk*. To save it, **Select** the resulting merged session to make it current, and then save the session.

Comparing loaded sessions

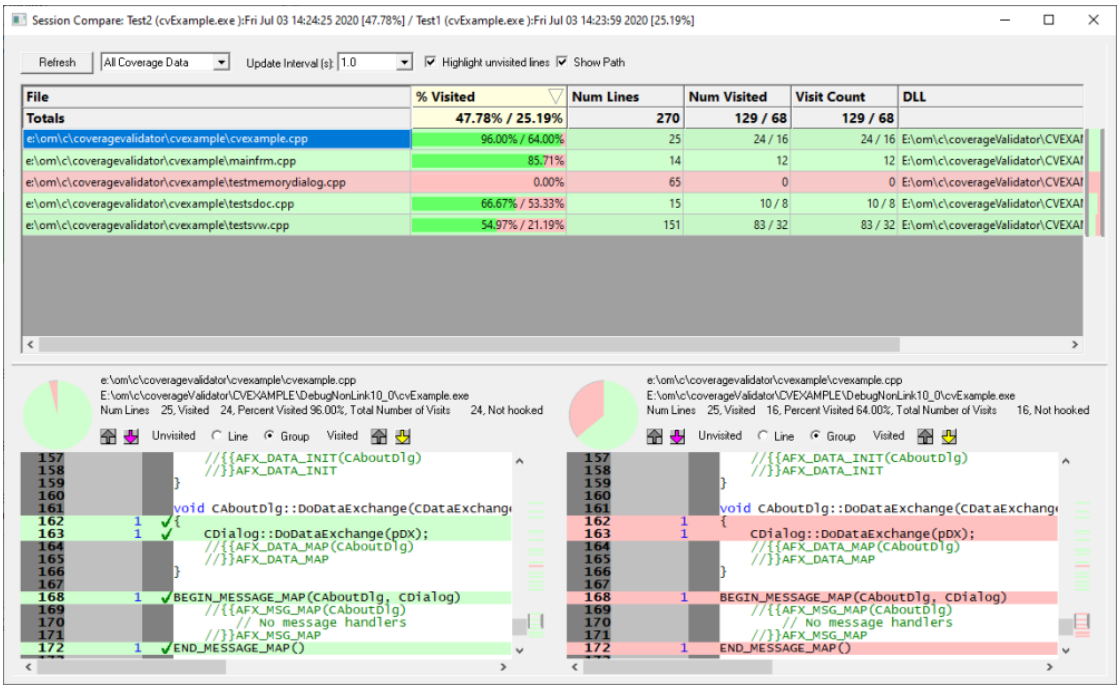
When two different sessions are loaded they can be compared as part of a manual regression test.

- **Compare...** › shows the **Compare Session** dialog for comparing coverage.



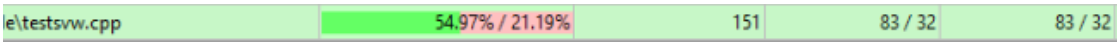
Select the two sessions to compare:

- **Baseline session** › the session you want to compare *against*
- **Comparison session** › to compare against the baseline
- **Compare** › shows the **Session Compare** dialog



The results of the comparison are shown in the top half.

In the visit statistics, entries with the same value are displayed once, while values that differed in each session have both values displayed.



To view the code coverage comparison for a particular file, select that file in the list at the top.

Scrolling one source code window will scroll the other source code window.

Using the drop-down list at the top, you can show **All Coverage Data**, or just the **Differences only**.

The other options in the dialog are identical to those in the Coverage tab.

Limiting the number of sessions

You can choose to limit the maximum number of sessions open at once. Once the maximum is reached, then each time a new session is added, the oldest session may automatically be removed:

- **Auto purge sessions** › ensures that the number of loaded sessions is limited to the maximum (below)
- **Maximum number of sessions** › sets the maximum number of sessions allowed if auto-purge is on

3.12 Query and Search

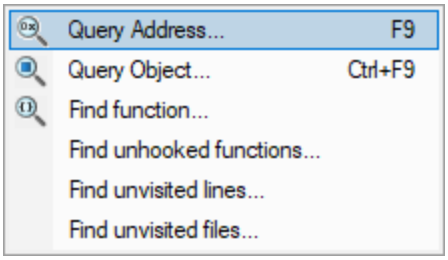
About Query and Search

Coverage Validator provides a handful of query tools to enable you to find particular coverage data collected in each session:

Searches can include:

- Query Address › Search the data for lines at or near an address
- Query Object › Search for lines belonging to a C++ object type
- Find function › Search for lines occurring in functions matching a function name of interest
- Find unhooked functions › Browse a list of functions that have not been hooked (or those that have)
- Find unvisited lines › Browse all lines that have not been visited, or those that have
- Find unvisited files › Browse all files that have not been visited, or those that have.

Click on an item in the picture below to jump to the relevant topic:



3.12.1 Finding addresses

Finding lines by address

Using the **Find Line by Address** dialog below, you can search the data for lines at or near an address.

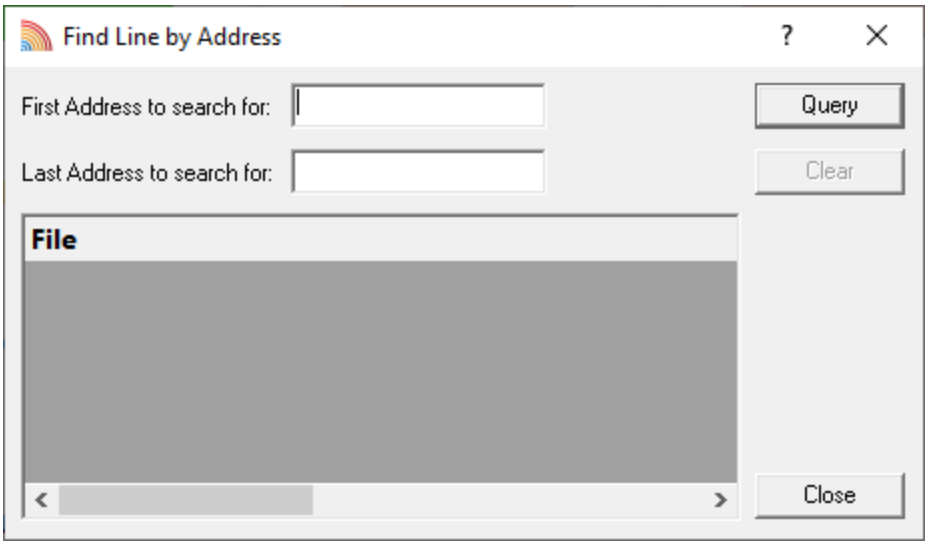
Searches can include:

- lines exactly at an address of your choice
- lines appearing within two chosen address

To show the dialog, choose the menu option below:

Query menu › choose **Query Address...** › displays the Find Line by Address dialog

Or use the following icon on the Query Toolbar.



Search criteria

The query is made at an exact address, or between a range of addresses.

The first search address is essential, but the second search address allows for some optional tolerance or range.

- **First address to search for** › enter the memory address of interest

  The format of the values can be decimal or prefixed with 0x for a hexadecimal format.

- **Last address to search for** › enter the end of the address range in which to search (optional)

  The last address must be the *larger* of the two.

Query results

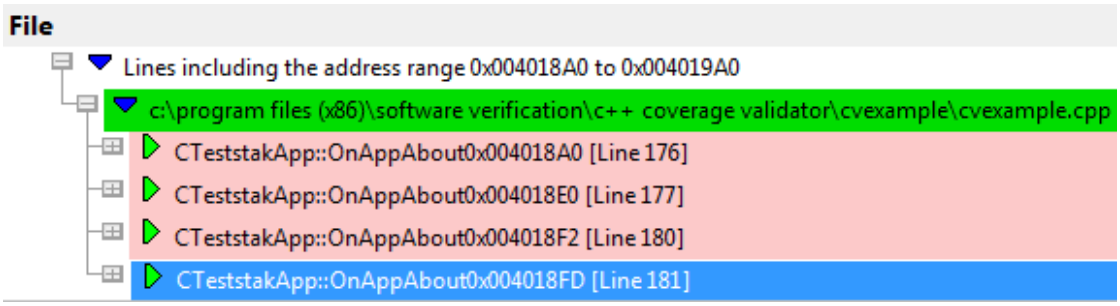
- **Query** › performs the search

  The search results are added to the list in the dialog.

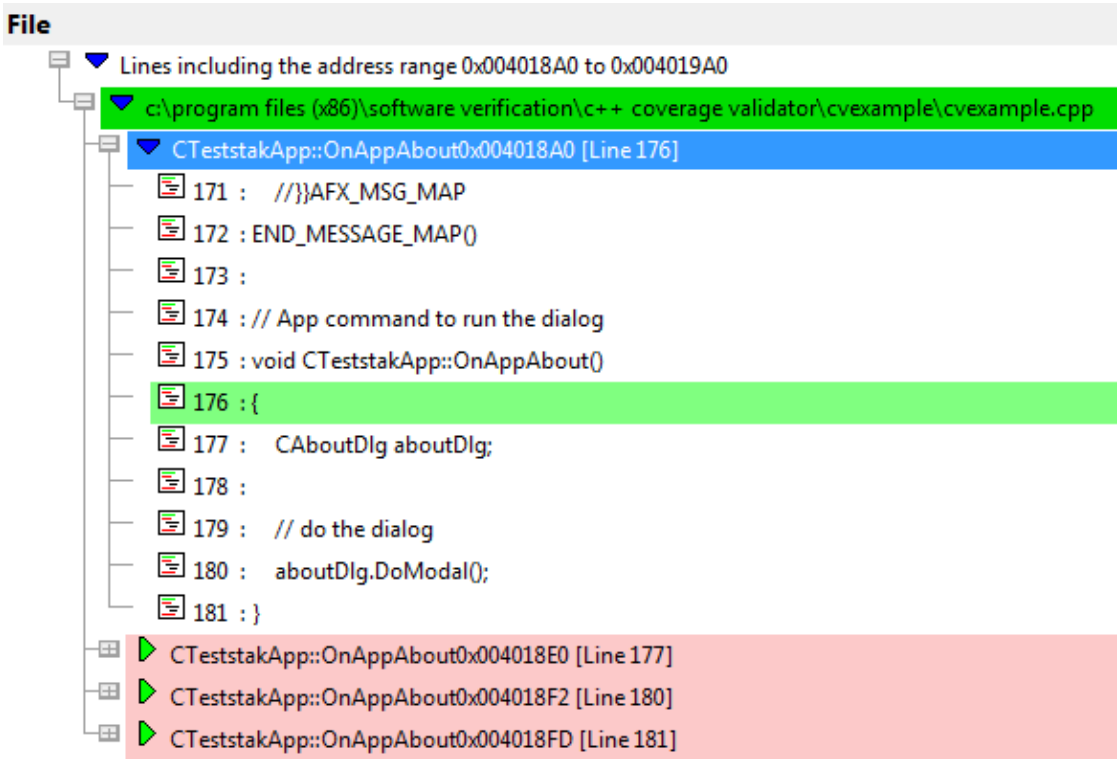
- **Clear** › clears all the results from the list

  The list is not automatically cleared with each search, so you can accumulate or compare results of different searches

Example: While running the example application, searching between addresses 0x004018a0 and 0x004019a0 you might get a result like this:

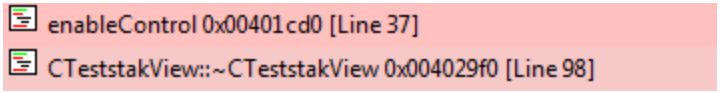


You can expand the search results, and double click the data items to edit source code in your preferred editor.



The amount of code shown can be controlled via the source browsing settings, for example the whole function or a few lines either side of the result.

If you want to try this out, but don't know what addresses to use, the Functions tab shows addresses against each function listed.



3.12.2 Finding objects

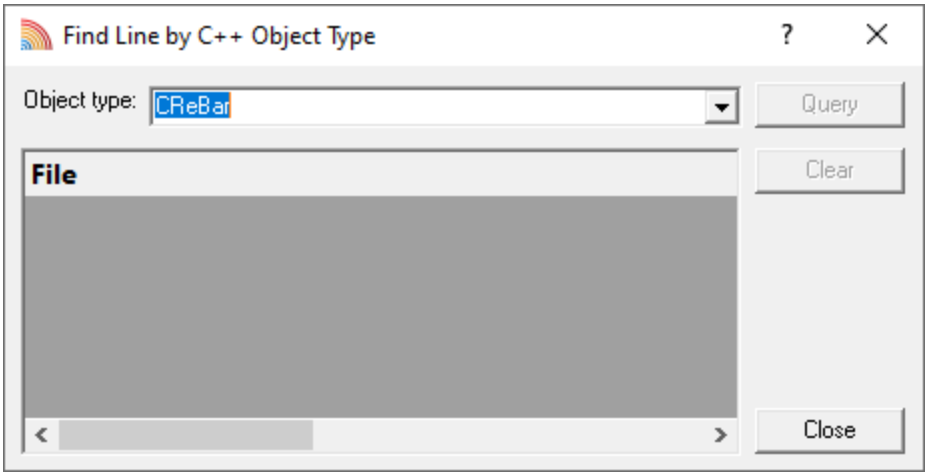
Finding lines by object types

Using the **Find Line by C++ Object Type** dialog below, you can search for lines belonging to a C++ object type in a similar way to finding addresses.

To show the dialog, choose the menu option below:

Query menu › choose **Query Object...** › displays the Find Line by C++ Object Type dialog

Or use the following icon on the Query Toolbar.



Search criteria

Just choose the object type for which you want to find related lines.

- **Object type** › choose the data-type of the objects you're interested in

  The class names that Coverage Validator uses are taken from the list of functions that are hooked by Coverage Validator.

Query results

- **Query** › performs the search

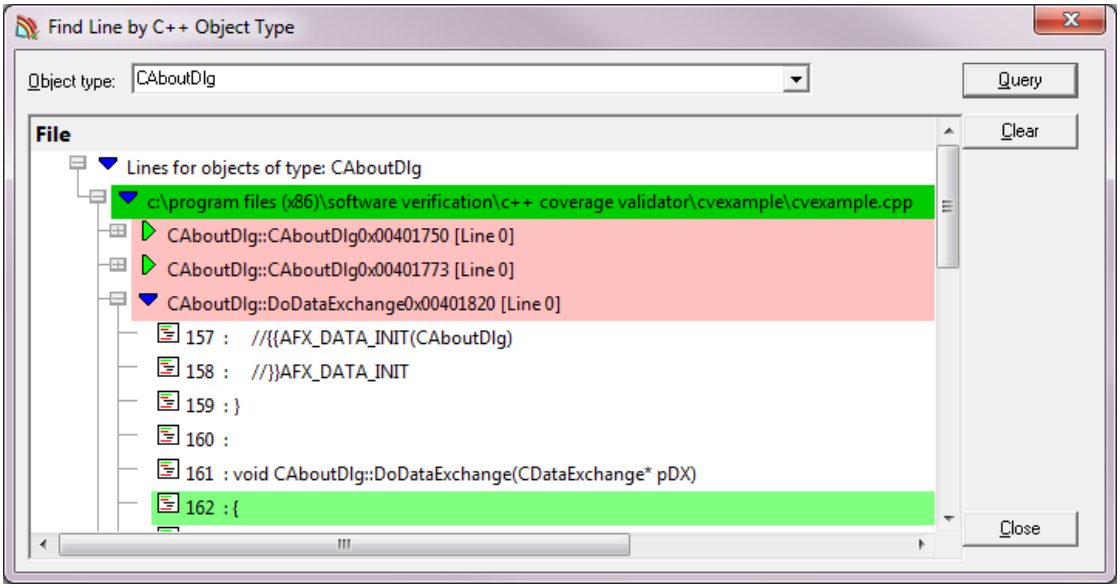
The search results are added to the File list in the dialog.

- **Clear** › clears all the results from the list

  The results are not automatically cleared with each search, allowing you to accumulate searches of multiple object types.

You can expand the search results, and double click the data items to edit source code in your preferred editor.

The results are displayed as a tree of files, functions and lines, as in the example below.



3.12.3 Finding functions

Finding lines in functions

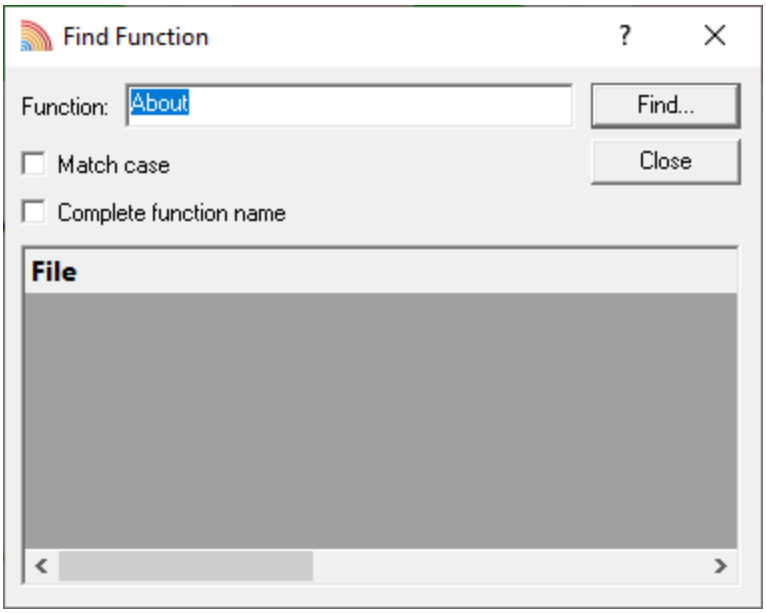
Using the **Find Function** dialog below, you can search for lines occurring in functions matching a function name of interest.

This feature is similar in behaviour to finding objects or addresses except here the results are cleared each time.

To show the Find Function dialog, choose the menu option below:

Query menu › choose **Find function...** › displays the Find Function dialog

Or use the following icon on the Query Toolbar.



Search criteria

Enter a function name, and optionally whether to match case or the complete name

- **Function** › enter full or partial function name
- **Match case** › tick to do a case-sensitive match
- **Complete function name** › tick to only match the whole name

  For C++ methods, complete names must be of the form classname::methodname.

Function match results

- **Find** › performs the search displaying results in the list

  Results *replace* any previous search, unlike querying addresses or objects where the results are *added* to the list.

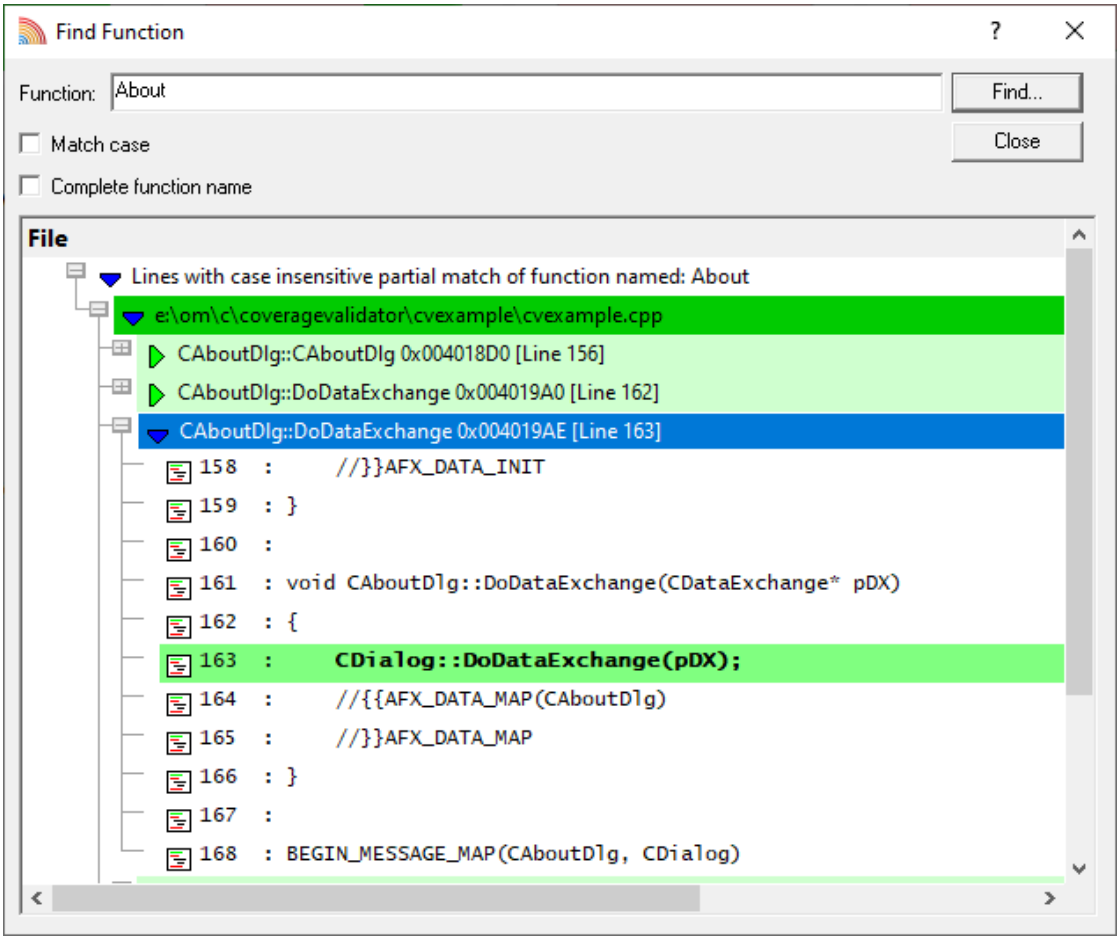
You can expand the search results, and double click the data items to edit source code in your preferred editor.

The picture above shows results of a search for 'About' in the example application.

The filename is shown in green, indicating that *some* parts of the file have been visited, but not *necessarily* in the functions matching the search.

The lines themselves are shown above in pink, indicating that they have not been visited.

Showing the about dialog in the example application and doing the search again would show the lines as green as below:



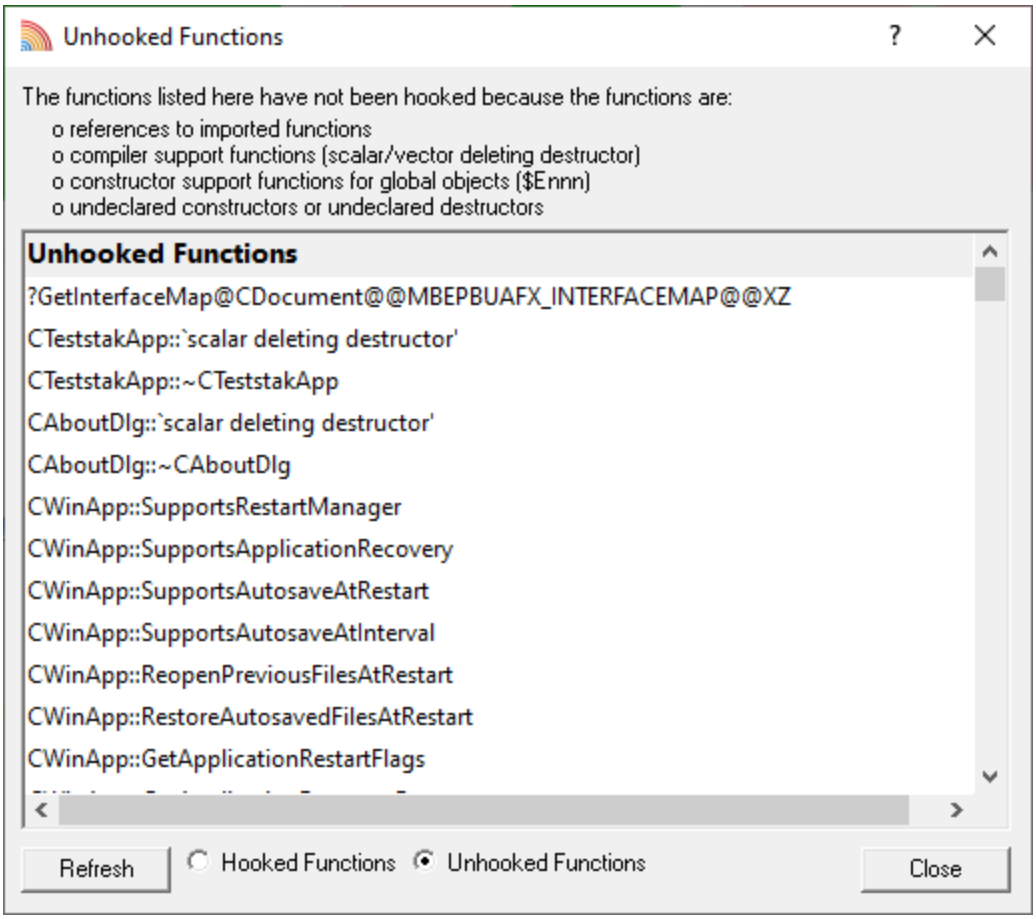
3.12.4 Find unhooked functions

Unhooked functions

Coverage Validator allows you to browse a list of functions that have not been hooked (or those that have).

Query menu › choose **Find unhooked functions...** › displays the **Unhooked Functions** dialog

The dialog initially displays the names of all functions that have *not* been hooked.



- **Refresh** › updates the list
- **Hooked Functions** › display functions that *have* been hooked
- **Unhooked Functions** › display functions that *have not* been hooked

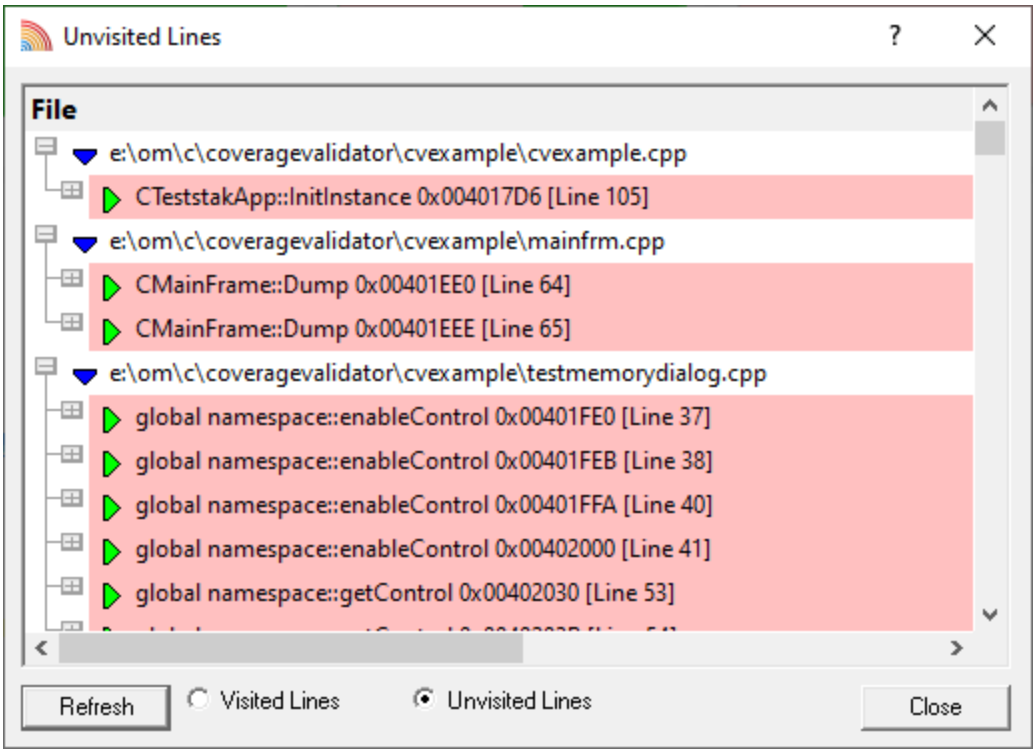
3.12.5 Find visited/unvisited lines

Unvisited and visited lines

You can browse all lines that have not been visited, or those that have.

Query menu › choose **Find unvisited lines...** › displays the Unvisited Lines dialog

The dialog displays a hierarchical view of the files, functions and lines.



In this colour scheme all lines will be shown in pink since they are all unvisited.

As with the other query dialogs, you can expand the search results, and double click the data items to edit source code in your preferred editor.

- **Refresh** › updates the list
- **Visited Lines** › displays lines that *have* been visited

  All lines would then be shown in green or whatever colour you set.
- **Unvisited Lines** › lines that *have not* been visited

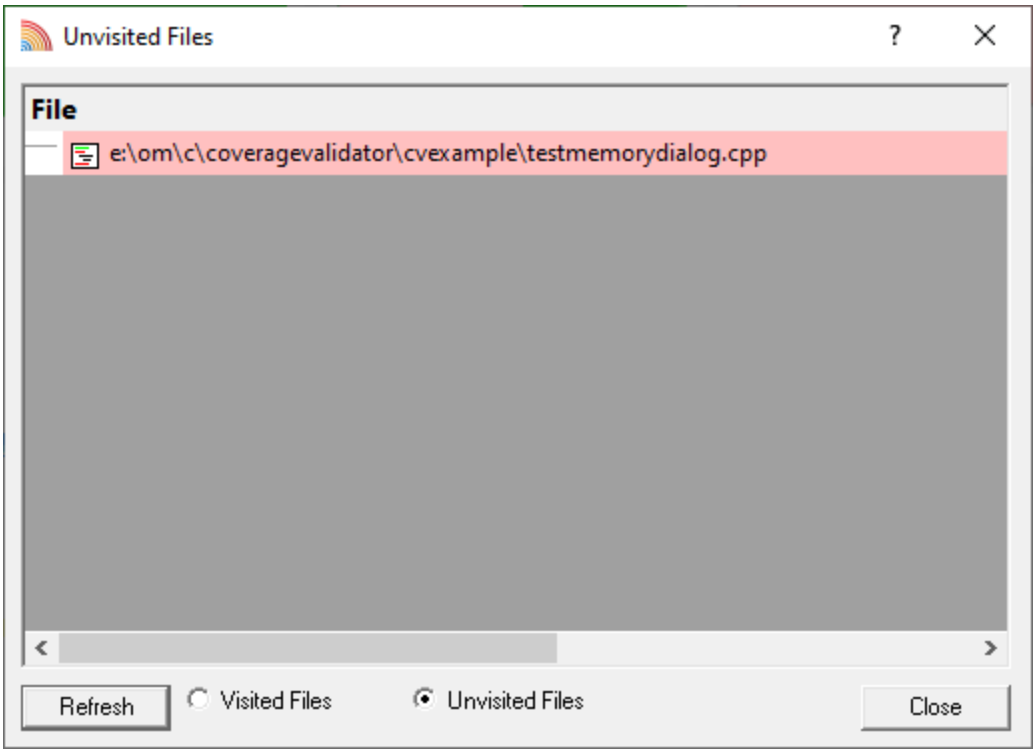
3.12.6 Find visited/unvisited files

Unvisited and visited files

Similar to unvisited lines, you can also browse all files that have not been visited, or those that have.

Query menu › choose **Find unvisited files...** › displays the **Unvisited Files** dialog

The dialog displays a list of files.

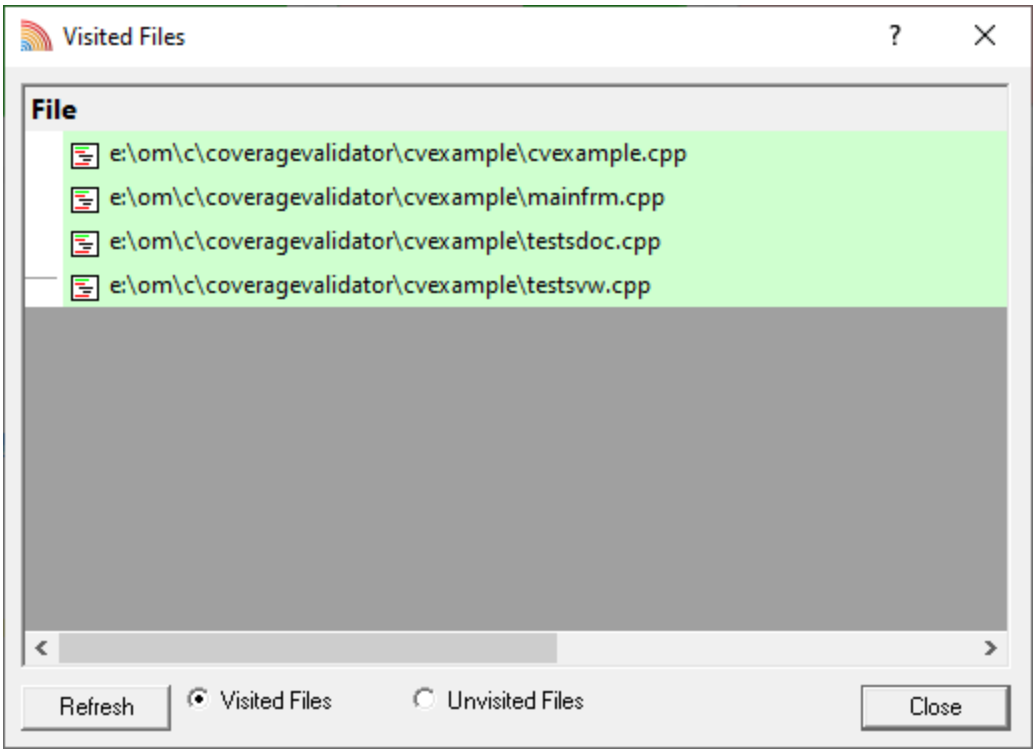


In this colour scheme all files will be shown in pink since they are all completely unvisited.

There are no hierarchical views to expand, and double clicking does not open the file in an editor.

- **Refresh** › updates the list
- **Unvisited Files** › files that *have not* been visited
- **Visited Files** › displays files that *have* been visited, even if only partially

  All files listed would then be shown in green or whatever colour you set:



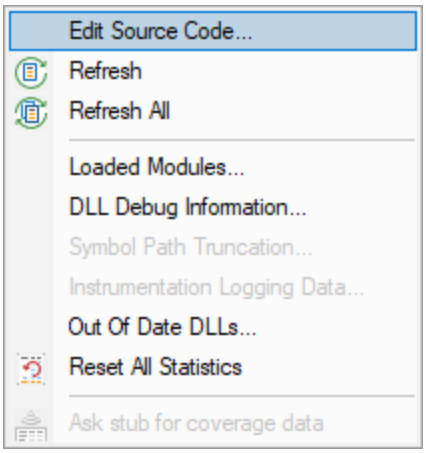
3.13 Tools

Tools

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
- A log of files, classes, functions, methods, or modules not instrumented, and reasons why not

Click on a menu item in the picture of the Tools Menu below to find out more:



3.13.1 Edit Source Code...

Source code editing

The editing settings let you set an editor of your choice to view or edit source code. Coverage Validator's built-in editor is one of those options.

The built-in editor can be started in several ways:

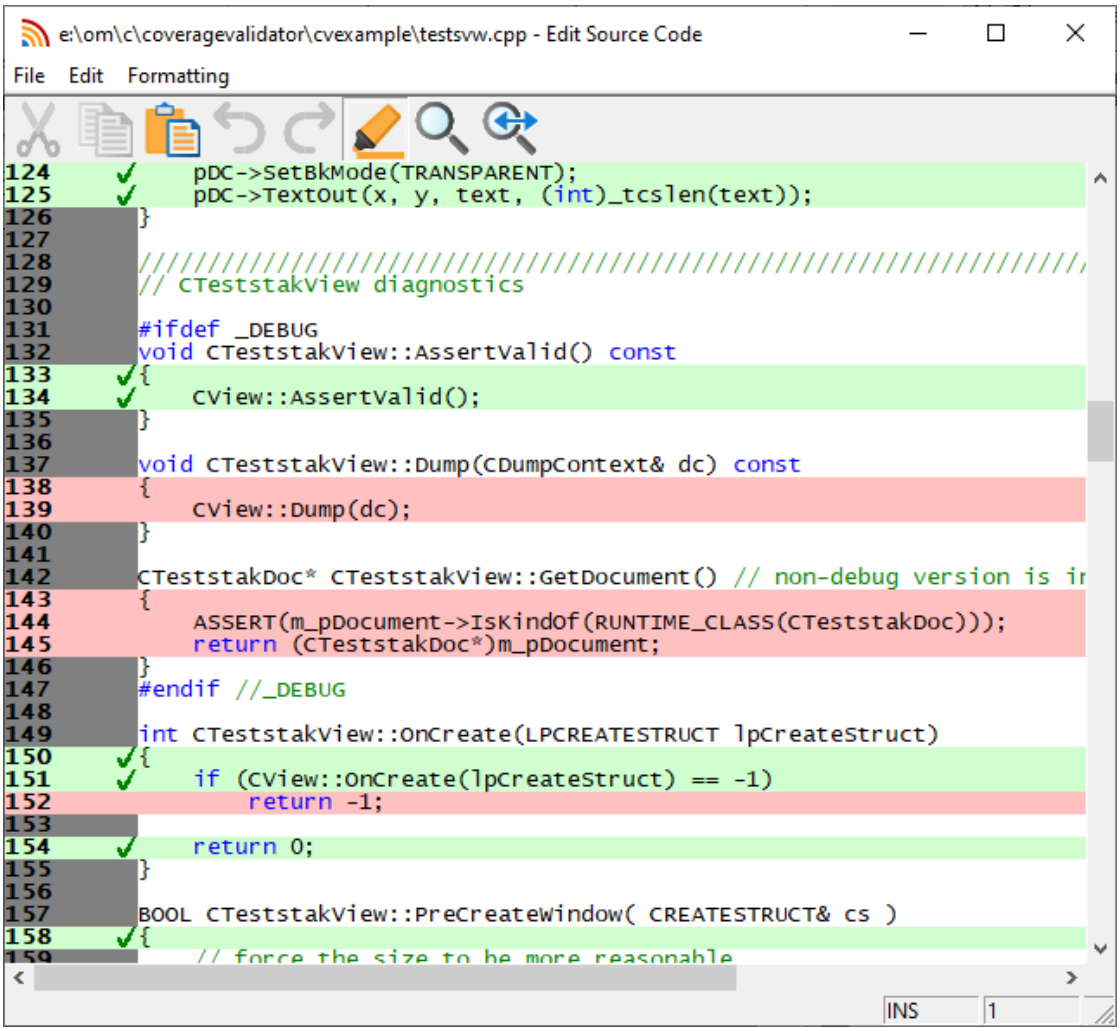
- Double click on a source code fragment (e.g. in the Query tools)
- **popup** menu > **Edit Source Code...**
- **Tools** menu > **Edit Source Code...**

Using the built-in editor

The built-in editor supports the basic operations expected for editing source code:

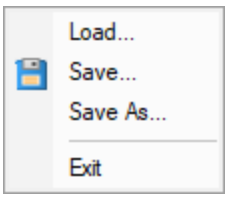
The highlighting is identical to that in the source code views of the main tabs:

- The lines in green (for this colour scheme) have been visited
- Lines that have not been *visited* are displayed in pink
- Lines with a green tick next to them indicate that they have been successfully hooked
- Lines that could not be *hooked* have a red cross against them
- An arrow indicates the source code line of interest when the source code editor was displayed



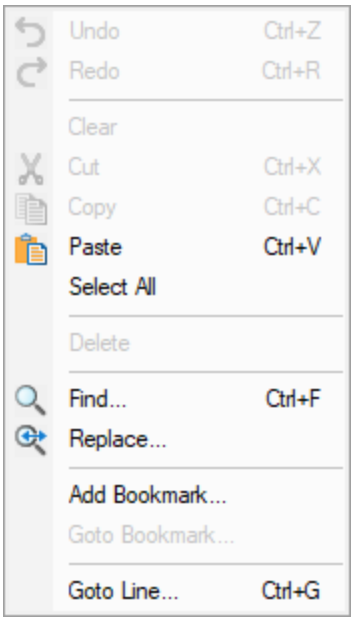
File menu

The file options need no explanation:



Edit menu

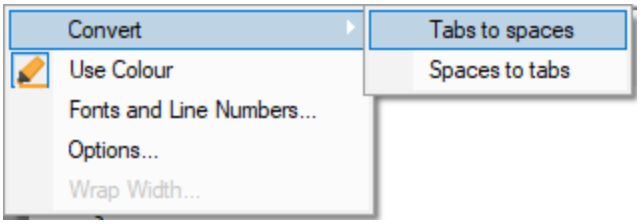
All the following edit options should also be familiar:



Undo/Redo is unlimited by default, but this can be changed in the options below.

Formatting menu

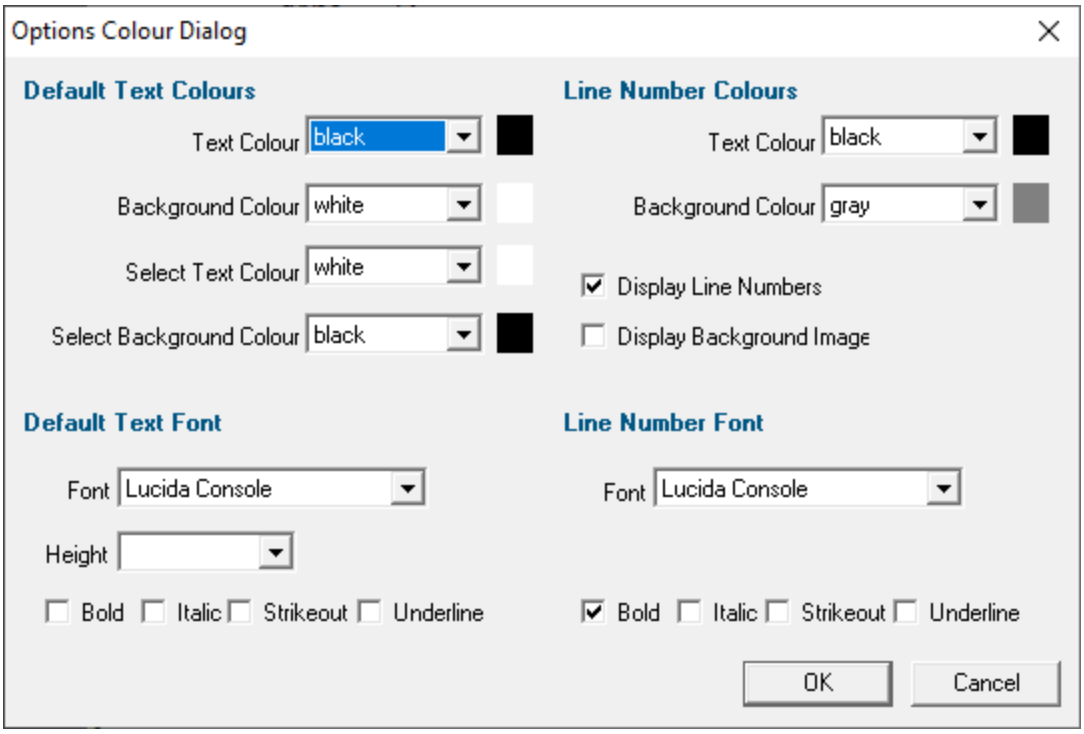
The formatting menu has general display and editing options



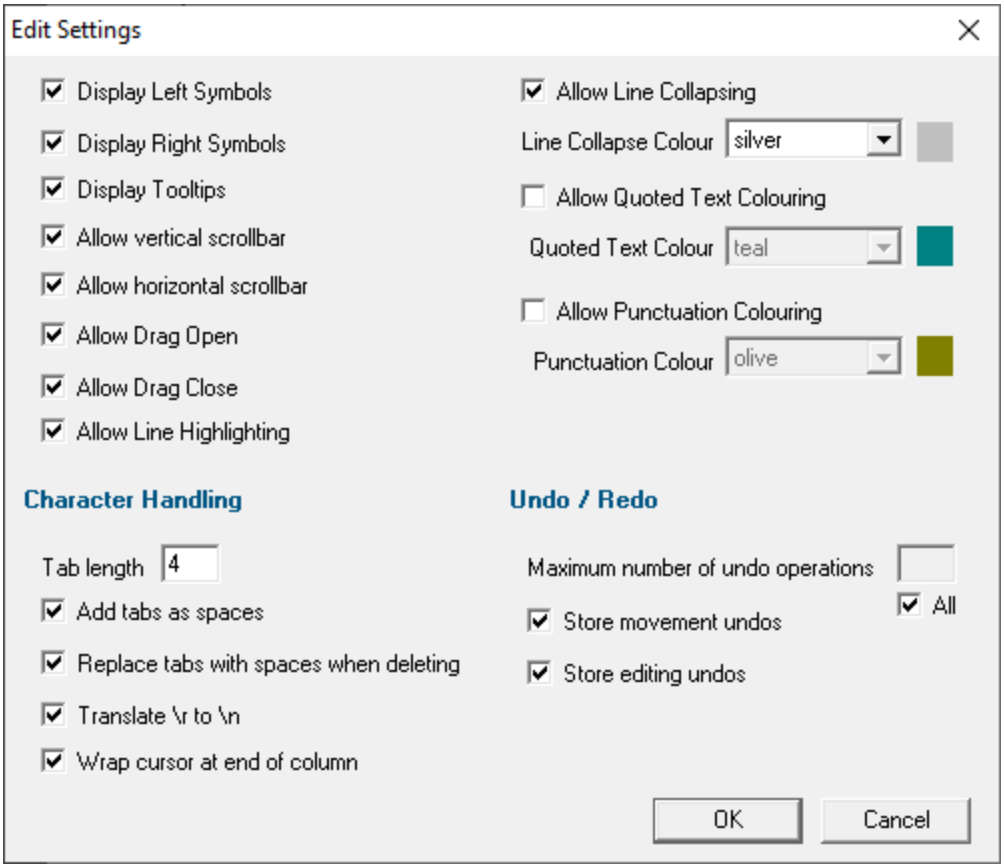
- **Convert** › **Tabs to spaces** › turns all tabs into spaces
- **Convert** › **Spaces to tabs** › turns all spaces into spaces

- **Use Colour** › toggles the colour coded display

- **Fonts and Line Numbers...** › change text colours, fonts and line numbers



- **Options...** › set tab length and other options



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

Status bar

The status bar shows help text at the bottom as you hover over menu and toolbar options.

To the right of the status bar are insert mode, column number and line number.

Line collapsing

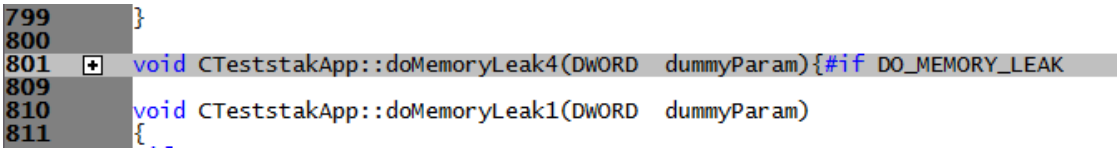
You can temporarily collapse sections of code as follows:

- **Left click** in the margin to start the section › **Drag** to define the length › **Release** to set the end of the section

  Click anywhere on the resulting indicator to collapse, and on the + to expand a section.

Expanded:

Collapsed:



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 the view at an interval of your choice.

Sometimes you need to refresh the data when *you* want to, especially while inspecting the data.

Most views have a local refresh button, which updates the data.

The same function is found in the Tools menu, as well as an option to update all views at once:

Tools menu › **Refresh** › refresh the data displayed only on the current tabbed view

Tools menu › **Refresh All** › refresh the data on *all* the tabbed views

Or use the **Refresh** and **Refresh All** icons on the Tools toolbar.

3.13.3 Loaded Modules

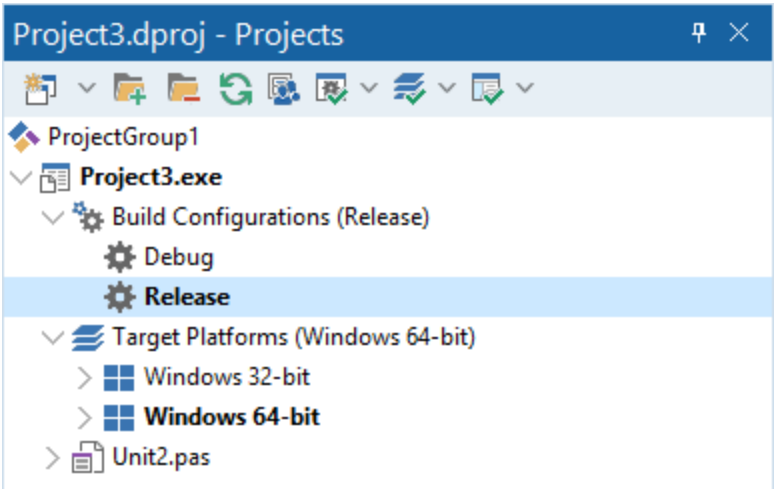
Viewing the loaded modules

You can view a list of the modules which are loaded by your target application.

Tools menu › **Loaded Modules...** › shows the **Loaded Modules** dialog

The dialog shows:

- the **Address** space occupied by the module (DLL or EXE)
- the type of **Code** in the module (native, managed, mixed mode or resources only
- the type of **Build** - is the code debug or release?
- the **Path** the module was loaded from



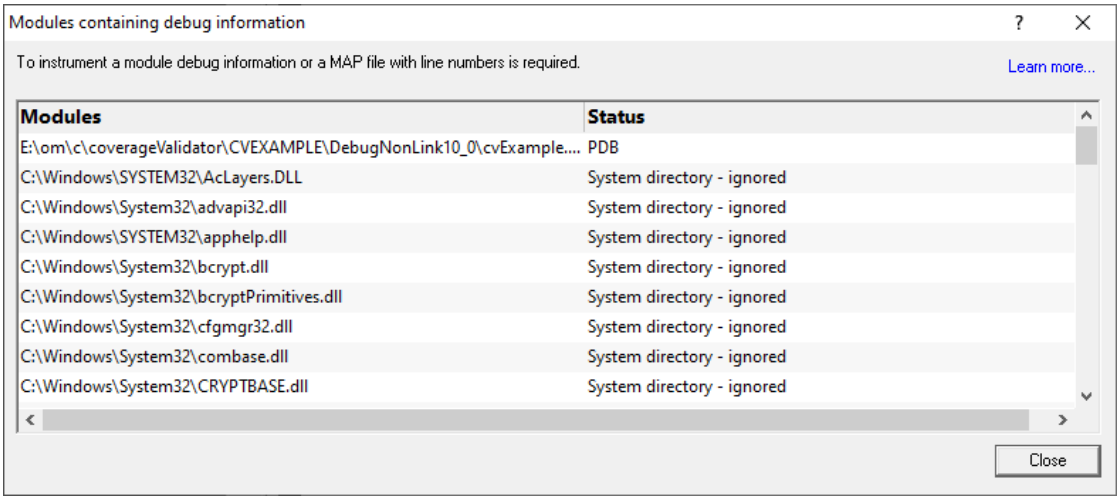
3.13.4 DLL Debug Information

Viewing the DLL debug information

If you are having problems collecting coverage data for a particular EXE/DLL the problem may be that the debug information that is required to perform the instrumentation of the software cannot be found.

You can view a list of the debug information status of modules loaded by your target application.

Tools menu › **DLL Debug Information...** › shows the DLL Debug Information dialog below



The dialog shows:

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

- if any symbol server is not reachable (offline or doesn't exist) a message will be shown in red at the bottom of the dialog. You can edit the symbol server definitions here.

Debug status

There are various reasons why a module may not have its debug information read.

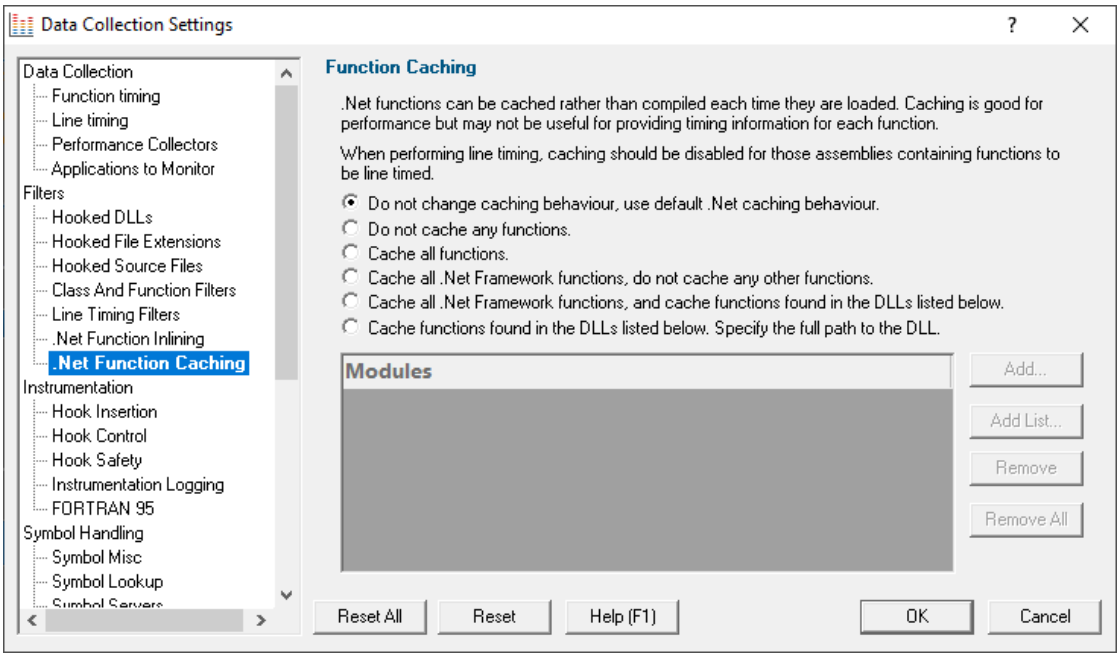
The dialog shows a comment or reason in the status column. Examples might be:

- PDB or MAP if the debug information *was* found and used
- Debug information not present
- A reason for being ignored
- Module is a part of the C Run-time Library (CRT) or Standard Template Library (STL)
- Location is a system directory
- Ignored due to Hooked DLLs advanced settings
- File is a Software Verify own module
- Module has been specified as a 3rd party
- No executable code is contained
- The module only has GUI resources

More information about PDB and MAP files

Clicking on the **Learn more...** link at the top right of the dialog shows some more details with additional links to topics in this help.


Click the links below to see read more in our frequently asked questions.

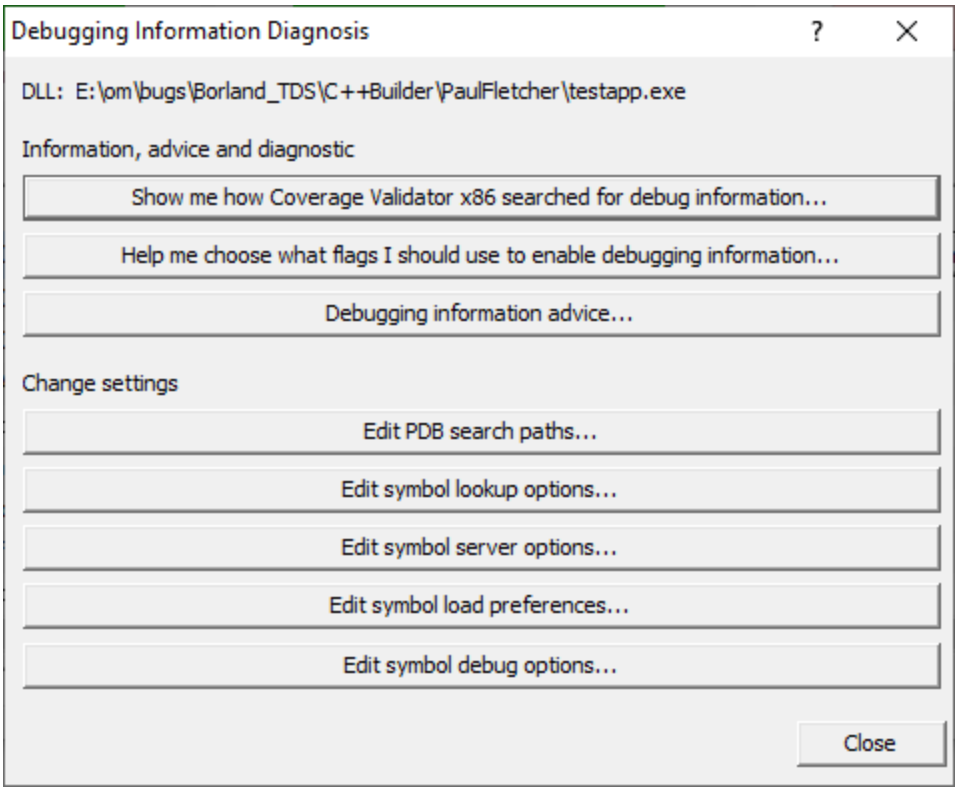


Finding out more using the Debugging Information Diagnosis Dialog

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:

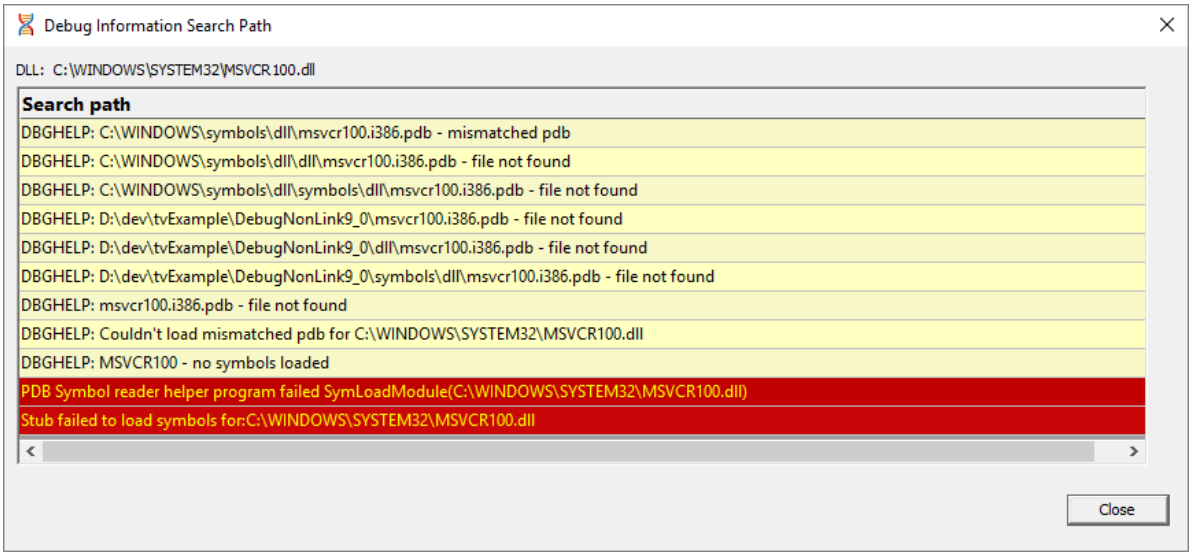
- Information, advice, and diagnostic help
- Quick links to change settings



The information options include:

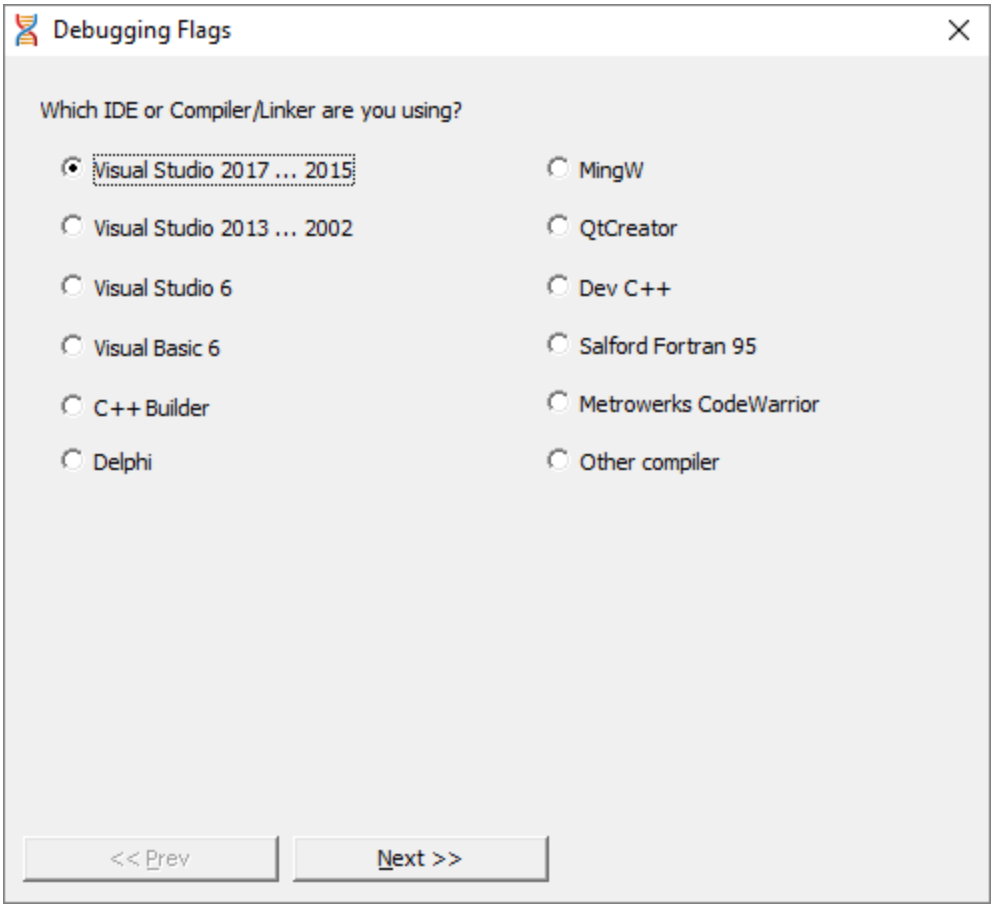
- **Show me how debug information was searched for...** › shows the Debug Information Search Path dialog

This information is extracted from the Diagnostic tab and shows only the relevant information for the module selected in the DLL Debug Information dialog.

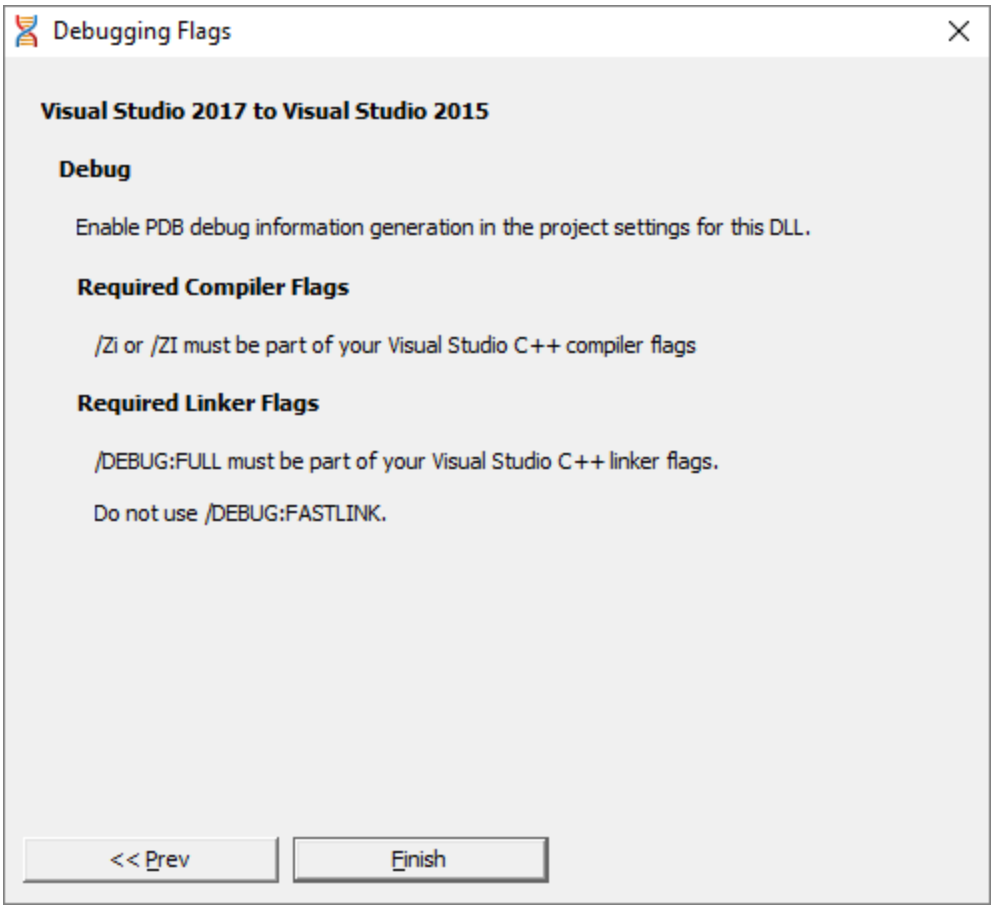


- **Help me choose what flags...** › shows the Debugging Flags wizard

  Use the wizard to first select the compiler or linker you're using



Next >> › Provides the relevant debug compiler and linker flags. An example for Visual Studio 2017 to 2015 is below:



- **Debugging information advice...** › shows the Symbols and Debugging Information dialog above.

The options for changing settings include quick links to the following pages from the Global Settings Dialog

- **Edit PDB search paths...** › shows the File Location settings page for PDB files.
- **Edit symbol lookup options...** › shows the Symbol Lookup settings page
- **Edit symbol server options...** › shows the Symbol Servers settings page
- **Edit symbol load preferences...** › shows the Symbol Load Preferences settings page
- **Edit symbol debug options...** › shows the Symbol Misc settings page

3.13.5 Instrumentation Logging Data

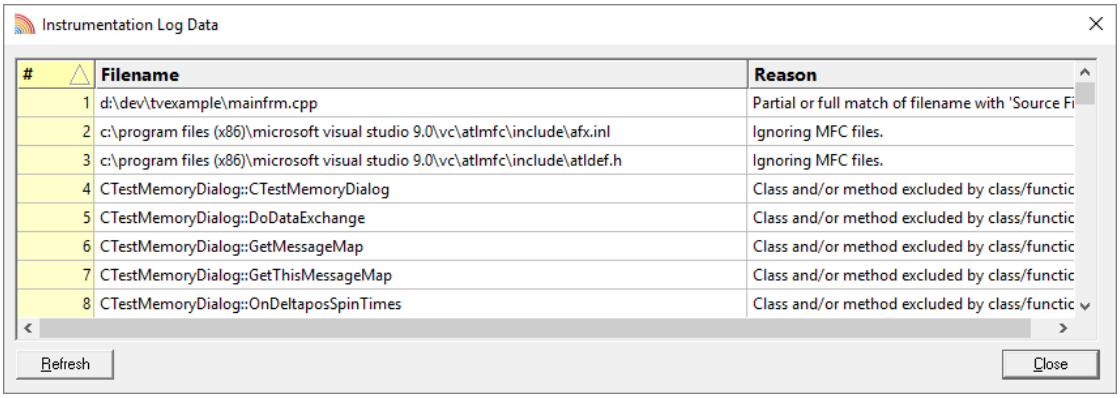
Instrumentation Log Data

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 class, file, DLL or or other filters in the Global Settings dialog.

To log details of why dlls, source files, classes, methods and functions are not instrumented, first switch on the instrumentation logging settings.

Once enabled, and a session has started, you can view a list of the files that have not been instrumented via the Tools menu.

Tools menu › **Instrumentation Logging Data...** › shows the Instrumentation Log Data dialog



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
- Class or function excluded by class and function filters
- File extension excluded by hook source file type filters
- Files excluded by source files filters

3.13.6 Out Of Date DLLs

It may happen that if you forget to build a DLL, or if a build error occurs that you perform code coverage on DLLs that are not built with the most recent version of your source code.

We refer to these DLLs are out of date DLLs because they are out of date compared to the source code that is compiled to create the DLLs.

Coverage Validator can detect this, and warn you about it.

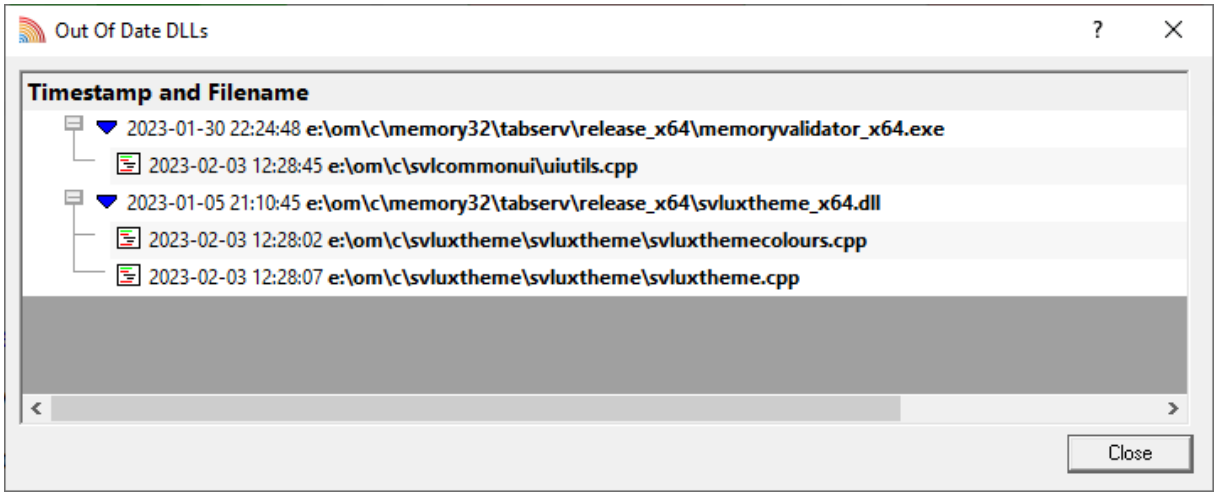
Summary tab

When out of date DLLs are found a warning is displayed on the summary tab, in the lower section of the display.

Out Of Date DLLs Dialog

The **View...** link will display the Out Of Date DLLs dialog.

There is also an option to display the Out Of Date DLLs dialog on the Tools menu.



The Out of date DLLs dialog shows the DLLs that are out of date, and the source files for each DLLs. The dates of both the DLLs and the source files are displayed.

The above image shows 2 DLLs that are out of date, with 2 files in each DLL being more recently edited than the build timestamp for the respective DLL.

Data Displays

The various data displays in Coverage Validator change the text colour to the Out Of Date colour.

File	% Visit...	Num Lines	Num Visited	Visit Count	DLL
Totals	**42.97%**	**128**	**55**	**55**	
e:\om\c\coveragevalidator\examples\nativeexample\nativeexample.cpp	65.22%	23	15	15	E:\om\c\coverageValidator\examples\native
e:\om\c\coveragevalidator\examples\nativeexample\mainfrm.cpp	100.00%	9	9	9	E:\om\c\coverageValidator\examples\native
e:\om\c\coveragevalidator\examples\nativeexample\testsdoc.cpp	83.33%	6	5	5	E:\om\c\coverageValidator\examples\native
e:\om\c\coveragevalidator\examples\nativeexample\testsvw.cpp	28.89%	90	26	26	E:\om\c\coverageValidator\examples\native

The source code displays in Coverage Validator change the text colour to the Out Of Date colour, and display a prominent warning in red text.

Sessions that are exported as HTML also change the text colour to the Out Of Date colour, and display a prominent warning in red text.

3.13.7 Reset All Statistics

Resetting statistics

It's not uncommon to want to clear all the coverage data gathered during the session so far, and then continue calculating coverage without restarting the session.

Resetting the statistics does exactly that.

Tools menu › **Reset All Statistics** › reset all the collected statistics

Or use the **Reset All Statistics** icons on the Session toolbar.

The way this works is that a message gets sent to the stub monitoring the target program.

All threads in the target program get suspended whilst the statistics are reset.

Once reset, all the threads are resumed and new values for the statistics are calculated.

3.13.8 Ask stub for coverage data

Real-time updates with NT services

When working with NT services, Coverage Validator can't access the data in the stub due to security differences between itself (an application) and your service.

To overcome this problem, the stub sends all the coverage data to the user interface when your application stops (e.g. you stop it from the service manager or the control panel, etc).

This allows the user interface to display the coverage data *after* your service has completed, but it's not unreasonable that you might want to see coverage data *while the service is running*.

Coverage Validator provides an on-demand utility to retrieve updated coverage information when you want it.

Tools menu › **Ask stub for coverage data** › updates the coverage statistics from the stub

or use the **Ask stub for coverage data** icon on the Session toolbar.

There will be a short pause whilst coverage data is sent from the stub to the user interface.

3.14 Software Updates

This topic covers the three 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

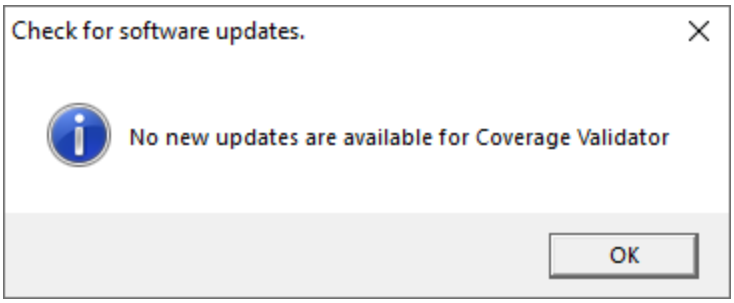
If you've been notified of a new software release to Coverage Validator or just want to see if there's a new version, this feature makes it easy to update.

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

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

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

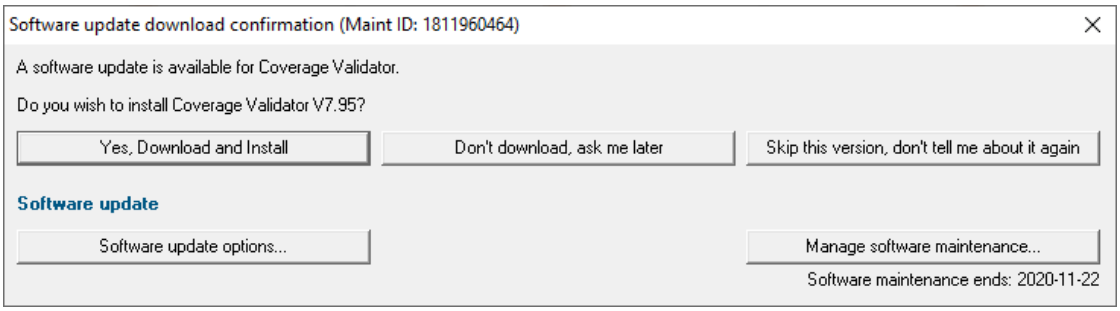
If no updates are available, you'll just see this message:



Note that evaluation versions cannot be updated.

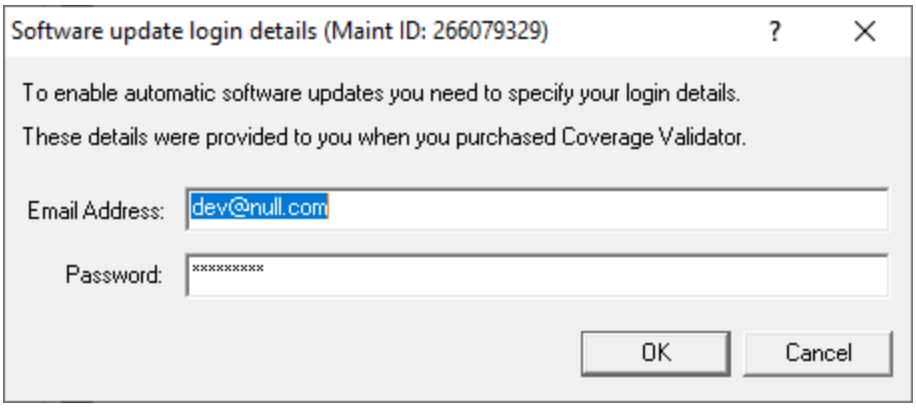
Software Update dialog

If a software update is available for Coverage Validator you'll see the software update dialog, unless your maintenance has expired.

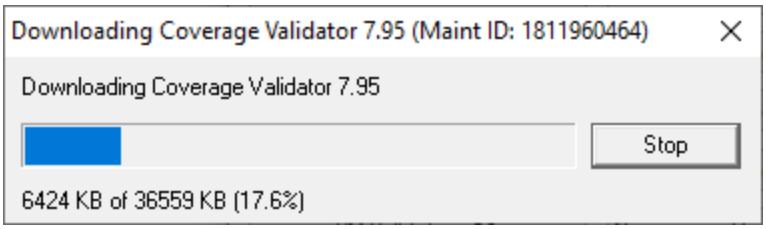


- **Download and install** › prompts you for log-in details if not known, and then downloads the update, showing progress

You may be asked for your log-in credentials, which you'll have received when you purchased Coverage Validator.



Once logged in, the download will start:



Once the update has downloaded, Coverage 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 Coverage 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 Coverage 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 Coverage Validator as follows:

- Ensure any open sessions are completed
- Ensure any target programs started by Coverage Validator are closed
- Ensure Coverage Validator and its help manual is also closed
- Ensure any error dialogs from the previous installation are closed

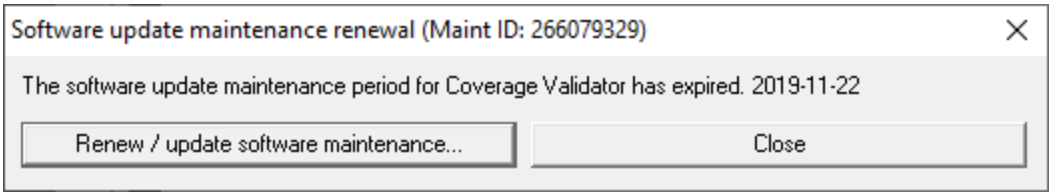
Have your license details handy as you may need to copy information into the license dialog.

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 Coverage 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

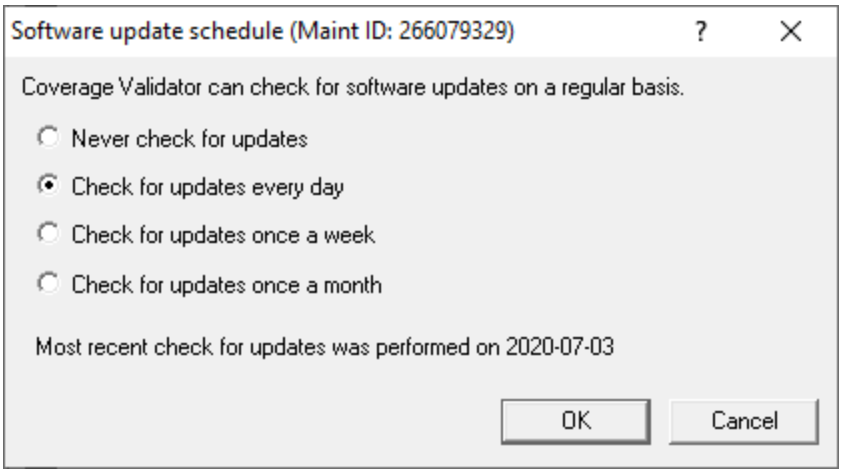
Coverage Validator can automatically check to see if a new version of Coverage Validator is available for downloading.

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

The update options are:

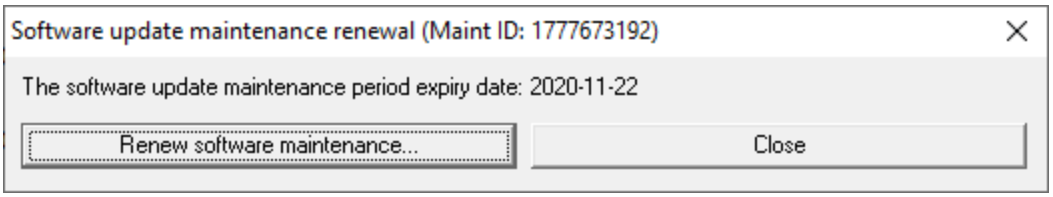
- never check for updates
- check daily (the default)
- check weekly
- check monthly

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



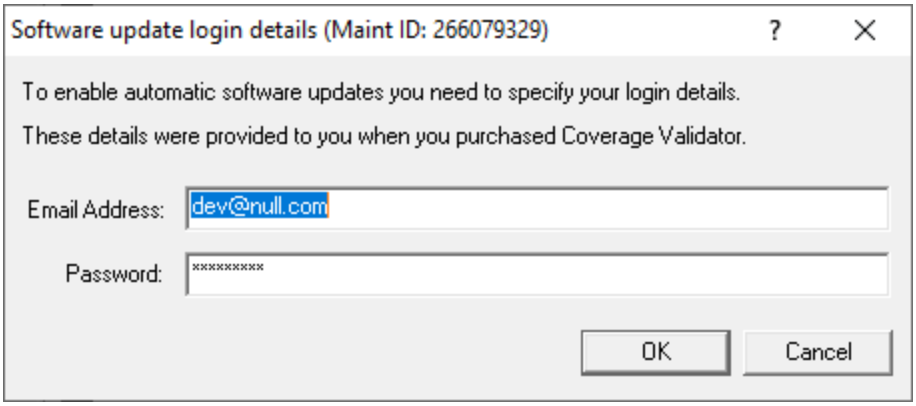
Your maintenance expiry date is shown. If you don't need to do anything just **Close** the dialog.

- **Renew software maintenance** › Opens your browser, logging you in to our website from which you can purchase maintenance

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

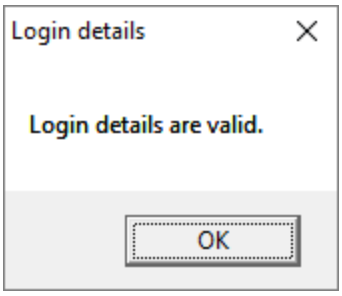


The text will be shown in red if the email address looks incorrectly formatted.

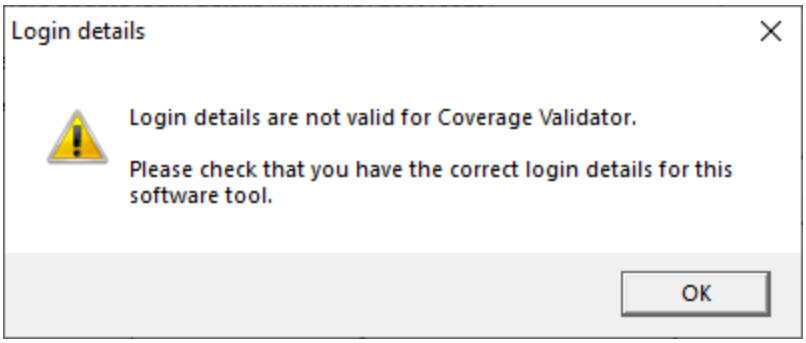
Testing the login details checks they're valid:

- **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.



You should have received the correct credentials when you purchased Coverage Validator, or with any software update emails.

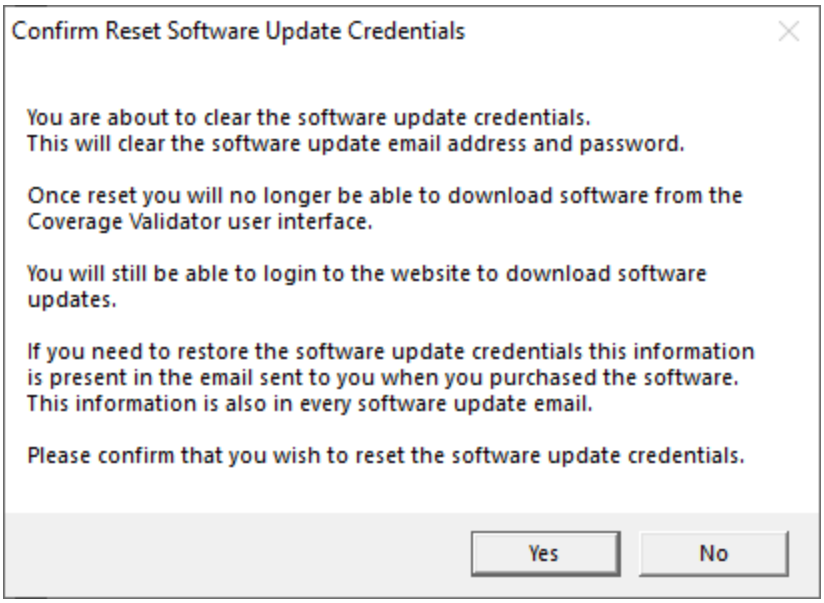
If you experience problems, check with your system administrator or contact Software Verify.

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 Coverage Validator, or with any software update emails.

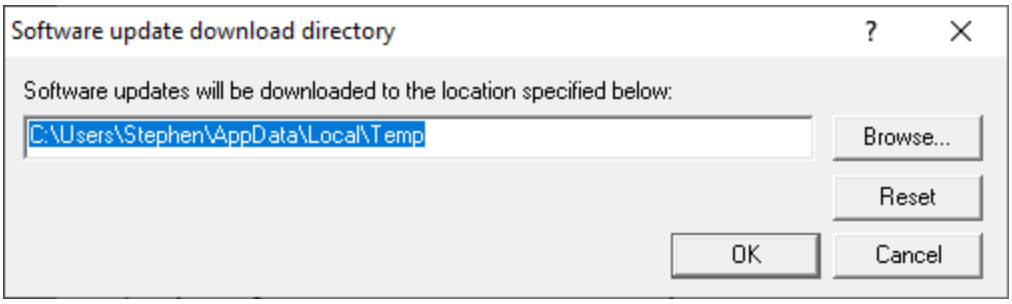


Software update directory

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 Sessions: Load, Save, Export, Close

Working with sessions

Sessions with Coverage Validator can be saved to and loaded from a file so that you can:

- share the session with a colleague
- examine the session at a later date
- compare the session with another session
- create baseline sessions for use in regression tests

Sessions can be even exported in HTML and XML formats.

You can have multiple sessions open at once, necessary for comparing or merging loaded sessions.

Closing a session

When you've finished working with a session, it can be closed.

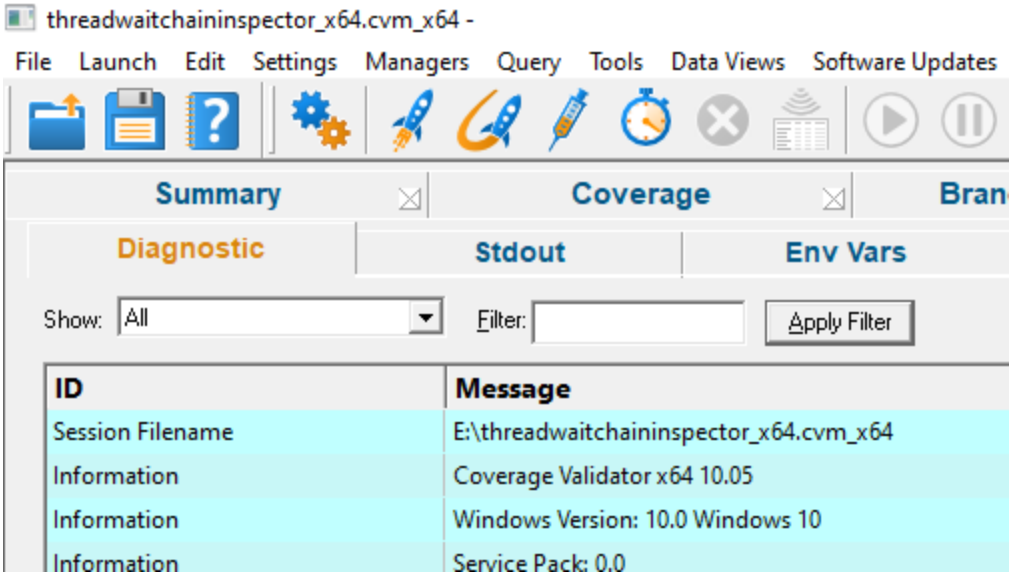
File menu › **Close Session...** › closes the session, clearing the displays

Closing a session may happen automatically if you start a new session and the session count limit is 1.

If the maximum session count allows, closed sessions still appear in the Session Manager, where they can be reopened or deleted.

Session Filename

The session filename is displayed as the first line of the diagnostic data on the Diagnostic tab.



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 (*.cvm)

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 1 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 (*.cvm), prompting for a file name if necessary

File menu › **Save As...** › saves the session to a new file

Or click on the Save Session icon on the standard toolbar.

Or use the shortcut:

Ctrl + S Save session

Unlike exports, there are no options here, as all 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 an **HTML** or **XML** Report › shows the Export Session dialog below

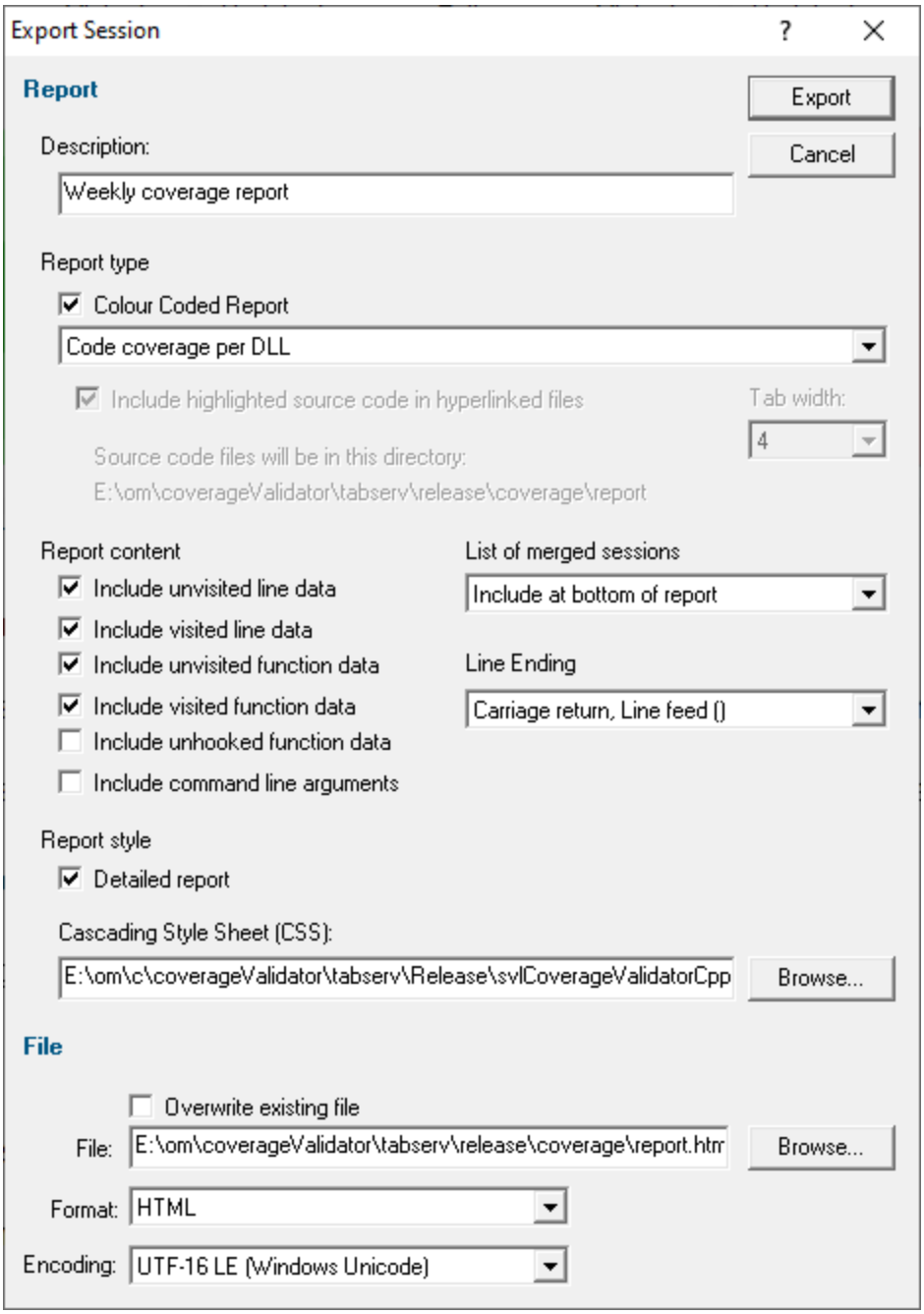
Exporting is not saving

You can't import session data.

Use save and load if you want to save session data for loading back into Coverage Validator at a later date.

The HTML and XML export session dialog

For regular HTML or XML export, the same dialog is used. The Cobertura XML export dialog has fewer options.



Report description

The optional description is included at the top of the exported data in the description field.

- **Description** › enter a meaningful description or just leave blank

  If no text is entered the description is omitted from the export.

Report Type

- **Colour Coded Report** › generate a colour coded report

  Only HTML reports can be colour coded:

  Colours used will be those set for use in the various displays.

The exported report can be generated on a per file or class basis, or as a summary of coverage.

- **Per file** › generate a report that lists data by filename
- **Per class** › list data by C++ class

  All functions that are *not* members of a class will be listed together in the "global namespace" class.

- **Per DLL** › list data by module name (EXE, DLL, BPL, etc)
- **Summary report** › generate a report comprising only of images of the graphs on the summary tab

- **Summary report, +coverage...** › includes the summary images, code coverage and branch coverage (also unit test coverage if present)

  Within the report, you can optionally link source code showing highlighted coverage lines and visit counts

  - **Include highlighted source code...** › check to include marked up source code files in a sub directory

Report content

When exporting data by file or class, you can optionally include data for visited or unvisited lines and functions, as well as unhooked functions.

Tick any or all of the following to **Include**:

- **unvisited lines** (included by default)
- **visited lines**

- **unvisited functions** (included by default)
- **visited functions**

- **unhooked functions**

List of merged sessions

If it's important to know what merged sessions were included in the statistics, you can optionally list the session information at the top or bottom of the report:

- **Do not include** › prevent the list of merged sessions being included
- **Include at top** › include the list of merged sessions at the *top* of the report
- **Include at bottom** › include merged sessions at the *bottom* of the report

Report style

Some areas of the report can be detailed or terse.

- **Detailed Report** › check box to include more detail (the default)

HTML reports are usually colour coded according to the current colour scheme.

- **Colour Coded Report** › check box to include more detail (the default)

  A colour coded view would be essential if including both visited *and* unvisited lines or functions.

  If you want a custom style (e.g. not using tables), export a detailed XML report and process that to generate the HTML report.

Cascading Style Sheet (CSS)

Cascading style sheets are used to control the formatting of the exported HTML reports.

By default the style sheet **svlCoverageValidatorCpp.css** is used from the Coverage Validator install directory.

You can specify your own style sheet in this field:

- **Cascading Style Sheet (CSS)** › enter the full path to the CSS file or **Browse** to the file

At the time of export a *copy* of the CSS file is placed in the same directory as the report.

Line ending

Depending on how you want to use the reports, you may prefer a certain line ending character

- **Carriage Return, Line Feed** › `\r\n` (e.g. Windows)
- **Line Feed** › `\n` (e.g. Linux)
- **Carriage Return** › `\r` (e.g. Macintosh)

File section

- **Overwrite existing file** › check if you *don't* want to be warned about overwrites
- **File** › type the filename or **Browse** to a location
- **Format** › set whether exporting HTML or XML

  Defaults to the original menu option selected, but included here to more easily export one format and then the other.
- **Encoding** › set whether UTF-16 LE, UTF-8 or ASCII encoding. By default the exported file is saved in the Windows Unicode format UTF-16 little endian. You can also save in UTF-8 and ASCII. ASCII has no byte order mark at the start of the file.

Ready to export?

Use the export button at top right when you're ready to go

- **Export** › export the session data

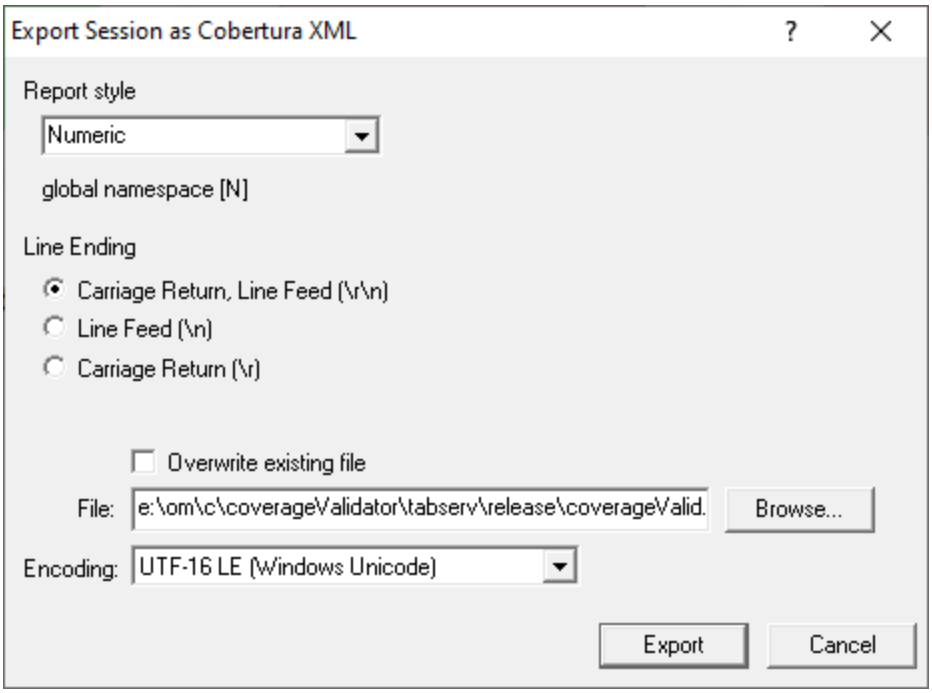
The Cobertura XML export session dialog

Cobertura is a free Java tool that calculates the percentage of code accessed by tests.

Cobertura's coverage output can be used in other systems such as the Jenkins continuous integration (CI) server.

Coverage Validator can output its own coverage results in the same XML format as Cobertura so that they can be used in a similar way.

For Cobertura XML export the options are much reduced:



The line ending and output file details are as for regular XML output, but the report style options differ.

Report style

The Cobertura XML report lists data by class, which is a problem for functions that aren't in a class, but in the global namespace.

You to choose how to work around this:

- **Numeric** › The class name is listed as `global namespace [N]`

  `N` is a unique numeric identifier for each file containing unclassed functions

- **Filename** › The class name is listed as `global namespace [filename]`

  `filename` is the complete path and filename of the file for the unclassed function

- **All** › The class name is listed as `global namespace`

3.15.2.1 XML Export Tags

This section describes the XML tags used to export session data from Coverage Validator.

Application and program details

An exported XML file starts with a few details about Coverage Validator and the target program:

```
<XML>
    <VALIDATORINFO>Coverage Validator information online</VALIDATORINFO>
    <VALIDATOR>Coverage 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>
    <INFO>Information about the report type</INFO>
```

Code fragments

Code fragments can be included as child tags of several of the tags used for the various types of exported XML data.

Exporting by class will just list individual **codeFragment** tags while exporting by file uses the **codeFragments** tags.

```
    <codeFragments>

        <codeFragment>...</codeFragment>
        ...

    </codeFragments>
```

For example:

```
    <codeFragments>

        <codeFragment
        function="CAboutDlg::CAboutDlg"
        visitCount="0"
        numLines="2"
        numLinesVisited="0"
        firstLine="156"
        lastLine="159">

        { }

        </codeFragment>

        ...

    </codeFragments>
```

Source files

When exporting by file, the **SOURCEFILE** tags details coverage data for a given source code file and will contain all the code fragments

For example

```
<SOURCEFILE
file="c:\program files (x86)\software verify\c++ coverage validator\examples\nativeexample\nat
dll="C:\Program Files (x86)\Software Verify\Coverage Validator\examples\nativeExample\DebugNon
numLines="32"
numLinesVisited="17"
numLinesUnhooked="0"
numFunctions="10"
numFunctionsVisited="4">

   ...

</SOURCEFILE>
```

Classes

When exporting by class, the **CLASS** tags details coverage data for a particular class and will contain all the code fragment children

For example

```
<CLASS name="CAboutDlg">
```

File names

Class information identifies a file using the **FILENAME** tag:

```
<FILENAME file="c:\program files (x86)\software verification\c++ coverage validator\examples\n
```

3.16 Starting your target program

Starting options

There are four ways to start a target program and have Coverage Validator collect data from it.

- Launch your program in a specified directory, with as many command line arguments as you want
- Inject Coverage 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 Coverage Validator to be started whenever the program is started

There are seven ways to start a target program and have Coverage Validator collect data from it.

- Launch your program in a specified directory, with as many command line arguments as you want
- Inject Coverage Validator into an already running program
- Wait until a specific program starts to run before attaching to it - e.g. for programs started as an OLE server
- Monitor a service
- Monitor IIS and ISAPI
- Use the Native API to start Coverage Validator from code that you control
- Start Coverage Validator from the command line, allowing you to automate your use of Coverage Validator

Modules without PDB files and without MAP files

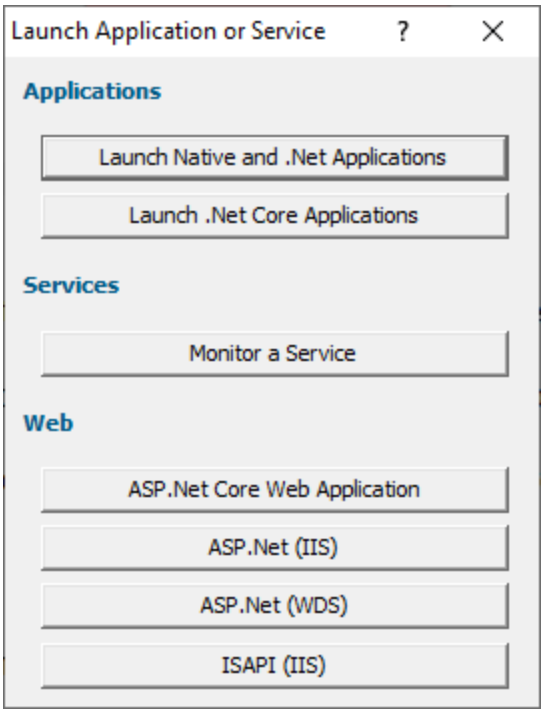
For your application to be processed for coverage data, each module to be monitored must have a PDB file with debug data, or a MAP file with line number data.

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 Launch Chooser

The launch chooser is displayed when you click on the rocket icon on the toolbar.

There are multiple application types and services that you may wish to use. The launch chooser provides the mechanism for making that choice.



Each button will display the launch dialog associated with the instruction displayed on the button.

Applications

- Launch Native and .Net Applications
- Launch .Net Core Applications

Services

- Monitor a Service

Web

- ASP.Net Core Web Application
- ASP.Net with IIS
- ASP.Net with Web Development Server
- ISAPI with IIS

You can repeat the choice made using the launch chooser by using Ctrl + F4

3.16.2 Launching the program (native and .Net)

Launching the application

Having Coverage Validator launch your program is the most common way to start up

When you're ready to start running a target program:

File menu › **Start Application...** › Shows the launch program wizard or dialog below

Or click on the launch icon on the session toolbar.

Or use the shortcut:

F4 Start application

➡ You can easily re-launch the most recently run program.

User interface mode

There are two interface modes used while starting a program

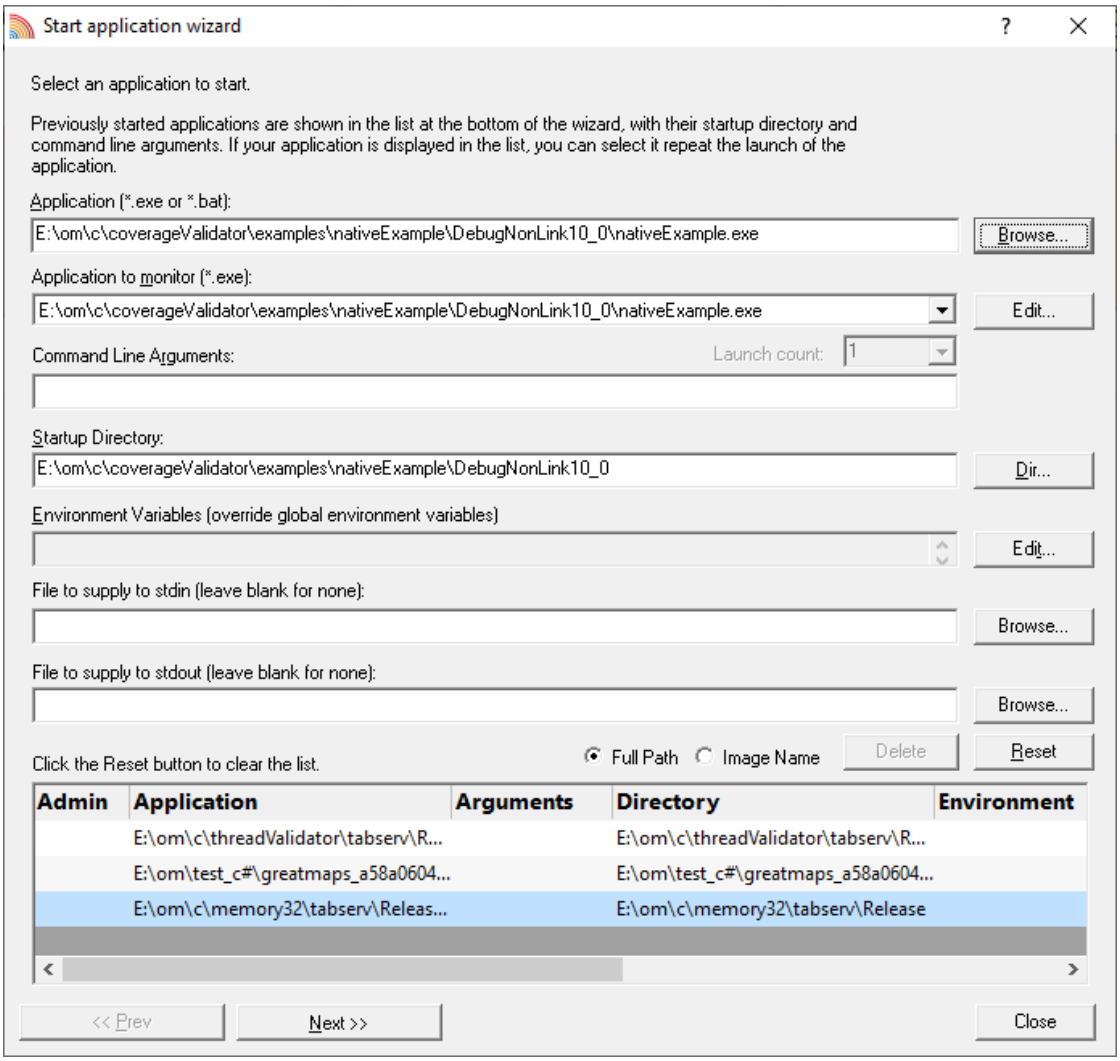
- **Wizard** mode guides you through the tasks in a linear fashion
- **Dialog** mode has all options contained in a single dialog

All the options are the same - just in different places.

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

The start application wizard

On first use, the wizard appears with fields cleared, but here's an example with a few fields set:



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 Coverage Validator in administrator mode.

Anywhere you see the icon indicates that administrator privileges will be required to proceed.

If you started Coverage Validator in administrator mode, you won't see any of these warnings, and everything will behave as normal.

Page 1: Entering details

- **Application** › type or **Browse** to set the program name to launch

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

  You can also choose a powershell script and the first executable started in the powershell script will be launched.

  Manually typing a path will show red text until a valid path is entered, after which the text becomes black.

- **Application to monitor** › choose the application that actually gets monitored

  This will typically just be the program that you set to start - unless otherwise specified.

  Alternatively you can monitor another application which will get launched by the start program.

  If the start application has already been added to the Applications to Monitor settings with a default application then that default will be entered here automatically.

  Otherwise, if nothing has been set up yet, you can do it from here:

  - **Edit...** › set the child applications that can be monitored for the start program

    This uses the Applications to Monitor dialog - which is exactly equivalent to using the Applications to Monitor settings page.

  A fallback option is to start monitoring `<<Any application that is launched>>`.

   If in doubt, just use the same as the start application.

  ➜ See also: Application to Monitor settings

- **Launch Count** › when monitoring a *child* application, set its 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.

- **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

- **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

Page 2: Data collection and redirection

- **Type of data collection** › Are you only interested in Native data, .Net data or both Native data and .Net data?
  - **Native Only** › Ignore all .Net data in the target application.
  - **.Net Only** › Ignore all Native data in the target application.
  - **Mixed Mode** › Collect both Native and .Net data from the target application

  This setting cannot be changed after the application is launched

- **Collect data from application** › If it's the startup procedure you want to validate, obviously start collecting data from launch.

  Depending on your application, and what you want to validate, you may want to start collecting data immediately, 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.

➡ See the section on controlling data collection for how to turn collection on and off after launch.

➡ The data collection option may be disabled because of the instrumentation mode that is selected.

- **Redirect standard output** › Controls redirection of stdout and stderr

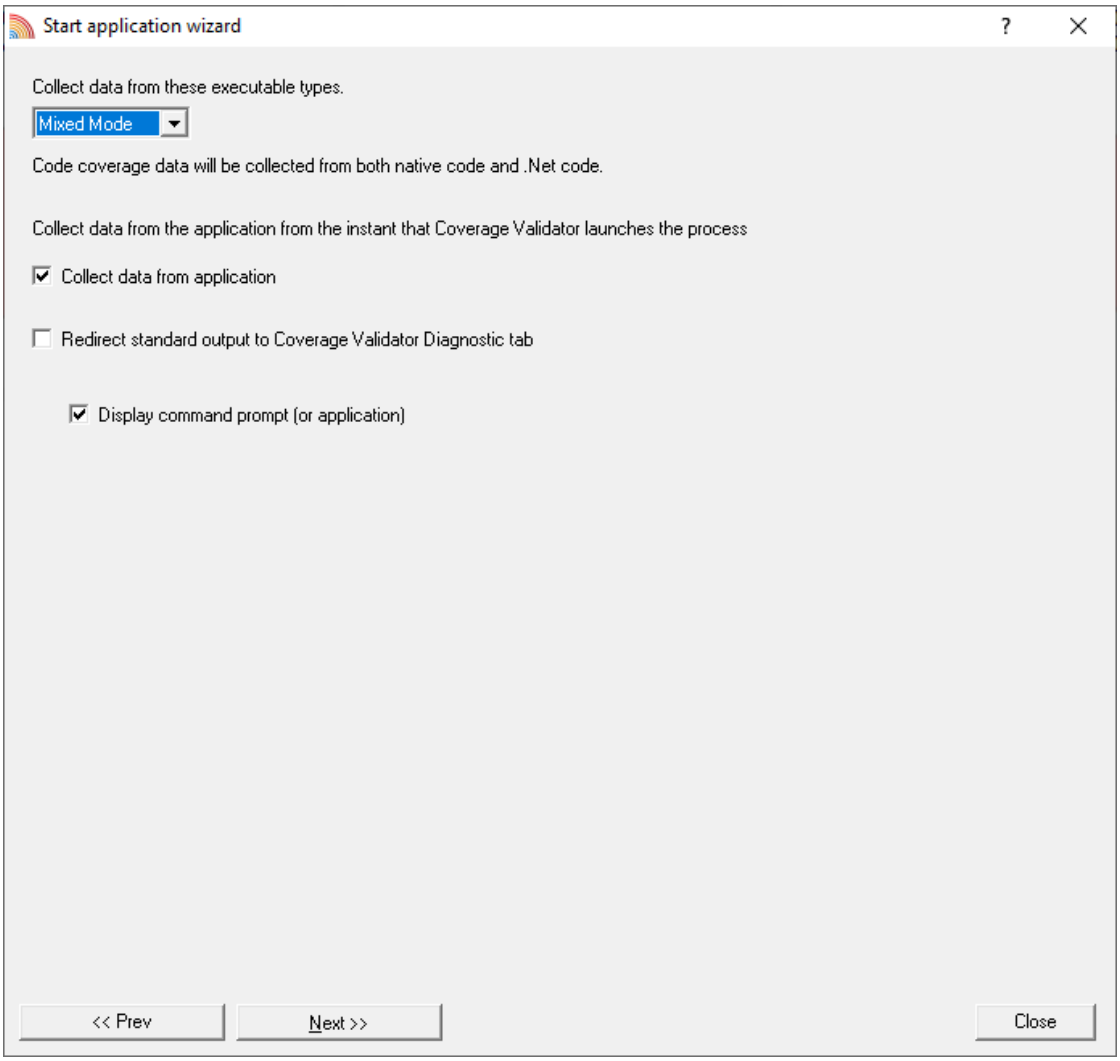
  Use this option if you want to collect the output of stdout and stderr for later analysis.

  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.



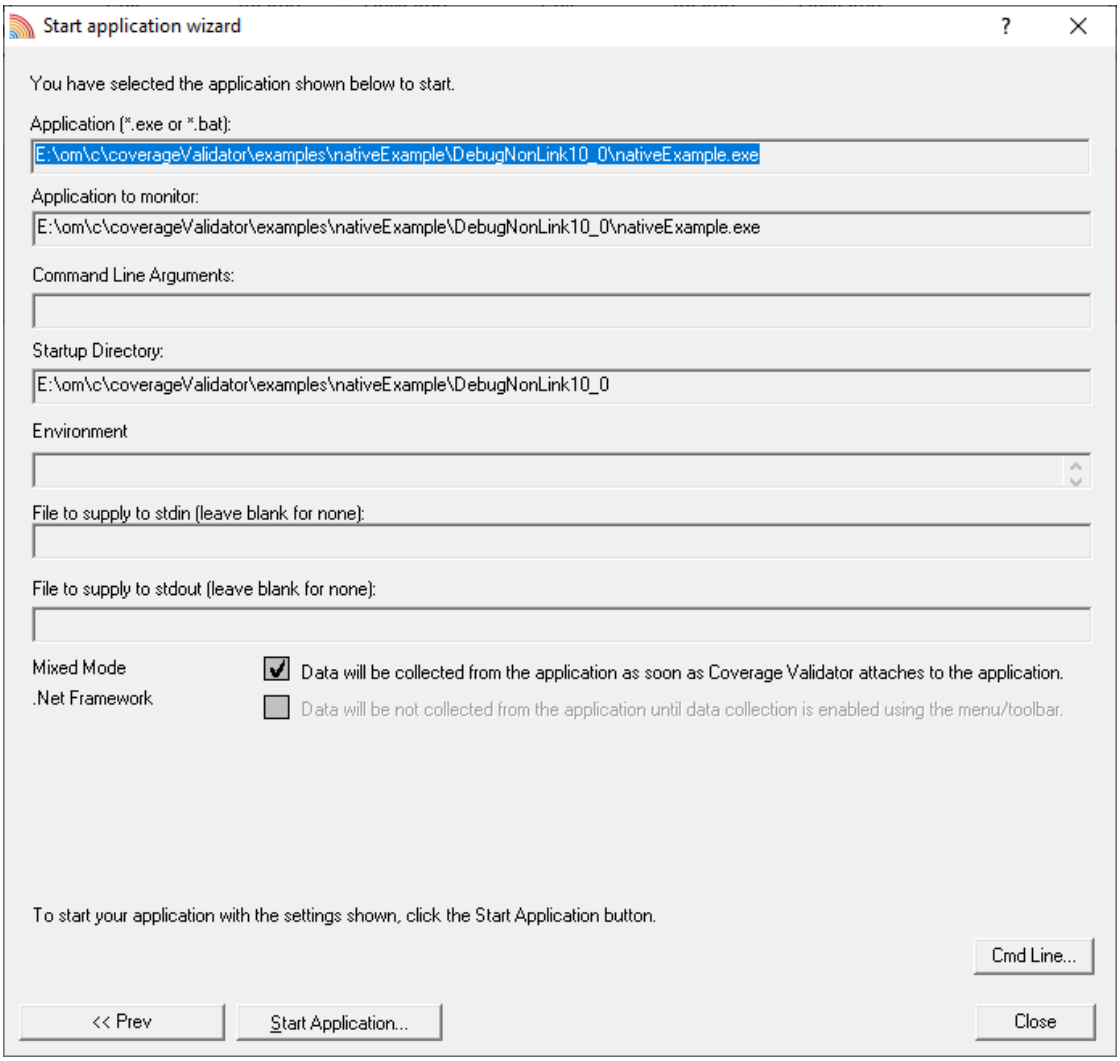
Page 3: Summary and starting your program

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.

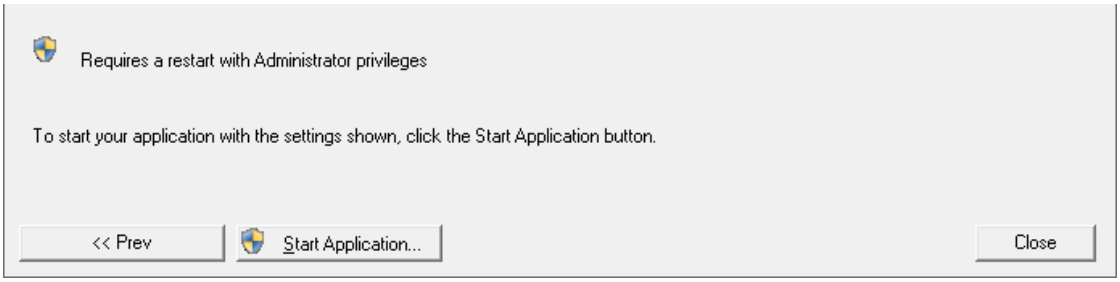
If you're happy with the settings, go ahead:

- **Start Application...** › start your program and attach Coverage Validator to it
- **Cmd Line...** › display the command line builder

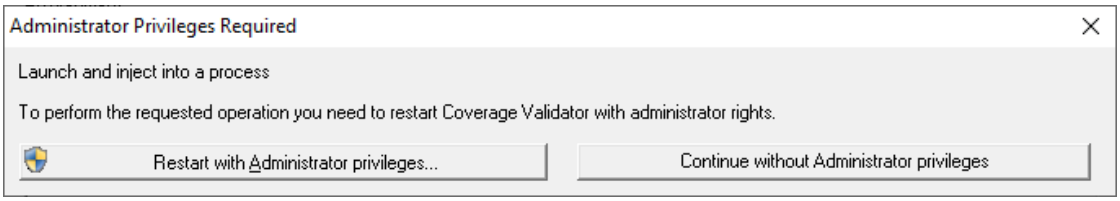


Administrator privileges in wizard mode

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



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



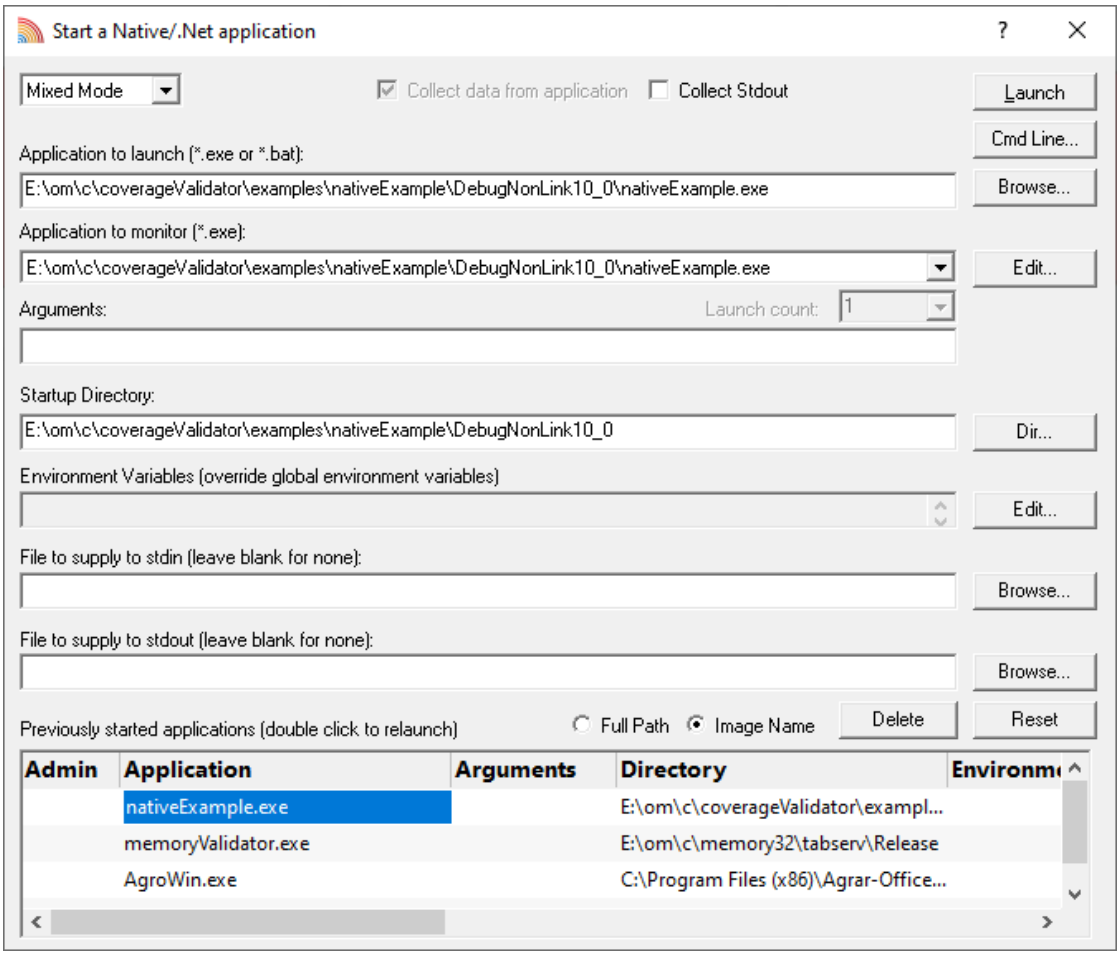
Dialog mode

In Dialog mode, all the settings are in one dialog which looks very much like the first page of the launch wizard above.

The option to start collecting data is at the top.

- **Launch** › start your program and attach Coverage Validator to it
- **Cmd Line...** › display the command line builder

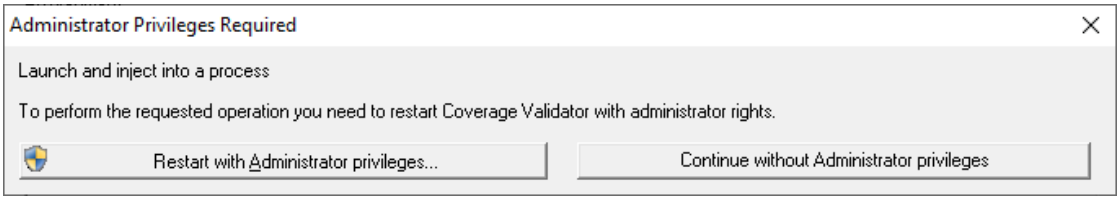
Double clicking a program in the list will also start it immediately.



Administrator privileges in dialog mode

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

- **Launch** > shows the Administrator Privileges Required confirmation dialog before restarting.



If you started Coverage 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 Coverage Validator.

Let's say we have a program **P**.

In the simplest case, simply:

- *start* **P**
- *monitor* **P**
- the Launch Count defaults to **1** and cannot be changed.

If **P** launches an application and you just want to monitor whatever that is:

- *start* **P**
- *monitor* `<<Any application that is launched>>`
- leave the Launch Count at **1**

If **P** launches an application **A** and maybe others as well, and you specifically want to monitor only **A** as it's launched:

- use the Application to Monitor settings to add a definition for P and child applications **A**
- *start* **P**
- *monitor* **A**
- leave the Launch Count at **1**

If **P** launches an application **A** many times and you specifically want to monitor the third invocation:

- use the Application to Monitor settings to add a definition for P and child applications **A**
- *start* **P**
- *monitor* **A**
- set the Launch Count to **3**

3.16.3 Launching the program (.Net Core)

Launching the application

Having Coverage Validator launch your program is the most common way to start up

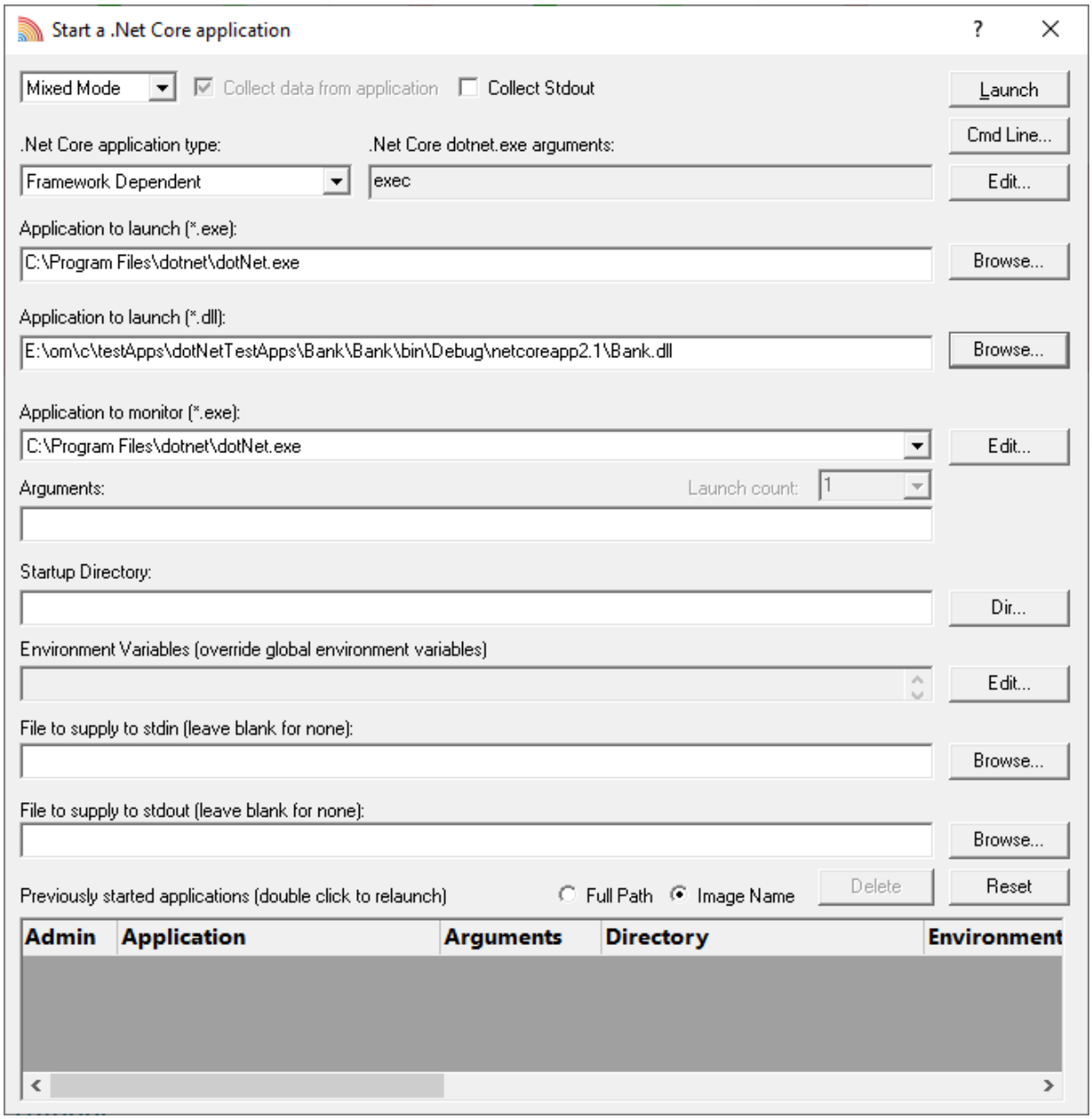
When you're ready to start running a .Net Core program

Launch menu › **Applications** › **Launch .Net Core Application** › Shows the .Net Core launch dialog below

Or use the shortcut

Shift + F4 Launch .Net Core Application

➜ You can easily re-launch the most recently run program.



.Net Core Application Type

.Net Core applications can be self contained or framework dependent. This changes how the launch dialog works.

- **.Net Core application type** › choose which type of .Net Core application you are launching

.Net Core Self Contained Application

- **Application to launch (*.exe)** › type or **Browse** to set the program name (*.exe) to launch

When you set this value the Application to launch DLL field will be automatically populated to have the same name as the EXE field but with a .DLL extension.

- **Application to launch (*.dll)** › type or **Browse** to set the program name (*.dll) to launch
- **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

.Net Core Framework Dependent Application

- **Application to launch (*.exe)** › type or **Browse** to set the program name (*.exe) to launch

  We don't auto-populate this field when you choose the Framework dependent application type. This because you may have your .Net Core runtime stored in a location that we can't auto-detect.

  To accommodate alternate locations for the .Net Core runtime we only auto-populate this field if it is empty when you choose the application DLL.

- **Application to launch (*.dll)** › type or **Browse** to set the program name (*.dll) to launch

  If you set this when Application to launch EXE field is empty, the EXE field will be automatically populated with the path to the system .Net Core framework dependent runtime.

  This is typically **c:\program files\dotnet\dotnet.exe**.

- **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

- **.Net Core dotnet.exe arguments** › any arguments that will be passed to the .Net Core runtime to control how the .Net Core runtime behaves.
- **Edit...** › displays the .Net Core runtime arguments editor

Fields common to all .Net Core applications

- **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.

- **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

- **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

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

Data collection and redirection

- **Type of data collection** › Are you only interested in Native data, .Net data or both Native data and .Net data?
  - **Native Only** › Ignore all .Net data in the target application.
  - **.Net Only** › Ignore all Native data in the target application.
  - **Mixed Mode** › Collect both Native and .Net data from the target application

  This setting cannot be changed after the application is launched
- **Collect data from application** › If it's the startup procedure you want to validate, obviously start collecting data from launch.

  Depending on your application, and what you want to validate, you may want to start collecting data immediately, 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.

  ➡ See the section on controlling data collection for how to turn collection on and off after launch.

  ➡ The data collection option may be disabled because of the instrumentation mode that is selected.
- **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.

The option to start collecting data is at the top.

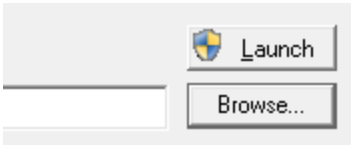
- **Launch** › start your program and attach Coverage Validator to it

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

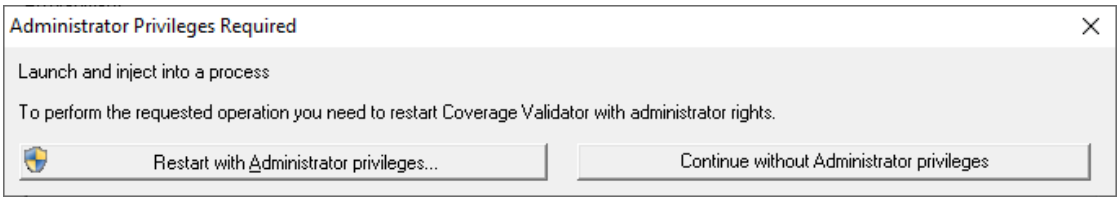
- **Cmd Line...** › display the command line builder

Administrator privileges in dialog mode

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



- **Launch** › shows the Administrator Privileges Required confirmation dialog before restarting



If you started Coverage 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 Coverage Validator.

Let's say we have a program **P**.

In the simplest case, simply:

- *start* **P**
- *monitor* **P**
- the Launch Count defaults to **1** and cannot be changed.

If **P** launches an application and you just want to monitor whatever that is:

- *start* **P**
- *monitor* `<<Any application that is launched>>`
- leave the Launch Count at **1**

If **P** launches an application **A** and maybe others as well, and you specifically want to monitor only **A** as it's launched:

- use the Application to Monitor settings to add a definition for P and child applications **A**
- *start* **P**
- *monitor* **A**
- leave the Launch Count at **1**

If **P** launches an application **A** many times and you specifically want to monitor the third invocation:

- use the Application to Monitor settings to add a definition for P and child applications **A**
- *start* **P**
- *monitor* **A**
- set the Launch Count to **3**

3.16.4 Re-launching the program

Re-launching the application

It's very easy to start another session using the most recently run program and settings:

Launch menu › **Applications** › **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

If the previously launched program was Native, .Net or .Net Core the application will be restarted immediately. No wizards or dialogs appear.

If the previously launched program was a service the appropriate monitor service dialog will be displayed.

➜ In the general questions see *Why might Inject or Launch fail?* for troubleshooting launch problems.

There is no difference between wizard and dialog interface mode when re-launching.

3.16.5 Injecting into a running program

Injecting into a running program

Coverage Validator attaches to a running process by injecting the stub into the process so it can start collecting data.

Choose one of these methods of starting the injection:

File menu › **Inject...** › shows the Attach to Running Process wizard or dialog below

Or click on the Inject icon on the session toolbar.

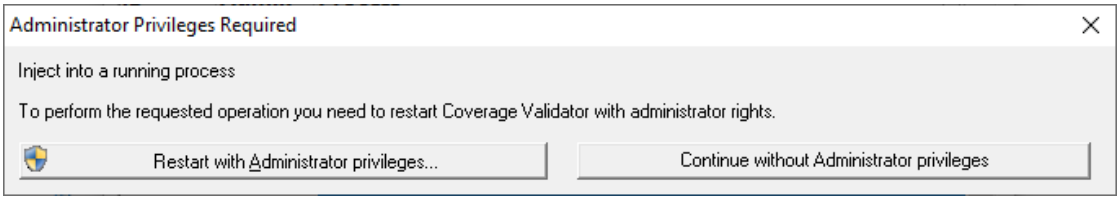
Or use the shortcut

F3 Inject into running application

Administrator privileges

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

When choosing the inject method, a restart with administrator privileges will be required to proceed.



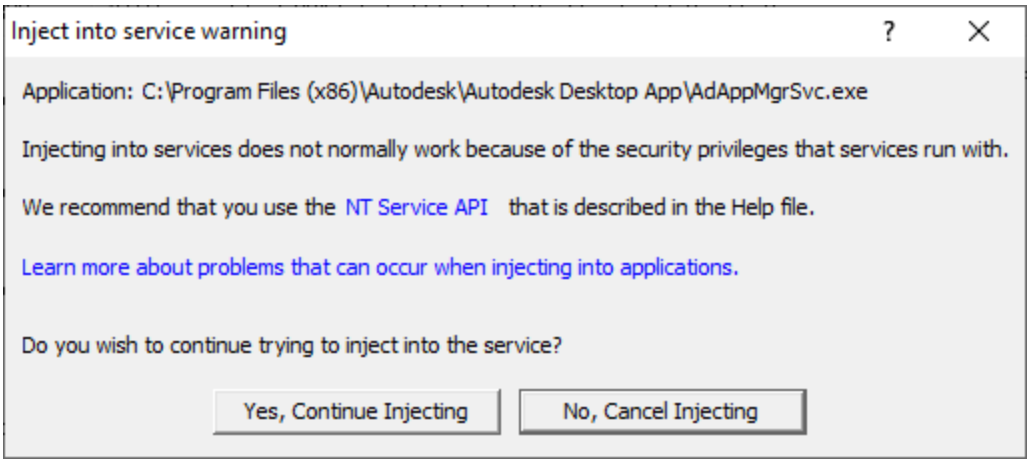
Injecting into a service?

If your process is a service, Coverage 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

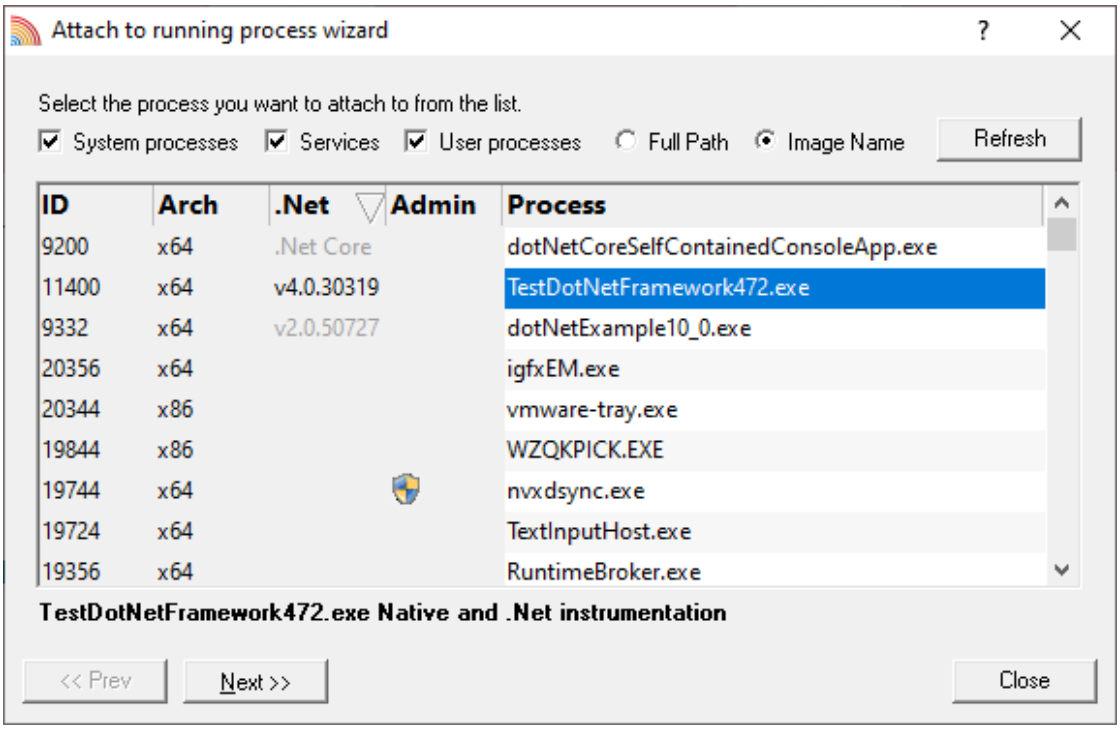
In this section we'll cover the Wizard mode first and the Dialog mode later.

The attach to running process wizard

The first page of the wizard shows a list of running system and user processes.

The **Arch** column is not shown when running 32 bit Coverage Validator because only 32 bit processes are listed.

Any processes that have grayed out **.Net** values cannot instrument the .Net part of the application (native components will be instrumented).



Choose the process and click **Next >>** for the next page of the wizard.

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

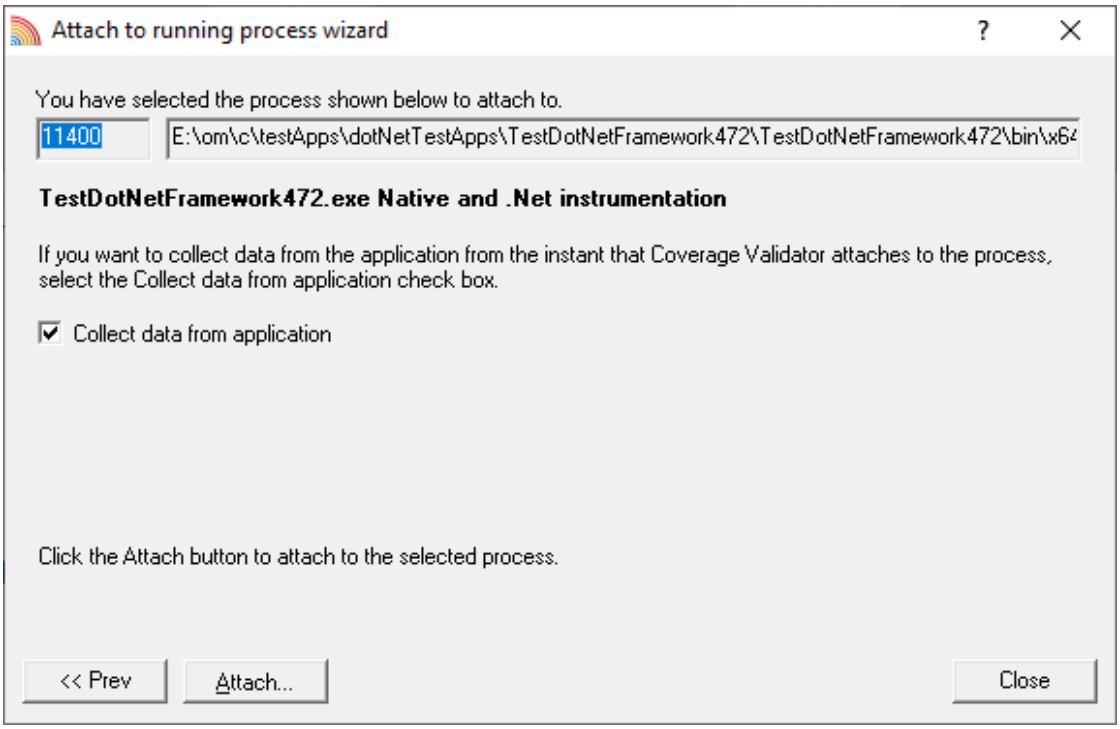
Depending on your application, and what you want to test, you may want to start collecting data as soon as injection happens, or do it later.

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.

If it's the startup procedure you want to test, then obviously start collecting data from launch.

➡ See the section on controlling data collection for how to turn collection on and off after launch.

➡ The data collection option may be disabled because of the instrumentation mode that is selected.



Currently we only support attaching to native applications and the native part of mixed mode applications.

Summary and starting your program

The second page confirms the process you have selected to inject into, and prompts you to attach:

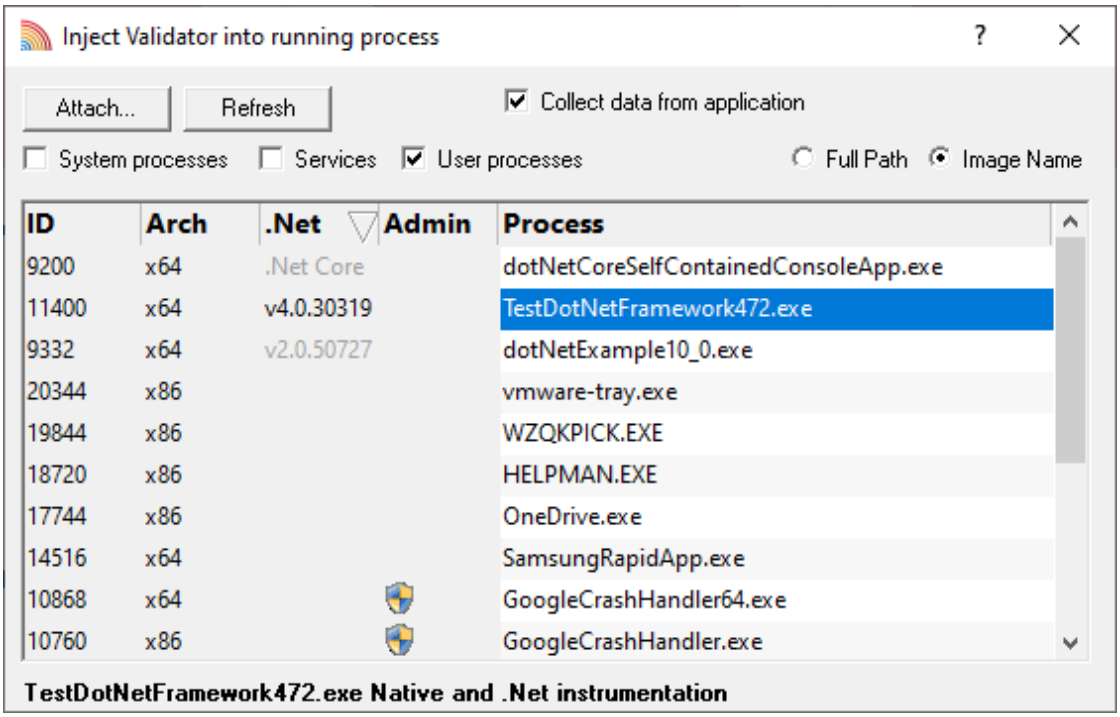
- **Attach...** › injects Coverage Validator into the specified process, showing progress status

➡ 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 very much like the first page of the wizard above.

The option to start collecting data is at the top, as is the **Attach...** button



3.16.6 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, Coverage Validator watches for the process starting up and *then* injects.

If the process is a service, Coverage 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.

Choose one of these methods of waiting:

Launch menu › **Applications** › **Wait for Application...** › shows the Wait for application wizard or dialog below

or click on the Wait (timer) icon on the session toolbar.

or use the shortcut

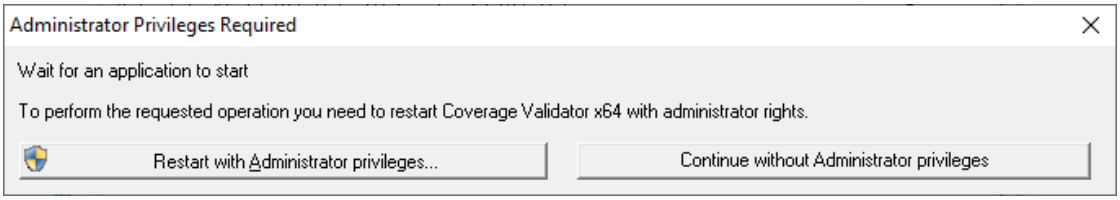
F2 Wait for application

Administrator privileges

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

If the application you want to wait for is running with Administrator privileges, Coverage Validator will also need to run with Administrator privileges.

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



Waiting for a service?

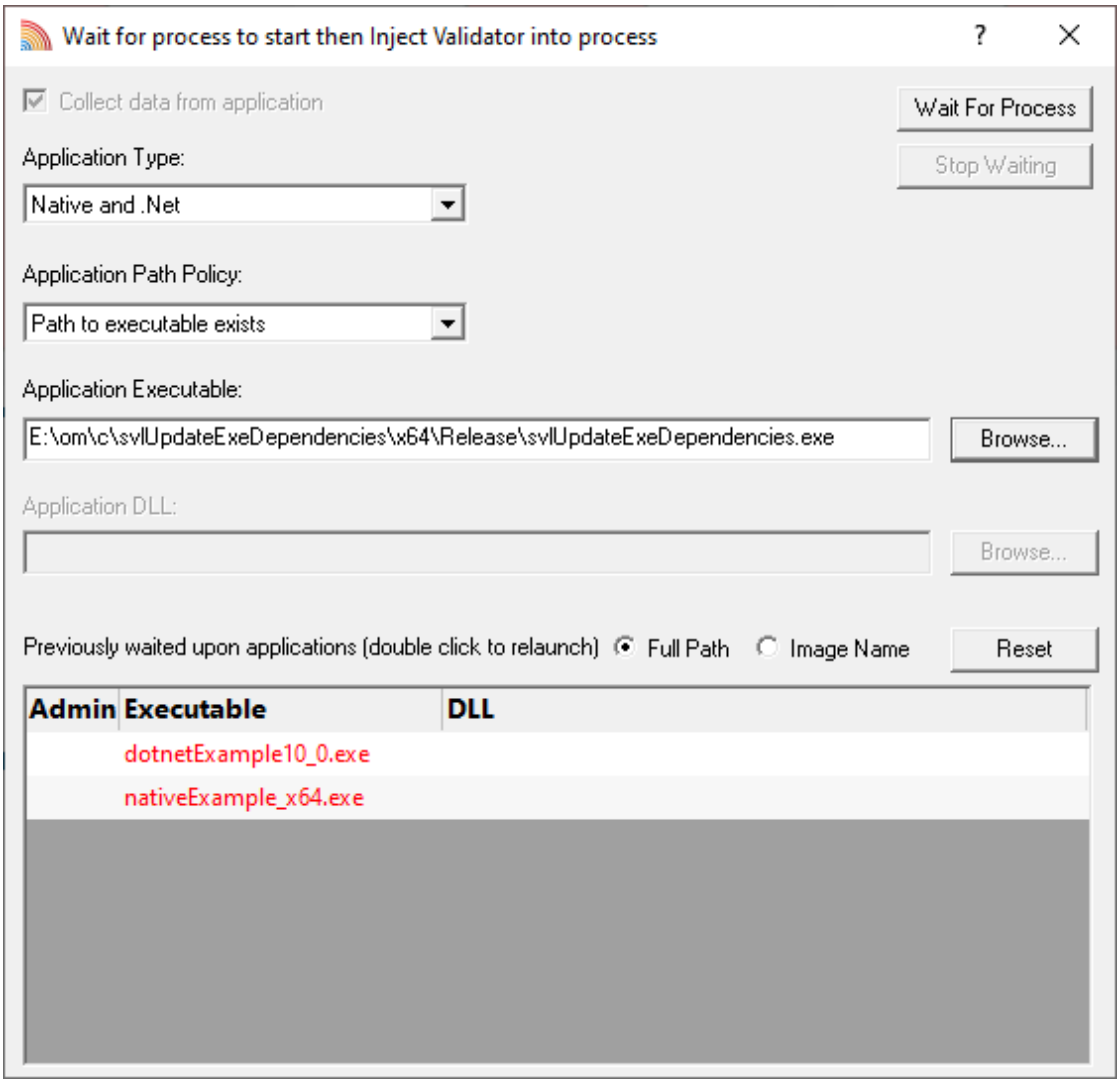
If your process is a service, Coverage Validator won't be able to attach to it.

Services can't have process handles opened by third party applications, even with Administrator privileges.

In order to work with services, you can use the NT service API and monitor the service

The wait for application dialog

The wait for application dialog lets you specify the application or choose one that you've waited for previously.



- **Collect data from application** › do want to collect data from the instant you attach to the application?

  Depending on your application, and what you want to validate, you may want to start collecting data as soon as injection has happened, or do it later.

  If your program has a complex start-up procedure, initialising lots of data, it may be much faster *not* to collect data until the program has launched.

  If it's the startup procedure you want to validate, obviously start collecting data from launch.

  ➡ See the section on controlling data collection for how to turn collection on and off after launch.

  ➡ The data collection option may be disabled because of the instrumentation mode that is selected.

- **Application Path Policy** › specify how the specified executable is treated

  - Path to executable exists › the executable will be checked that it exists and is appropriate for Coverage Validator to work with
  - Path to executable is created dynamically › most pre-wait checks are not performed - use this if the path the executable is on does not exist at the time you start waiting for the process to start

- **Application type** › choose the type of application
  - Native and .Net
  - .Net Core (Framework Dependent)
  - .Net Core (Self Contained)
- **Application Executable** › edit or **Browse...** the application to wait to start.

  The name of the executable. For example **c:\unitTests\test.exe** or **test.exe**.

  If Application Path Policy is **Path to executable exists** this must be the full path to the executable. For example **c:\unitTests\test.exe**.

  For native applications this is the application executable.

  For .Net Framework applications this is the application executable.

  For .Net Core Framework-dependent applications this is most likely going to be **c:\program files\dotnet\dotnet.exe**.

  For .Net Core Self-contained applications this is the application executable.
- **Application DLL** › edit or **Browse...** the application DLL to wait to start. This field is only needed for .Net Core applications.

  The name of the DLL. For example **c:\unitTests\test.dll** or **test.dll**.

  If Application Path Policy is **Path to executable exists** this must be the full path to the dll. For example **c:\unitTests\test.dll**.

  For native applications this is not used.

  For .Net Framework applications this is not used.

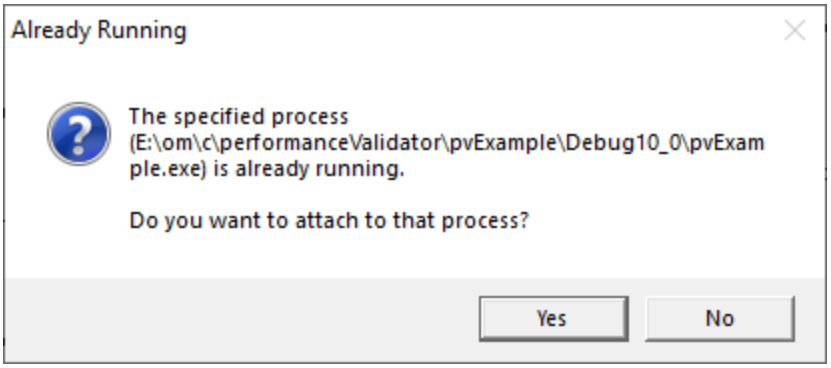
  For .Net Core Framework-dependent applications this is the application dll. (the name of the dll that you would pass to dotnet.exe on the command line).

  For .Net Core Self-contained applications this is the dll that has the same name as the application executable. (for theApp.exe, the dll name is theApp.dll).
- **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

- **Wait for Process...** > starts waiting and then injects Coverage Validator into the specified process, showing progress status
- **Stop Waiting** > stops the wait

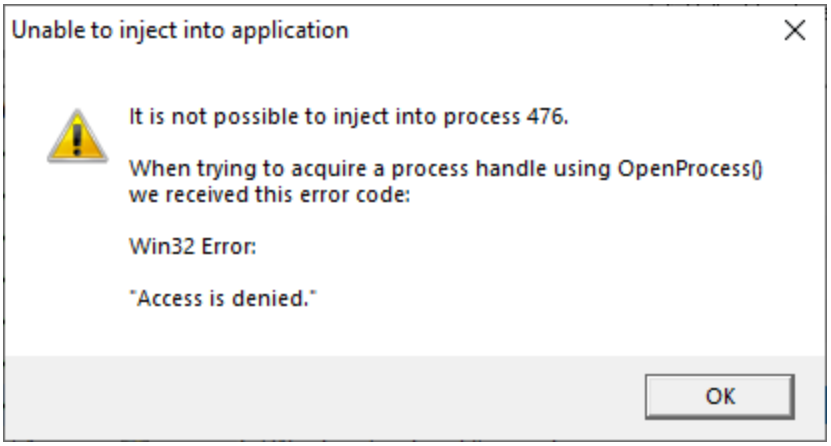
What could go wrong?

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:



Timing issues are inherit with native injecting into a program as it starts up.

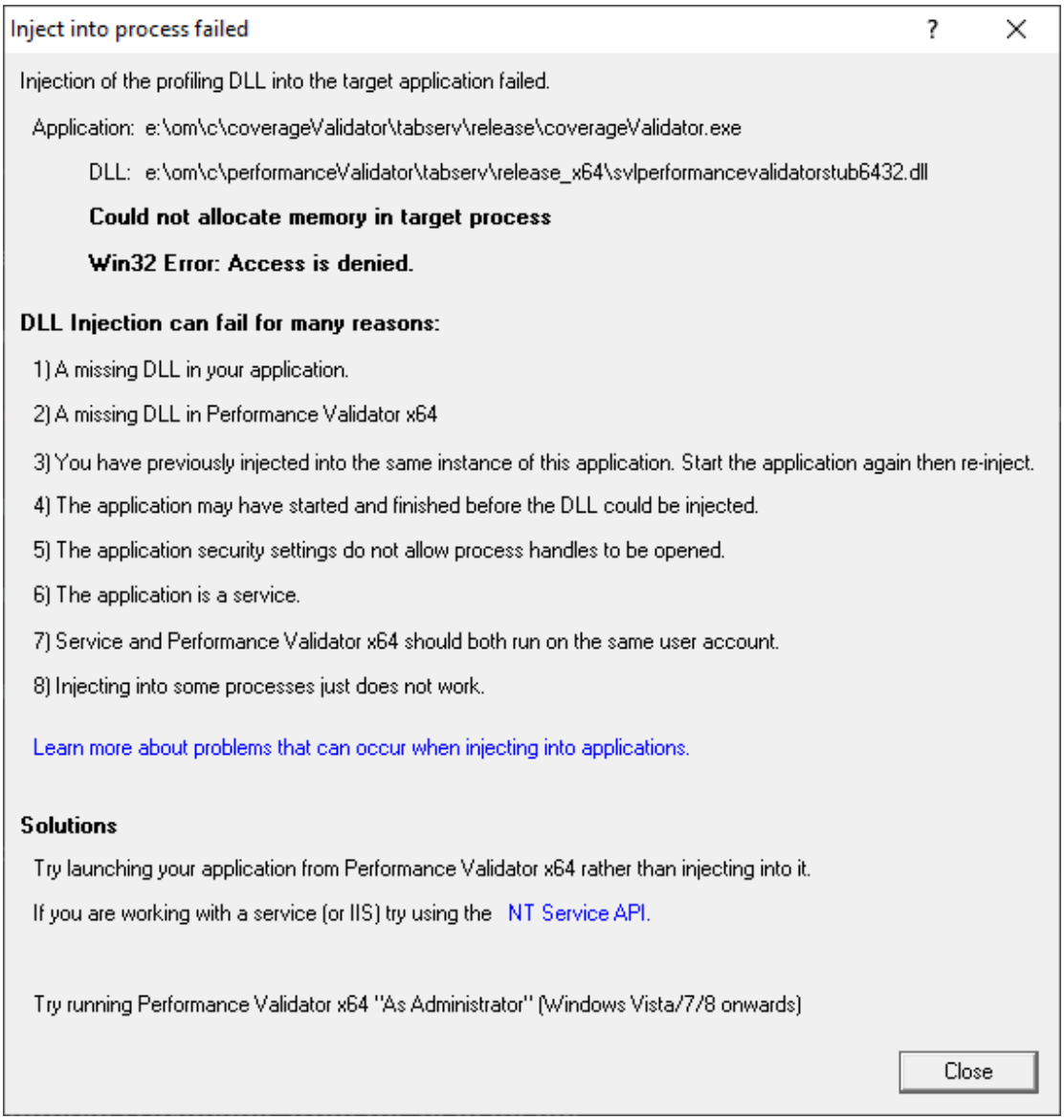
This could cause the injection to fail in unpredictable ways and you may see dialogs like that below:



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

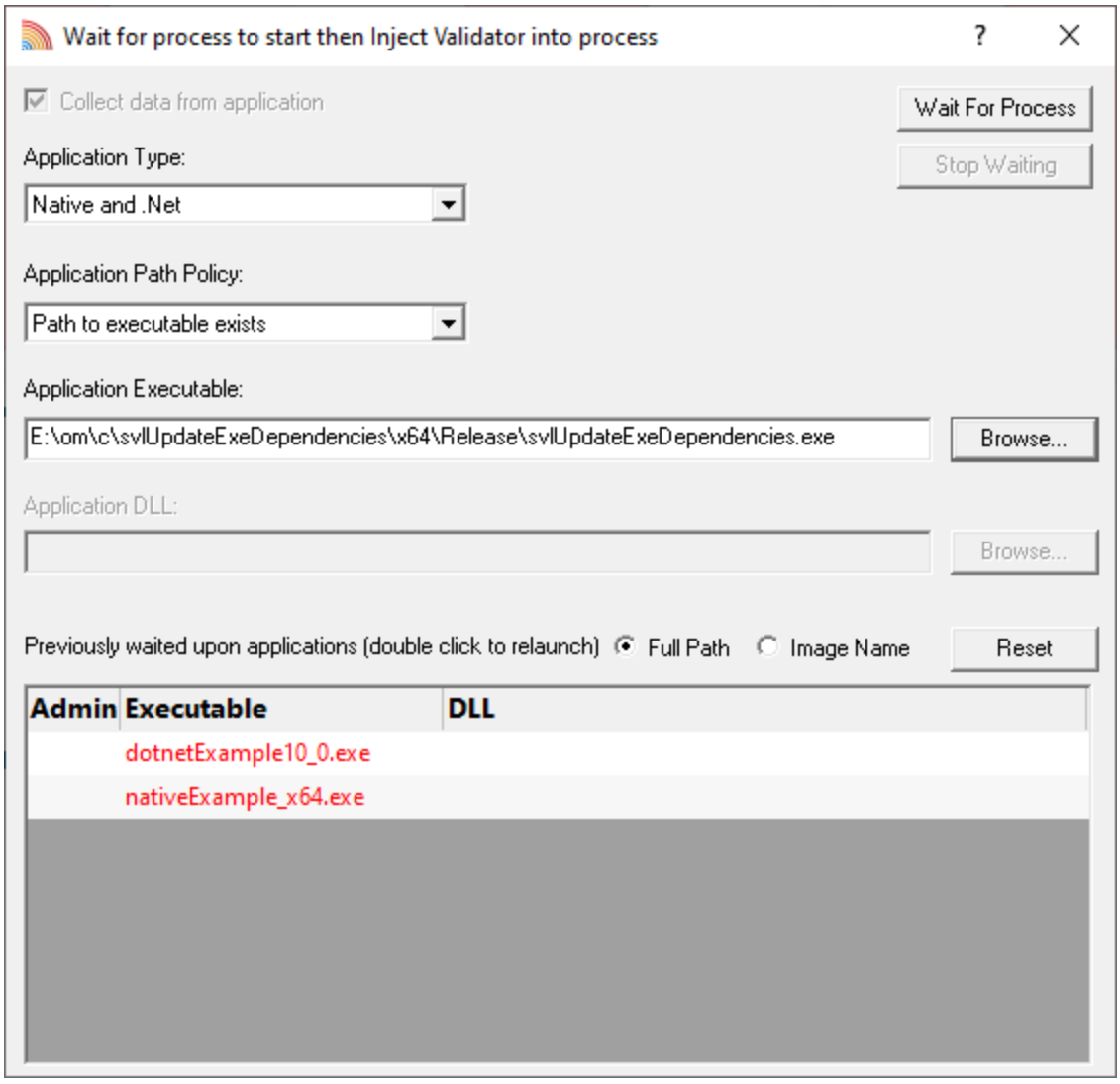


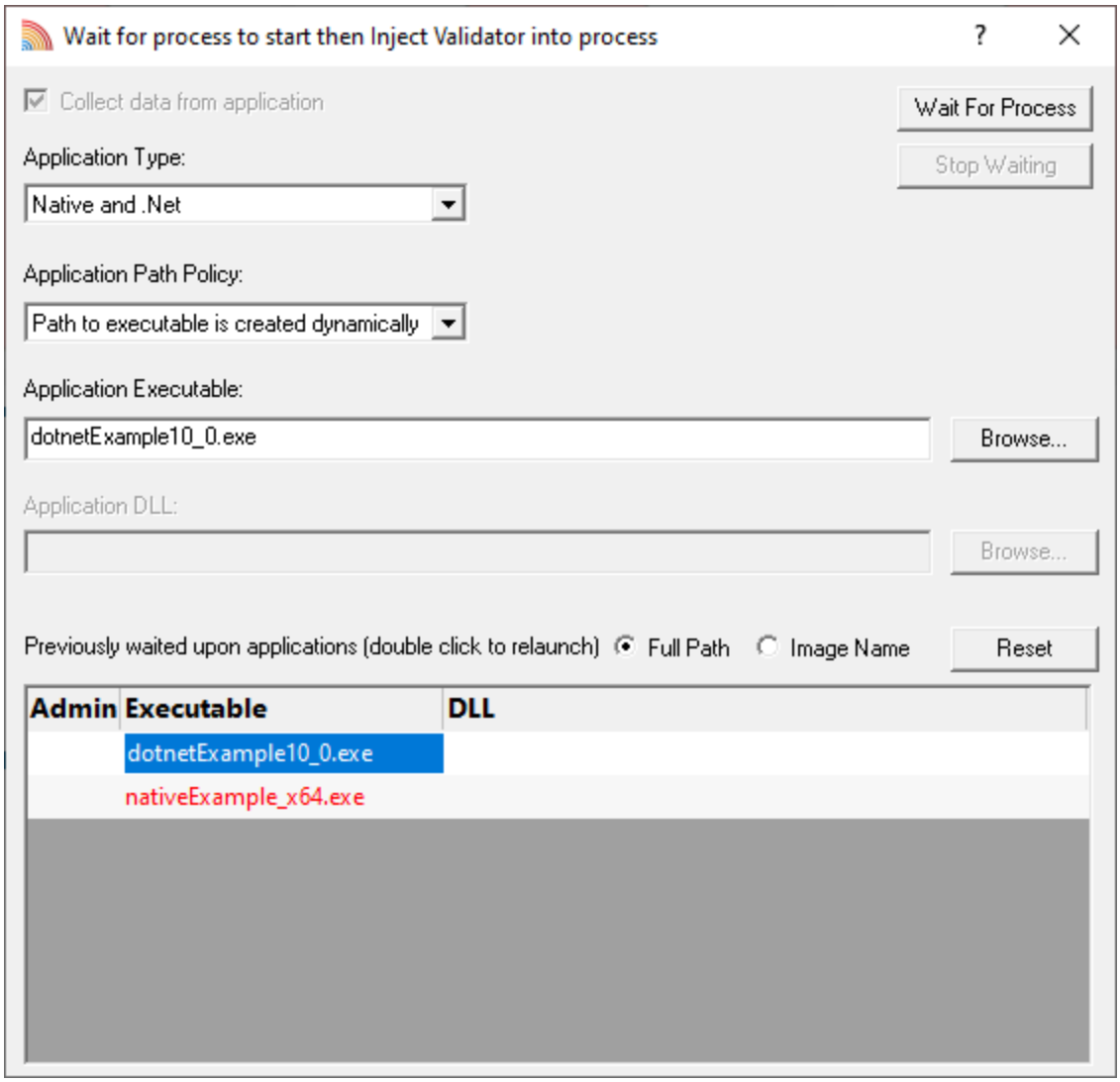
Sometimes retrying a few times might catch a better moment for attaching to the process.

➡ In the general questions see *Why might Inject or Launch fail?* for troubleshooting launch problems.

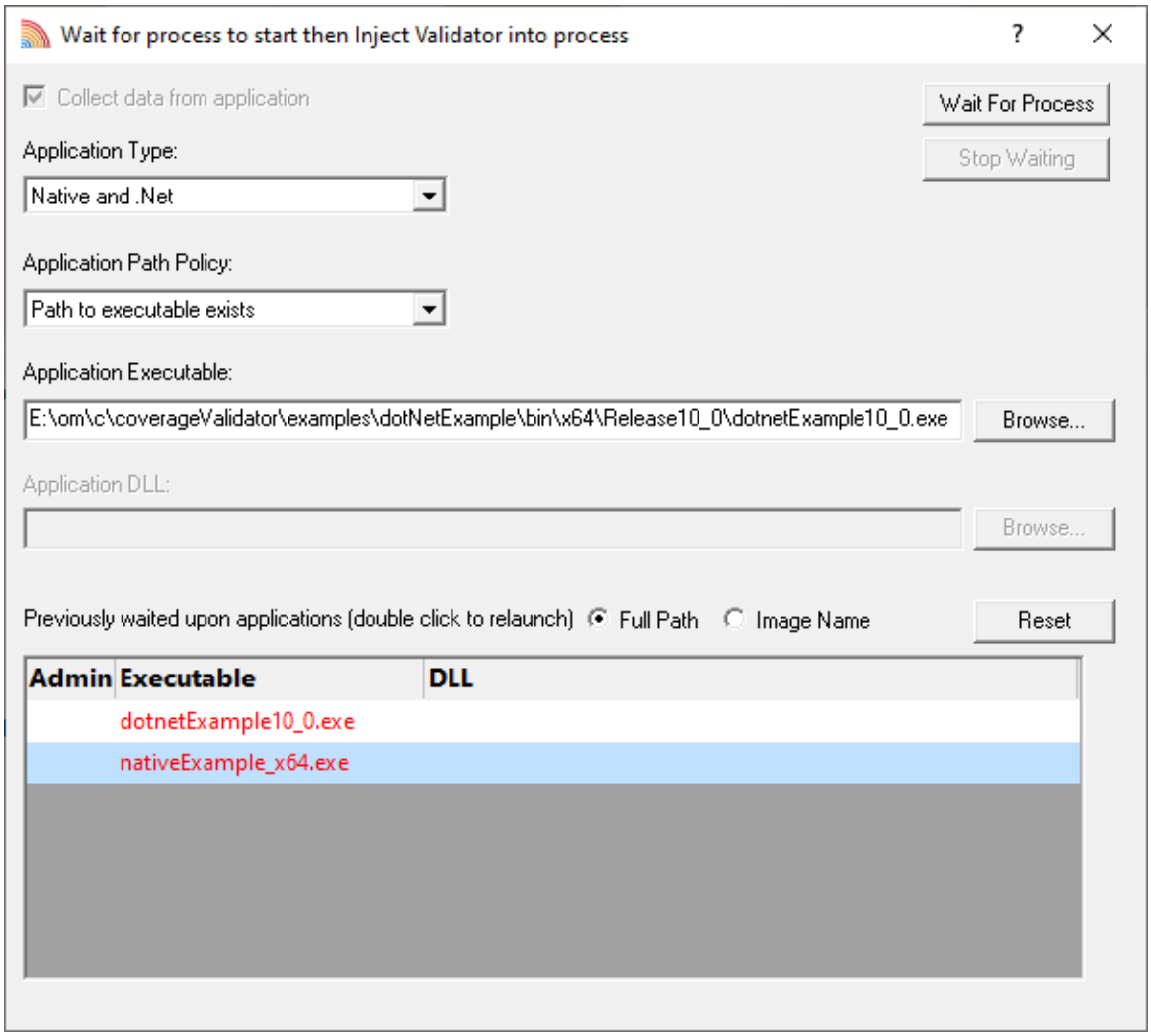
Example Dialogs

Native

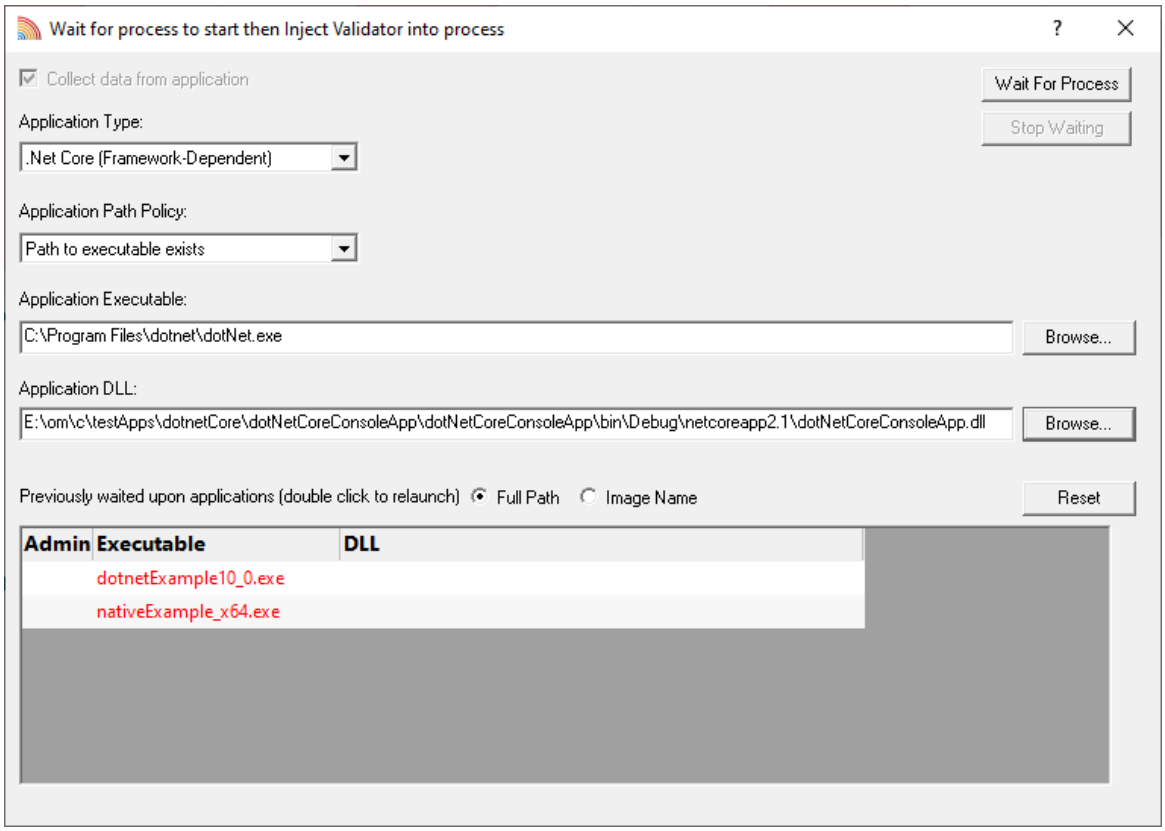




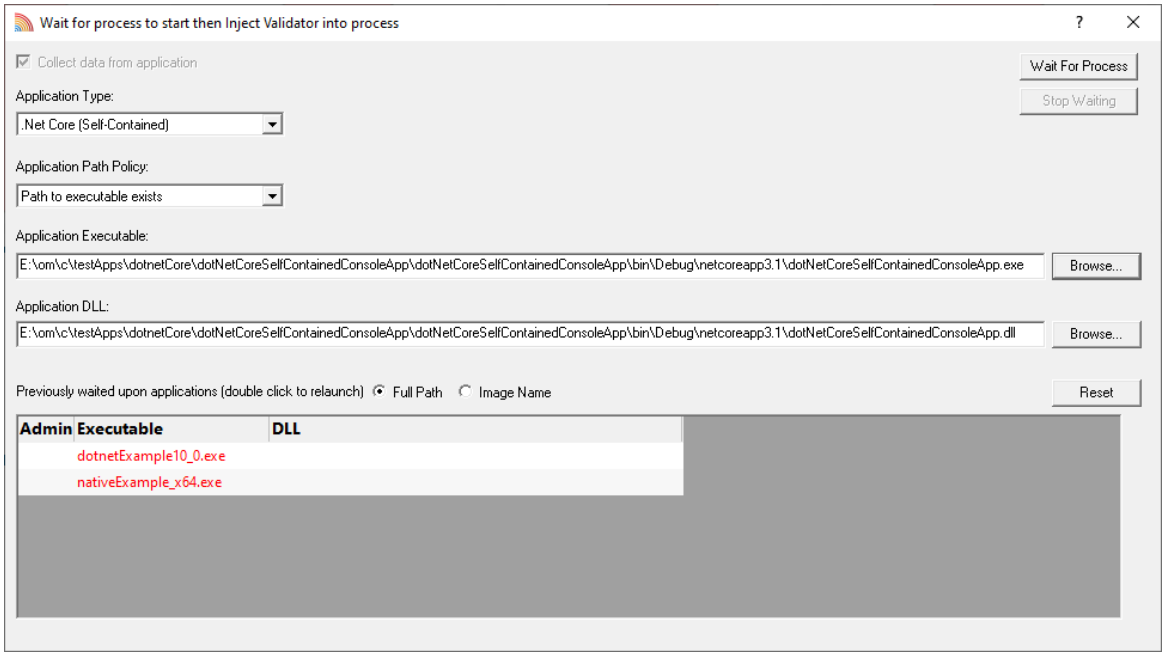
.Net



.Net Core (Framework-dependent)



.Net Core (Self-contained)



3.16.7 Monitor a service

Monitoring a service

Monitoring a service works for:

- Native services
- .Net services
- Mixed mode services.

Native Services

If you are 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

Coverage Validator won't attach until some .Net code is executed.

If there is native code being called prior to the .Net code, Coverage 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, Coverage Validator will recognize the service when the .Net runtime starts executing the .Net service main.

 When working with Coverage 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 Coverage Validator, which you should have running before starting the service.

To start monitoring a service:

 **Launch** menu > **Services** > **Monitor a service...** > shows the Monitor a service dialog below

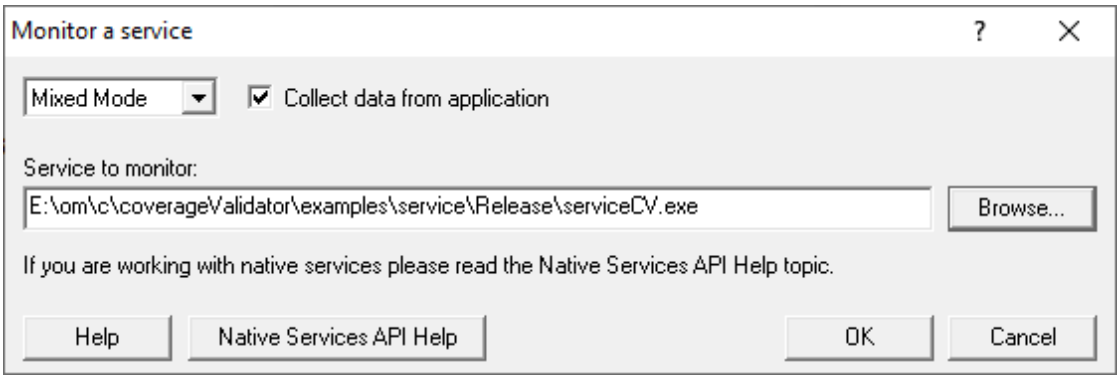
Or use the shortcut

 Monitor a service

The monitor a service dialog

First ensure the service is installed, but not running.

Set the service to monitor, choose whether to start collecting data right away, and click OK.



- **Service to monitor** › type or **Browse** to set the service name to monitor
- **OK** › waits for the service to start before injecting into it

Start the service in the normal manner, e.g. from the control panel, the command line or programmatically.

Data collection

- **Type of data collection** › Are you only interested in Native data, .Net data or both Native data and .Net data?
  - **Native Only** › Ignore all .Net data in the target application.
  - **.Net Only** › Ignore all Native data in the target application.
  - **Mixed Mode** › Collect both Native and .Net data from the target application

  This setting cannot be changed after the application is launched
- **Collect data from application** › If it's the startup procedure you want to validate, obviously start collecting data from launch.

  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.

  ➡ The data collection option may be disabled because of the instrumentation mode that is selected.

Examples

Example demonstrating how to monitor a service.

Example demonstrating how to monitor an application launched from a service (how to monitor any application running on a service account).

3.16.8 IIS

Enter topic text here.

3.16.8.1 Monitor IIS and ISAPI

Monitoring ISAPI

Monitoring ISAPI works for:

- Native ISAPI extensions.
- Mixed mode ISAPI extensions (native ISAPI that uses .Net code).

If you are working with native ISAPI you must use the NT Service API in your service *as well as* using the Monitor ISAPI method below.

To start monitoring ISAPI:

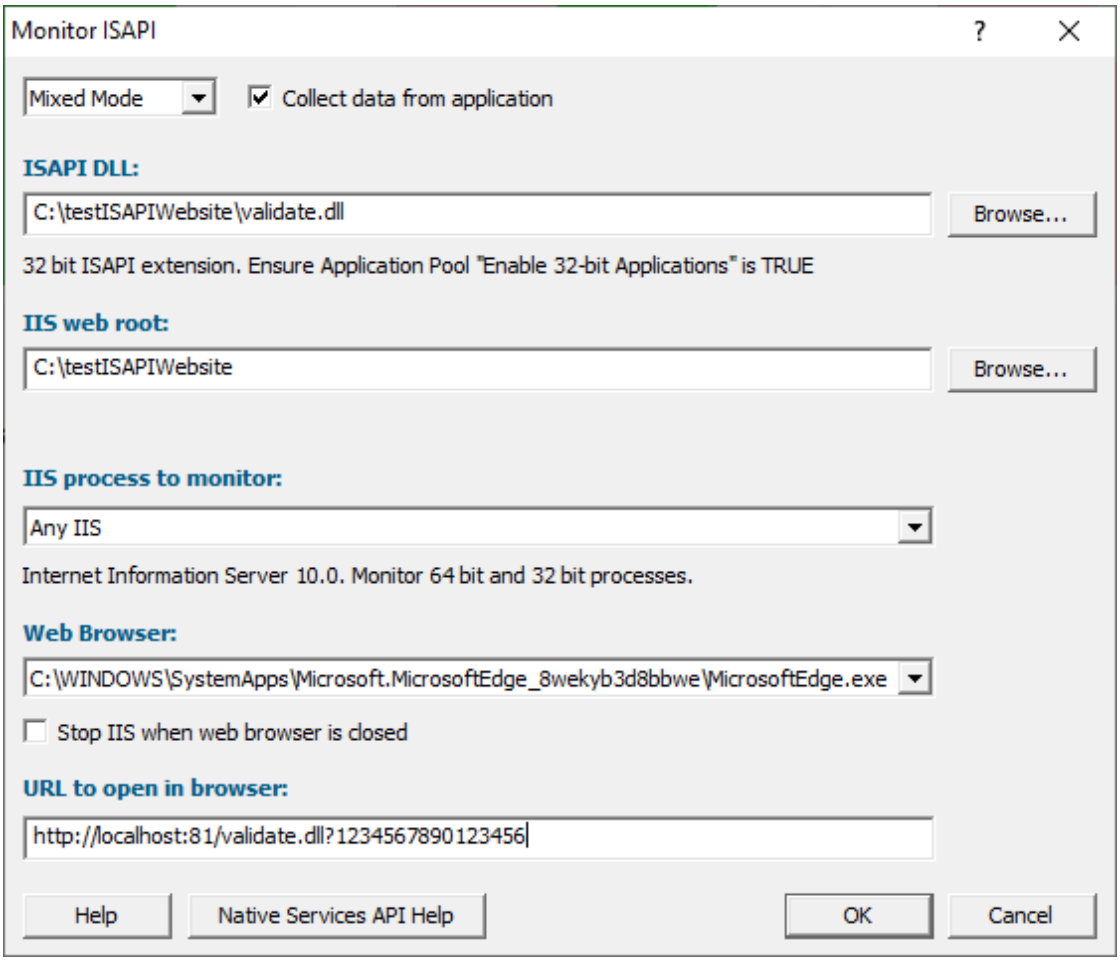
 **Launch** menu › **IIS** menu › **Monitor IIS and ISAPI...** › shows the Monitor ISAPI dialog below

Or use the shortcut

 Monitor IIS and ISAPI

The monitor ISAPI dialog

Set the dll to monitor, the web root, the IIS process, an optional web browser to use and an optional url to launch, and click OK.



- **ISAPI DLL** › type or **Browse** to set the ISAPI DLL that we're monitoring
- **IIS web** › type or **Browse** to set the web root for the IIS website we're working with
- **IIS process to monitor** › select the IIS process we're working with
- **Web Browser** › select the web browser that you're going to use to load the web page
- **URL to open in browser** › type the web page and arguments you want to load to cause the ISAPI to be loaded in IIS
- **OK** ›resets IIS, setups all the variables, copies DLLs and settings into the web root and starts the web browser to load the specified web page

IIS is a protected process and can only execute, read and write files in specific directories. That's why Coverage Validator copies data to the web root so that it can be read, written or executed.

Data collection

- **Type of data collection** › Are you only interested in Native data, .Net data or both Native data and .Net data?
  - **Native Only** › Ignore all .Net data in the target application.
  - **.Net Only** › Ignore all Native data in the target application.
  - **Mixed Mode** › Collect both Native and .Net data from the target application

  This setting cannot be changed after the application is launched

- **Collect data from application** › If it's the startup procedure you want to validate, obviously start collecting data from launch.

  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.

  ➡ The data collection option may be disabled because of the instrumentation mode that is selected.

Slow Startup

The first time you work with Web Development Server and Coverage Validator you may experience a delay during startup. This is most like because symbols are being downloaded from Microsoft's symbol servers to match the DLLs and assemblies on your machine.

3.16.8.2 Monitor IIS and ASP.Net

Monitoring ASP.Net Application (IIS)

To start monitoring ASP.Net application running in IIS:

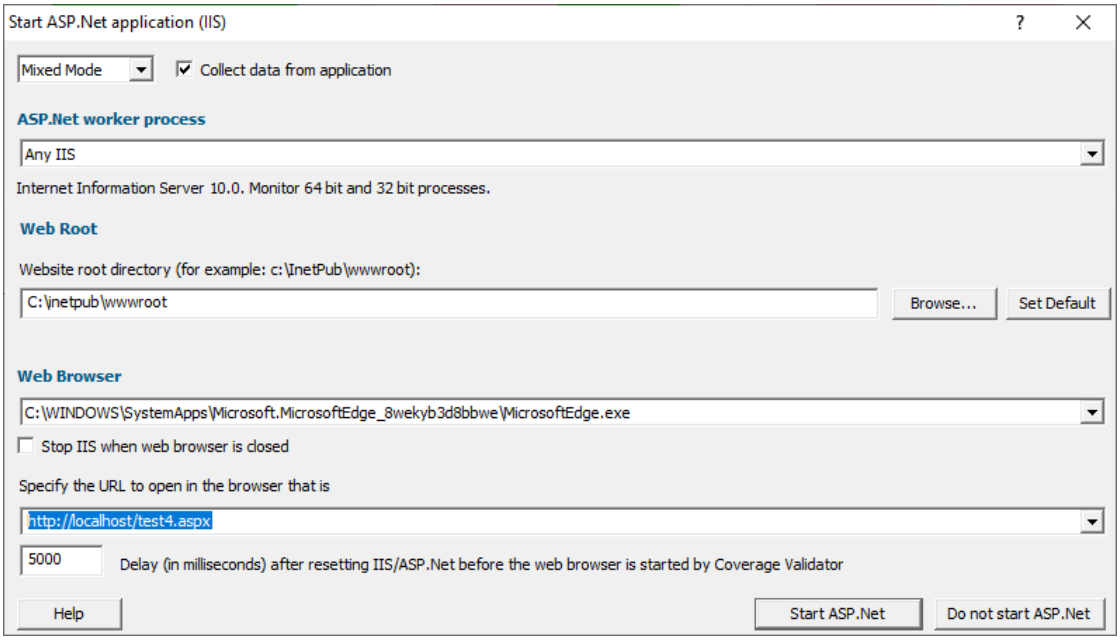
 **Launch** menu › **IIS** menu › **Start ASP.Net Application...** › shows the Start ASP.Net Application dialog below

Or use the shortcut

 Monitor IIS and ASP.Net

The Start ASP.Net application dialog

Set the asp worker process, the web root, an optional web browser to use and an optional url to launch, and click OK.



- **ASP.Net worker process** › select the IIS process we're working with. This can be any ASP.Net process or a specific one. The default is **Any IIS**.
- **IIS web** › type or **Browse** to set the web root for the IIS website we're working with
- **Web Browser** › select the web browser that you're going to use to load the web page
- **URL to open in browser** › type the web page and arguments you want to load to cause the ISAPI to be loaded in IIS
- **OK** › resets IIS, setups all the variables, copies DLLs and settings into the web root and starts the web browser to load the specified web page

IIS is a protected process and can only execute, read and write files in specific directories. That's why Coverage Validator copies data to the web root so that it can be read, written or executed.

Data collection

- **Type of data collection** › Are you only interested in Native data, .Net data or both Native data and .Net data?
  - **Native Only** › Ignore all .Net data in the target application.
  - **.Net Only** › Ignore all Native data in the target application.
  - **Mixed Mode** › Collect both Native and .Net data from the target application

This setting cannot be changed after the application is launched

- **Collect data from application** > If it's the startup procedure you want to validate, obviously start collecting data from launch.

  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.

  ➔ The data collection option may be disabled because of the instrumentation mode that is selected.

Slow Startup

The first time you work with IIS and Coverage Validator you may experience a delay during startup. This is most like because symbols are being downloaded from Microsoft's symbol servers to match the DLLs and assemblies on your machine.

3.16.8.3 Reset & Stop IIS

Reseting IIS

Launch menu > **IIS** menu > **Reset IIS** > resets Internet Information Server (stops it and starts it again).

The session is not discarded when IIS is reset.

Stopping IIS

Launch menu > **IIS** menu > **Stop IIS** > stops Internet Information Server.

The session is not discarded when IIS is stopped.

3.16.9 Web Development Server

Enter topic text here.

3.16.9.1 Monitor Web Development Server and ASP.Net

Monitoring ASP.Net Application (WDS)

To start monitoring ASP.Net application running in IIS:

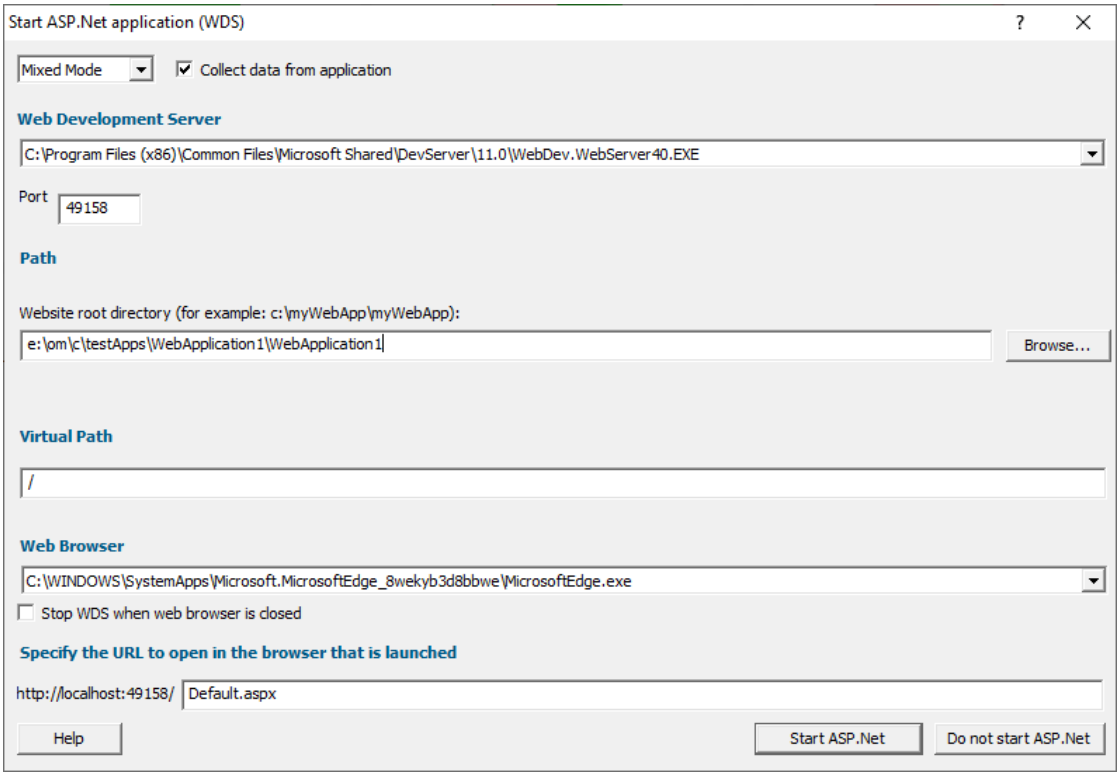
Launch menu › **Web Development Server** menu › **Start ASP.Net Application...** › shows the Start ASP.Net Application dialog below

Or use the shortcut

F9 Monitor Web Development Server and ASP.Net

The Start ASP.Net application dialog

Set the web development server, the port to use, path to the web application, virtual path, an optional web browser to use and an optional url to launch, and click OK.



- **Web Development Server** › select the WDS process we're working with. The default is the most recent version installed on the computer.
- **Port** › select the port that the server will serve pages on. The default is 49158 (the same value that Visual Studio uses).
- **Path** › type or **Browse** to set the path to the ASP.Net application.
- **Virtual Path** › type the path on the server that corresponds to the web application. The default is /.

- **Web Browser** › select the web browser that you're going to use to load the web page.
- **URL to open in browser** › type the web page and arguments you want to load to cause the ISAPI to be loaded in IIS.
- **OK** › resets IIS, setups all the variables, copies DLLs and settings into the web root and starts the web browser to load the specified web page.

Web Development Server is not a protected process like IIS. This can make working with WDS much easier than working with IIS.

Data collection

- **Type of data collection** › Are you only interested in Native data, .Net data or both Native data and .Net data?
  - **Native Only** › Ignore all .Net data in the target application.
  - **.Net Only** › Ignore all Native data in the target application.
  - **Mixed Mode** › Collect both Native and .Net data from the target application

  This setting cannot be changed after the application is launched
- **Collect data from application** › If it's the startup procedure you want to validate, obviously start collecting data from launch.

  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.

  ➡ The data collection option may be disabled because of the instrumentation mode that is selected.

Slow Startup

The first time you work with Web Development Server and Coverage Validator you may experience a delay during startup. This is most like because symbols are being downloaded from Microsoft's symbol servers to match the DLLs and assemblies on your machine.

3.16.9.2 Stop Web Development Server

Stopping Web Development Server

Launch menu > **Web Development Server** menu > **Stop Web Development Server** > stops Web Development Server.

The session is not discarded when Web Development Server is stopped.

3.16.10 ASP.Net Core Web Application

Enter topic text here.

3.16.10.1 Start ASP.Net Core Web Application

Monitoring ASP.Net.Core Web Application

To start monitoring ASP.Net application running in IIS:

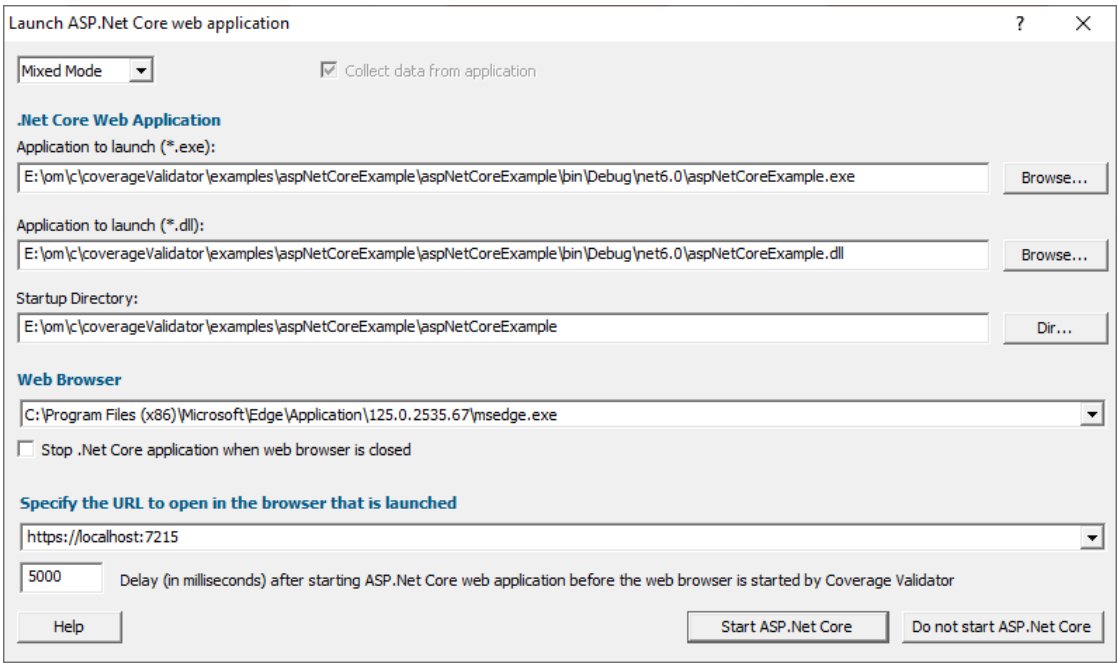
Launch menu > **Web** menu > **ASP.Net Core Web Application...** > shows the Start ASP.Net Core Web Application dialog below

Or use the shortcut

F7 Start ASP.Net Core Web Application

The Start ASP.Net Core application dialog

Set the web development server, the port to use, path to the web application, virtual path, an optional web browser to use and an optional url to launch, and click OK.



- **.Net Core Web Application (exe)** › select your ASP.Net Core web application to launch.
- **.Net Core Web Application (dll)** › select your ASP.Net Core web application to launch.
- **Startup Directory** › type or **Dir...** to set the path to the ASP.Net application.

  This value will be auto-populated based on the path you specify for your application.
- **Web Browser** › select the web browser that you're going to use to load the web page.
- **URL to open in browser** › type the web page, port and arguments you want to load to cause the ISAPI to be loaded in IIS.

  This value will be auto-populated based on the path you specify for your application.

  Example: https://localhost:7215
- **Start ASP.Net Core** › starts your ASP.Net web application, then starts the web browser to load the specified web page.

Data collection

- **Type of data collection** › Are you only interested in Native data, .Net data or both Native data and .Net data?
  - **Native Only** › Ignore all .Net data in the target application.
  - **.Net Only** › Ignore all Native data in the target application.

- **Mixed Mode** › Collect both Native and .Net data from the target application

This setting cannot be changed after the application is launched

- **Collect data from application** › If it's the startup procedure you want to validate, obviously start collecting data from launch.

  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.

  ➡ The data collection option may be disabled because of the instrumentation mode that is selected.

Slow Startup

The first time you work with ASP.Net Core and Coverage Validator you may experience a delay during startup. This is most like because symbols are being downloaded from Microsoft's symbol servers to match the DLLs and assemblies on your machine.

3.16.10.2 Stop ASP.Net Core Web Application

Stopping ASP.Net Core Web Application

Launch menu › **Web** menu › **Stop ASP.Net Core Web Application** › stops ASP.Net Core web application.

The session is not discarded when the ASP.Net Core web application is stopped.

3.16.11 Linking to a program

Why link Coverage Validator into your program?

There are cases when you might need to link Coverage 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 work 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:

- `svlCoverageValidatorStubLib.lib` for 32 bit
- `svlCoverageValidatorStubLib_x64.lib` for 64 bit

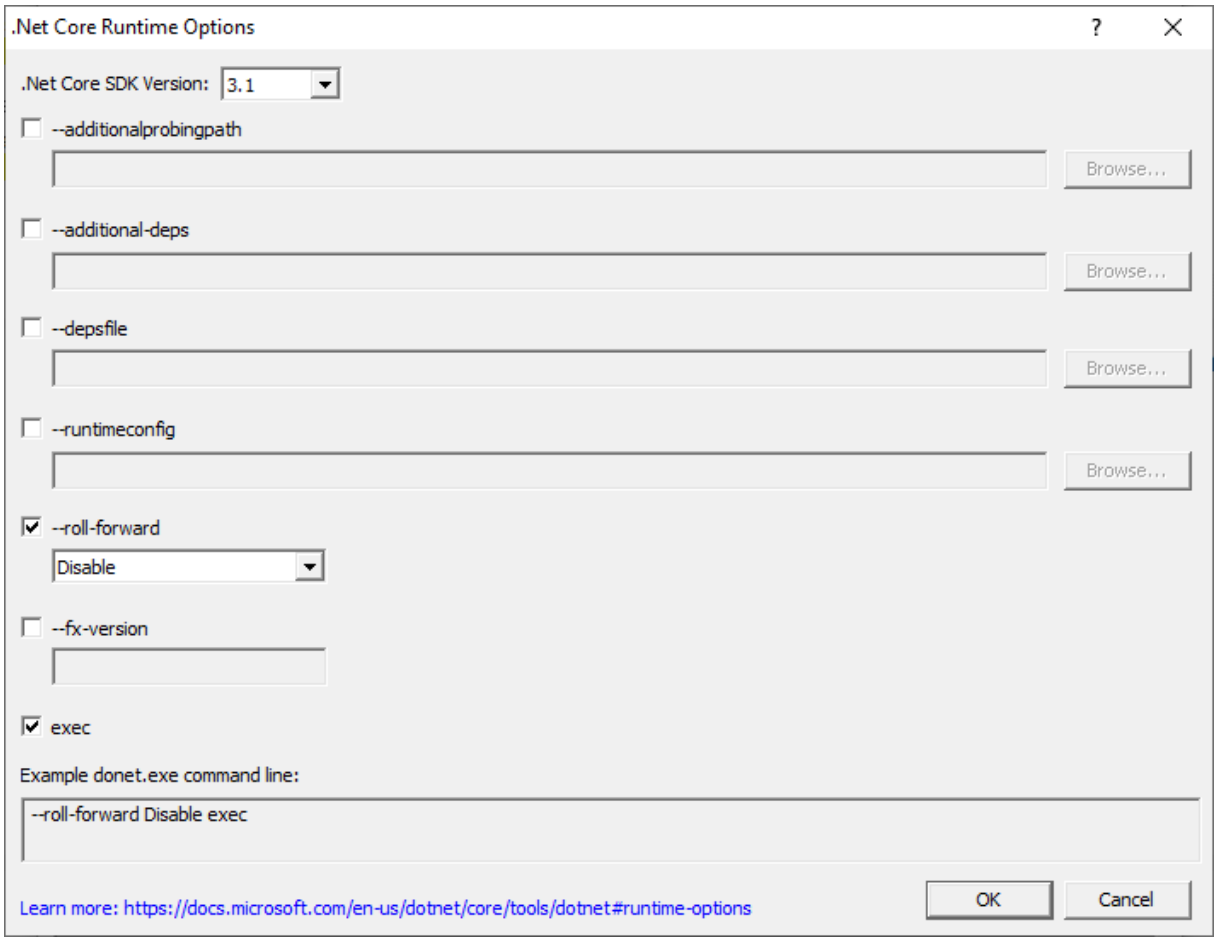
When linked and started, your program will automatically start Coverage Validator.

 The libraries should be linked to your program's **.exe**, not to a DLL that is loaded into your program.

3.16.12 .Net Core Runtime Arguments Editor

The .Net Core runtime arguments editor allows you to choose which options you pass to the .Net Core runtime.

Typically no arguments or just "exec" are passed to the runtime, assuming everything that is needed is in the same directory.



Enabling each check box will add the appropriate option to the runtime arguments.

The field can populated using the **Browse...** button to display a file or folder browser, or edited directly if you wish to add more than one path.

Some fields change depending on the SDK version chosen.

- **--additionalprobingpath** › path containing probing policy and assemblies to probe
- **--additional-deps** › path to an additional .deps.json file
- **--depsfile** › path to the deps.json file.
- **--runtimeconfig** › path to a runtimeconfig.json file
- **--roll-forward** › how to apply roll forward for .Net Core SDK 3.0 and above
- **--roll-forward-on-no-candidate-fx** › how to apply roll forward for .Net Core SDK 2.x
- **--fx-version** › version of .Net runtime to use to run the application
- **exec** › adds the "exec" keyword at the end of the arguments list

If you don't know what these options are you should read the .Net runtime options document shown in the link below before changing any of them.

We cannot advise you on how to set these values - except to say that if you're not setting them when using .Net Core outside of this software tool you shouldn't be setting them when using this software tool.

.Net Core Runtime Options

.Net Core provides some options that allow you to control how .Net Core performs.

Detailed information on these options is provided by Microsoft at https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet#runtime-options.

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 Coverage 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.

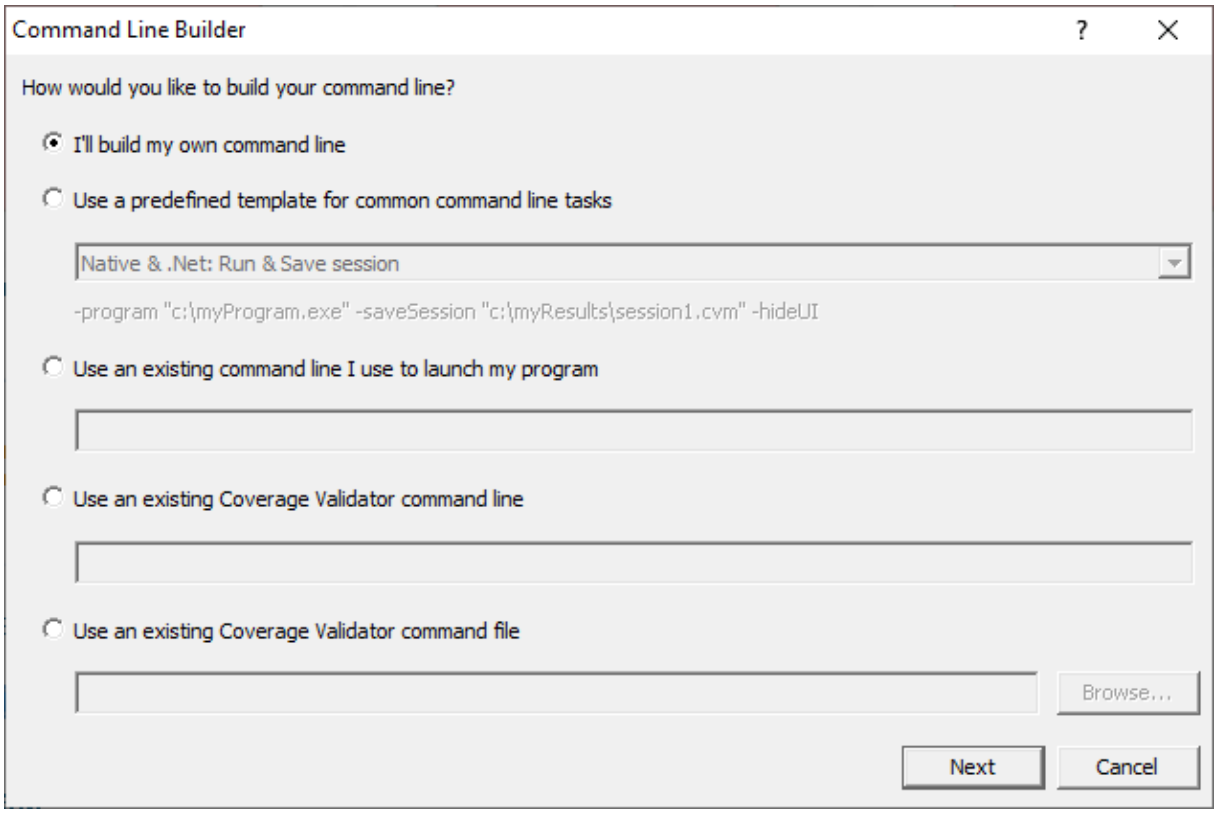
Since the session is discarded, using Coverage Validator to stop the target program is usually quicker and more convenient than external stop methods.

➡ You can easily re-launch the program again using the same settings as before.

3.18 Command Line Builder

The command line builder helps you create command lines with valid options.

The command line builder is a two stage process, the first stage helping your choose how you want to build the command line, and the second stage actually building the command line based on the choices in the first stage.



There are five options for building your command line:

- **I'll build my own** › you'll build your command line from scratch, with no predefined options
- **Use a predefined template** › choose from a list of predefined command lines that you can customize

The predefined templates cover a range of tasks you may want to perform from the command line. These include running sessions, saving sessions, exporting to HTML, XML, Cobertura, and merging code coverage data.

Examples are provided for both Native and .Net applications, and .Net Core applications.

- **Use an existing command line** › use the command line you use to start the tool you want to collect code coverage for

Example: `e:\dev\paintpot\release\paintpot.exe e:\testimages\venn.png -invert -mirror -phaseChange`

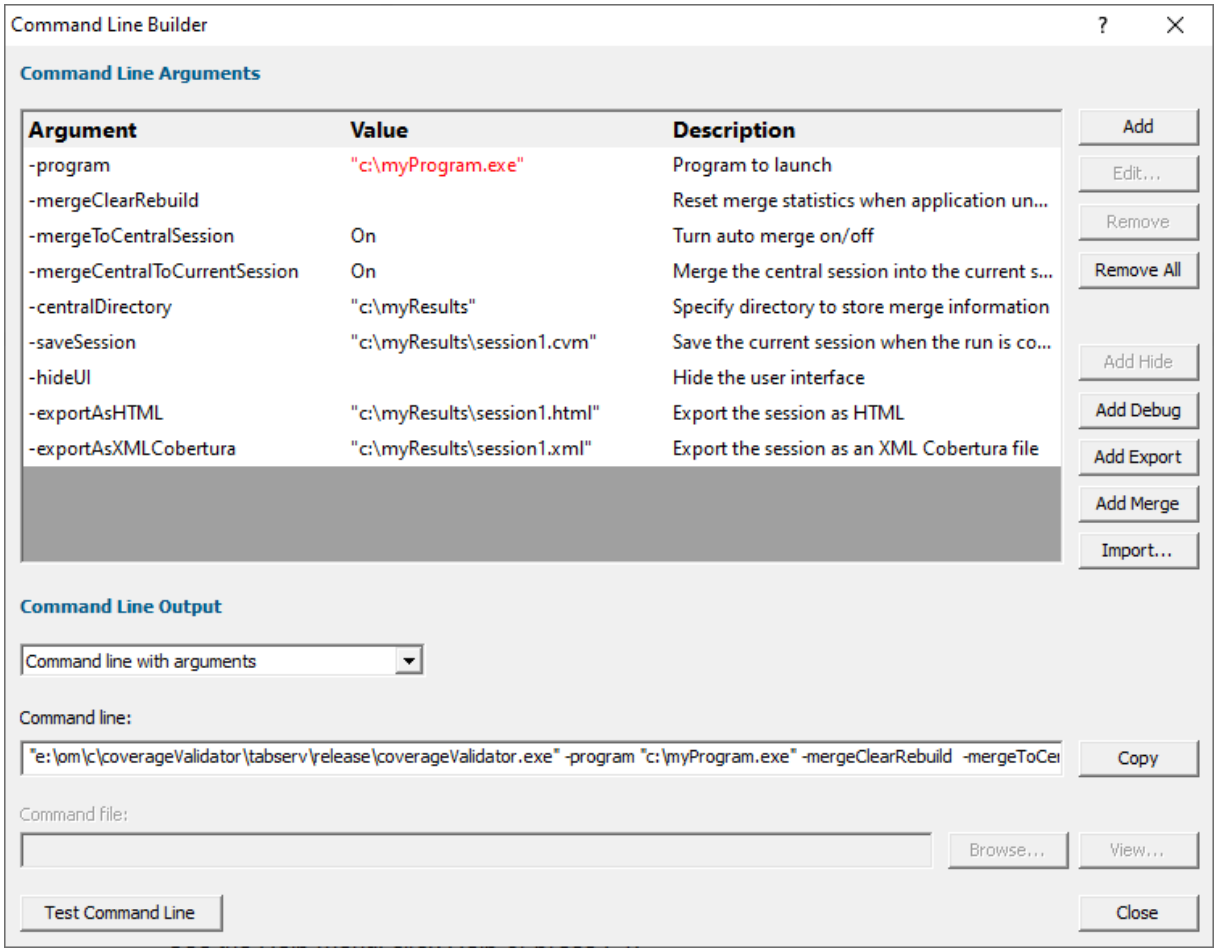
- **Use an existing Coverage Validator command line** › use an existing Coverage Validator command line and customize that

Example: `-program e:\dev\paintpot\release\paintpot.exe -hideUI -exportAsHTML e:\testResults\gannt.html -allArgs e:\testimages\gantt.png -inflate:3`

- **Use an existing Coverage Validator command file** › use an existing Coverage Validator command file and customize that

  Example: `-commandFile e:\commandFiles\paintpot_gantt.cf`

When you have made your choice the **Next** button moves you to the customization part of the command line builder.



The image above shows the command line builder populated with one of the predefined template choices. You can see a few entries refer to directories and files that do not exist on disk (they are red).

These are items you will need to customize to match the program you are testing.

For example, the **-centralDirectory** will need to created before you run the test.

Any entries that will only exist after they have been created by the test will also be shown in red.

Editing

To edit an argument, double click the argument. A combo box will display a list of valid arguments you can choose.

To edit a value, double the value. If the argument type has a list of known values a combo box will be provided, directories will display a directory chooser, files will display a file chooser, numbers will only allow numeric editing. All other values will be edited as text.

- **Add** › add a new argument to the grid
- **Remove** › remove the selected item
- **Remove All** › removes all items in the grid
- **Add Hide** › adds a -hideUI argument which will cause Coverage Validator to hidden when running. Coverage Validator will close after the target program finishes running
- **Add Debug** › adds various arguments which will cause Coverage Validator to display error messages if there are problems with the command line.
- **Add Export** › adds export options that will cause Coverage Validator to export html and/or xml reports after the target program finishes running
- **Add Merge** › adds merge options that will cause Coverage Validator to load a session and and merge sessions after the target program finishes running
- **Import...** › you can import a command file, the contents of which will replace all the items in the grid

Command Line Output

There are two command line output styles.

- **Command line with arguments** › generates a command line containing all arguments and values shown in the grid
- **Command line with command file** › generates a command file containing all arguments and values shown in the grid, and a command line that references the command file

When this option is chosen the command file edit field and the **Browse...** and **View...** buttons are enabled, allowing you to specify a command file name, and to view it's contents.

If a command file has not been specified when this option is selected you will be prompted to select a name for the command file.

When the command file name is selected the command file will be created with the arguments and values shown in the grid.

- **Copy** › copies the command line to the clipboard so that you can paste the command line in cmd prompts, batch files and automated scripts (Jenkins etc)

- **Browse...** › opens a Windows file dialog to allow you to specify the command file location
- **View...** › opens the command file using the Windows shell, this allows you to view the command file in your favourite editor

Testing

If you wish to test the command line, you have two options:

- **Manual test** › use the **Copy** button to copy the command line, then paste it into a cmd prompt and press return.
- **Test Command Line** › a new instance Coverage Validator is started with the specified command line.

3.19 Data Collection

Collecting data

Once you've launched or injected into a program, you can stop and start data collection whilst the program is running.

This is a high level switch that controls *all* data collection, regardless of any other settings.

With data collection off, the target program runs at close to normal speed.

Temporarily turning off collection can be a good idea if you need to take actions to get the program into the right state for validation.

You can also turn data off from the start and only turn it on when you need it.

Starting and stopping data collection

File menu › **Start collecting data...** › starts collecting data immediately

or click on the green icon on the session toolbar to start collecting.

File menu › **Stop collecting data...** › stops collecting

or click on the red icon on the session toolbar to stop.

Collect data from application is disabled. Why?

If the instrumentation detail setting is set to use breakpoints, the collect data from application setting is disabled - data collection is always on - you can't turn it off.

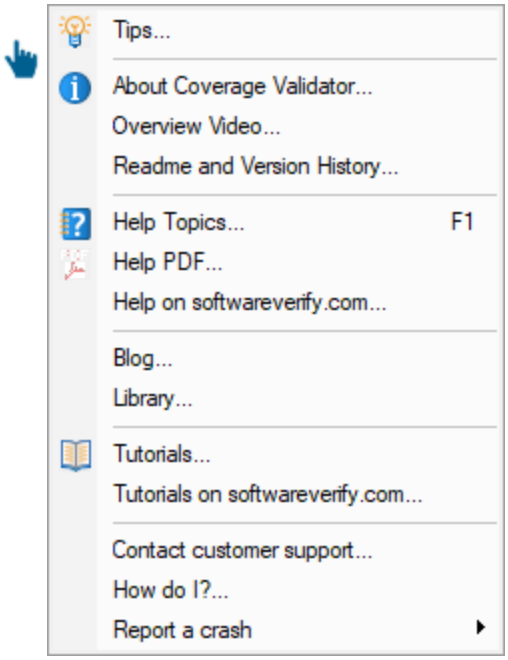
When the ability to change this setting is disabled both of the above icons will be greyed out, and so will any "collect data from application" checkboxes on launch dialogs etc.

3.20 Help

The help menu

The help menu provides access to useful help, tips and tutorials.

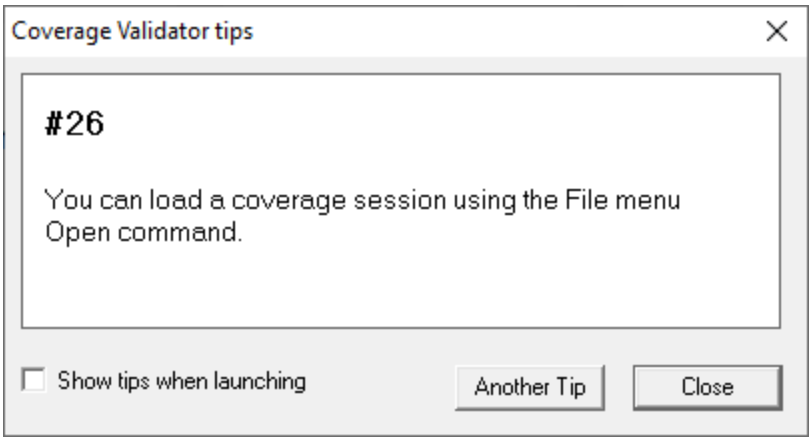
Each item is covered briefly below, in menu order.



Tips

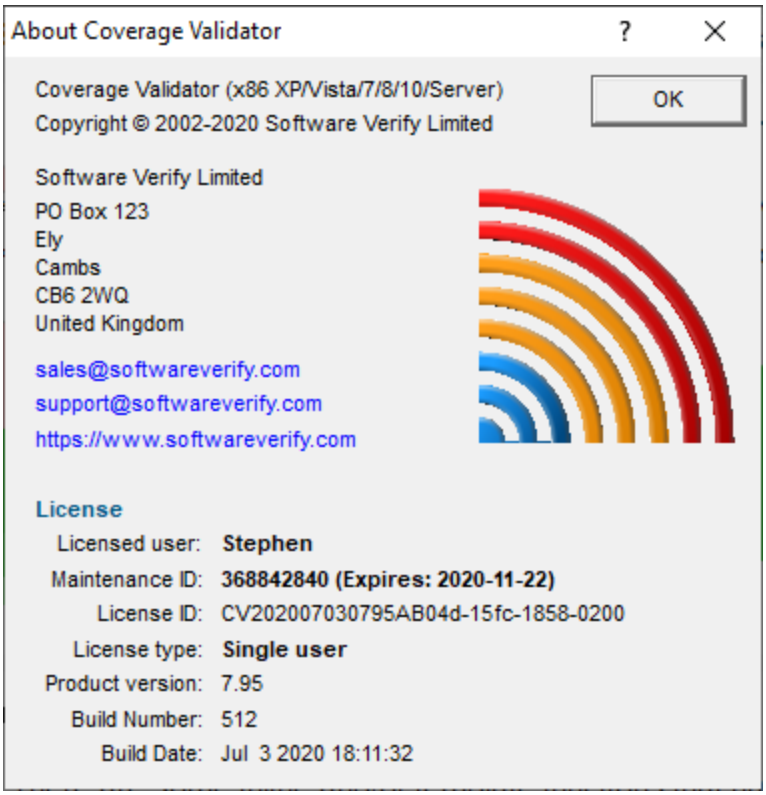
Help menu › **Tips...** › shows the tip dialog where you can browse tips in random order

Here you can also choose whether to display the tips dialog while launching programs.



About box

Help menu › **About Coverage Validator...** › shows contact and copyright information, as well as details of your license

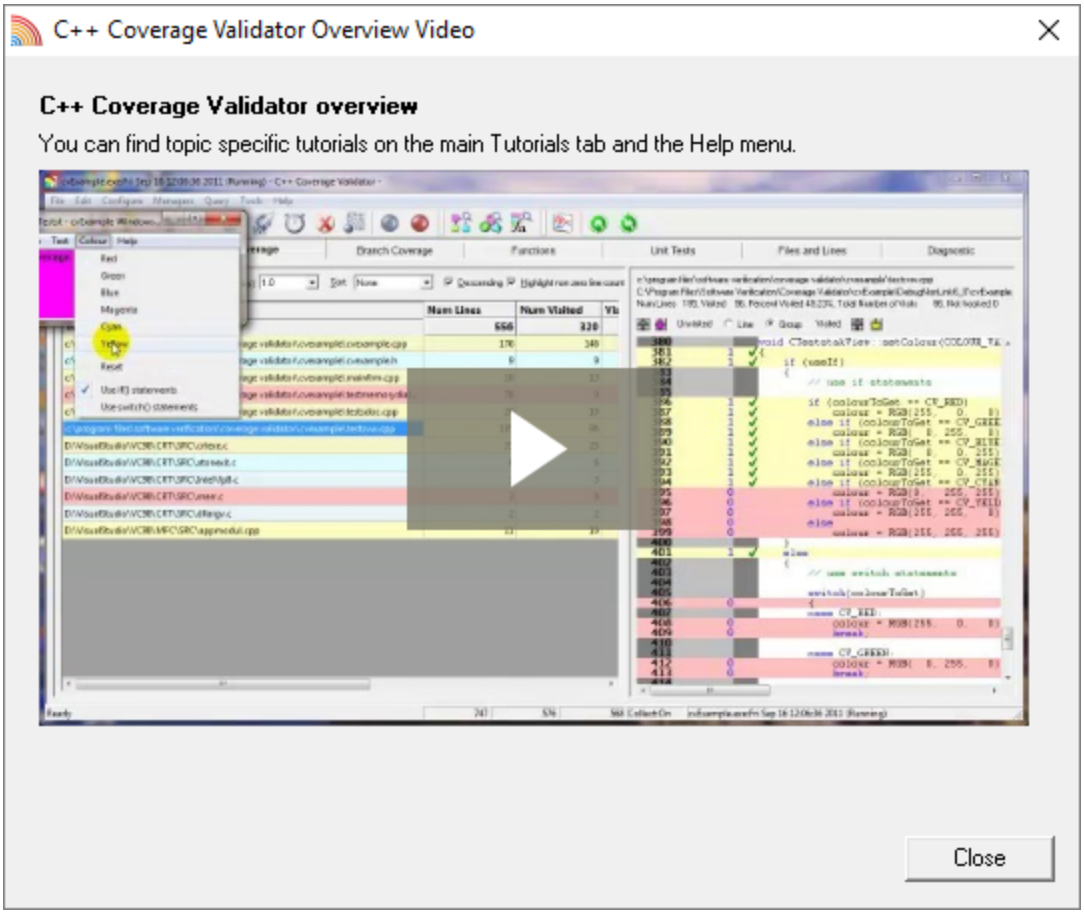


Overview Video

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 Coverage Validator.



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 Coverage Validator including:

- basic information about getting started and where to go for support
- known issues
- version history

To see what's changed since the version you have installed see the latest version history.

Help HTML

Help menu › **Help Topics...** › shows the HTML help dialog

You might be reading this right now!.

Or click on the question mark icon on the standard toolbar:

The F1 key also shows the help, but has the added bonus of jumping directly to the page relevant for the current view or dialog.

We occasionally get reports of customers seeing exception errors while viewing the HTML help. Unfortunately, we don't have a solution for this!

Help PDF

Help menu › **Help PDF** › shows the PDF version of this help

You will need a suitable PDF reader such as Adobe Acrobat Reader, but do beware of unwanted add-on installs.

PDF help for *all* our products are online.

Help on softwareverify.com

Help menu › **Help on softwareverify.com** › shows the online version of this help

Blog

Help menu › **Blog** › shows the Software Verify Blog.

Library

Help menu › **Library** › shows the Software Verify Library - all our best articles organised into related topics for easy access. You will find many articles about getting the most out of Coverage Validator in here.

Tutorials

The tutorials are intended to guide you through learning how to use aspects of Coverage Validator.

All tutorials are available online in the form of short videos and examples covering popular topics.

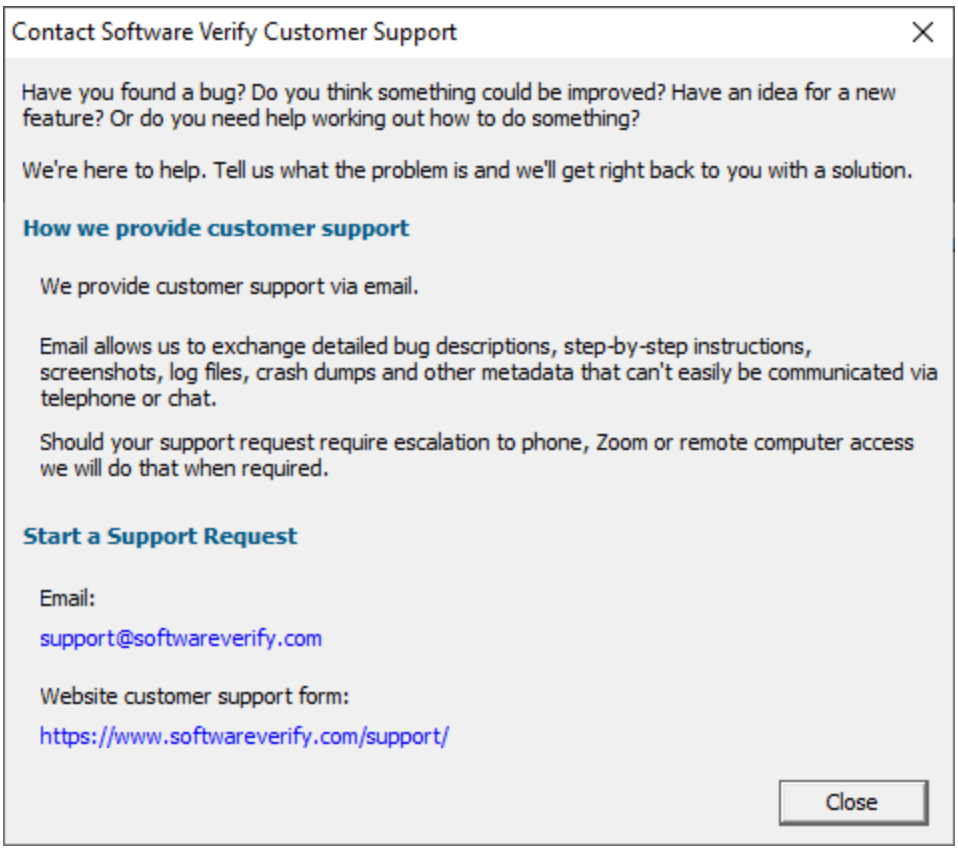
Help menu › **Tutorial...** › simply selects the Tutorial tab to show a list of the tutorials

Double click on the row of a tutorial in the list to open it in a browser.

Help menu › **Tutorials on softwareverify.com...** › opens the online tutorials in a browser

Contact customer support

Help menu › **Contact customer support** › shows the Contact customer support dialog.



How do I?

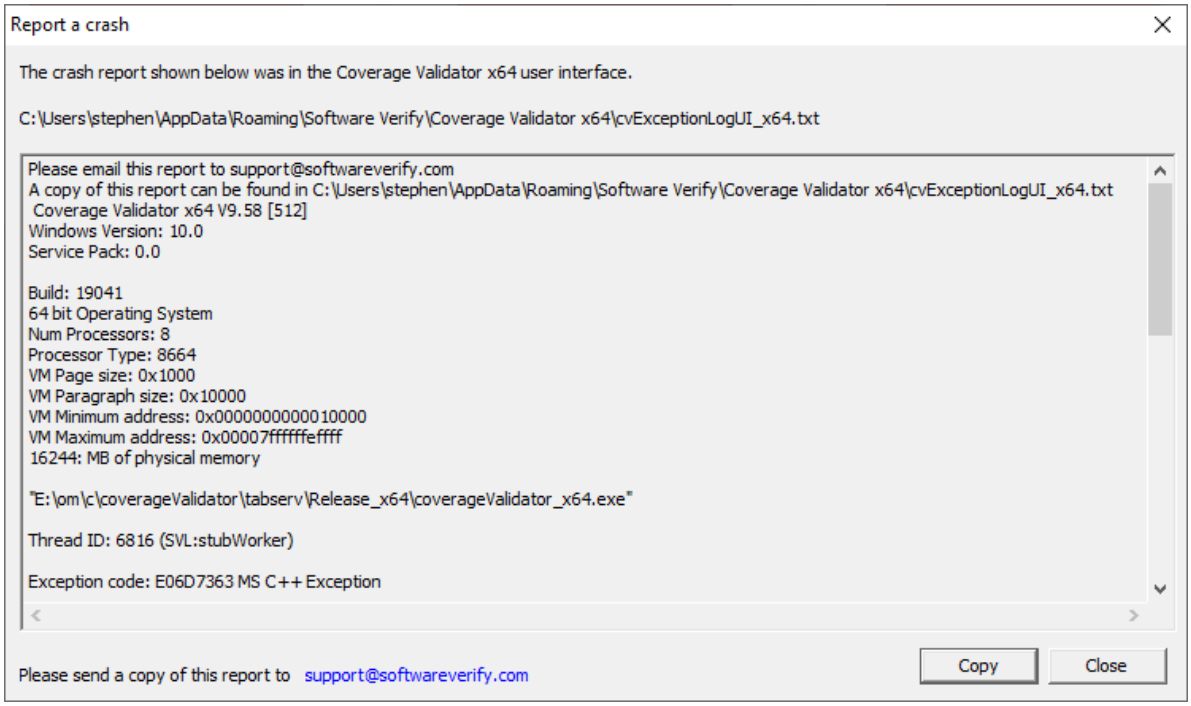
Help menu › **How do I?** › shows the How do I? dialog.

DbgHelp Search Info	DBGHELP: Symbol Search Path: C:\WINDOWS\symbols\dll;C:\Program Files (x86)\Software Verify\Performance Validator x86\examples\nativeExample\Release10_0;
DbgHelp Search Info	DBGHELP: C:\WINDOWS\symbols\dll\nativeExample.pdb - file not found
DbgHelp Search Info	DBGHELP: C:\WINDOWS\symbols\dll\exe\nativeExample.pdb - file not found
DbgHelp Search Info	DBGHELP: C:\WINDOWS\symbols\dll\symbols\exe\nativeExample.pdb - file not found

Report a crash

Help menu › **Report a crash** › displays the options for reporting a crash.

If an exception report for the Coverage Validator user interface, or an exception report for an application that Coverage Validator was monitoring is available it will be displayed with options to copy it to the clipboard and contact customer support at Software Verify.



Part

4 Environment Variables

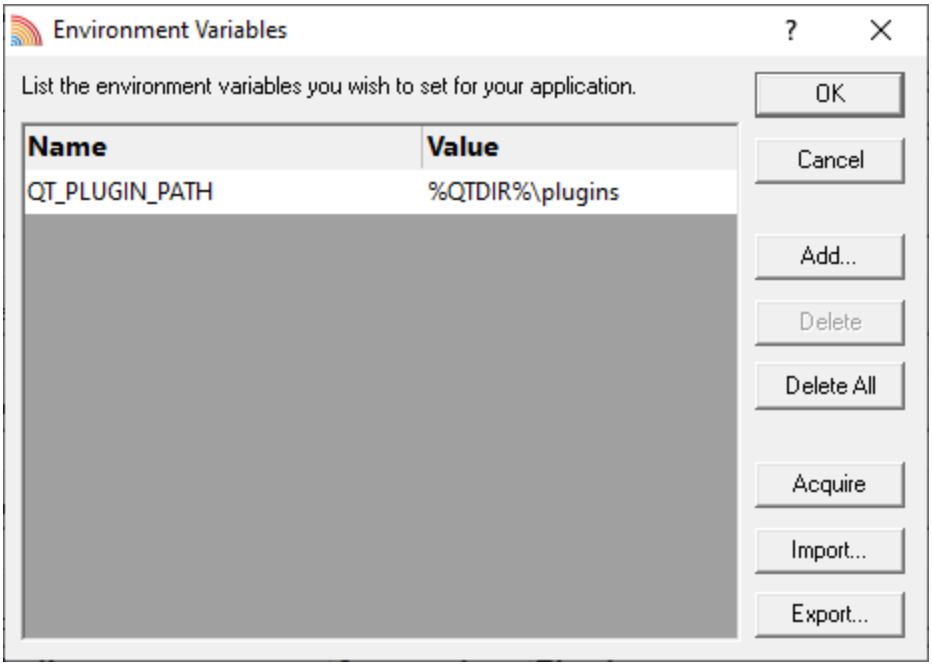
When launching an application, you might want to pass in some environment variables to your program.

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 › enter name in the first column, value in the second
- **Delete** › deletes a selected item in the list
- **Delete All** › clears the list
- **Acquire** › fetches all system environment variables, *adding* them to the list
- **Import...** › loads variables from a previously exported file, *adding* them to the list
- **Export...** › saves all entries in the list to a file of your choice

  The exported file is a simple ascii file with one entry per line of the form `name=value`
- **OK** › accepts all changes

- **Cancel** › ignores changes

Part V

5 Command Line Interface

Coverage Validator provides a command line interface to allow you to perform automated coverage data collection.

To run 32 bit coverage validator run **`C:\Program Files (x86)\Software Verify\Coverage Validator x86\coverageValidator.exe`**

To run 64 bit coverage validator run **`C:\Program Files (x86)\Software Verify\Coverage Validator x64\coverageValidator_x64.exe`**

Automated coverage data collection

Potential uses for automated code coverage analysis are:

- In the regression test suite to ensure code coverage of a certain level
- In unit testing to ensure code coverage of a certain level
- Quality assurance

Results from coverage data collection sessions can be merged to form an aggregate result.

Typically, command line options allow Coverage 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 Coverage Validator would exit between automated tests, but it can be made to stay running if necessary.

➡ See the command line reference for an alphabetical listing all the available commands.

Command line argument usage

There are a few basic rules to remember when using the command line arguments:

- separate arguments by spaces
- quote arguments if they contain spaces
- some arguments are only useful in conjunction with others
- some arguments are incompatible with others

If your command line is very long, consider using **-commandFile** to specify a command file for your arguments.

Unrecognised arguments

Any unrecognised arguments found on the command line are simply ignored, whether or not they are prefixed with a hyphen.

Arguments intended for your program will not conflict with the Coverage Validator arguments in this manual as you should use -arg (or -allArgs) to redirect them to your program.

Need some help building the command line?

If you find creating command lines from nothing to be a bit daunting we've created a Command Line Builder tool to help you build command lines.

You'll still need to complete some details, but the builder will help prevent you making some mistakes.

Command Lines and Floating Licences

Coverage Validator works from the GUI and from the command line with all types of software licence (floating licences and non-floating licences).

When floating licences are being used Coverage Validator will wait to acquire a floating licence before proceeding to process the command line options.

There are no options to:

- Checkout a floating licence
- Release a floating licence
- Query for available licences

These options are managed automatically by Coverage Validator, there is no need for such options to be manually controlled from the command line.

5.1 Example Command Lines

Typical command line examples

The following examples demonstrate a few different scenarios in which you might want to use Coverage Validator via the command line.

To run 32 bit coverage validator run `C:\Program Files (x86)\Software Verify\Coverage Validator x86\coverageValidator.exe`

To run 64 bit coverage validator run `C:\Program Files (x86)\Software Verify\Coverage Validator x64\coverageValidator_x64.exe`

Example 1 - 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 Functions tab being the current tab.

coverageValidator_x64.exe **-program** "c:\myProgram.exe" **-saveSession** "c:\myResults\session1.cvm_x64" **-displayUI -refreshResults -refreshCoverage -refreshFunctions -refreshFilesAndLines**

A brief explanation of each argument:

Option	Argument	Description
-program	"c:\myProgram.exe"	The target program to launch
-saveSession	"c:\myResults\session1.cvm_x64"	After the application finishes, the session should be saved in this file
-displayUI		Show the user interface during the coverage test
-refreshResults		The current directory for the application to work in
-refreshCoverage		All main data tabs should be refreshed when the test completes, including the Summary Results
-refreshFunctions		
-refreshFilesAndLines		

Example 2 - running a session and exiting when the session is complete

This example starts the application, showing no progress dialog whilst attaching to the process.

On completion, the resulting session is saved, and Coverage Validator exits.

coverageValidator_x64.exe **-program** "c:\myProgram.exe" **-saveSession** "c:\myResults\session2.cvm_x64" **-hideUI**

A brief explanation of each argument:

Option	Argument	Description

-program	"c:\myProgram.exe"	The target program to launch
-saveSession	"c:\myResults\session2.cvm_x64"	After the application finishes, the session should be saved in this file
-hideUI		Hide the user interface during the coverage test. Coverage Validator closes when the test application finishes.

Example 3 - running a session (.Net Core, Self Contained)

This example starts a .Net Core 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 Functions tab being the current tab.

coverageValidator_x64.exe **-program** "c:\myDotNetCoreApp.exe" -dotNetCoreLaunchType SelfContained **-saveSession** "c:\myResults\session3.cvm_x64" **-displayUI -refreshCoverage -refreshFunctions -refreshResults**

A brief explanation of each argument:

Option	Argument	Description
-program	"c:\myDotNetCoreApp.exe"	The target program to launch
-dotNetCoreLaunchType	SelfContained	The .Net Core program is self contained
-saveSession	"c:\myResults\session3.cvm_x64"	After the application finishes, the session should be saved in this file
-displayUI		Show the user interface during the performance test
-refreshCoverage		Main data tabs should be refreshed when the test completes, including the Summary Results
-refreshFunctions		
-refreshResults		Coverage Validator will be left open at the Summary Results

Example 4 - running a session (.Net Core, Framework Dependent)

This example starts a .Net Core 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 Functions tab being the current tab.

coverageValidator_x64.exe **-program** "c:\dotNetCoreApp.dll" -dotNetCoreLaunchType FrameworkDependent **-saveSession** "c:\myResults\session4.cvm_x64" **-displayUI** -**refreshCoverage** -**refreshFunctions** -**refreshResults**

A brief explanation of each argument:

Option	Argument	Description
-program	"c:\dotNetCoreApp.dll"	The target program to launch with the .Net runtime
-dotNetCoreLaunchType	FrameworkDependent	The .Net Core program is framework dependent
-saveSession	"c:\myResults\session4.cvm_x64"	After the application finishes, the session should be saved in this file
-displayUI		Show the user interface during the performance test
-refreshCoverage		Main data tabs should be refreshed when the test completes, including the Summary Results
-refreshFunctions		
-refreshResults		Coverage Validator will be left open at the Summary Results

Example 5 - running and merging a session

Add the following to the first example to load a previous session, and after the application has finished and this session saved, merge in the loaded session, and save the combined results:

coverageValidator_x64.exe **-program** "c:\myProgram.exe" **-directory** "c:\testarea" **-loadSession** "c:\myResults\session1.cvm_x64" **-mergeSessions** **-saveMergeResult** "c:\myResults\session5.cvm_x64"

Option	Argument	Description
-program	"c:\myProgram.exe"	The target program to launch

-directory	"c:\testarea"	Set the working directory in which the program is executed
-loadSession	"c:\myResults\session1.cvm_x64"	Loads this previously saved session into the session manager
-mergeSessions		The loaded session is to be merged with the newly recorded session recorded *after* the application has terminated
-saveMergeResult	"c:\myResults\session5.cvm_x64"	The merged session should be saved here

Example 6 - merging previous results

This example loads two previously saved sessions and merges them without any user intervention, saving the merged session as a new session.

coverageValidator_x64.exe **-hideUI -loadSession** "c:\myResults\session1.cvm" **-loadSession2** "c:\myResults\session2.cvm_x64" **-mergeSessions -saveMergeResult** "c:\myResults\session1+2.cvm_x64"

Option	Argument	Description
-hideUI		The user interface should not be shown during the test
-loadSession	"c:\myResults\session1.cvm_x64"	Loads this previously saved session into the session manager
-loadSession2	"c:\myResults\session2.cvm_x64"	Also loads this session
-mergeSessions		The loaded session is to be merged with the newly recorded session recorded *after* the application has terminated
-saveMergeResult	"c:\myResults\session1+2.cvm_x64"	The merged session should be saved here

Example 7 - merging multiple previous results

This example loads two previously saved sessions and merges them without any user intervention, saving the merged session as a new session.

coverageValidator_x64.exe **-hideUI -mergeSessions -mergeMultiple** e:\cv_merge_multiple.txt **-saveMergeResult** e:\cv_merge_result.cvm_x64

Option	Argument	Description
-hideUI		The user interface should not be shown during the test
-mergeSessions		Merge all sessions
-mergeMultiple	e:\cv_merge_multiple.txt	Loads all sessions identified by the filenames listed in this file, one filename per line.
-saveMergeResult	e:\cv_merge_result.cvm_x64	Save the merged sessions in this new session

Example 8 - merging runs into a central session

This example profiles two different applications (each of which is a unit test) merging them into a central session.

It demonstrates the use of **-mergeClearNone**, **-mergeUsingSymbol** and relative file specifications (rather than explicit full path file specifications).

Because the two applications are different we must prevent the merge from resetting its data when the application changes and when the application timestamp changes. We do this with the **-mergeClearNone** option.

There is also a very real chance that symbols from each application will use conflicting addresses, thus we must merge using symbol names, not addresses. We do this using the **-mergeUsingSymbol** option.

First unit test:

coverageValidator_x64.exe **-resetSettings -showErrorsWithMessageBox -mergeUsingSymbol -mergeClearNone -numSessions** 2 **-hideUI -program** .\bin\tstdate.exe **-mergeToCentralSession**:On **-centralFileName** .\temp\cvmerge.cvm_x64 **-saveSession** .\temp\libdate.cvm_x64 **-sourceFileFilterHookFile** .\cv\libdate.srchook **-dllHookFile** .\cv\libdate.dllHook

Second unit test, with identical arguments in grey:

coverageValidator_x64.exe **-resetSettings -showErrorsWithMessageBox -mergeUsingSymbol -mergeClearNone -numSessions** 2 **-hideUI -program** .\bin\tsttime.exe **-mergeToCentralSession**:On **-centralFileName** .\temp\cvmerge.cvm **-saveSession** .\temp\libtime.cvm_x64 **-sourceFileFilterHookFile** .\cv\libdate.srchook **-dllHookFile** .\cv\libdate.dllHook

Each command line is essentially the same, but runs a different application and saves a different session.

You can add more unit tests by running more command lines.

Because of the repeatable nature of the command line this is easily scriptable.

A brief explanation of the arguments not already covered above:

Option	Argument	Description
-resetSettings		Resets the settings to the default state, providing a known starting point to modify the settings from. An alternative would be to use **-loadSettings** to load a known set of settings
-showErrorsWithMessageBox		A message box is displayed if any errors are encountered when processing options such as **-dllHookFile** and **-sourceFileFilterHookFile**
-mergeUsingSymbol		Merges sessions by symbol, rather than by address
-mergeClearNone		Prevents merged data in the central session from being reset, e.g. because the application has changed or because the timestamp has changed
-numSessions		Allows the session manager to load 2 sessions, ensuring there is enough workspace to hold the sessions in use
-mergeToCentralSession	:On	Indicates that code coverage sessions should be merged into a central session
-centralFileName	.\temp\cvmerge.cvm_x64	The central session location in the **temp** sub directory of the current directory
-sourceFileFilterHookFile	.\cv\libdate.srchook	A file listing source files that should be processed for code coverage
-dllHookFile	.\cv\libdate.dllHook	A file listing the DLLs that should be processed for code coverage

Example 9 - using command files

Functionally this example does the same as the previous example, but uses command files to achieve the same result

The following code would be placed in a `.bat` or `.cmd` file:

```
@echo off
del /Q .\temp\*.*
"C:\Program Files (x86)\Software Verify\Coverage Validator\coverageValidator.exe" -commandFile
"C:\Program Files (x86)\Software Verify\Coverage Validator\coverageValidator.exe" -commandFile
"C:\Program Files (x86)\Software Verify\Coverage Validator\coverageValidator.exe" -commandFile
```

The code empties the local results 'temp' directory and then runs two tests as above, before finally running CoverageValidator once more to create some HTML reports.

The batch file assumes it will be run from the directory that has the `bin`, `cv`, and `temp` sub-directories with the appropriate data in them.

The contents of the `.cf` files in this example would be as follows:

libdate.cf

```
-resetSettings
-showErrorsWithMessageBox
-mergeUsingSymbol
-mergeClearNone
-numSessions 2
-hideUI
-program .\bin\tstdate.exe
-dllHookFile .\cv\libdate.dllhook
-sourceFileFilterHookFile .\cv\libdate.srchook
-mergeToCentralSession:On
-centralFileName .\temp\cvmerge.cvm_x64
-saveSession .\temp\libdate.cvm_x64
```

libtime.cf

```
-resetSettings
-showErrorsWithMessageBox
-mergeUsingSymbol
-mergeClearNone
-numSessions 2
-hideUI
-program .\bin\tsttime.exe
-dllHookFile .\cv\libtime.dllhook
-sourceFileFilterHookFile .\cv\libtime.srchook
-mergeToCentralSession:On
-centralFileName .\temp\cvmerge.cvm_x64
-saveSession .\temp\libtime.cvm_x64
```

export.cf

```
-resetSettings
-loadSession .\temp\cvmerge.cvm_x64
-hideUI
-exportAsHTML .\temp\cvreport.html
-exportDescription "Time and date tests"
-exportDetailedReport:On
-exportDoColourCode:On
-exportSourceCode:On
-exportType SummaryAndCoverage
-exportUnhooked:On
-exportUnvisitedFunctions:On
-exportUnvisitedLines:On
-exportVisitedFunctions:On
-exportVisitedLines:On
-showMergeWithReport none
```

Example 10 - monitoring a process that restarts itself

This example shows you how to collect code coverage data for a process that restarts itself. Because it restarts itself it will create multiple coverage sessions - to deal with this we use the central session to collect all the different coverage data into one session.

coverageValidator_x64.exe **-waitName "E:\test\myProgram.exe" -hideUI -monitorAllRestarts -maxNumRestarts 3 -mergeUsingSymbol -mergeClearNone -mergeToCentralSession:On -centralFileName e:\cvmerge.cvm_x64**

Option	Argument	Description
-waitName	"E:\test\myProgram.exe"	Wait for this program to start
-hideUI		The user interface should not be shown during the test
-monitorAllRestarts		Monitor this program when it restarts as well as the first time it starts.
-maxNumRestarts	3	When the program has been restarted 3 times and then shutdown, close Coverage Validator.
-mergeUsingSymbol		Merges sessions by symbol, rather than by address
-mergeClearNone		Prevents merged data in the central session from being reset, e.g. because the application has changed or because the timestamp has changed
-mergeToCentralSession	:On	Indicates that code coverage sessions should be merged into a central session

-centralFileName	e:\cvmerge.cvm_x64	The central session location in the root directory of e: drive.

5.2 Environment variables

Environment variables can be referenced on the command line.

This allows you to set an environment variable outside of Coverage Validator (cmd prompt, batch file, etc) and then reference it on the command line.

For example:

```
-program %BUILD_DIR%\testProgram.exe
```

If the BUILD_DIR environment variable is set to e:\dev\debug the above would evaluate to `-program e:\dev\debug\testProgram.exe`

What if I can't set an environment variable?

There are situations where it isn't desirable, or possible to set the environment variable value prior to starting Coverage Validator.

In those situations you can set the environment variable on the command line using -setenvironment.

```
-setenvironment BUILD_DIR=e:\dev\debug -program %BUILD_DIR%\testProgram.exe
```

Problems with environment variable substitution

If you are running from a command prompt, or batch file, or any process that will handle environment variable substitution using %ENV_VAR% you will find that referencing the environment variable on the command line won't work when using -setenvironment, because by the time Coverage Validator sees the command line the %ENV_VAR% values have already been substituted.

To get around this, using ENV_VAR instead of %ENV_VAR%.

```
-setenvironment BUILD_DIR=e:\dev\debug -program $BUILD_DIR$\testProgram.exe
```

-setenvironment

Set environment variables for Coverage Validator, as a series of name/value pairs.

Use this option once for each environment variable you wish to set.

Usage of **-setenvironment** for any environment variable must appear on the command line prior to any reference to that environment variable on the command line.

To pass quotes along with the string, escape a pair of inner quotes like the example below

Examples:

-setenvironment APP_FLAG=ON;
-setenvironment "APP_FAG=ON;"
-setenvironment "APP_COMMS=ON; APP_DEBUG=OFF;"
-setenvironment "APP_MSG=\"A quoted string with spaces\";"
-setenvironment BUILD_DIR=e:\dev\debug

Note that this is not the same as -environment, which allows you to specify environment values that you can pass to the program being launched.

5.3 Target Program & Start Modes

Resetting global settings

-resetSettings

Forces Coverage Validator to reset all settings to the default state, except for any configured colours and the UI Global Hook settings which must be reset manually.

If using this option, it's recommended that you list this *first* on your command line or in your command file.

Specifying the target application

The following options let you launch a program (with various start-up modes), inject into a running program or wait for a program to start before attaching.

Launching a program

-program

Specifies the full file system path of the executable target program to be started by Coverage 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.

See -programToMonitor to monitor a different program than the one you initially launch.

Examples:

-program c:\testbed.exe
-program "c:\new compiler\version2\testbed.exe"

If you specify the file *without* a path then:

- If you used -directory to set a startup directory then the filename in that directory is used if it exists
- Otherwise, the directories in the PATH environment variable are used to look for the filename

-programToMonitorEXE
-programToMonitor

-programToMonitor has been replaced by **-programToMonitorEXE**. **-programToMonitor** will be honoured to provided backward compatibility.

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

This program will be monitored by Coverage Validator only when the program specified using -program starts it.

If no path is specified, the first child process that has the same name will be monitored.

To monitor *any* program that is launched specify `<<Any>>` as the program argument. In batch files and powershell scripts you will need to quote this to get it accepted by the file parser.

See -programToMonitorLaunchCount to change which instance of the program is monitored.

Only valid in conjunction with -program.

Examples:

-programToMonitorEXE c:\testbed-child-process.exe
-programToMonitorEXE "c:\new compiler\version2\testbedChildProcess.exe"
-programToMonitorEXE testbed-child-process.exe
-programToMonitorEXE "<<Any>>"

-program c:\testbed.exe **-programToMonitorEXE** c:\testbed-child-process.exe

In this last example `c:\testbed.exe` is launched but not monitored. Only when `testbed.exe` launches a child process `c:\testbed-child-process.exe` is that child process monitored.

-programToMonitorDLL

This option provides a qualifying DLL to identify different .Net Core processes, which are typically launched using the same .Net Core runtime. *Include the dll extension*.

Only valid in conjunction with -program and -programToMonitorEXE.

Examples:

-programToMonitorDLL c:\test\dotNetCoreApp.dll

-program c:\testbed.exe **-programToMonitorEXE "**c:\program files\dotnet\dotnet.exe" **-programToMonitorDLL** c:\test\dotNetCoreApp.dll

In this last example `c:\testbed.exe` is launched but not monitored. Only when `testbed.exe` launches a child process `c:\program files\dotnet\dotnet.exe` to run the application `c:\test\dotNetCoreApp.dll` is that child process monitored.

-programToMonitorLaunchCount

Specify the n^{th} invocation of the 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 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.

-arg

Passes the following element on the command line to the target program.

-arg can be used multiple times, or you can use -allArgs

 To pass quotes along with the string, escape a pair of inner quotes like the example below

Only valid with: -program

Examples:

-arg myProgram.exe
-arg "c:\Program Files\myApp\myProgram.exe"
-arg "-in infile -out outfile"
-arg "\"a quoted string\""

-allArgs

Passes the remainder of the command line (after -allArgs) to the program being launched.

Unlike -arg above, there is no need to escape the quotes as the content is passed verbatim.

Only valid with: -program

Example:

-allArgs anything put here is passed to the target program "even stuff in quotes" is passed

-directory

Sets the working directory in which the program is executed. If -directory is not specified the program is run in its current directory.

Only valid with: -program

Examples:

-directory c:\development\
-directory "c:\research and development\"

-environment

Environment variables for program, as a series of name/value pairs. Not to be confused with -setenvironment.

Use this option once for each environment variable you wish to set.

 To pass quotes along with the string, escape a pair of inner quotes like the example below

Only valid with: -program

Examples:

-environment APP_FLAG=ON;
-environment "APP_FAG=ON;"
-environment "APP_COMMS=ON; APP_DEBUG=OFF;"
-environment "APP_MSG=\"A quoted string with spaces\";"

-dataCollectType

Specifies the type of data collection that you want. Native, .Net or mixed mode (both native and .Net).

Valid with -program and -monitorAService.

Examples:

-dataCollectType native

-dataCollectType dotNet
-dataCollectType mixedMode

-stdin

Specifies a file to be read and piped to the standard input of the application being tested.

If the filename contains spaces, the filename should be quoted.

An error occurs if the file does not exist. See **-ignoreMissingStdin** to avoid this error.

Examples:

-stdin c:\settings\input.txt
-stdin "c:\reg tests settings\input.txt"

-stdout

Specifies a file to be written with data piped from the standard output of the application being tested.

If the filename contains spaces, the filename should be quoted.

An error occurs if the file does not exist. See **-ignoreMissingStdout** to avoid this error.

Examples:

-stdout c:\settings\output.txt
-stdout "c:\reg tests results\output.txt"

-ignoreMissingStdin

If this flag is specified and the file specified by **-stdin** does not exist, no error is reported.

-ignoreMissingStdout

If this flag is specified and the file specified by **-stdout** does not exist, no error is reported.

Target program startup modes

-createProcessStartupThread
-normalStartupThread
-idleStartupThread
-suspendStartupThread
-pauseStartupThread
-noSuspendInStubDuringAttach

All these options are obsolete and will be ignored if present on command lines or in command files.

Injecting into a program

-injectName

Sets the name of the process for Coverage 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 (decimal) id of a process for Coverage Validator to attach to.

Not compatible with -program, -injectName, -waitName or -monitorAService.

Example:

-injectID 1032

Waiting for a program

-waitNameEXE
-waitName

-waitName has been replaced by **-waitNameEXE**. **-waitName** will be honoured to provided backwards compatibility.

Sets the name of a process that Coverage Validator will wait for.

When the named process starts Coverage Validator will attach to the process.

Not compatible with -program, -injectName, -injectID or -monitorAService.

Examples:

-waitNameEXE c:\testbed.exe
-waitNameEXE "c:\new compiler\version2\testbed.exe"

-waitNameDLL

Sets the name of a process DLL that Coverage Validator will wait for.

When the named process starts Coverage Validator will attach to the process.

Examples:

-waitNameDLL c:\dotNetApp.dll
-waitNameDLL "c:\new compiler\version2\dotNetApp.dll"

For use with -waitNameEXE when you want to wait for .Net Core applications.

-waitNameEXE "c:\program files\dotnet\dotnet.exe" **-waitNameDLL** "c:\testApps\dotNetCoreApp\release\dotNetCoreApp.dll"

-maxNumRestarts

Set the number of times a process monitored using -waitName can be restarted before Coverage Validator will exit.

To use this option you must use -waitName and -monitorAllRestarts.

Not compatible with -program, -injectName, -injectID or -monitorAService.

Examples:

-maxNumRestarts 1
-maxNumRestarts 5
-maxNumRestarts 600

-monitorAllRestarts

Allow Coverage Validator to monitor a process starting multiple times.

To use this option you must use -waitName.

If you use this option without specifying -maxNumRestarts the restart count is set to 1.

Not compatible with -program, -injectName, -injectID or -monitorAService.

Examples:

-monitorAllRestarts

Monitoring a service

-monitorAService

Sets the full file system path of a service *including any extension*.

Coverage 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"

.Net Core specific arguments

-dotNetCoreArg

Specifies and argument to pass to the .Net Core runtime. You can specify -dotNetCoreArg as many times as you need to pass as many arguments as you need.

See this Microsoft document for the list of .Net Core runtime configuration options https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet#runtime-options.

Use this argument with -program.

Examples:

-dotNetCoreArg "--roll-forward LatestPatch"
-dotNetCoreArg "--runtimeconfig ./configUnitTest.json"

-dotNetCoreLaunchType

Specifies the type of program being launched by the .Net Core runtime. You can specify -dotNetCoreLaunchType once. If specified more than once, the last definition is used.

Use this argument with -program.

Examples:

-dotNetCoreLaunchType SelfContained
-dotNetCoreLaunchType FrameworkDependent

Data Collection

-collectData

Enables or disables the collection of flow tracing data

Examples:

-collectData:On
-collectData:Off

The data collection option may be ignored because of the instrumentation mode that is selected.

-collectStdout

Enables or disables the collection of standard output (stdout)

Examples:

-collectStdout:On
-collectStdout:Off

5.4 User interface visibility

User interface visibility

You can choose to hide or show Coverage Validator during the test, as well as the window of the target application.

-displayUI

Forces the Coverage 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 coverage 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 Coverage 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 Coverage Validator user interface during the test.

-launchAppHide

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's 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 Coverage Validator via the **-arg** option.

-launchAppShow

Shows the target application during the test.

This is equivalent to setting the `wShowWindow` member of the `STARTUPINFO` struct to `SW_SHOW` when using the Win32 `CreateProcess()` function.

-launchAppShowMaximized
-launchAppShowMinimized
-launchAppShowMinNoActive
-launchAppShowNA
-launchAppShowNoActivate
-launchAppShowNormal

As well as the previous two options to show or hide the target application during the test there are other options equivalent to values that can be used in the `STARTUPINFO` struct.

The options are equivalent to the setting the `wShowWindow` member to the following values

Option	wShowWindow member	Launched application is shown...
-launchAppShowMaximized	`SW_SHOWMAXIMIZED`	Maximized and activated
-launchAppShowMinimized	`SW_SHOWMINIMIZED`	Minimized and activated
-launchAppShowMinNoActive	`SW_SHOWMINNOACTIVE`	Minimized and not active
-launchAppShowNA	`SW_SHOWNA`	Shown at current size and position but not activated
-launchAppShowNoActivate	`SW_SHOWNOACTIVATE`	Show at most recent size and position but not activated

-launchAppShowNormal	SW_SHOWNORMAL	Show at original size and position and activated

-showCommandPrompt
-hideCommandPrompt

Causes any launched console window to be shown or hidden during the test.

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 Coverage Validator once a test is complete.

-refreshResults (Summary tab)
-refreshCoverage
-refreshFunctions
-refreshFilesAndLines

Error reporting user interface visibility

-dontShowWindowsErrorReportingUI

Does not show the Windows Error Reporting dialog during tests if any test crashes the target program.

-showWindowsErrorReportingUI

Does show the Windows Error Reporting dialog during tests if any test crashes the target program.

5.5 Session Management

Session management

The following options let you control the sessions during testing

-numSessions

Sets the number of sessions that can be loaded at once.

This is equivalent to the same setting in the session manage and can't be less than 1.

Example:

-numSessions 2

-saveSession

Saves the session data when all data has finished being collecting from the target program.

Coverage Validator 32 and 64 bit use the file extension .cvm and .cvm_64 respectively.

A missing or incorrect filename extension will be corrected automatically

Examples:

-saveSession c:\results\testMacro1.cvm
-saveSession "c:\test results\testMacro1.cvm_x64"
-saveSession c:\results\testMacro1

-loadSession
-loadSession2

-loadSession loads a previously created session to be merged with the data from the session being *recorded*.

-loadSession2 loads a session to be merged with the session loaded via **-loadSession**.

These options might be used when a series of tests have already been performed and their sessions saved.

Coverage Validator 32 and 64 bit use the file extension .cvm and .cvm_64 respectively.

A missing or incorrect filename extension will be corrected automatically

Examples:

-loadSession c:\results\testMacro1.cvm
-loadSession "c:\test results\testMacro1.cvm"
-loadSession 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.

5.6 Merging sessions

Merging sessions

-mergeSessions

Merge a previously loaded session (using -loadSession) with the current session just recorded (by the command line arguments to -program, etc).

The resulting merged session is placed as the current session.

-saveMergeResult

Results of a session merge are saved to a file.

If the name contains spaces, it should be quoted.

Examples:

-saveMergeResult c:\results\testMacro1.cvm
-saveMergeResult "c:\test results\testMacro1.cvm"

➡ See also -showMergeWithReport for whether a list of merged files is included in exported data.

-mergeUsingAddress

Merge coverage data based on symbol addresses.

Typically this is used when merging coverage data from tests on the *same* build of software (same executable modules etc).

➡ See also -mergeUsingSymbol

-mergeUsingSymbol

Merge coverage data based on symbol names and symbol filenames.

Typically this is used when merging coverage data from tests on the *different* builds of software (e.g. different executable modules using the same source code).

➡ See also -mergeUsingAddress

-mergeMultiple

Merge coverage data from multiple sessions that are listed in file.

The contents of the file list one session file per line. Lines that identify files that don't exist or that are not sessions are ignored

Example merge multiple file:

e:\cv_help.cvm
e:\cv_red.cvm
e:\cv_green.cvm
e:\cv_blue.cvm
e:\cv_magenta.cvm

e:\cv_cyan.cvm
e:\this_file_doesnt_exist.cvm

Examples:

-mergeMultiple c:\results\merge_multiple.txt
-mergeMultiple "c:\test results\merge_multiple.txt"

Example command line:

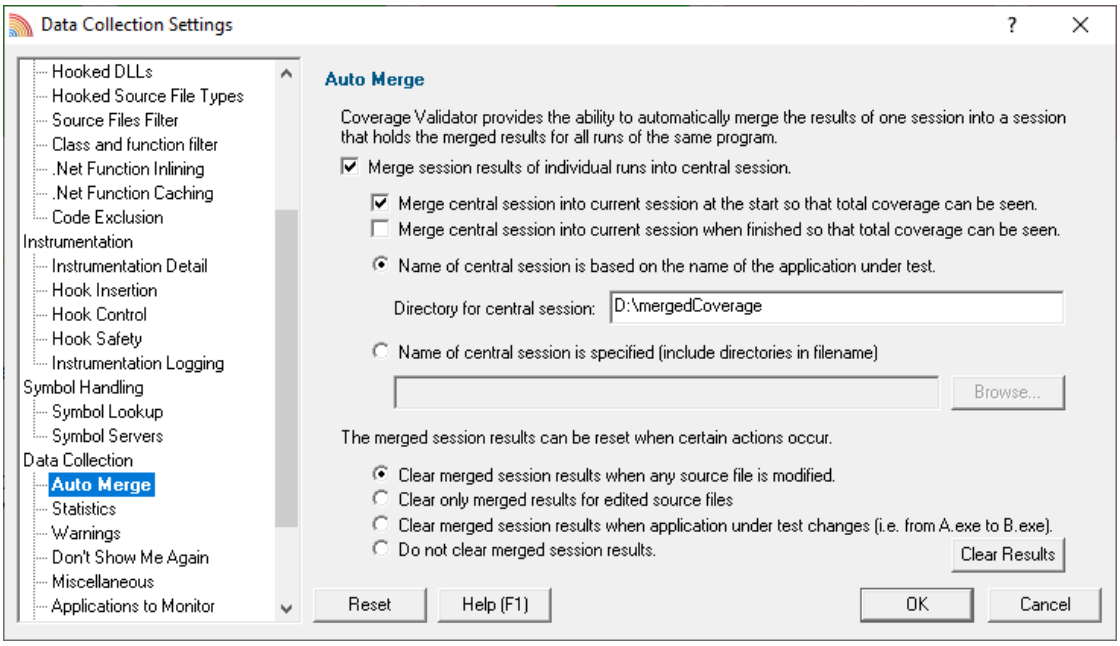
This command line merges every session listed in c:\cv_merge_multiple.txt, then saves the result in e:\cv_merge_result.cvm. Finally Coverage Validator is closed.

-mergeMultiple e:\cv_merge_multiple.txt -mergeSessions -saveMergeResult e:\cv_merge_result.cvm -hideUI

Merging central sessions

The following options let you control the merging of sessions into a central.

Options here are equivalent to the settings on the Auto Merge page (below) of the Data Collection Settings.



-mergeToCentralSession

Switch the auto merge feature on or off.

This option overrides the Merge session results of individual runs into central session check box.

 If you're using this to merge coverage data from different applications (e.g. running unit tests) we recommend also using -mergeClearNone and -mergeUsingSymbol.

Examples:

-mergeToCentralSession:On
-mergeToCentralSession:Off

-disableMergeIntoCurrentSession
-enableMergeIntoCurrentSession

When the regression test is complete, set whether the merged session results are then merged into the current session or not.

These options override the Merge central session into current session when finished... option which will prevail if *neither* of these options are present.

-centralDirectory

Names the central directory used for performing an auto merge.

Use the full directory path, or to reset the central directory use `<<None>>`.

 If you're merging coverage data from different applications (e.g. running unit tests) we recommend also using -mergeClearNone and -mergeUsingSymbol.

Examples:

-centralDirectory e:\unitTestResults\test1
-centralDirectory <<None>>

-centralFileName

Names the full path to the central file used for performing an auto merge.

 If you're merging coverage data from different applications (e.g. running unit tests) we recommend also using -mergeClearNone and -mergeUsingSymbol.

Example:

-centralFileName e:\unitTestResults\test1\unitTest1.cvm

Resetting merge results

The next four options are exactly equivalent to the reset options on the Auto Merge page of the Data Collection Settings.

The four triggers for clearing the merged session results are:

- **-mergeClearRebuild** › Clear all merged session results when any source file is modified (the application changes its timestamp)
- **-mergeClearEditedFilesOnly** › Clears only merged session results for source files that have been edited
- **-mergeClearChangeApplication** › When the application being tested changes from that used to create the merged session data
- **-mergeClearNone** › No clearing of merged session results occurs under any circumstance

5.7 Session Export Options

Session export format - HTML or XML

-exportAsHTML
-exportAsXML

Export the session data as an HTML or XML file when Coverage 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"

-exportMergedAsHTML
-exportMergedAsXML

Export the *merged* session data as an HTML or XML file when Coverage Validator has finished collecting data from the target program.

Example:

-exportMergedAsHTML c:\results\testMacro1Merged.html
-exportMergedAsXML "c:\xml results\testMacro1Merged.xml"

-exportAsXMLCobertura

Export the session data as a Cobertura XML file when Coverage 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:

-exportAsXMLCobertua "c:\test results\xm -cobertura\testMacro1.html"

-exportMergedAsXMLCobertura

Export the *merged* session data as a Cobertura XML file when Coverage Validator has finished collecting data from the target program.

Example:

-exportMergedAsXMLCobertura "c:\cobertura xml results\testMacro1Merged.xml"

-showMergeWithReport

Specify whether a list of merged files is included in the exported data when an export is performed, and if so, whether to include them at the top or bottom.

Examples:

-showMergeWithReport none › Do not include in the report
-showMergeWithReport top › Include at the top of the report
-showMergeWithReport bottom › Include at the bottom of the report

Session export encoding - HTML or XML

These options allow you to export the session data as UTF-16, UTF8 or ASCII. UTF-16 and UTF-8 will add a byte order mark (BOM) to the start of the exported file.

-exportAsHTML_BOM

The exported HTML will be exported with the appropriate format.

-exportAsHTML_BOM ASCII
-exportAsHTML_BOM UTF8
-exportAsHTML_BOM UTF16

-exportAsXML_BOM

The exported XML will be exported with the appropriate format.

-exportAsXML_BOM ASCII
-exportAsXML_BOM UTF8
-exportAsXML_BOM UTF16

-exportAsXMLCobertura_BOM

The exported XML will be exported with the appropriate format.

-exportAsXMLCobertura_BOM ASCII
-exportAsXMLCobertura_BOM UTF8
-exportAsXMLCobertura_BOM UTF16

-exportMergedAsHTML_BOM

The exported HTML will be exported with the appropriate format.

-exportMergedAsHTML_BOM ASCII
-exportMergedAsHTML_BOM UTF8
-exportMergedAsHTML_BOM UTF16

-exportMergedAsXML_BOM

The exported XML will be exported with the appropriate format.

-exportMergedAsXML_BOM ASCII
-exportMergedAsXML_BOM UTF8
-exportMergedAsXML_BOM UTF16

-exportMergedAsXMLCobertura_BOM

The exported XML will be exported with the appropriate format.

-exportMergedAsXMLCobertura_BOM ASCII
-exportMergedAsXMLCobertura_BOM UTF8
-exportMergedAsXMLCobertura_BOM UTF16

HTML and XML export options

The following options control settings for HTML and XML export of the recorded session results

Many of the export settings take the form option:On or Option:Off

Example:

-exportUnvisitedLines:On
-exportVisitedLines:Off

Most of these options are equivalent to the corresponding settings used in the Export Session dialog.

-exportDescription	Sets the description to be included the exported HTML/XML. Quotes should be used if spaces are present. **-exportDescription** "Quality control test 2a"

-exportDetailedReport:On/Off	Specifies that a detailed report is produced
-exportDoColourCode:On/Off	Make the HTML export colour coded (not relevant for XML export)
-exportFileFormatCR **-exportFileFormatCRLF** **-exportFileFormatLF**	The line ending in the output file will be one of the following: • Carriage Return: `\r` (e.g. Macintosh) • Carriage Return, Line Feed: `\r\n` (e.g. Windows) • Line Feed: `\n` (e.g. Linux)
-exportIncludeArgs:On/Off	Choose if program arguments are included in exported data.
-exportSourceCode:On/Off	Set whether source code is exported - only used when **-exportType** is set to SummaryAndCoverage or DLL
-exportType	Specify the format of the exported data: • File • Class • DLL • SummaryAndCoverage -**exportType** SummaryAndCoverage
-exportUnhooked:On/Off	Include information about functions that could not be hooked
-exportUnvisitedFunctions:On/Off	Include information about unvisited functions
-exportUnvisitedLines:On/Off	Include information about unvisited lines
-exportVisitedFunctions:On/Off	Include information about visited functions
-exportVisitedLines:On/Off	Include information about visited lines

5.8 Filter and Hook options

File extension hooking

-fileExt

Specify a file extension to be instrumented.

This is the same as listing a file extension on the Hooked Source File Types page in the Filters section of the settings dialog.

Example:

-fileExt cpp

Source code exclusion

Source code exclusion lets you control sections of a source file to be excluded by using pragmas (keyword directives) to mark the a region of code.

The pragmas can surround complete classes, functions or just single lines of code.

-codeExcludeDisable
-codeExcludeEnable

Disables or enables source code exclusion.

-startPragma

Sets the start pragma for use with multi-line code exclusion.

Example:

-startPragma `CVPragmaMultiLineStart`

-endPragma

Sets the end pragma for use with multi-line code exclusion.

Example:

-endPragma `CVPragmaMultiLineEnd`

-linePragma

Sets the line pragma for use with single line code exclusion.

Example:

-linePragma `CVPragmaSingleLineIgnore`

Source statistics per-DLL

Depending on how you build your software it's possible that some of your source files are present in more than one module (DLL / EXE) in your application.

These settings are equivalent to those for multiple source code file inclusion on the Statistics tab in the Data Collection section of the settings dialog for more details.

The options are:

-statisticsConsiderDLLForSource

Code coverage data is collected for each source on a per-DLL basis.

-statisticsIgnoreDLLForSource

Code coverage data is collected for each source regardless of which DLL the source code is compiled into.

Class and function hooks

You can indicate class and function names that are to be hooked or not hooked.

This is done either directly on the command line or from a file via **-classAndFunctionFile**

-classAndFunction

Specify classes and methods to be hooked or not hooked, depending on the use of the next two options.

Examples:

-classAndFunction MyClass::MyMethod
-classAndFunction MyClass::
-classAndFunction MyFunction

-classAndFunctionHook

Only the classes and methods specified by **-classAndFunction** will be hooked.

-classAndFunctionDoNotHook

The classes and methods specified by **-classAndFunction** will *not* be hooked.

-classAndFunctionAll

All classes and methods will be hooked.

-classAndFunctionNone

Discard any previous command line options specified for the class and method filter.

-classAndFunctionFile

Points to a file specifying the class and function to be hooked for the test.

If the filename contains spaces, the filename should be quoted.

 The settings dialog Class and Function Filter tab provides options for creating a ready made class and function hook file by using the **Export...** button on the dialog.

Examples:

-classAndFunctionFile c:\settings\testMacroClassAndMethods.cvxc
-classAndFunctionFile "c:\reg tests settings\testMacroClassAndMethods.cvxc"

 Class and function file format

The first line of text in the DLL hooks file is *one* of the following:

- `Rule:DoNotHookSpecificClasses`
  - DLLs marked as enabled *will not* be hooked. All other DLLs will be hooked
- `Rule:HookSpecificClasses`
  - DLLs marked as enabled *will* be hooked. All other DLLs will not be hooked
- `Rule:HookAllClasses`
  - All DLLs will be hooked regardless of the settings in the list

 Capitalization is important.

The remaining lines list one class and method per line.

Class and method are separated by `::` or omit the method name if all methods in the class are wanted..

To name a method and no class, use `::` to prefix the method.

Example:

Two methods in the `pageMemoryUsageTracker` class should be hooked, along with all methods in the class `rightMenuAnalysis`, and all methods and functions named `resizeData`.

```
Rule:HookSpecificClasses
pagedMemoryUsageTracker::setPage
pagedMemoryUsageTracker::reset
resizeData
rightMenuAnalysis::
```

Example:

Two methods in the `pageMemoryUsageTracker` class are not to be hooked.

```
Rule:DoNotHookSpecificClasses
pagedMemoryUsageTracker::setPage
pagedMemoryUsageTracker::reset
```

Example:

All classes and methods will be hooked.

```
Rule:HookAllClasses
```

The file can be ANSI or UNICODE text and paths with spaces do not need quotes.

Source file hooks

-sourceFileFilterHookFile

Points to a file specifying the source files to be hooked for the test.

If the filename contains spaces, the filename should be quoted.

The settings dialog Source Files Filter tab provides options for creating a ready made source file filter hook file by using the **Export...** button on the dialog.

Examples:

-sourceFileFilterHookFile c:\settings\testMacroFiles.cvxft
-sourceFileFilterHookFile "c:\reg tests settings\testMacroFiles.cvxft"

Source file filter format

The first line of text in the DLL hooks file is *one* of the following:

- `Rule:DoNotHook` > Source files that follow *will not* be hooked. All other files will be hooked
- `Rule:DoHook` > Source files that follow *will* be hooked. All other files will not be hooked

Capitalization is important.

The remaining lines list one source file (or source directory) per line.

The file is named with a path or without a path, i.e. the same way that Coverage Validator discovers the path.

Example:

Most files are listed with a full path but constituent files of MFC might be listed without a path.

```
Rule:DoNotHook
"c:\program files\software verification\coverage validator\examples\nativeExampl
appmodule.cpp
```

Example:

Using paths with and without spaces:

```
Rule:DoHook
"E:\OM\C\coverage Validator\examples\nativeExample"

Rule:DoHook
E:\OM\C\coverageValidator\examples\nativeExample
```

Example:

Using environment variables.

```
Rule:DoHook
%ENV_VAR%\examples\nativeExample
```

Using wildcards.

```
Rule:DoHook
c:\test\*\nativeExample
```

The file can be ANSI or UNICODE text and paths with spaces do not need quotes.

5.9 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 Coverage 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 coverage 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.cvs
-loadSettings "c:\coverage test settings\testMacro1.cvs"

 The -settings option is identical to -loadSettings and is provided for backwards compatibility

File locations for source, PDB or MAP files

File location files can be easily generated by exporting file locations from the File Locations page of the settings dialog.

-fileLocations

Specify a plain text file listing file locations to be used during testing. See the format of the file below.

Each set of file types (one per line) is preceded by a header line in the file.

- **[Files]** › Source files
- **[Third]** › Third party source files
- **[PDB]** › PDB files
- **[MAP]** › MAP files

Example:

-fileLocations c:\coverageTests\testFileLocations1.cvxfl

Example file:

```
[Files]
c:\work\project1\
[Third]
d:\VisualStudio\VC98\Include
[PDB]
c:\work\project3\debug
c:\work\project3\release
[MAP]
c:\work\project3\debug
```

```
c:\work\project3\release
```

Files listing classes and functions to hook

Set class and function names that are to be hooked or not hooked using -classAndFunctionFile in the Filter and Hook options.

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.cvx
-dllHookFile "c:\coverage tests settings\testMacroDLLs.cvx"

The first line of text in the DLL hooks file is one of the following:

- Rule: DoNotHook › DLLs marked as enabled *will not* be hooked. All other DLLs will be hooked
- Rule: DoHook › DLLs marked as enabled *will* be hooked. All other DLLs will not be hooked
- Rule: HookAll › All DLLs will be hooked regardless of the settings in the list

Capitalization is important.

The remaining lines list one DLL filename or folder path and an enabled state on each line.

Example:

```
Rule:DoNotHook
nativeExample.exe enable=FALSE
MFC42D.DLL enable=TRUE
MSVCRTD.dll enable=TRUE
KERNEL32.dll enable=TRUE
ole32.dll enable=TRUE
```

Example:

```
Rule:DoHook
E:\OM\C\coverageValidator\examples\nativeExample\DebugNonLink
enable=TRUE
```

Example:

```
Rule:DoHook
```

```
"E:\OM\C\coverageValidator\examples\nativeExample with
spaces\DebugNonLink" enable=TRUE
```

Example:

```
Rule:DoHook
%ENV_VAR%\DebugNonLink enable=TRUE
```

Here, the environment variable `ENV_VAR` is used to replace the text `%ENV_VAR%` in the path definition.

For example, if ENV_VAR was set to `e:\dev\` the resulting value would be `e:\dev\DebugNonLink`

The file can be ANSI or UNICODE text and paths with spaces do not need quotes.

5.10 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 limits.

Use -- to insert comments into the file, including when commenting out option.

Examples:

-commandFile c:\coveragetests\testMacro1.cf
-commandFile "c:\coverage 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.cvs

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

```
-saveSession c:\testbed\results\test1.cvm
```

For any argument that can be supplied to a command in a command file, you can also specify an environment variable substitution.

```
-directory %DIR%
-program %DIR%\testProgram.exe
```

The environment variables must have been set prior to starting Coverage Validator.

You cannot specify a command with an environment variable substitution.

5.11 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.

Debugging command driven testing

If you're having problems with using the command line, check the following, try displaying error messages using the option below, and look at the exit return codes.

.

- separate command line arguments with spaces
- all command line options that include spaces need to have quotes around them
- some arguments are only useful in conjunction with others - check notes against each option
- some arguments are incompatible with others - check notes against each option

-showErrorsWithMessageBox

Forces errors to be displayed using a message box when running from the command line.

This can be very useful when debugging a command line that does not appear to work correctly.

Checking for a heartbeat

-pulseToStdout

Coverage Validator will output '.' characters to the standard output on a regular basis.

Use this option if you want some activity on the standard output that can be monitored for signs of a non-responsive program.

Pipe warnings

-suppressPipeWarnings

Prevent Coverage Validator from displaying an error message box when the stub injected into the launched application cannot find the user interface.

This error condition rarely occurs, but for these situations this facility can be useful.

Do not use with **-terminatePipeWarnings** below.

-terminatePipeWarnings

Forces Coverage Validator to shutdown the launched application (using TerminateProcess) when the stub injected into the launched application cannot find the user interface.

This error condition rarely occurs, but for these situations this facility can be useful.

Do not use with the **-suppressPipeWarnings** option.

Exit return codes

Coverage 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 Coverage Validator uses it (obsolete)
- **-22** › Java xrun supplied that is not allowed because Java Coverage Validator uses it (obsolete)
- **-23** › Java cp supplied that is not allowed because Java Coverage Validator uses it (obsolete)
- **-24** › Java classpath supplied that is not allowed because Java Coverage Validator uses it (obsolete)
- **-25** › Firefox is already running, please close it (obsolete)
- **-26** › Lua runtime DLL version is not known (obsolete)
- **-27** › Not compatible software
- **-28** › InjectUsingCreateProcess, no DLL name supplied
- **-29** › InjectUsingCreateProcess, Unable to open PE File when inspecting DLL
- **-30** › InjectUsingCreateProcess, Invalid PE File when inspecting DLL
- **-31** › InjectUsingCreateProcess, No Kernel32 DLL
- **-32** › InjectUsingCreateProcess, NULL VirtualFree() from GetProcAddress
- **-33** › InjectUsingCreateProcess, NULL GetModuleHandleW() from GetModuleHandleW
- **-34** › InjectUsingCreateProcess, NULL LoadLibraryW() from LoadLibraryW
- **-35** › InjectUsingCreateProcess, NULL FreeLibrary() from FreeLibrary
- **-36** › InjectUsingCreateProcess, NULL VirtualProtect() from GetProcAddress
- **-37** › InjectUsingCreateProcess, NULL VirtualFree() from GetProcAddress
- **-38** › InjectUsingCreateProcess, unable to find DLL load address
- **-39** › InjectUsingCreateProcess, unable to write to remote process's memory
- **-40** › InjectUsingCreateProcess, unable to read remote process's memory
- **-41** › InjectUsingCreateProcess, unable to resume a thread
- **-42** › UPX compressed - cannot process such executables
- **-43** › Java class not found in CLASSPATH
- **-44** › Failed to launch the 32 bit svlGetProcAddressHelperUtil.exe
- **-45** › Uknown error with svlGetProcAddressHelperUtil.exe
- **-46** › Couldn't load specified DLL into svlGetProcAddressHelperUtil.exe
- **-47** › Couldn't find function in the DLL svlGetProcAddressHelperUtil.exe
- **-48** › Missing DLL argument svlGetProcAddressHelperUtil.exe
- **-49** › Missing function argument svlGetProcAddressHelperUtil.exe
- **-50** › Missing svlGetProcAddressHelperUtil.exe
- **-51** › Target process has a manifest that requires elevation
- **-52** › svlInjectIntoProcessHelper_x64.exe not found
- **-53** › svlInjectIntoProcessHelper_x64.exe failed to start
- **-54** › svlInjectIntoProcessHelper_x64.exe failed to return error code
- **-55** › getImageBase() worked ok
- **-56** › ReadFile() failed in getImageBase()
- **-57** › NULL pointer when trying to allocate memory
- **-58** › CreateFile() failed in getImageBase()
- **-59** › ReadProcessMemory() failed in getImageBase()
- **-60** › VirtualQueryEx() failed in getImageBase()
- **-61** › Bad /appName argument in svlInjectIntoProcessHelper_x64.exe
- **-62** › Bad /dllName argument in svlInjectIntoProcessHelper_x64.exe
- **-63** › Bad /procId argument in svlInjectIntoProcessHelper_x64.exe
- **-64** › Failed to OpenProcess in svlInjectIntoProcessHelper_x64.exe
- **-65** › A DLL that the .exe depends upon cannot be found
- **-66** › A stdin file was specified, but Validator could not open it
- **-67** › A stdout file was specified, but Validator could not open it
- **-68** › Failed to create the child output pipe

- **-69** › Failed to create a duplicate of the output write handle for the std error write handle. This is necessary in case the child application closes one of its std output handles
- **-70** › Failed to create the child input pipe
- **-71** › Failed to create a duplicate output read temporary file
- **-72** › Failed to create a duplicate input write temporary file
- **-73** › User was trying to launch a service as an application that was linked to CV APIs. User cancelled when informed of this fact
- **-74** › Returned by Coverage Validator if user performs a baseline comparison and memory leaks are detected
- **-75** › Shutdown and restart as 32 bit Coverage Validator
- **-76** › Shutdown and restart as 64 bit Coverage Validator
- **-77** › Entry point in executable is NULL.
- **-78** › Application is .Net Core.
- **-79** › Entry point is for a .Net application.

- **-80** › VirtualAllocEx() returned NULL
- **-81** › InjectUsingCreateProcess, NULL GetLastError() from GetProcAddress

5.12 Command Line Reference

Command line reference

The following alphabetical list provides a convenient look-up for all the command line arguments used in automated regression testing.

Option	Description
-?	Print command line help on the standard output.
-allArgs	Pass the remainder of the command line to the program being launched.
-arg	Pass command line arguments to the target program. Can be used multiple times.
-centralDirectory	Name a central directory used for performing an auto merge.
-centralFileName	Name the full path to the central file used for performing an auto merge.
-classAndFunction	Specify classes and methods to be hooked or not hooked.
-classAndFunctionAll	All classes and methods will be hooked.
-classAndFunctionDoNotHook	Classes and methods specified by -classAndFunction will *not* be hooked.
-classAndFunctionFile	Points to a file specifying the class and function to be hooked for the test.
-classAndFunctionHook	Only the classes and methods specified by -classAndFunction will be hooked.
-classAndFunctionNone	Discard any previous command line options specified for the class and method filter.

-codeExcludeDisable **-codeExcludeEnable**	Disable or enable source code exclusion.
-collectData	Turn data collection on or off
-collectStdout	Turn collection of stdout on or off
-commandFile	Specify a file from which to read the command line arguments.
-createProcessStartupThread	This option is obsolete.
-directory	Set the working directory in which the program is executed.
-disableMergeIntoCurrentSession	When the regression test is complete, the merged session results will not be merged into the current session.
-displayUI	Force the Coverage Validator user interface to be displayed during the test.
-dllHookFile	Points to a file listing the DLLs to be hooked for the test.
-doNotInteractWithUser	Never display dialog boxes in the target application that is being profiled.
-dotNetCoreArg	Specify a runtime configuration option to the .Net runtime.
-dotNetCoreLaunchType	Specify if you are launching a self contained or framework dependent .Net Core application.
-dontShowWindowsErrorReportingUI	Never display the Windows Error Reporting dialog if the target program crashes.
-enableMergeIntoCurrentSession	When the regression test is complete, set the merged session results to be merged into the current session.
-endPragma	Set the end pragma for use with multi-line code exclusion.
-environment	Environment variables for program, as a series of name/value pairs
-exportAsHTML **-exportAsXML**	Export the session data as an HTML or XML file when Coverage Validator has finished collecting data from the target program.
-exportAsXMLCobertura	Export the session data as a Cobertura XML file when Coverage Validator has finished collecting data.
-exportAsHTML_BOM **-exportAsXML_BOM** **-exportAsXMLCobertura_BOM** **-exportMergedAsHTML_BOM** **-exportMergedAsXML_BOM** **-exportMergedAsXMLCobertura_BOM**	Specify the file encoding for the exported file
-exportDescription	Set the description to be included the exported HTML/XML.
-exportDetailedReport	Produce a detailed export report.

-exportDoColourCode	Make the HTML export colour coded (not relevant for XML export).
-exportFileFormatCR **-exportFileFormatCRLF** **-exportFileFormatLF**	Set the line ending format in the output file.
-exportMergedAsHTML **-exportMergedAsXML**	Export the merged session data as an HTML or XML file when Coverage Validator has finished collecting data from the target program.
-exportSourceCode	Set whether source code is exported when -exportType is set to. SummaryAndCoverage
-exportType	Specify the format of the exported data (file, class, summary & coverage).
-exportUnhooked	Export information about functions that could not be hooked.
-exportUnvisitedFunctions	Export information about unvisited functions.
-exportUnvisitedLines	Export information about unvisited lines.
-exportVisitedFunctions	Export information about visited functions.
-exportVisitedLines	Export information about visited lines.
-fileExt	Specify a file extension to be instrumented.
-fileLocations	Specify a plain text file listing file locations to be used during testing. See the format of the file below.
-help	Print command line help on the standard output.
-hideCommandPrompt	Any launched console window will be hidden during the test.
-hideUI	Hide the Coverage Validator user interface during the test.
-idleStartupThread	This option is obsolete.
-ignoreMissingStdin	Allows you to specify stdin files that don't exist without getting an error.
-ignoreMissingStdout	Allows you to specify stdout files that don't exist without getting an error.
-injectID	Set the numeric (decimal) id of a process for Coverage Validator to attach to.
-injectName	Set the name of the process for Coverage Validator to attach to.
-launchAppHide	Hide the target application during the test.
-launchAppShow	Show the target application during the test.
-launchAppShowMaximized	Show the target application maximized and activated.
-launchAppShowMinNoActive	Show the target application minimized and activated.
-launchAppShowMinimized	Show the target application minimized and not active.
-launchAppShowNA	Show the target application at current size and position but not activated.

-launchAppShowNoActivate	Show the target application at most recent size and position but not activated.
-launchAppShowNormal	Show the target application at original size and position and activated.
-linePragma	Set the line pragma for use with single line code exclusion.
-loadSession	Load a previously created session to be merged with the data from the session being recorded.
-loadSession2	Load a session to be merged with the session that was loaded via -loadSession.
-loadSettings	Points to a previously saved settings file to be used for the test.
-maxNumRestarts	Sets the maximum number of times a process can restart itself to be monitor by -waitName.
-mergeClearChangeApplication	Reset merge results when the application being tested changes from that used to create the merged session data
-mergeClearEditedFilesOnly	Reset merge results only for source files that have been edited
-mergeClearNone	No clearing of merged session results occurs under any circumstance.
-mergeClearRebuild	Clear all merged session results when any source file is modified (the application changes its timestamp)
-mergeSessions	Merge a previously loaded session (using -loadSession) with the current session just recorded (by the command line arguments to -program, etc).
-mergeToCentralSession	Switch the auto merge feature on or off.
-mergeUsingAddress	Merge coverage data based on symbol addresses
-mergeUsingSymbol	Merge coverage data based on symbol names and symbol filenames
-monitorAllRestarts	Enables the ability to monitor additional runs of the applicaton monitored by -waitName.
-monitorAService	Specify the full file system path to the service to monitor with Coverage Validator, including any extension. The service is not started by Coverage Validator but my an external means.
-normalStartupThread	This option is obsolete.
-noSuspendInStubDuringAttach	This option is obsolete.
-numSessions	Set the number of sessions that can be loaded at once.
-pauseStartupThread	This option is obsolete.
-program	Specify the full file system path of the executable target program to be started by Coverage Validator, including any extension.
-programToMonitorDLL	Specify the .Net Core DLL that identifies the program being monitored. Use in conjunction with -programToMonitorEXE.

-programToMonitorEXE -programToMonitor	Specify the program to monitor if monitoring a different application than the launched application.
-programToMonitorLaunchCount	Specify the nth invocation of the programToMonitor which is to have its data collected.
-pulseToStdout	Output '.' characters to the standard output on a regular basis.
-refreshCoverage	Automatically refresh the Coverage tab once a test is complete.
-refreshFilesAndLines	Automatically refresh the File and Lines tab once a test is complete.
-refreshFunctions	Automatically refresh the Functions tab once a test is complete.
-refreshResults	Automatically refresh the Summary tab once a test is complete.
-resetSettings	Forces Coverage Validator to reset (nearly) all settings to the default state.
-saveMergeResult	Results of a session merge are saved to a file.
-saveSession	Save the session data when all data has finished being collecting from the target program.
-setenvironment	Environment variables for Coverage Validator, as a series of name/value pairs
-settings	Points to a previously saved settings file to be used for the test.
-showCommandPrompt	Any launched console window will be shown during the test.
-showErrorsWithMessageBox	Force errors to be displayed using a message box when running from the command line.
-showMergeWithReport	Specify whether a list of merged files is included in the exported data when an export is performed, and if so, whether to include them at the top or bottom.
-showWindowsErrorReportingUI	Show the Windows Error Reporting user interface if the target program crashes.
-sourceFileFilterHookFile	Points to a file specifying the source files to be hooked for the test.
-startPragma	Set the start pragma for use with multi-line code exclusion.
-statisticsConsiderDLLForSource	Code coverage data is collected for each source on a per-DLL basis.
-statisticsIgnoreDLLForSource	Code coverage data is collected for each source regardless of which DLL the source code is compiled into.
-stdin	Name a file to be read and piped to the standard input of the application being tested.
-stdout	Name a file to be written with data piped from the standard output of the application being tested.

-suppressPipeWarnings	Prevent Coverage Validator from displaying an error message box when the stub injected into the launched application cannot find the user interface.
-suspendStartupThread	This option is obsolete.
-terminatePipeWarnings	Force Coverage Validator to shutdown the launched application when the stub injected into the launched application cannot find the user interface.
-waitNameDLL	Name a .Net Core dll that identifies the process to wait for. Use in conjunction with -waitNameEXE.
-waitNameEXE **-waitName**	Name a process that Coverage Validator will wait for.

To run 32 bit coverage validator run **`C:\Program Files (x86)\Software Verify\Coverage Validator x86\coverageValidator.exe`**

To run 64 bit coverage validator run **`C:\Program Files (x86)\Software Verify\Coverage Validator x64\coverageValidator_x64.exe`**

5.13 Troubleshooting

Running from the command line can cause some problems, often because you can't be sure that what you put on the command line did what you thought would do.

Ensure the arguments supplied are what you expected.

-echoArgsToUser

If you are testing a console application, make sure you can see it.

-showCommandPrompt

If an errors occur when processing the command line, make sure you can see those.

-showErrorsOnCommandPrompt

-showErrorsWithMessageBox

Look on the diagnostic tab to ensure the diagnostic data collected makes sense.

If you've got -hideUI in your command line, comment it out temporarily (make it -xhideUI so that it's not recognised).

What if the tool hangs?

If you're running from the command line, most likely you'll be running from a cmd prompt, or possibly powershell.

We've only ever had one customer report a hang with any of our tools when running from the command line.

We eventually found the problem, and it wasn't with the software tool.

The problem was that they were running the tool in hidden mode (-hideUI) from a command prompt and for unknown reasons the tool would never exit.

When they added a simple change to their command the problem went away.

They added **cmd /c** to the start of their command line. This opens a new command prompt and instructs it to launch the command line and wait for it to exit.

Problem command line:

```
"c:\program files (x86)\Software Verify\Coverage Validator x64\coverageValidator_x64.exe" -program
```

Working command line:

```
cmd /c "c:\program files (x86)\Software Verify\Coverage Validator x64\coverageValidator_x64.exe" -
```

Part VI

6 API

The Coverage Validator API

There are some features of Coverage Validator that are useful to call directly from your program.

Working with services?

If you are working with services you to attach Coverage Validator to a service and to start Coverage Validator, you should use the NT Service API, not the functions in this API.

All the other functions in this API can be used with applications and with services.

Deploying on a customer machine

You can use the API without incurring any dependency on Coverage Validator.

If Coverage Validator is not installed on the machine the software runs on, nothing will happen.

This allows you to add the Coverage Validator API to your software without need to have a separate build for use with Coverage Validator.

Convenience functions

One convenience function is provided that will start the Coverage Validator GUI (if it is not already running), then load the Coverage Validator code coverage collector into your process and start collecting code coverage.

```
extern "C"
int loadValidatorIntoApplication();

Returns:
TRUE   Successfully loaded CV DLL into target application and successfully
started the profiler.
FALSE   Failed to load the CV DLL or failed to start the profiler.
```

To use this function #include `loadValidatorIntoApplication.h` into your code.

The source files can be found in the `API` directory in the Coverage Validator install directory.

```
loadValidatorIntoApplication.h
loadValidatorIntoApplication.c
```

Just add these files to your project and build.

6.1 Native API Reference

Unicode or ANSI?

All the API functions are declared as `extern "C"`, so they can be used by C users and C++ users.

To use these functions #include `svlCVAPI.h` into your code.

cvLoadProfiler

Loads the profiler DLL into memory, but does not start the profiler.

Use this for:
32 bit applications with a 32 bit Coverage Validator GUI
64 bit applications with a 64 bit Coverage Validator GUI

For most use cases won't need to load the profiler, as the profiler will have been loaded when your launched your program from Coverage Validator, or when you injected into your program using Inject or Wait For Application.

However if you're running your program from outside of Coverage Validator and want to load the profiler from inside your program you can use `cvLoadProfiler()` to do that. You'll then need to call `cvStartProfiler()` to start it.

```
extern "C"
int cvLoadProfiler();

Returns:
TRUE   Successfully loaded PV DLL into target application.
FALSE  Failed to load the PV DLL into target application.
        Check that the PATH environment variable points to the Coverage Validator
    install directory contains svlCoverageValidatorStub*.dll.
```

Do not use this function if you are working with services, use the ***NT Service API.***

cvLoadProfiler6432

Loads the profiler DLL into memory, but does not start the profiler.

Use this for:
32 bit applications with a 64 bit Coverage Validator GUI

For most use cases won't need to load the profiler, as the profiler will have been loaded when your launched your program from Coverage Validator, or when you injected into your program using Inject or Wait For Application.

However if you're running your program from outside of Coverage Validator and want to load the profiler from inside your program you can use `cvLoadProfiler6432()` to do that. You'll then need to call `cvStartProfiler()` to start it.

```
extern "C"
int cvLoadProfiler6432();

Returns:
TRUE   Successfully loaded PV DLL into target application.
FALSE  Failed to load the PV DLL into target application.
       Check that the PATH environment variable points to the Coverage Validator
    install directory contains svlCoverageValidatorStub*.dll.
```

Do not use this function if you are working with services, use the ***NT Service API.***

cvStartProfiler

To start the profiler from your API code you need to call the function `cvStartProfiler()` from your code before you call any API functions. Ideally you should call this function as early in your program as possible.

```
extern "C"
int cvStartProfiler();

Returns:
TRUE   Successfully started PV profiler.
FALSE  Failed to start the PV profiler.
```

If you prefer to start the profiler from the user interface or command line you can omit the `cvStartProfiler()` call. You can leave it present if you wish to start Coverage Validator from your program.

Do not use this function if you are working with services, use the ***NT Service API.***

cvSetCollect()

Enables or disables data collection - i.e. whether data is sent to Coverage Validator from your target application.

```
extern "C"
void cvSetCollect(int enable); // TRUE to enable, FALSE to disable
```

cvGetCollect()

Returns whether data collection is on.

```
extern "C"
int cvGetCollect();            // Returns TRUE or FALSE
```

6.2 C# API

The C# API is a wrapper around the native API.

For all of these APIs see the native API for more details.

Adding the API to your application

The C# API is provided as a source code `svlCVAPI.cs` file that you add to your application. The source file is in the API directory in the Coverage Validator install directory.

The C# API does not add any dependencies to your application - if Coverage Validator is present the API functions work, if Coverage Validator is not present the API functions do nothing.

The C# API

The C# API is implemented by the `CoverageValidator` class in the `SoftwareVerify` namespace.

collectOn()

Turn data collection on.

```
public static void collectOn();
```

collectOff()

Turn data collection off.

```
public static void collectOff();
```

setCollect()

Turn data collection on or off.

```
public static void setCollect(bool enable);
```

getCollect()

Determine if data collection is turned on or off.

```
public static bool getCollect();
```

6.3 Calling the API via GetProcAddress

Calling API functions using GetProcAddress

If you don't want to use the svlCVAPI.c/h files you can use `GetProcAddress()` to find the interface functions in the Coverage Validator DLL.

The interface functions have different names and do not use C++ name mangling, but have identical parameters to the API functions.

To determine the function name take any native API name, replace the leading **cv** with **api**. For example `cvGetCollect()` becomes `apiGetCollect();`

Example usage

```
typedef int (__cdecl *apiGetCollect_FUNC)();


HMODULE getValidatorModule()
{
   HMODULE   hModule;

   hModule = GetModuleHandle(_T("svlCoverageThreadValidatorStub6432.dll"));    // 32 bit DLL wi
   if (hModule == NULL)
      hModule = GetModuleHandle(_T("svlCoverageValidatorStub_x64.dll"));       // 64 bit DLL wi
   if (hModule == NULL)
      hModule = GetModuleHandle(_T("svlCoverageValidatorStub.dll"));           // 32 bit DLL wi

   return hModule;
}

HMODULE   hMod;

// get module, will only succeed if Coverage Validator launched this app or is injected into t

hMod = getValidatorModule();
if (hMod != NULL)
{
    // CV is present, lookup the function and call it to get the data collection status

   apiGetCollect_FUNC   theFunc;

   theFunc = (apiGetCollect_FUNC)getFunctionFromValidatorModule("apiGetCollect");
   if (theFunc != NULL)
      return (*theFunc)();
}
```

API functions and their GetProcAddress names

For any API functions not listed, try looking up the name in `svlCoverageValidatorStub.dll` using depends.exe or PE File Browser.

Show API functions and GetProcAddress names

API Name	GetProcAddress() Name
cvLoadProfiler	
cvIsValidatorPresent	
cvStartProfiler	
cvSetCollect	apiSetCollect
cvGetCollect	apiGetCollect
cvShutdownCoverageVa	apiShutdownCoverageValidator

Other exported functions

You may see some other functions exported from `svlPerformanceValidatorStub.dll(_x64).dll`.

These other functions are for Performance Validator's use. Using them may damage memory locations and/or crash your code. Best not to use them!

6.4 Convenience functions

Convenience functions

One convenience function is provided that will start the Coverage Validator GUI (if it is not already running), then load the Coverage Validator coverage profiler into your process and start profiling it.

```
extern "C"
int loadValidatorIntoApplication();

Returns:
TRUE    Successfully loaded CV DLL into target application and successfully
started the profiler.
FALSE   Failed to load the CV DLL or failed to start the profiler.
```

To use this function #include `loadValidatorIntoApplication.h` into your code.

The source files can be found in the `API` directory in the Coverage Validator install directory.

```
loadValidatorIntoApplication.h
loadValidatorIntoApplication.c
```

Just add these files to your project and build.

Part VII

7 Working with IIS and Services

 When working with NT services your account must have the appropriate privileges described in the User Permissions topic.

Attaching to your service

To use Coverage Validator with NT Services you need to link a small library to your application and call two functions in the library.

The NT Service API

The NT Service API is provided to enable Coverage 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 Coverage Validator user interface.

Monitoring the service

When working with Coverage Validator and services using the NT Service API you don't start the service using Coverage 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 Coverage Validator, which you should have running *before* starting the service.

Once you've exercised your service and stopped it, Coverage Validator will show the usual coverage information, although you can manually request an update while the session is running

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 Coverage Validator with services, please contact us at support@softwareverify.com.

7.1 NT Service API

The Coverage Validator stub service libraries

The NT Service API is very simple, consisting of functions to load and unload the Coverage Validator DLL.

We have also provided some debugging functions to help you debug the implementation of the NT Service API because getting data into and out of services is not always straightforward.

The stub service libraries used for this are shown in the following table:

	32 bit Coverage Validator	64 bit Coverage Validator

32 bit service	`svlCVStubService.lib`
	`svlCVStubServiceMD.lib`
	`svlCVStubServiceMT.lib`
		`svlCVStubService6432.lib`

64 bit service	N/A	`svlCVStubService_x64.lib`
		`svlCVStubServiceMD_x64.lib`
		`svlCVStubServiceMT_x64.lib`

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

Library name suffixes

The MD suffix indicates the library was built with the /MD compiler switch.
The MT suffix indicates the library was built with the /MT compiler switch.

Directory Name: 2010 or 2012?

Visual Studio 6 to Visual Studio 2010
If you are using Visual Studio 2010 or earlier, use libraries from a directory with 2010 in the directory name.

Visual Studio 2010 to Visual Studio 2022
If you are using Visual Studio 2012 or later, use libraries from a directory with 2012 in the directory name.

Header files

The header files can be found in the `svlCVStubService` directory in the Coverage Validator install directory.

The headers file provide an error enumeration and the NT Service API functions.

```
svlCVStubService.h
svlServiceError.h
```

Linker Problems

Some linkers cannot link the stub service library file. If you have this problem see What do I do if I cannot use svlCVStubService.lib?

Loading the Coverage Validator DLL into your service

To load the Coverage Validator stub dll `svlCoverageValidatorStub(_x64).dll` into your service, call `svlCVStub_LoadCoverageValidator()`, *not* `LoadLibrary()`.

If you are monitoring a 32 bit service with the 64 bit Coverage Validator user interface you should use `svlCVStub_LoadCoverageValidator6432()`.

Shutting down the Coverage Validator DLL from your service.

To shutdown Coverage Validator's monitoring of the service , call `svlCVStub_ShutdownCoverageValidator()`.

This sends the shutting down notification and removes any hooks for your process.

Calling this function is optional. You can stop your service without calling this function.

Unloading the Coverage Validator DLL from your service.

To unload the Coverage Validator stub dll, call `svlCVStub_UnloadCoverageValidator()`, not `FreeLibrary()`.

Calling this function is optional. You can stop your service without calling this function.

Setting a service notification callback

Once you have successfully loaded the Coverage 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 *Coverage Validator* to periodically call a user supplied callback from which you can regularly inform the SCM of the appropriate status.

You can set the service callback with `svlCVStub_SetServiceCallback(callback, userParam)`.

Usage

Here is an example callback which ignores the userParam.

```
void serviceCallback(void   *userParam)
{
    static DWORD dwCheckPoint = 1;

    ssStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    ssStatus.dwServiceSpecificExitCode = 0;

    ssStatus.dwControlsAccepted = 0;

    ssStatus.dwCurrentState = dwCurrentState;
    ssStatus.dwWin32ExitCode = dwWin32ExitCode;
    ssStatus.dwWaitHint = dwWaitHint;

    ssStatus.dwCheckPoint = dwCheckPoint++;

    // Report the status of the service to the service control manager.

    return SetServiceStatus( sshStatusHandle, &ssStatus);
}
```

Once your service is running (rather than starting) your service callback should set the appropriate running status `SERVICE_RUNNING` rather than `SERVICE_START_PENDING`.

```
if (!ReportStatusToSCMgr(SERVICE_RUNNING,       // service state
                         NO_ERROR,              // exit code
                         0))                    // wait hint
{
   dwErr = GetLastError();
   if (bLogging)
      svlCVStub_writeToLogFileW(L"ReportStatusToSCMgr:5\r\n");
   goto cleanup;
}
```

An alternative solution is to prevent the service callback from being called once the service status has been set to running.

```
svlCVStub_SetServiceCallback(NULL, NULL);.
```

Starting Coverage Validator DLL in your service

To start Coverage Validator inspecting your service call `svlCVStub_StartCoverageValidator()`.

Starting Coverage Validator DLL in IIS

To start Coverage Validator inspecting IIS call `svlCVStub_StartCoverageValidatorForIIS()`.

Setting a filename for all logging data to be written to

To set the filename for all debugging/logging information to be written to call `svlCVStub_setLogFileName()`.

Deleting the logfile

To delete the log file call `svlCVStub_deleteLogFile()`.

Writing text to the logfile

To write a standard ANSI character string to the log file call `svlCVStub_writeToLogFileA(text)`. The ANSI string will be converted to Unicode prior to writing to the log file.

To write a Unicode character string to the log file call `svlCVStub_writeToLogFileW(text)`.

Writing error code descriptions to the logfile

To write a human readable description of the SVL_SERVICE_ERROR error code to the log file call `svlCVStub_writeToLogFile(errCode)`.

Writing LastError code descriptions to the logfile

To write a human readable description of the Windows error code to the log file call `svlCVStub_writeToLogFileLastError(errCode)`.

Dumping the PATH environment to the logfile

To write the contents of the PATH environment variable to the log file call `svlCVStub_dumpPathToLogFile()`.

This can be useful if you want to know what the search path is when trying to debug why a DLL wasn't found during an attempt to load the Validator DLL.

7.1.1 Changes to the NT Service API

API changes - February 2018

To make the API easier to use with services we made the following changes:

- Changed the API by adding many debugging functions to allow you to easily log information.
- We also extended the error enumeration to provide additional error status values.
- We also split the function of loading and starting Coverage 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 Coverage Validator when the service is non-trivial in scope.

  Failure to inform the service control manager results in the service being killed by the service control manager because it thinks the service has hung.

  This change in the API is to ensure you get better results from using our software.

What do you need to do to move from the old API to the new API?

Change all SVL_ERROR declarations to SVL_SERVICE_ERROR.

Your previous startup code probably looked like this:

```
SVL_ERROR errCode;

errCode = svlCVStub_LoadCoverageValidator();
```

Change it to this:

```
SVL_SERVICE_ERROR errCode;

errCode = svlCVStub_LoadCoverageValidator();

errCode = svlCVStub_SetServiceCallback(serviceCallback, NULL);

errCode = svlCVStub_StartCoverageValidator();
```

The serviceCallback would look something like this:

```
void serviceCallback(void   *userParam)
{
    static DWORD dwCheckPoint = 1;

    ssStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    ssStatus.dwServiceSpecificExitCode = 0;

    ssStatus.dwControlsAccepted = 0;

    ssStatus.dwCurrentState = dwCurrentState;
    ssStatus.dwWin32ExitCode = dwWin32ExitCode;
    ssStatus.dwWaitHint = dwWaitHint;
    ssStatus.dwCheckPoint = dwCheckPoint++;

    // Report the status of the service to the service control manager.

    return SetServiceStatus(sshStatusHandle, &ssStatus);
}
```

In the code above we have omitted error handling. To see how to use the new logging function with error handling please

Important.

Once your service is running (rather than starting) your service callback should set the appropriate running status `SERVICE_RUNNING` rather than `SERVICE_START_PENDING`.

```
if (!ReportStatusToSCMgr(SERVICE_RUNNING,       // service state
                         NO_ERROR,              // exit code
                         0))                    // wait hint
{
   dwErr = GetLastError();
   if (bLogging)
      svlCVStub_writeToLogFileW(L"ReportStatusToSCMgr:5\r\n");
   goto cleanup;
}
```

An alternative solution is to prevent the service callback from being called once the service status has been set to running.

```
svlCVStub_SetServiceCallback(NULL, NULL);.
```

7.1.2 NT Service API Reference

The API consists of the following functions.

SVL_SERVICE_ERROR Enumeration

```
typedef enum _svlServiceError
{
   SVL_OK,                                                  // Normal behaviour
   SVL_ALREADY_LOADED,                                      // Stub DLL already loaded into serv
   SVL_LOAD_FAILED,                                         // Failed to load stub DLL into serv
   SVL_FAILED_TO_ENABLE_STUB_SYMBOLS,                       // Loaded DLL, but failed to enable
   SVL_NOT_LOADED,                                          // Couldn't unload DLL because DLL n
   SVL_FAIL_UNLOAD,                                         // Couldn't unload DLL because could
   SVL_FAIL_TO_CLEANUP_INTERNAL_HEAP,                       // Couldn't get the internal stub he
   SVL_FAIL_MODULE_HANDLE                                   // Couldn't get the stub DLL handle
   SVL_FAIL_SETSERVICECALLBACK,                             // Couldn't call the set service cal
   SVL_FAIL_COULD_NOT_FIND_ENTRY_POINT,                     // Couldn't find the DLL entry point
   SVL_FAIL_TO_START,                                       // Failed to start the Validator
   SVL_FAIL_SETSERVICECALLBACKTHRESHOLD,                    // Couldn't call the set service cal
   SVL_FAIL_PATHS_DO_NOT_MATCH,                             // Path to service in env vars doesn
   SVL_FAIL_INCORRECT_PRODUCT_PREFIX,                       // Wrong validator
   SVL_FAIL_X86_VALIDATOR_FOUND_EXPECTED_X64_VALIDATOR,     // Found wrong bit depth validator
   SVL_FAIL_X64_VALIDATOR_FOUND_EXPECTED_X86_VALIDATOR,     // Found wrong bit depth validator
   SVL_FAIL_DID_YOU_MONITOR_A_SERVICE_FROM_VALIDATOR,       // Looks like Monitor A Service wasn
   SVL_FAIL_ENV_VAR_NOT_FOUND,                              // Env Var not found
   SVL_FAIL_VALIDATOR_ENV_VAR_NOT_FOUND,                    // Env Var identifying validator not
   SVL_FAIL_VALIDATOR_ID_NOT_SPECIFIED,                     // Validator process not specified
   SVL_FAIL_VALIDATOR_ID_NOT_A_PROCESS,                     // Validator process identified does
   SVL_FAIL_VALIDATOR_NOT_FOUND,                            // Validator process identified does
```

```
} SVL_SERVICE_ERROR;
```

svlCVStub_LoadCoverageValidator

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

To load the Coverage Validator stub `svlCoverageValidatorStub.dll` into your service, use `svlCVStub_LoadCoverageValidator()`, *not* `LoadLibrary()`.

This loads the DLL and sets up a few internal variables in the DLL to ensure that symbols are sent from the stub to the Coverage Validator user interface.

This is necessary because the Coverage 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 Coverage Validator user interface and you won't get meaningful function names in your stack traces.

This function can be used when monitoring:

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

If you are monitoring 32 bit applications with Coverage Validator x64 you should use `svlCVStub_LoadCoverageValidator6432()`.

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

	32 bit Coverage Validator	64 bit Coverage Validator
32 bit service	`svlCVStub_LoadCoverageValidator()`	`svlCVStub_LoadCoverageValidator6432()`
64 bit service	N/A	`svlCVStub_LoadCoverageValidator()`

svlCVStub_LoadCoverageValidator6432

```
extern "C"
SVL_SERVICE_ERROR svlCVStub_LoadCoverageValidator6432();
```

To load the Coverage Validator stub `svlCoverageValidatorStub6432.dll` into your service, use `svlCVStub_LoadCoverageValidator6432()`, *not* `LoadLibrary()`.

This loads the DLL and sets up a few internal variables in the DLL to ensure that symbols are sent from the stub to the Coverage Validator user interface.

This is necessary because the Coverage 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 Coverage Validator user interface and you won't get meaningful function names in your stack traces.

This function should only be used when monitoring 32 bit services or applications with Coverage Validator x64.

svlCVStub_StartCoverageValidator

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

To start Coverage Validator inspecting the service call `svlCVStub_StartCoverageValidator()`.

svlCVStub_StartCoverageValidatorForIIS

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

To start Coverage Validator inspecting IIS call `svlCVStub_StartCoverageValidatorForIIS()`.

Example usage.

svlCVStub_ShutdownCoverageValidator

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

To stop Coverage Validator inspecing the service call `svlCVStub_ShutdownCoverageValidator()`.

This sends the shutting down notification and removes any hooks for your process.

Calling this function is optional. ***You can stop your service without calling this function.***

svlCVStub_UnloadCoverageValidator

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

To unload Coverage Validator call `svlCVStub_UnloadCoverageValidator()`, *do not call* `FreeLibrary()`.

Calling this function is optional. ***You can stop your service without calling this function.***

svlCVStub_SetServiceCallback

```
extern "C"
```

```
SVL_SERVICE_ERROR svlCVStub_SetServiceCallback(serviceCallback_FUNC callback,
                                               void*                userParam);
```

`svlCVStub_SetServiceCallback` is used to setup a service callback that is used to inform the Windows service cor

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

Why is a service callback needed?

When a service is starting, Windows requires the service to inform the Service Control Manager (SCM) that is starting at least every ten seconds.

Failure to do so results in Windows concluding that the service has failed to start, and the service is terminated.

Instrumenting your service may well take *more* than 10 seconds, depending on the complexity and size of your service.

The solution is for *Coverage Validator* to periodically call a user supplied callback from which you can regularly inform the SCM of the appropriate status.

We strongly recommend that you setup a service callback. Not setting a service callback can result in failure of your

Debugging functions

The following functions are provided to help you log information about the progress, success or failure of the NT Service API attaching Coverage Validator to your service.

We strongly recommend that you use these logging functions so that you can understand why Coverage Validator might fail to connect to a service.

To see example usage of these debugging functions please look in `service.cpp` in the `examples\service` directory in the Coverage Validator install directory.

svlCVStub_setLogFileName

```
extern "C"
void svlCVStub_setLogFileName(const wchar_t* fileName);
```

Call `svlCVStub_setLogFileName` to set the name of the filename used for logging.

This function must be called before you can use any of the other debugging functions.

Setting this filename also sets the filename used by some of these API functions - you will find additional logging data from those functions that will help debug any issues with the service.

svlCVStub_deleteLogFile

```
extern "C"
void svlCVStub_deleteLogFile();
```

This function deletes the log file.

svlCVStub_writeToLogFileA

```
extern "C"
void svlCVStub_writeToLogFileA(const char* text);
```

This function writes a standard ANSI character string to the log file.

The ANSI string will be converted to Unicode prior to writing to the log file.

svlCVStub_writeToLogFileW

```
extern "C"
void svlCVStub_writeToLogFileW(const wchar_t* text);
```

This function writes a Unicode character string to the log file.

svlCVStub_writeToLogFile

```
extern "C"
void svlCVStub_writeToLogFile(SVL_SERVICE_ERROR errCode);
```

This function writes a human readable description of the SVL_SERVICE_ERROR error code to the log file.

svlCVStub_writeToLogFileLastError

```
extern "C"
void svlCVStub_writeToLogFileLastError(DWORD errCode);
```

This function writes a human readable description of the Windows error code to the log file.

The errCode parameter is the error code returned from GetLastError().

svlCVStub_dumpPathToLogFile

```
extern "C"
void svlCVStub_dumpPathToLogFile();
```

This function writes the contents of the PATH environment variable to the log file.

This can be useful if you want to know what the search path is when trying to debug why a DLL wasn't found during an attempt to load the Validator DLL.

7.1.3 Troubleshooting

Troubleshooting - Service fails to start

If a service takes too long to start the service control manager kills the service.

The way to stop this is for a service to call `ReportStatusToSCMgr()` to tell the service control manager that the service is still OK.

Coverage Validator can't do this for you as the call requires some data from any earlier call you have made.

The solution is that you provide a callback using `svlCVStub_SetServiceCallback()` that Coverage Validator can call during the process of attaching to the service, and you can call the appropriate function.

Example code to set the callback:

```
	errCode = svlCVStub_SetServiceCallback(serviceCallback,		// the
callback
						NULL);			// some
user data (we don't have any, so set NULL)
	if (bLogging)
	{
		if (errCode != SVL_OK)
		{
			svlCVStub_writeToLogFileW(L"Setting service callback failed.
\r\n");
			svlCVStub_writeToLogFile(errCode);
		}

		svlCVStub_writeToLogFileW(L"Starting Coverage Validator\r\n");
	}
```

Example callback:

```
static void serviceCallback(void  *userParam)
{
	// just tell the Service Control Manager that we are still busy
	// in this example userParam is not used
	//
	// note that prior to the Validator loading it's DLL ssStatus.dwCurrentState
must have been initialised, most likely to SERVICE_START_PENDING
	// you could pass a fixed value here, but it would need to change once the
service has finished starting up so that you don't unintentionally change the service
state
```

```
      // when this callback is called. This callback is called whenever
instrumentation happens (when a DLL is loaded). Thus you can't assume this is only
called during service startup,
      // it may also get called later in the service lifetime.

      ReportStatusToSCMgr(ssStatus.dwCurrentState,    // service state
                          NO_ERROR,                   // exit code
                          3000);                      // wait hint
}
```

We strongly recommend that you set a service callback. It won't harm your program and it will remove any likelihood of your service being killed by the service control manager.

Troubleshooting - Service starts, Coverage Validator gets no data

If you have problems getting Coverage Validator to monitor your service you'll need to find out what's failing.

Until Coverage Validator loads correctly and successfully connects to the graphical user interface you have no way of knowing what is happening.

The solution is to set a log file that Coverage Validator can write status messages to. You can also write your own status messages to this log file.

Set the log file using `svlCVStub_setLogFileName`. Write to it using `svlCVStub_writeToLogFile()`, `svlCVStub_writeToLogFileA()`, `svlCVStub_writeToLogFileW()`.

Then when things are not working as expected take a look at the log file to see the errors. The Coverage Validator will often suggest what the problem is.

We strongly recommend that you configure the log file and use it when working with services. It has saved us a lot of time.

7.2 Working with IIS

Configuring IIS for use with ISAPI

We assume that you are familiar with IIS. This is not a topic we can provide advice for.

That said, we wrote a blog article about configuring IIS for use with ISAPI.

Example ISAPI

We have provided an example ISAPI extension configured for use with Coverage Validator.

This example is provided as source code and project files. You will need to build it yourself, you may need to change an include path to find the appropriate headers. The resulting ISAPI will need to be copied to your website for testing and the website configured to allow the ISAPI to execute (please see the above mentioned blog article for details on that).

You can find the example ISAPI in the **isapiExample** folder in the Coverage Validator installation directory.

Using Coverage Validator with IIS

IIS is a service application. It runs as one of the more restricted applications on Microsoft Windows.

Coverage mapped files created by IIS cannot be opened by user mode programs (Coverage Validator, for example). DLLs, executables and files cannot be opened by IIS except if they are in directories which IIS has access to. These are security measures intended to make your computer secure from attack.

These security measures make it hard for tools like Coverage Validator to work.

- We have to communicate settings information to Coverage Validator via text file
- All DLLs and helper programs we want to use need to be copied to the web root (or a subdirectory within the web root) so that they can be used.
- We need to have our own data transport because our usual high speed memory mapped data transport is not available.

It's also not possible to launch IIS or inject into a running IIS instance.

The only way to work with IIS is by using the NT Service API, and using the `svlCVStub_StartCoverageValidatorForIIS()` function instead of `svlCVStub_StartCoverageValidator()`.

We've provide some example code to show you how to attach and detach from your ISAPI extension.

Workflow

1) Start monitoring your ISAPI by using the Monitor ISAPI dialog.

Launch menu > **IIS** menu > **Monitor ISAPI...**

2) When you have finished interacting with the web pages that use the ISAPI component shutdown IIS, wait for Coverage Validator's status to indicate "Ready" and examine the results.

Launch menu > **IIS** menu > **Stop IIS**

7.3 Example Source Code

Service Example

Example demonstrating how to monitor a service.

Also see the example service that ships with Coverage Validator.

You can find this in the `\examples\service` directory in the Coverage Validator install directory.

Also see the example service and child process that ships with Coverage Validator.

You can find this in the `\examples\serviceWithAChildProcess` directory in the Coverage Validator install directory.

IIS Example

Example demonstrating how to monitor an ISAPI DLL.

Also see the example ISAPI DLL that ships with Coverage Validator.

You can find this in the `\examples\isapiExample` directory in the Coverage Validator install directory.

7.3.1 Example Service Source Code

Where to put your code

When you use the functions to load and unload Coverage 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 Coverage 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 Coverage 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 Coverage 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);
}
```

```
void WINAPI service_main(DWORD   dwArgc,
                         LPTSTR   *lpszArgv)
{
   if (bLogging)
   {
      svlCVStub_setLogFileName(SZLOGFILENAME);
      svlCVStub_deleteLogFile();
   }

   // register our service control handler:

   sshStatusHandle = RegisterServiceCtrlHandler(TEXT(SZSERVICENAME), service_ctrl);
   if (sshStatusHandle != 0)
   {
      DWORD   dwErr = 0;

      // **CV_EXAMPLE** start

      if (bCoverageValidator)
      {
         // load Coverage Validator (but if monitoring a 32 bit service with Coverage

         if (bLogging)
         {
            svlCVStub_writeToLogFileW(_T("About to load Coverage Validator\r\n"));
         }

         SVL_SERVICE_ERROR   errCode;

#ifdef IS6432
         // x86 with x64 GUI
         errCode = svlCVStub_LoadCoverageValidator6432();
#else    //#ifdef IS6432
         // x86 with x86 GUI
         // x64 with x64 GUI
         errCode = svlCVStub_LoadCoverageValidator();
#endif   //#ifdef IS6432
         if (bLogging)
         {
            if (errCode != SVL_OK)
            {
               DWORD   lastError;

               lastError = GetLastError();
               svlCVStub_writeToLogFileW(_T("Coverage Validator load failed. \r\n"));
               svlCVStub_writeToLogFileLastError(lastError);
               svlCVStub_writeToLogFile(errCode);

               svlCVStub_dumpPathToLogFile();
            }
            else
            {
               svlCVStub_writeToLogFileW(_T("Coverage Validator load success. \r\n"));
            }
```

```
         }

         // setup a service callback so that the Service Control Manager knows the ser
         // is starting up even if instrumentation takes longer than 10 seconds (which
         // for a non-trivial application)

         if (bLogging)
            svlCVStub_writeToLogFileW(_T("Setting service callback Coverage Validator\
         errCode = svlCVStub_SetServiceCallback(serviceCallback,     // the callback
                                                NULL);               // some user data
(we don't have any, so set NULL)
         if (bLogging)
         {
            if (errCode != SVL_OK)
            {
               svlCVStub_writeToLogFileW(_T("Setting service callback failed. \r\n"));
               svlCVStub_writeToLogFile(errCode);
            }

            svlCVStub_writeToLogFileW(_T("Starting Coverage Validator\r\n"));
         }

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

               lastError = GetLastError();
               svlCVStub_writeToLogFileW(_T("Starting Coverage Validator failed.
\r\n"));
               svlCVStub_writeToLogFileLastError(lastError);
               svlCVStub_writeToLogFile(errCode);
            }

            svlCVStub_writeToLogFileW(_T("Finished loading Coverage Validator\r\n"));
         }
      }
      else
      {
         if (bLogging)
            svlCVStub_writeToLogFileW(_T("Not using Coverage Validator, DLL will not
be loaded\r\n"));
      }

      // **CV_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
      {
         // do work

         dwErr = ServiceStart(dwArgc, lpszArgv);

         // finished doing work
      }

      // **CV_EXAMPLE** start

      if (bCoverageValidator)
      {
         // unload Coverage Validator here
         // IMPORTANT.
         // Because of the way services work, you can find that this thread which is
trying to gracefully unload
         // Coverage Validator is ripped from under you by the operating system. This
prevents Coverage Validator from
         // removing all its hooks successfully. If Coverage 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 Coverage
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 Coverage Validator somewhere else. The stack trace may be
different, but a fundamental point is the
         // code calling through doexit(), exit() and ExitProcess()
         //
         //NTDLL! 77f64e70()
         //SVLPERFORMANCEVALIDATORSTUB!
         //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()
         //SVLPERFORMANCEVALIDATORSTUB!
         //SVLPERFORMANCEVALIDATORSTUB!
         //SVLPERFORMANCEVALIDATORSTUB!
```

```
            //SVLPERFORMANCEVALIDATORSTUB!
            //SVLPERFORMANCEVALIDATORSTUB!
            //SVLPERFORMANCEVALIDATORSTUB!
            //SVLPERFORMANCEVALIDATORSTUB!
            //SVLPERFORMANCEVALIDATORSTUB!
            //MSVCRT! 78001436()
            //MSVCRT! 780057db()
            //KERNEL32! 77f19fdb()
            //SVLPERFORMANCEVALIDATORSTUB! ExitProcess hook
            //doexit(int 0x00000000, int 0x00000000, int 0x00000000) line 392
            //exit(int 0x00000000) line 279 + 13 bytes
            //mainCRTStartup() line 345
            //KERNEL32! 77f1b9ea()

            svlCVStub_UnloadCoverageValidator();
        }

        // **CV_EXAMPLE** end

        // try to report the stopped status to the service control manager.

        (VOID)ReportStatusToSCMgr(SERVICE_STOPPED, dwErr, 0);
    }

    return;
}
```

7.3.2 Example ISAPI Source Code

Where to put your code

When you use the functions to load and unload Coverage Validator from your service, it is important that you put the function calls in the correct place in your ISAPI extension.

Typically, this means that Coverage Validator is:

- loaded as the first action in the `GetExtensionVersion()` function of your ISAPI extension.
- unloaded in the `TerminateExtension()` function of your ISAPI extension.

Example source code

The source code shown below shows an example `GetExtensionVersion()` and an example `TerminateExtension()` used in an ISAPI, demonstrating where to load and unload Coverage Validator.

This example code logs errors. We strongly recommend that you do this in your example. Because IIS is a protected process that can't communicate to the outside world except via HTTP/HTTPS when anything fails during the loading and start of Coverage Validator the only means we have of

communicating that failure to you is via the log file. Please use the log file, it will make debugging any mistakes very much easier, simpler and quicker than any other method.

This process is almost identical to working with a regular service, except that `svlCVStub_StartCoverageValidator()` is replaced with `svlCVStub_StartCoverageValidatorForIIS()`.

This example assumes the web root is located `C:\\testISAPIWebsite`

Show the C++ example ISAPI functions

```
#include "svlCVStubService.h"
#include "svlServiceError.h"

BOOL WINAPI GetExtensionVersion(HSE_VERSION_INFO *pVer)
{
	// some setup work to define what the extension is

	pVer->dwExtensionVersion = HSE_VERSION;
	strncpy(pVer->lpszExtensionDesc, "Validate ISAPI Extension",
HSE_MAX_EXT_DLL_NAME_LEN);

	// load Validator here

	svlCVStub_setLogFileName(L"C:\\testISAPIWebsite\\svl_CV_log.txt");
	svlCVStub_deleteLogFile();

	SVL_SERVICE_ERROR   errCode;
#ifdef IS6432
	// x86 with x64 GUI
	errCode = svlCVStub_LoadCoverageValidator6432();
#else   //#ifdef IS6432
	// x86 with x86 GUI
	// x64 with x64 GUI
	errCode = svlCVStub_LoadCoverageValidator();
#endif   //#ifdef IS6432
	if (errCode != SVL_OK)
	{
		DWORD   lastError;

		lastError = GetLastError();
		svlCVStub_writeToLogFileW(L"Coverage Validator load failed. \r\n");
		svlCVStub_writeToLogFileLastError(lastError);
		svlCVStub_writeToLogFile(errCode);

		svlCVStub_dumpPathToLogFile();
	}
	else
	{
		svlCVStub_writeToLogFileW(L"Coverage Validator load success. \r\n");

		errCode = svlCVStub_StartCoverageValidatorForIIS();
		if (errCode != SVL_OK)
		{
			DWORD   lastError;
```

```
                    lastError = GetLastError();
                    svlCVStub_writeToLogFileW(L"Starting Coverage Validator failed.
\r\n");
                    svlCVStub_writeToLogFileLastError(lastError);
                    svlCVStub_writeToLogFile(errCode);
                }

                svlCVStub_writeToLogFileW(L"Finished starting Coverage Validator\r\n");
        }

        return TRUE;
}


BOOL WINAPI TerminateExtension(DWORD     dwFlags)
{
        // unload Validator here

        svlCVStub_UnloadCoverageValidator();

        return TRUE;
}
```

Part

8 Working With VBUnit

This page gives information about using Coverage Validator with programs that use VBUnit.

About VBUnit

VBUnit works by spawning a worker service process, `vbUnitTestServer.exe` which works in conjunction with the main process `RunVBUnit.exe`.

Because `vbUnitTestServer.exe` is a service and is not launched directly using `CreateProcess` from `RunVBUnit.exe` we can't monitor and hook this process.

This means that to get the Coverage Validator stub dll into your Visual Basic process you'll have to load the stub dll yourself.

At the time of writing, the current VBUnit is VBUnit3.

Using Coverage Validator with VBunit

There are two steps: preparing the executable and running the test.

Step 1: Modifying the VB DLL/EXE

To load the Coverage Validator stub dll into your Visual Basic process do the following steps:

1) Copy **`svlCoverageValidatorStub.dll`** to the same directory (or any directory on the **`$PATH`**) as the Visual Basic executable (or DLL) you wish to test.

2) Copy **`DbgHelp.dll`** from the Coverage Validator install directory to the same directory (or any directory on the **`$PATH`**) as the Visual Basic executable (or DLL) you wish to test.

> Don't copy the **`DbgHelp.dll`** from elsewhere as you may get an earlier version of the **`DbgHelp.dll`** and not be able to read symbols as a result.

3) Modify the start of your test DLL or exe so that the first thing it does is load the Coverage Validator stub DLL.

> Do this as follows:
>
> a) Add these lines to the *start* of your Visual Basic code.

```
   Private Declare Function FreeLibrary Lib "kernel32" (ByVal hLibModule As Long)
As Long
   Private Declare Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal
lpLibFileName As String) As Long
```

b) Add these lines where you wish to *load* the Coverage Validator DLL.

```
Dim lbCVStub As Long

lbCVStub = LoadLibrary("svlCoverageValidatorStub")
```

c) Add this line where you wish to *unload* the Coverage Validator DLL.

```
FreeLibrary lbCVStub
```

This step is optional, but do it as close to the end of the execution of your DLL or EXE as possible.

4) Ensure each VB exe or DLL has been built with debug symbols. Debug symbols are required so that Coverage Validator can monitor each line visit.

Do this as follows:

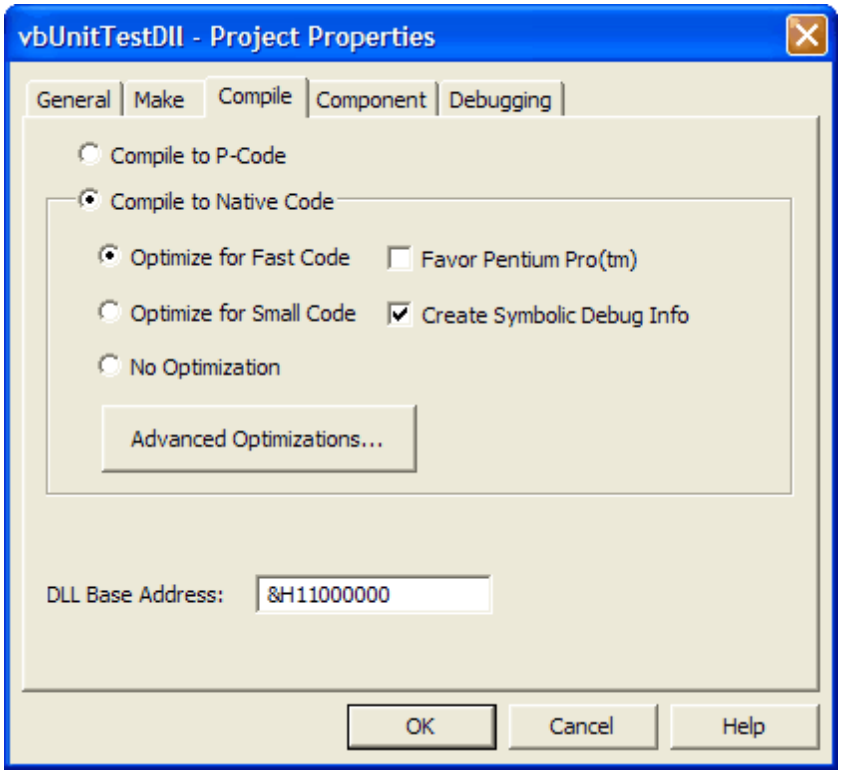
a) Open the Visual Studio properties dialog for the project. **Project** Menu › **Properties...**

b) Go to the **Compile** tab.

c) Select the **Compile to Native Code** radio box.

d) Check the **Create Symbol Debug Info** check box. Click **OK**.

e) Make the project. **File** Menu › **Make** [name of project].



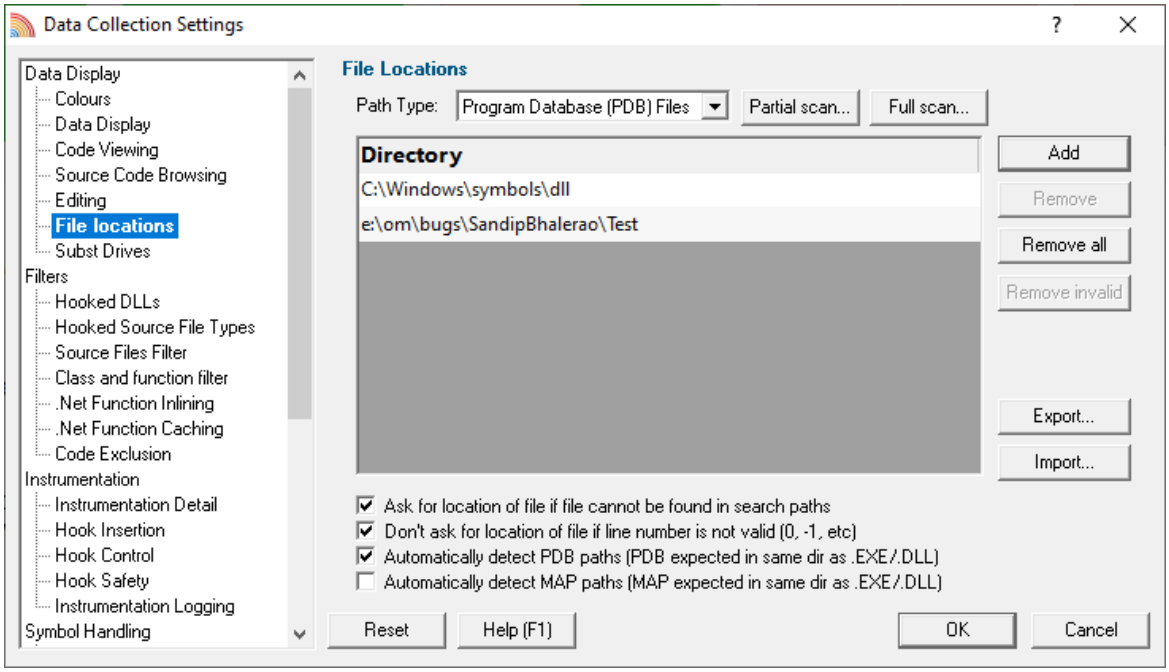
Step 2: Setting up Coverage Validator

1) Setup where the Visual Basic PDB files are stored

Do this as follows:

a) Open the settings dialog.

b) Go to **"File Locations"**.



c) Choose "**Program Database (PDB) Files**" in the combo box.

d) Click **Add**. Enter the directory name where the PDB files are located.

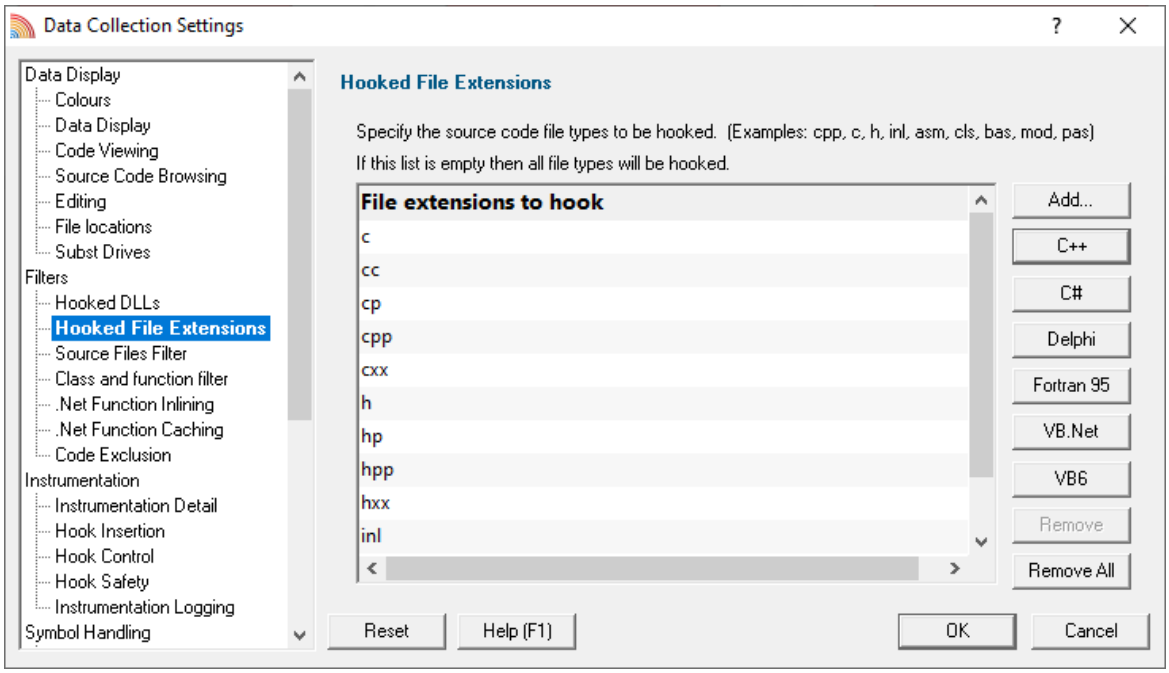
e) Click **OK**.

2) Setup Visual Basic file associations.

Do this as follows:

a) Open the settings dialog.

b) Go to **"Hooked File Extensions"**.



d) Check if the file extensions "cls" and "bas" are present, or remove all file types, If no file types are present every type of file will be instrumented.

e) For any file extension that is not present click **Add** then enter the extension.

e) Click **OK**.

Step 3: Running the test

To run the test do the following:

1) Start Coverage Validator.

2) Start your Unit tests from your command line or batch file.

When the Coverage Validator DLL loads into your DLL/EXE it will instrument your DLL/EXE.

It will then contact the Coverage Validator UI and proceed as if you had launched the unit tests from Coverage Validator.

Part

9 Working with Visual Basic 6 (VB6)

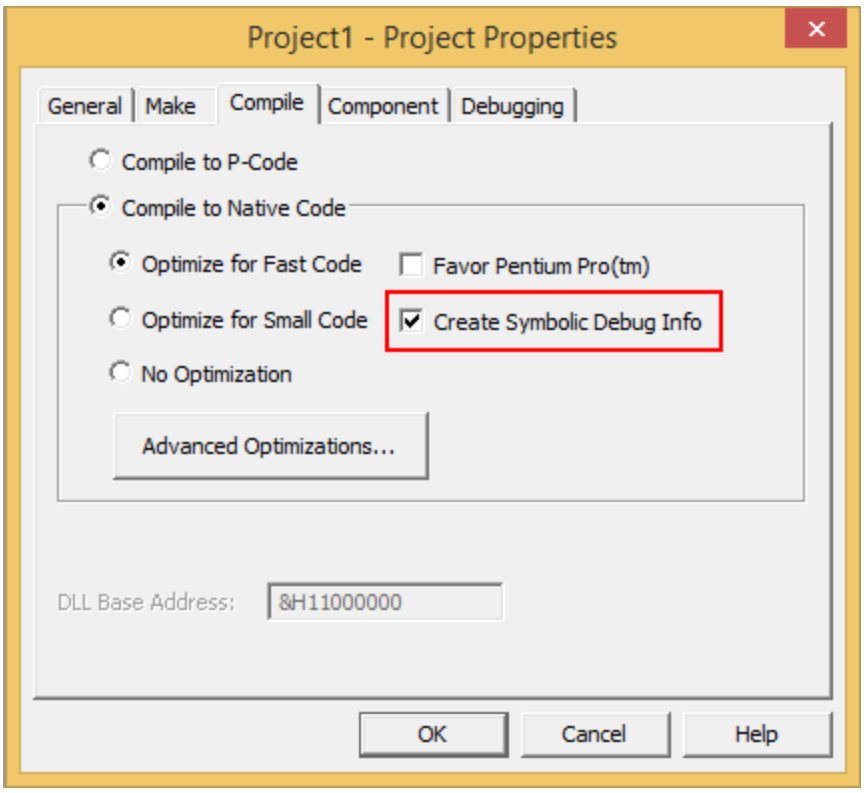
To work with VB6 you need to make two simple changes to your VB6 way of working.

Enable debug information

Debug information needs to be generated to enable Coverage Validator to be able to report function names, filename and line numbers.

To do this, in Visual Basic 6, go to the **Project** menu, choose **Project Properties...**.

Then go to the **Compile** tab and select **Create Symbolic Debug Info**. Click **OK**.



Compile your program into an executable

The next step is to build your program as an executable.

From the **File** menu, choose **Make <name-of-program.exe>**.

Your program will be compiled as an executable. Another file will be created with the same name, but instead of .exe (or .dll) as an extension, the extension will be .pdb. This file contains the debugging information.

For example *test.exe* will create a debugging information file called *test.pdb*.

If you always keep the pdb file in the same directory as the .exe Coverage Validator will be able to find the debugging symbols.

Part

10 Examples

The need for examples

We know Coverage 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 Coverage Validator in an easy and repeatable way.

Having said that, Coverage Validator provides coverage information and doesn't detect error conditions, so the example program is quite simplistic.

The example application provides a safe test demonstration:

- It lets you trigger events in your own time so you can observe coverage changes
- 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 Coverage 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.

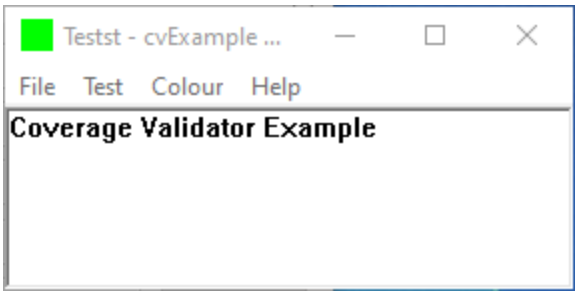
10.1 Example application

The example application

The example application is a great way to explore the capabilities of Coverage Validator.

The source and projects are included in the installation, but you'll need to build the example application yourself.

You can then use nativeExample.exe in conjunction with Coverage Validator to monitor the coverage of the application as you use it.



How to use these examples

The best way to understand how Coverage Validator works is by example.

We recommend launching the example application from Coverage Validator and observing how the menu actions affect coverage information.

Examining the source code is the best way to see what's going on in the example application.

Resetting the statistics before and between using the menu items is a good way to easily see exactly what new code was executed and marked as included in the coverage.

For convenience, below we have provided the source locations where each menu action runs a test.

Most test locations are in the `CTeststakView` class of `nativeExample\TESTSVW.CPP`

File menu

File menu › **Exit** › closes the example application, which itself increases the code coverage.

Test menu

Test menu › **...**

› **Test**	`OnTestPerformtest()` Calls a small number of test functions
› **Test2**	`OnTestTest2()` Calls a small number of other test functions
› **LoadLibrary Test...**	`OnCommandLoadLibrary()` Opens a file browser for you to choose a test dll to load but doesn't c
› **FreeLibrary Test**	`OnCommandFreeLibrary()` Frees up any previously loaded library above

Colour menu

Colour menu › **...**

Red, Green, Blue, Cyan, Magenta, Yellow	`OnColourRed(), OnColourGreen(), etc` Sets the background colour of the main window and repaints the window
Reset	`OnColourReset()` Calls the same method that accesses two more different methods
Use if() statements	`OnColourUseifstatements()` Affects the code path used when applying the background colours at
Use switch() statements	`OnColourUseswitchstatements()` Affects the code path used when applying background colours

Help menu

Help menu > **...**

About Coverage Validator Tester...	`CTeststakApp::OnAppAbout(), etc` Shows a simple help dialog using code in nativeExample.cpp

10.1.1 Building the example application

Where to find the example application

The example project is in the **examples\nativeExample** sub-directory of the Coverage Validator installation directory.

If the directory is not present, reinstall your software and choose *custom* or *full* installation to include the examples.

Solutions and projects

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

- **nativeExample_VSx_x.sln** > for Microsoft® Visual Studio / .net
- **nativeExample.dsp** > for Microsoft® Developer Studio® 6.0

Configurations

There are a small number of configurations in each project:

- **Debug Non Link** / **Release Non Link** > with the `wWinMainCRTStartup` unicode entry point
- **Debug Non Link ANSI** / **Release Non Link ANSI** > without the 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, Coverage Validator will still work with it just fine.

10.2 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 Coverage Validator with NT Services.

The following tasks are performed when the service is started:

- Loads the Coverage Validator stub DLL into the service
- Performs the normal work of the service until it's stopped
- Unloads the Coverage Validator stub DLL from the service
- Informs the service control manager that a stop is pending

➡ Read more about working with NT Services.

10.2.1 Building the example service

Example service project files

The example project can be found in the `service` sub-directory in the directory where Coverage 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 `svlCVStubService(_x64).lib` demonstrating use with the NT Service API

Using the service

The service is named C**V Simple Service** in the control panel services dialog, and provides the following command line options:

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

Open a cmd prompt in administrator mode, navigate to the location of the service executable, and use one of these commands to install, remove, start, stop the service.

Examples:

```
serviceCV.exe -install

serviceCV.exe -start

serviceCV.exe -stop

serviceCV.exe -remove
```

10.2.2 Building the example client

If you've already built the sample service, the process is very similar

Project files

The example project can be found in the `serviceClient` sub-directory in the directory where Coverage 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 a small number of configurations in each project:

- **Debug / Release** › dynamically links to the `svlCVStubService(_x64).lib` demonstrating use with the NT Service API

Using serviceClient

The service is named **CV Simple Service** in the control panel services dialog, and provides the following command line options:

- **-string** › Sends the following (optionally quoted) text to the service. If the service is running the service will return the string in reverse order

  For example: `serviceClient.exe -string "The quick brown fox"` returns `"xof nworb kciuq ehT"`

- **-help** › Display the help message

10.2.3 Building the example service utility

The serviceMutex project demonstrates a way of controlling whether Coverage 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 Coverage Validator was installed.

If the directory is not present, reinstall your software and choose custom or full installation.

There are two project files in the directory:

- **serviceMutex.dsp** › for Microsoft® Developer Studio® 6.0
- **serviceMutex.vcproj** › for Microsoft® Visual Studio / .net

Configurations

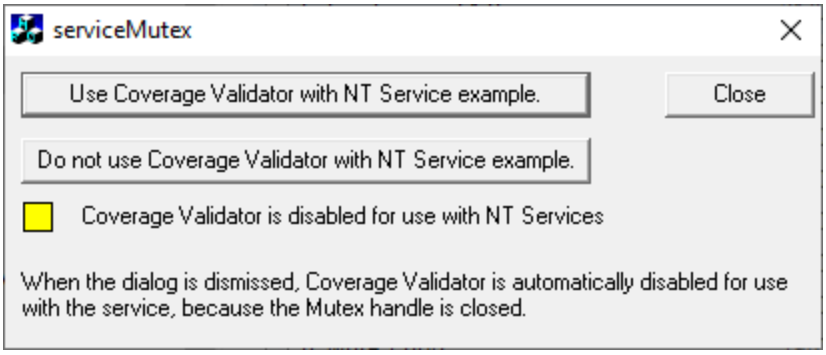
There are just two configurations in each project:

- **Debug / Release** › dynamically links to the `svlCVStubService(_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 Coverage 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.

10.2.4 Monitoring the service

Once the example service and example client has been built, the next step is to test them using Coverage Validator.

Installing the service

If you haven't installed the service, do the following:

- open an administrator mode cmd prompt
- navigate to the directory containing the serviceCV.exe to install
- serviceCV.exe -install

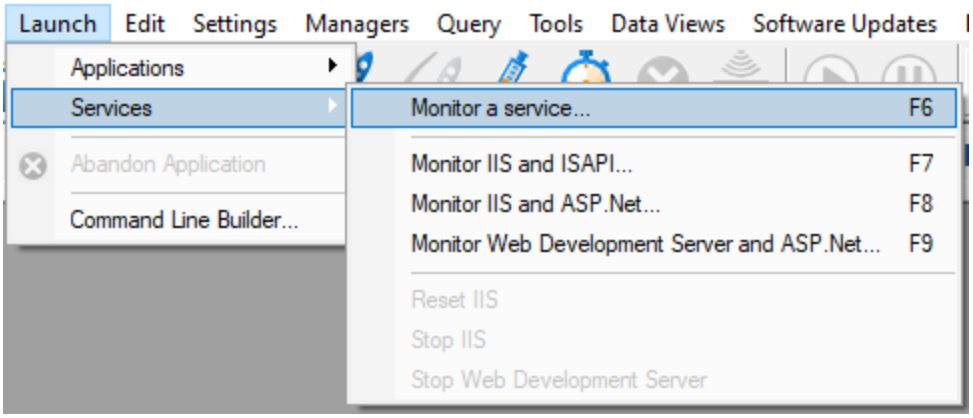
Monitoring the service

Prerequisites

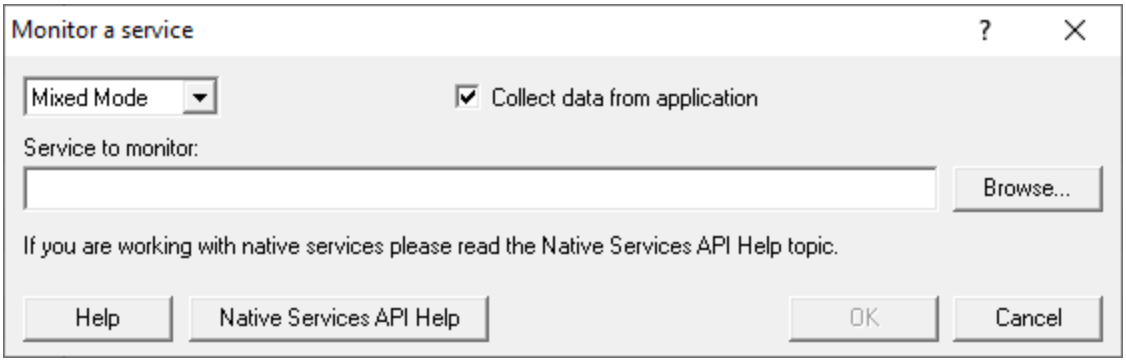
- example service has been installed, but not started (if service has been started, stop the service)
- example service and example client have been built

The following process is used to monitor the application launched by the service:

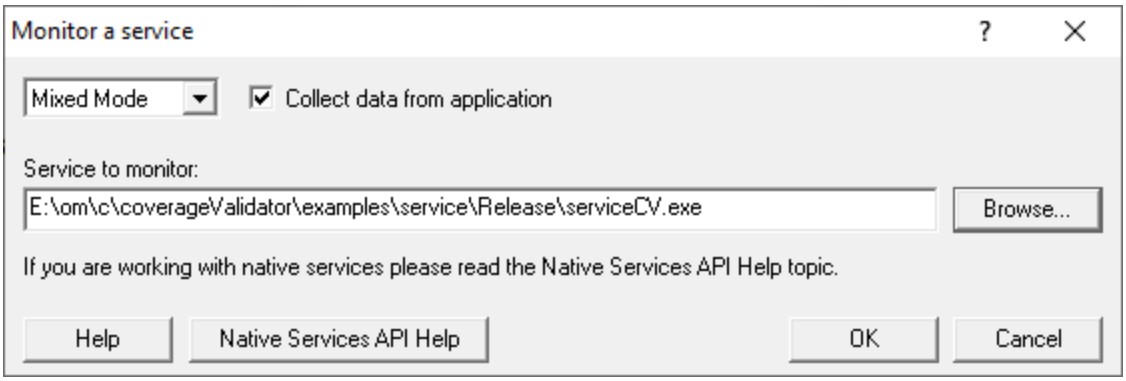
- From the Launch menu choose `Services > Monitor a Service...`



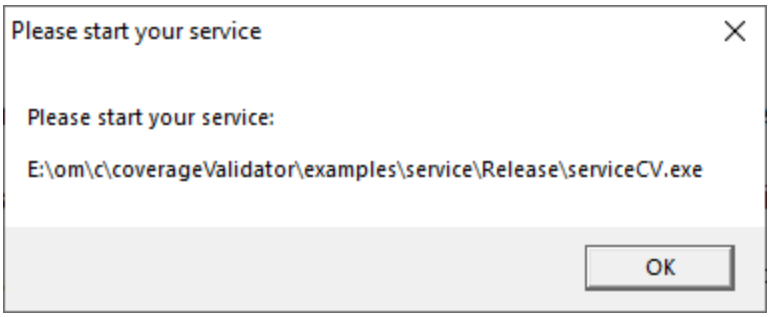
- The Monitor a service dialog is displayed



- Use **Browse...** to open the file chooser dialog and choose the service that will be monitored by Coverage Validator.



- Click **OK**
- Coverage Validator sets up a variety of parameters then displays a dialog box asking you to start you service. Click **OK** to dismiss the dialog



- Start your service. For the example `serviceCV.exe` do the following
  - open an administrator mode cmd prompt
  - navigate to the directory containing the `serviceCV.exe` to start
  - `serviceCV.exe -start`
  - `serviceCV.exe` starts will be monitored by Coverage Validator
- The target application contacts Coverage Validator
- Data is collected until the service finishes executing
- Coverage Validator displays the results

10.3 Example Application Launched from a Service

The example Application launched from a Service

This pair of projects create an application that is launched from a service.

The purpose of this example is to show how to monitor the application that is launched from the service. This is also the same process for monitoring an application launched by an application launched from a service.

This process is subtly different to the method for working with services (see the example service for that).

Service

The service project is `serviceWithAChildProcess.vcxproj`

The following tasks are performed when the service is started:

- the test application is launched from the service

Application

The application project is `serviceChildProcess.vcxproj`

The application's first task is to load Coverage Validator into the application.

- Loads the Coverage Validator stub DLL into the application
- Configures the NT Service API to communicate to Coverage Validator
- Does some work that can be monitored by Coverage Validator
- Exits

Implementation Details

For implementation details see `attachToCoverageValidator();` in `serviceChildProcess.cpp`.

The application will need to link to the NT Service API, for example `..\..\..\svlCVStubService\release_2010_x64\svlCVStubService_x64.lib` (for a release x64 EXE/DLL).

Important. Call `attachToCoverageValidator() as close to the start of your application as possible, before any threads have been created.`

➡ Read more about working with NT Services.

10.3.1 Building the service and application

Example solution files

The example solution can be found in the `examples\serviceWithAChildProcess` subdirectory in the directory where Coverage Validator was installed.

If the directory is not present, reinstall your software and choose custom or full installation.

Example project files

The example projects can be found in the subdirectories in the directory where Coverage Validator was installed.

`examples\serviceWithAChildProcess\serviceWithAChildProcess`

- **serviceWithAChildProcess.vcproj** › for Microsoft® Visual Studio / .net

`examples\serviceWithAChildProcess\serviceChildProcess`

- **serviceChildProcess.vcproj** › for Microsoft® Visual Studio / .net

Configurations

There are a small number of configurations in each project:

- **Debug / Release** › dynamically links to the `svlCVStubService(_x64).lib` demonstrating use with the NT Service API

Using the service

The service is named **SVL *** CV Child Process** in the control panel services dialog (*** changes depending on the build configuration), and provides the following command line options:

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

Open a cmd prompt in administrator mode, navigate to the location of the service executable, and use one of these commands to install, remove, start, stop the service.

Examples:

```
serviceWithAChildProcess.exe -install

serviceWithAChildProcess.exe -start

serviceWithAChildProcess.exe -stop

serviceWithAChildProcess.exe -remove
```

10.3.2 Monitoring the application launched from the service

Once the example service and example application are built, the next step is to test them using Coverage Validator.

Installing the service

If you haven't installed the service, do the following:

- open an administrator mode cmd prompt
- navigate to the directory containing the serviceWithAProcess.exe to install
- serviceWithAProcess.exe -install

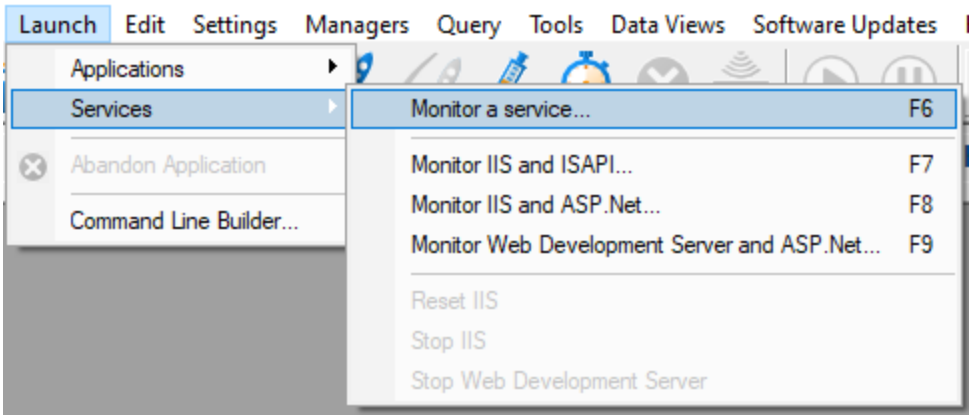
Monitoring the application launched by the service

Prerequisites

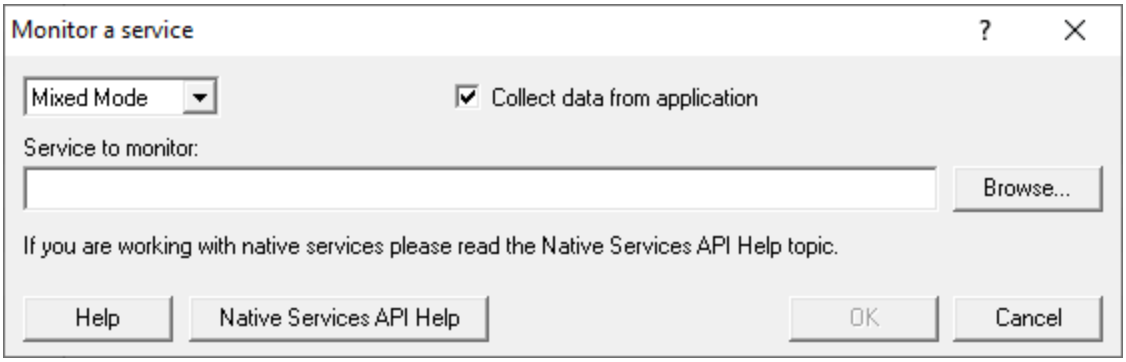
- example service has been installed, but not started (if service has been started, stop the service)
- example service and example application have been built (application must use the NT Service API as demonstrated in attachToCoverageValidator())
- example application executable is in the same directory as the example service (this is only a requirement for the example)

The following process is used to monitor the application launched by the service:

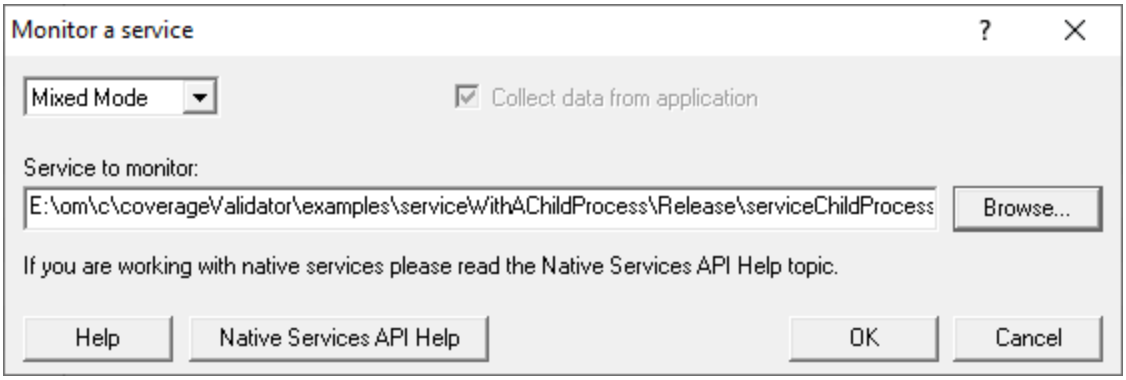
- From the Launch menu choose `Services > Monitor a Service...`



- The Monitor a service dialog is displayed

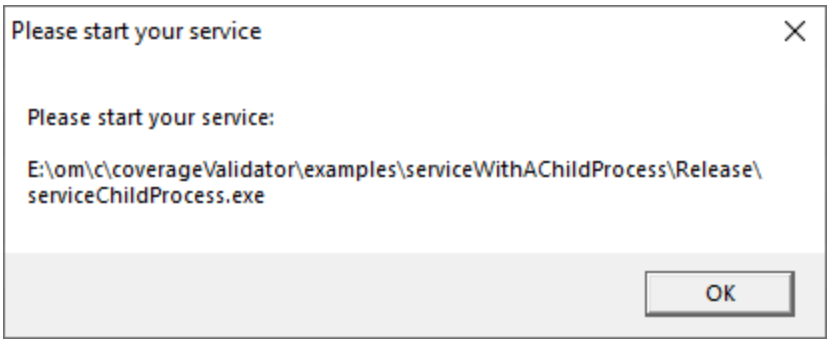


- Use **Browse...** to open the file chooser dialog and choose the application that will be monitored by Coverage Validator. This is the application that is launched by the service. Do not choose the service



- Click **OK**

- Coverage Validator sets up a variety of parameters then displays a dialog box asking you to start you service. Click **OK** to dismiss the dialog



- Start your service. For the example `serviceWithAChildProcess.exe` do the following
  - open an administrator mode cmd prompt
  - navigate to the directory containing the `serviceWithAProcess.exe` to start
  - `serviceWithAProcess.exe -start`
  - `serviceWithAProcess.exe` starts and launches the child process `serviceChildProcess.exe` that will be monitored
- The target application contacts Coverage Validator
- Data is collected until the target process finishes executing
- Coverage Validator displays the results

Part

11 Debug Information, Symbols, Filenames, Line Numbers

Depending on which IDE or compiler/linker combination the process to create debug information to ensure that you have symbols, filenames and line numbers is different.

This section shows you what to do to ensure you have symbols for your compiler and linker.

11.1 Visual Studio

Enabling debug information in Visual Studio has changed over the years depending on the version of Visual Studio you are using.

It's generally the same, but there have been some changes in recent versions that can cause confusion.

By default debug configurations create debug information, but for some versions of Visual Studio, release configurations do not create debug information.

You need to set both compiler and linker settings to get debug information. ***Setting just one or the other will not give you debug information you can use.***

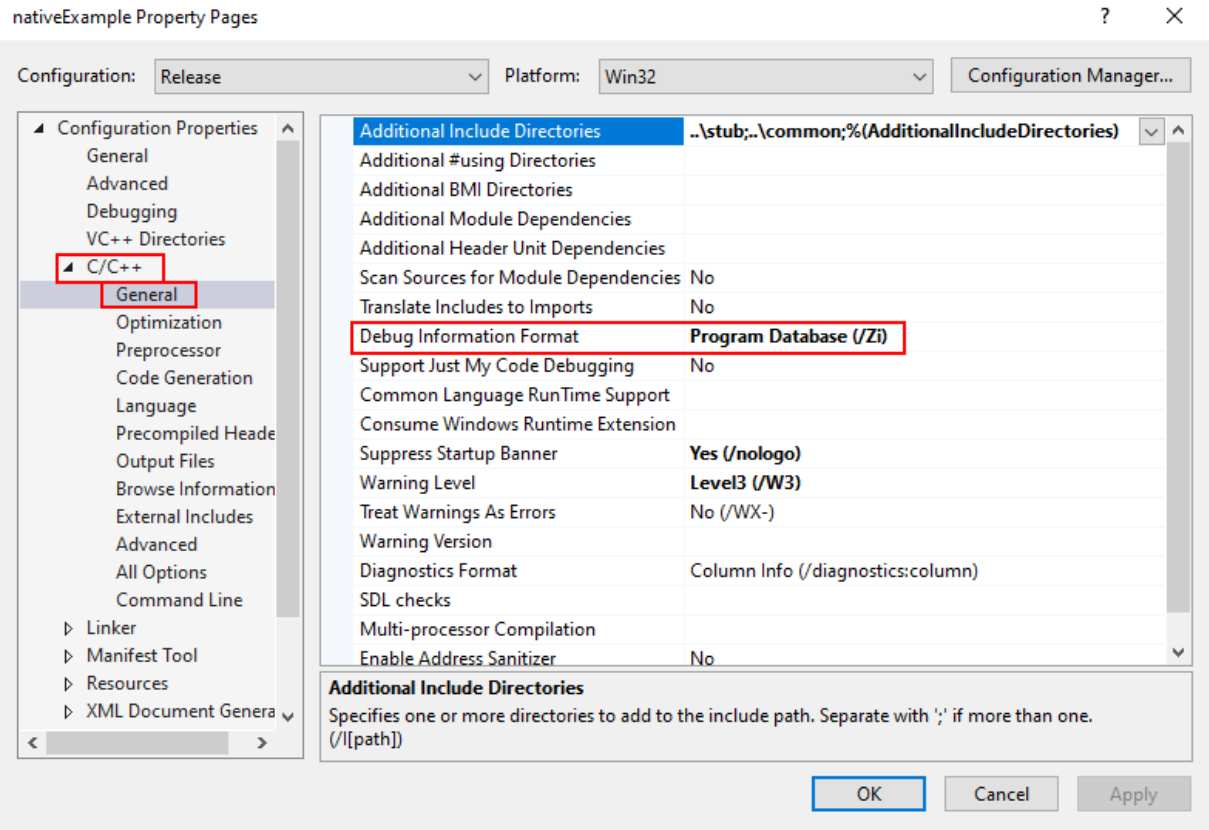
Configurations

In the help below we show you how to modify one configuration, for example Release | Win32.

You need to modify all configurations appropriately. Release, Debug, Win32, Win64 and any other configurations you are using.

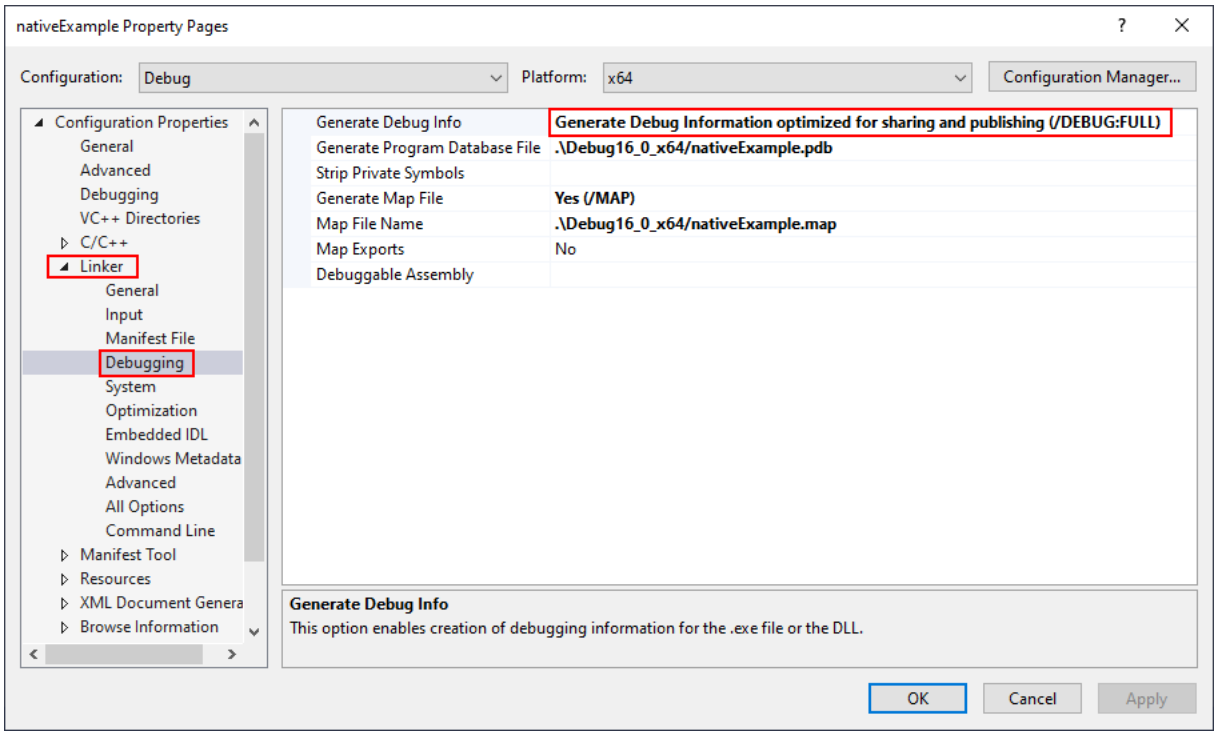
Visual Studio 2017 - 2021

Compiler Settings



Linker Settings

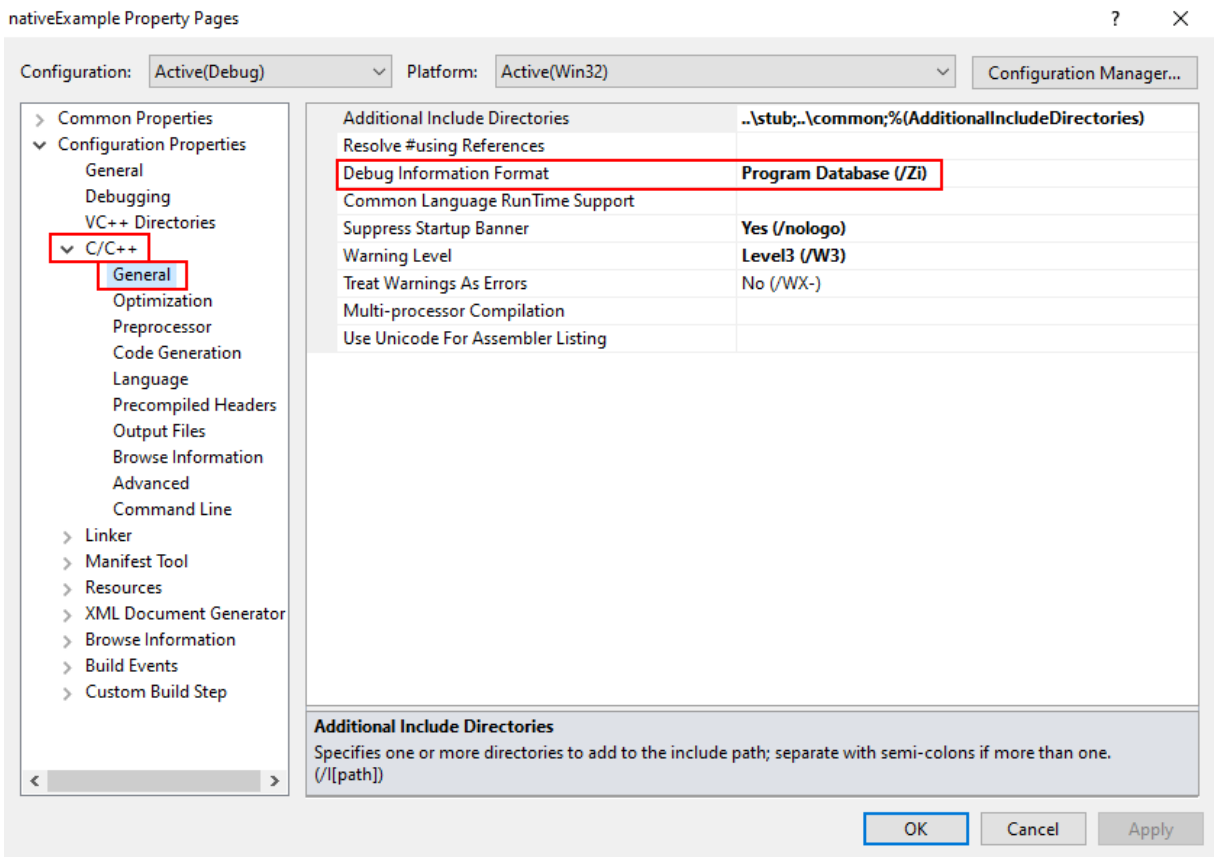
If you're building on a different machine to the machine you're working on (for example a build server), you should choose **/DEBUG:FULL**, not /DEBUG or /DEBUG:FASTLINK.



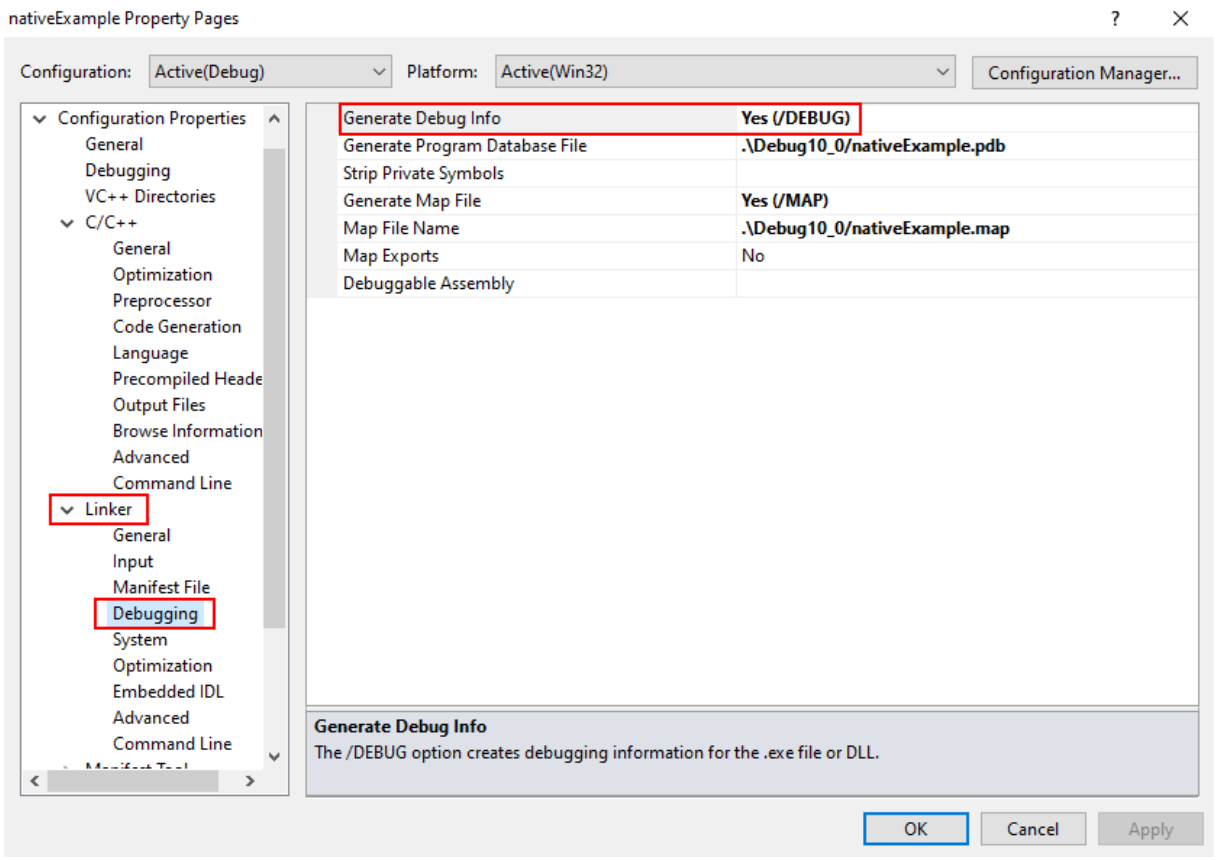
When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Visual Studio 2010 - 2015

Compiler Settings



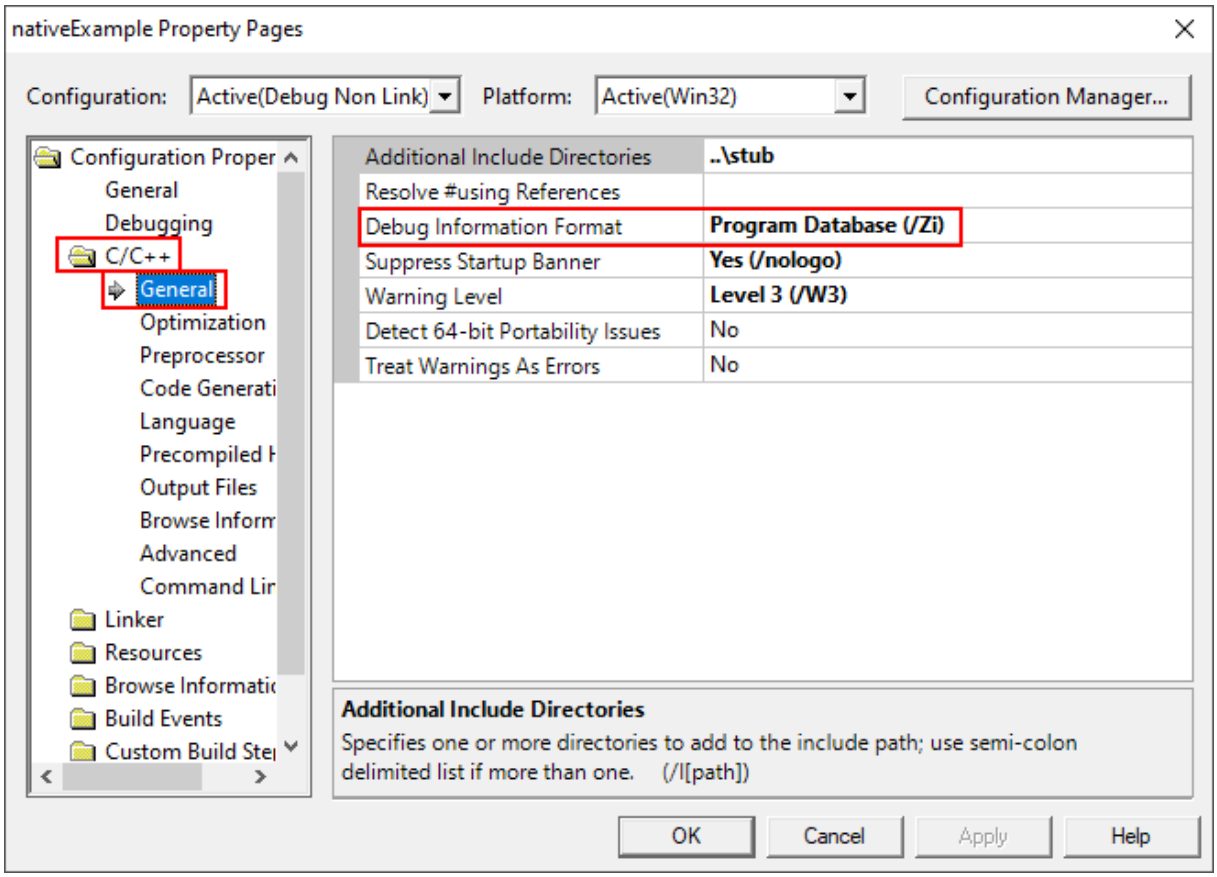
Linker Settings



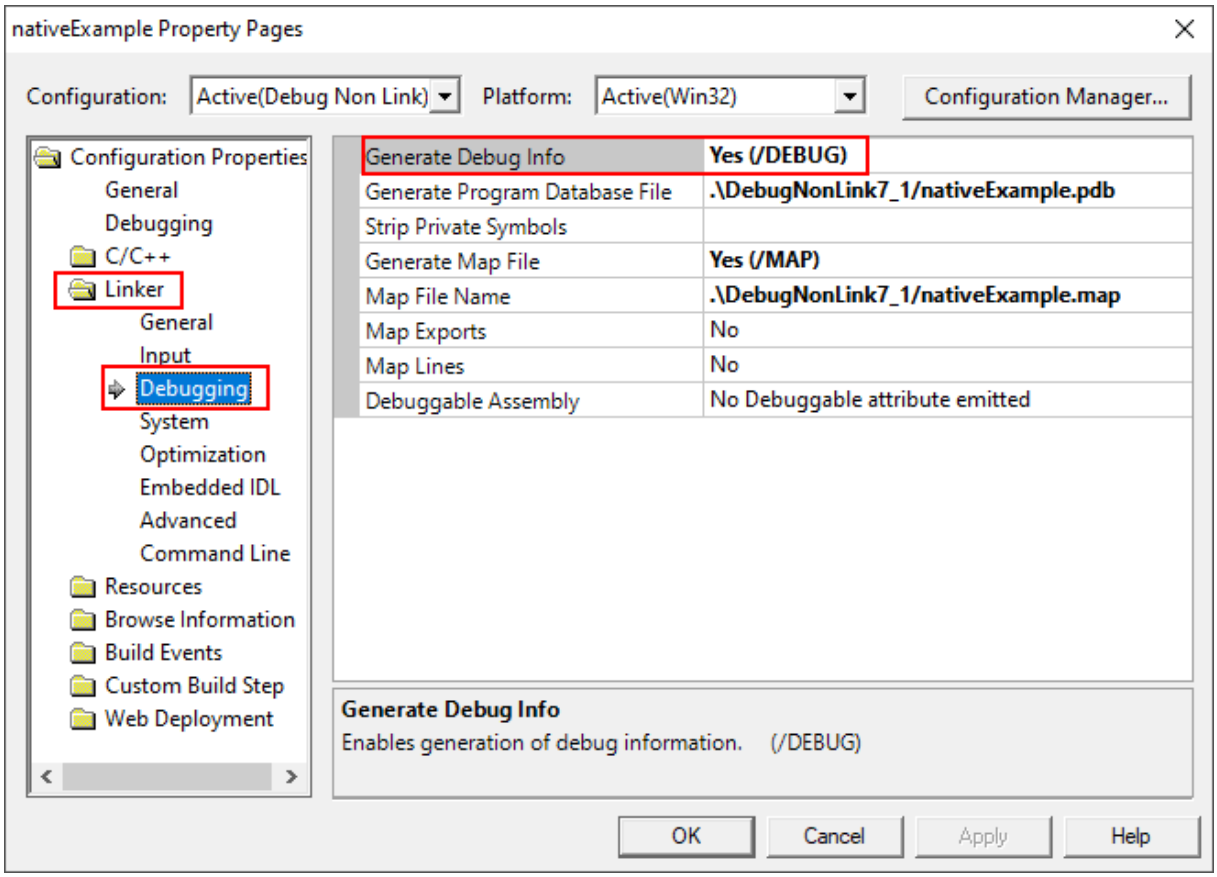
When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Visual Studio 2002 - 2008

Compiler Settings



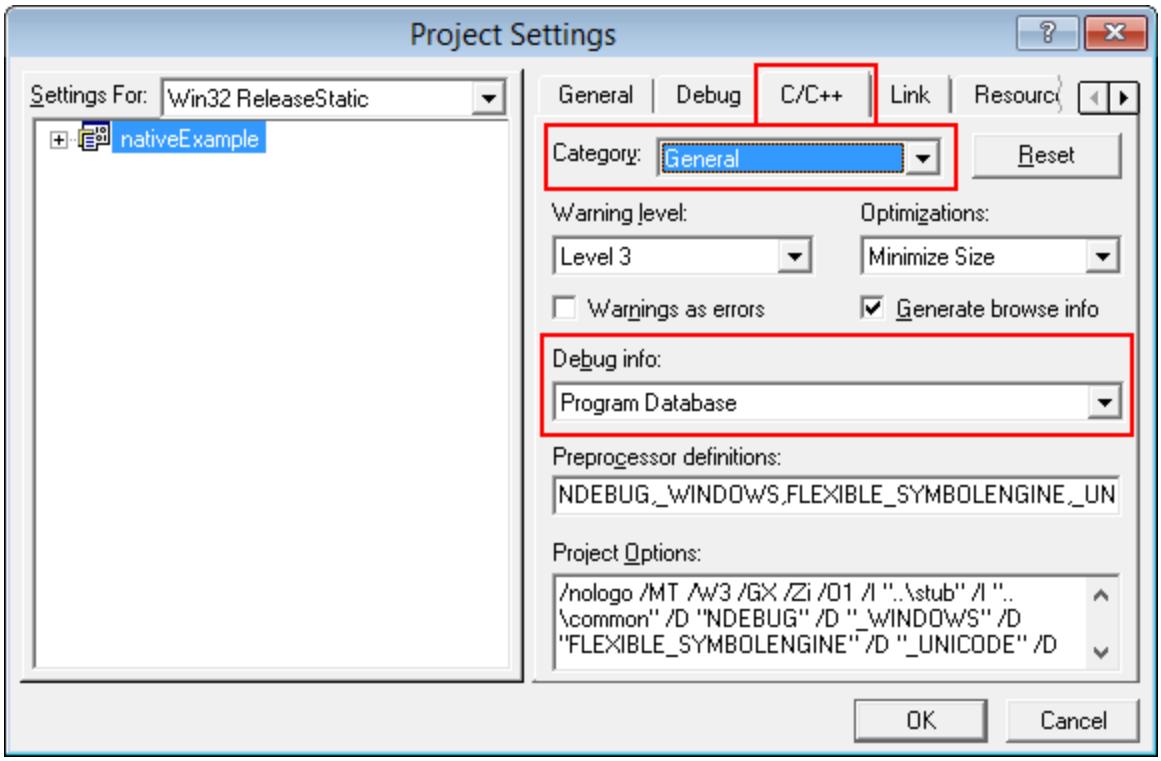
Linker Settings



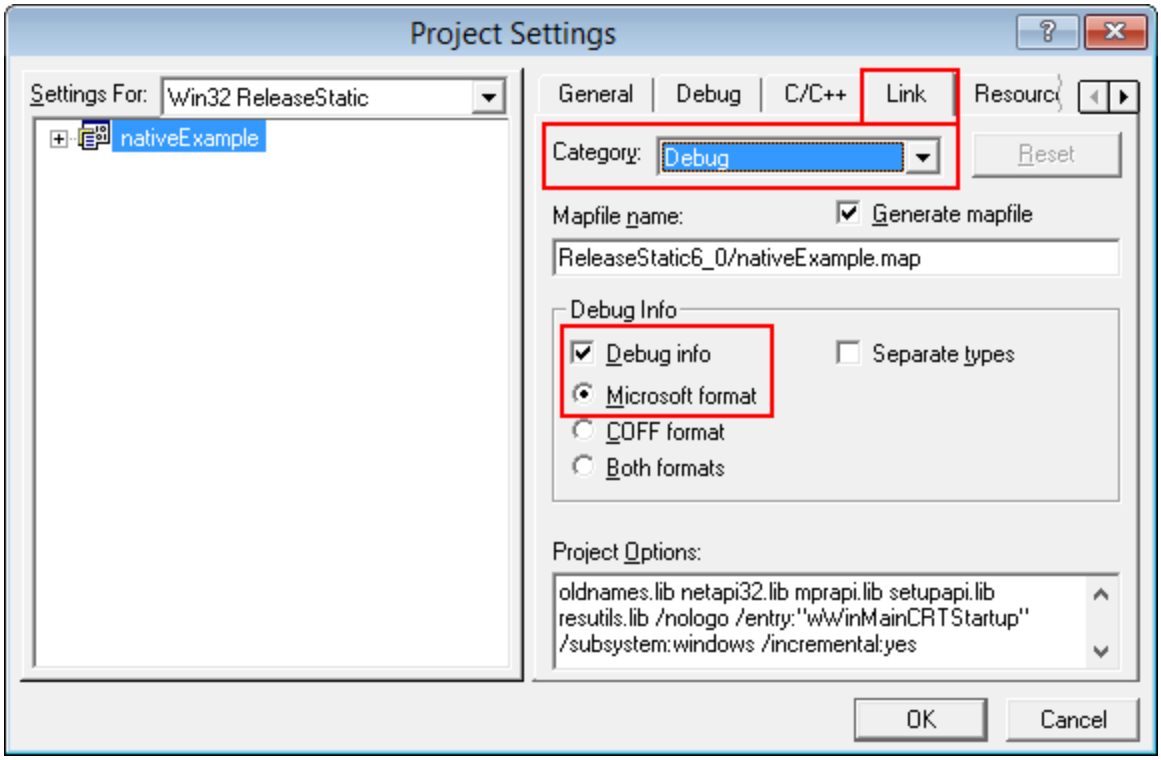
When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Visual Studio 6.0

Compiler Settings



Linker Settings



When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

11.2 C++ Builder

Debug information can be provided using two methods.

- Debugging information (TDS or DWARF format)
- MAP files

Debugging Information

Debug configurations of C++ Builder projects automatically generate debug information that provides symbols, filenames and line numbers.

However the release configurations of C++ Builder projects do not automatically generate debug information. You need to configure that yourself.

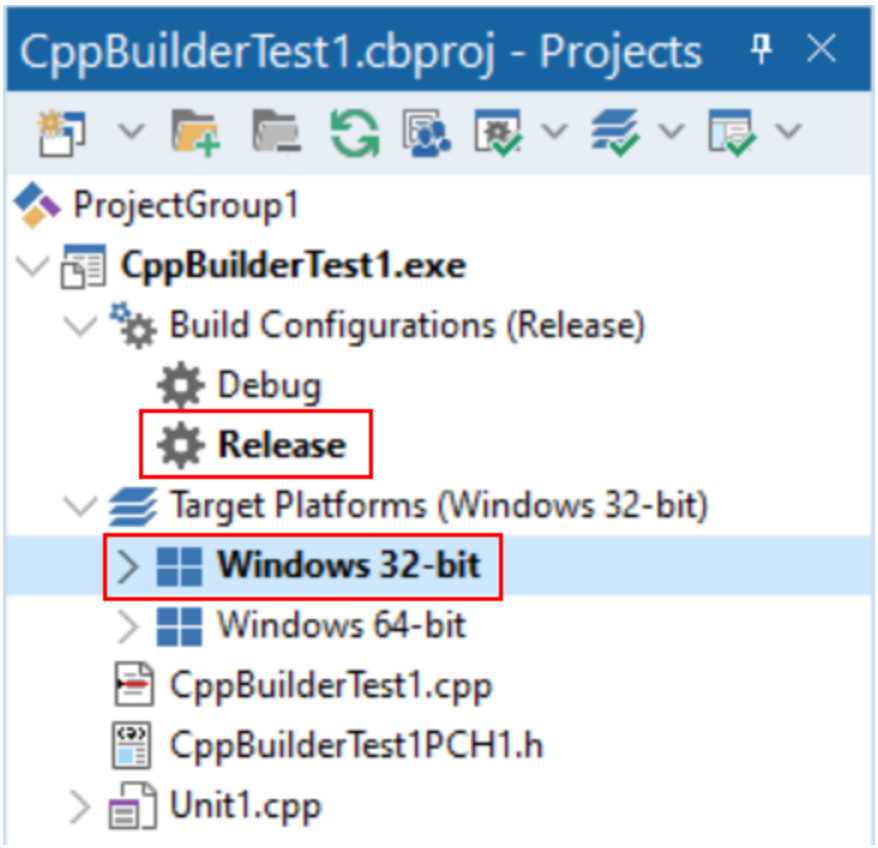
Here's how you do that. It's slightly different if you're building 32 bit applications rather 64 bit applications.

You need to set both compiler and linker settings to get debug information. Setting just one or the other will not give you debug information you can use.

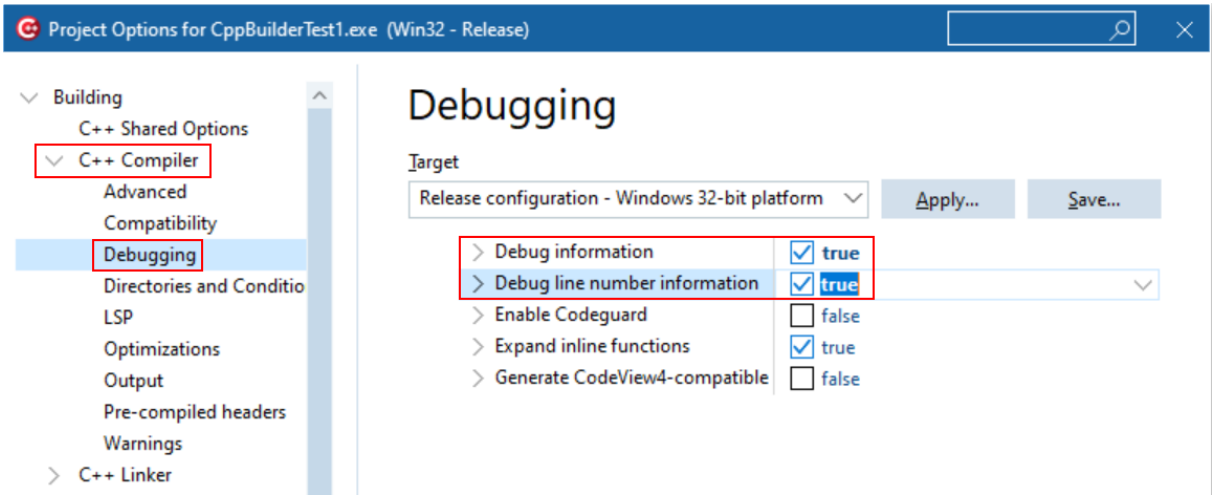
32 bit C++ Builder

Project Configuration

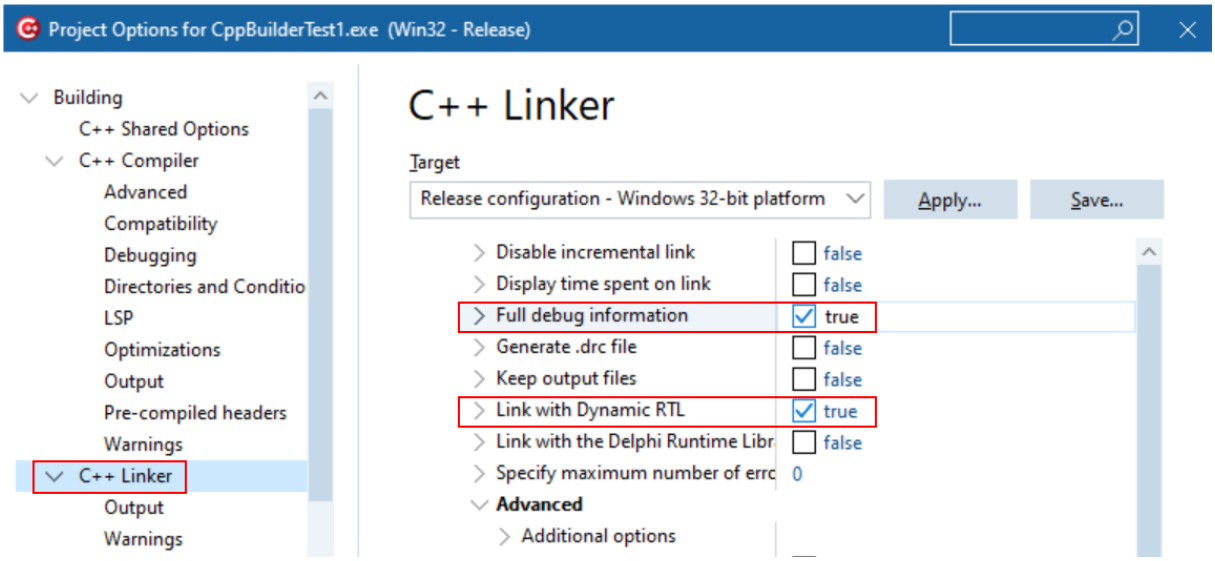
Change your project settings to target 32 bit builds.



Compiler Settings



Linker Settings

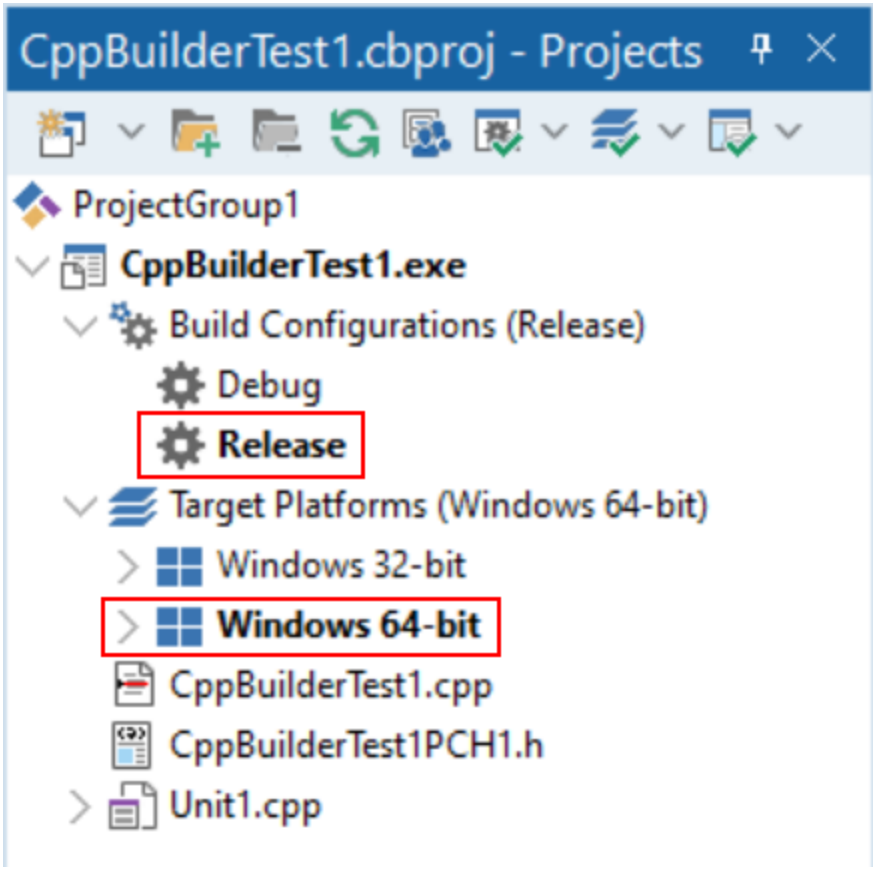


When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

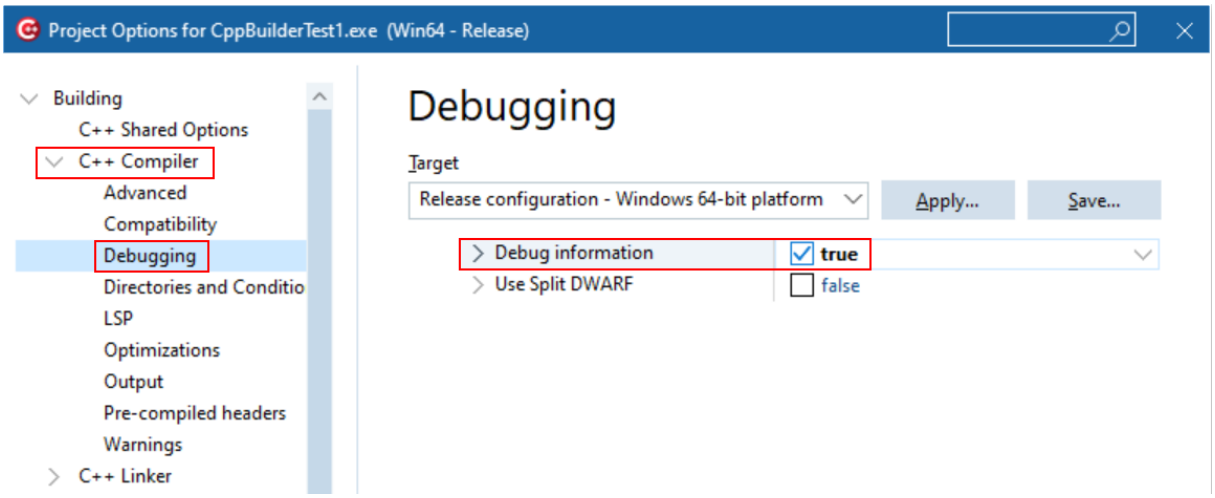
64 bit C++ Builder

Project Configuration

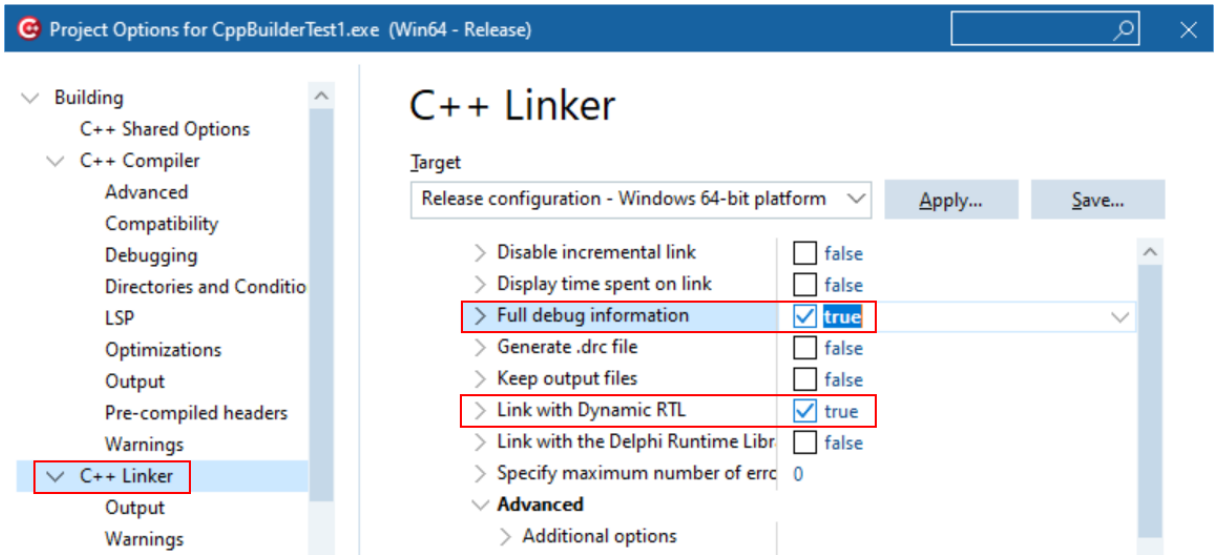
Change your project settings to target 64 bit builds.



Compiler Settings



Linker Settings



When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

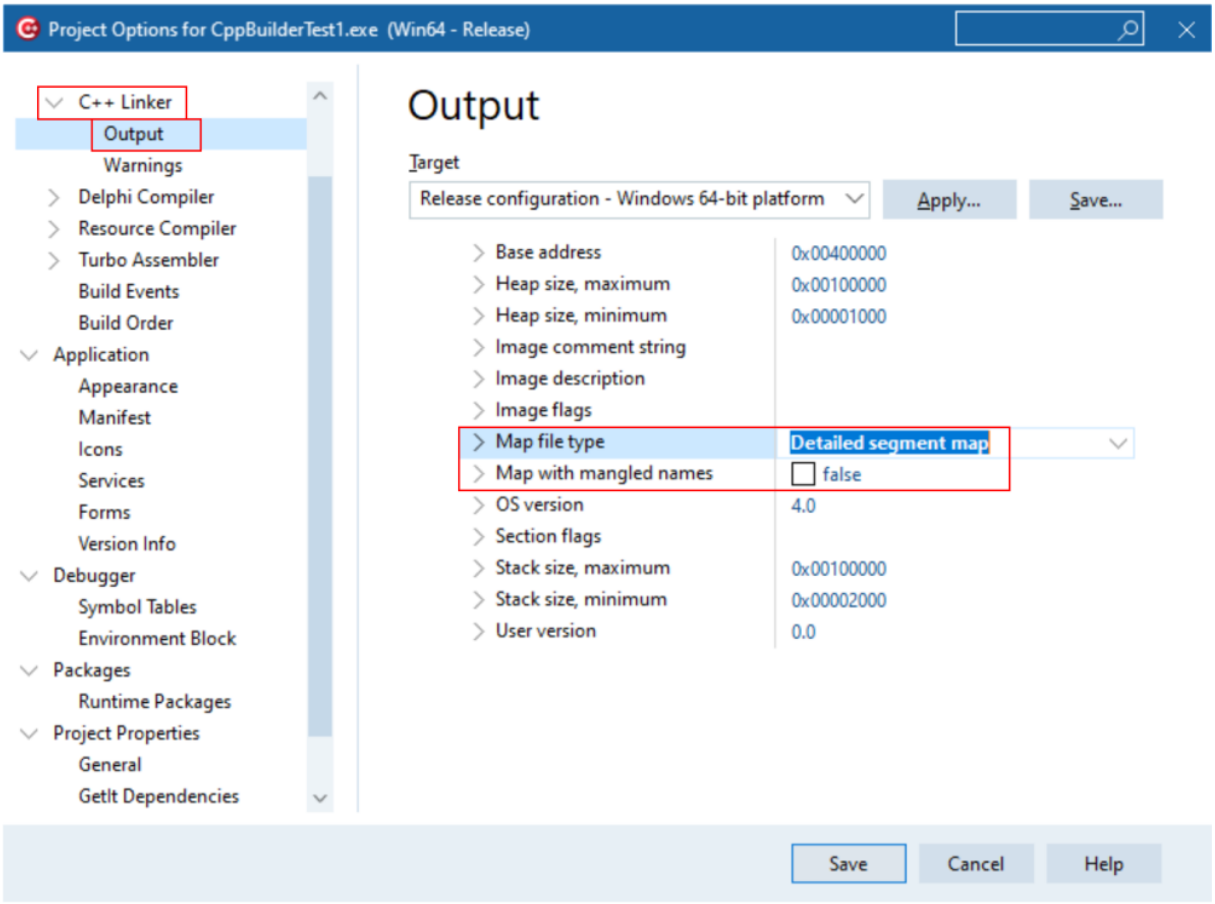
MAP files

MAP files are not generated by default. You need to enable the option to generate a detailed map file.

The method is the same for 32 bit and 64 bit C++ Builder.

Select the project configuration as shown in the Debugging Information section above, then modify the C++ Linker, Output settings.

Linker Settings



When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Debugging Information or MAP files?

If you can create both debugging information and MAP files which should I use?

Coverage Validator uses this information to provide symbols, filenames and line numbers.

For the purposes of instrumenting your modules (EXE / DLL / etc) this information is used to identify functions and to identify line numbers.

For this purpose ***it does*** matter whether you use Debugging Information or MAP files.

Debugging Information

TDS format and DWARF format debugging information both appear to be accurate, in that they reflect the correct location of functions and line numbers in the module they represent.

Some additional data is present in the last symbol in any given source file. Our symbol reader handles this and removes the unwanted information.

MAP files

MAP file information does not appear accurate. It is good enough for resolving addresses into symbols, filenames and line numbers for creating callstacks and crash addresses, but it is not good enough for placing hooks at the correct place for every line in the module. Some modules get instrumented perfectly, while others fail for no apparent reason. Given the lack of information in a MAP file we can only assume that some of the data identified as lines indicating code are in fact lines indicating data in the code segment. Instrumenting data is not going to work - you're corrupting the data. This would explain why instrumenting these modules with MAP file information doesn't work.

Our recommendation

Although in some circumstances working with MAP file data from C++ Builder will work, we strongly recommend that you use TDS debugging information (32 bit builds) and DWARF debugging information (64 bit builds).

11.3 Delphi

Debug information can be provided using two methods.

- Debugging information (TDS or DWARF format)
- MAP files

Debugging Information

Debug configurations of Delphi projects automatically generate debug information that provides symbols, filenames and line numbers.

However the release configurations of Delphi projects do not automatically generate debug information. You need to configure that yourself.

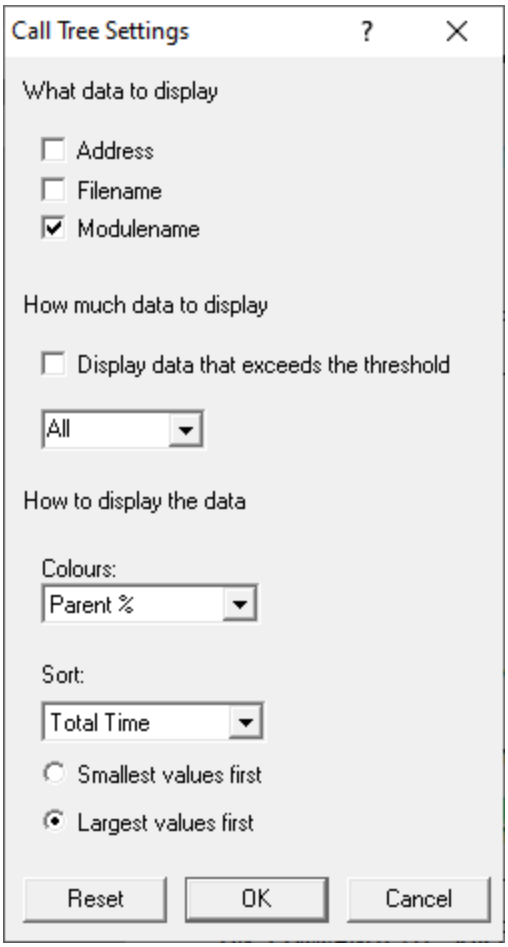
Here's how you do that. It's slightly different if you're building 32 bit applications rather 64 bit applications.

You need to set both compiler and linker settings to get debug information. Setting just one or the other will not give you debug information you can use.

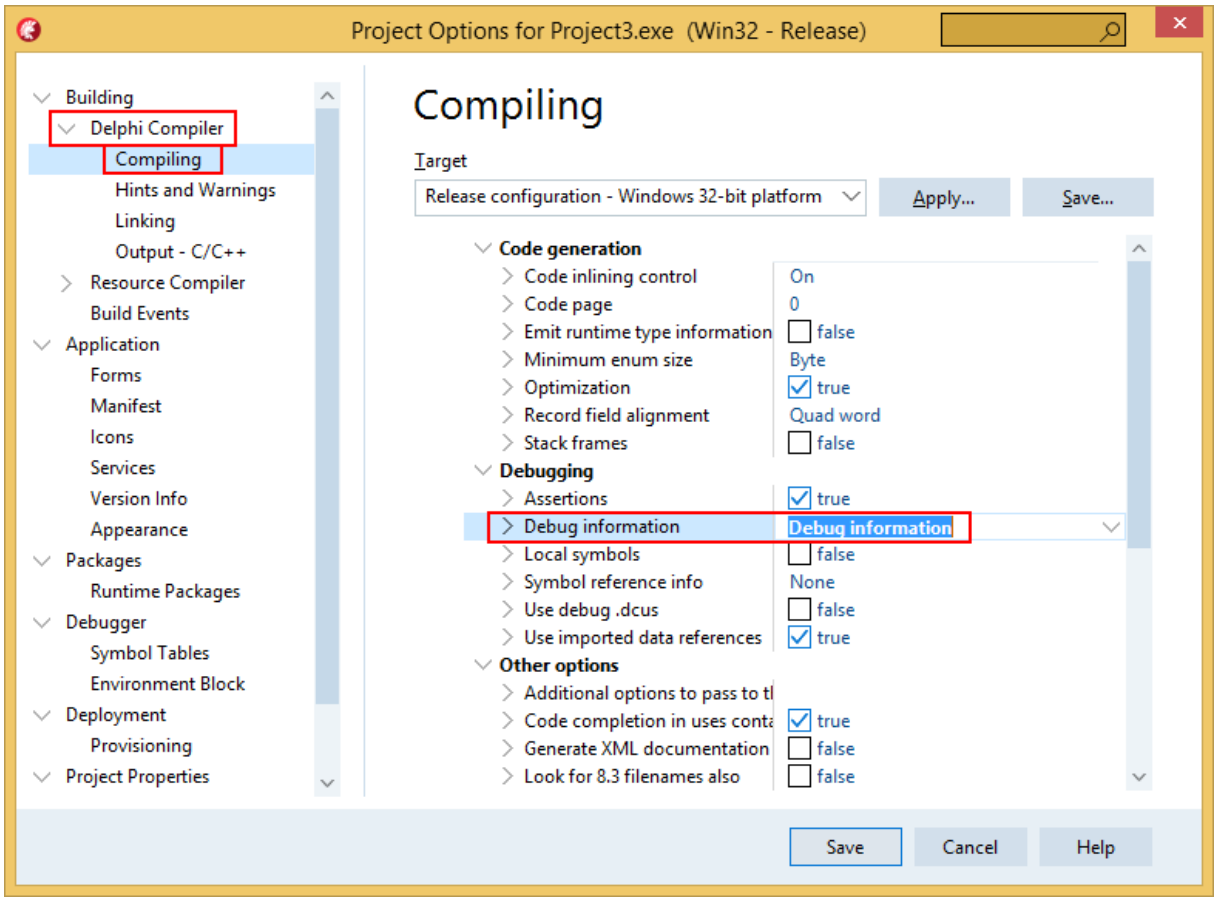
32 bit Delphi

Project Configuration

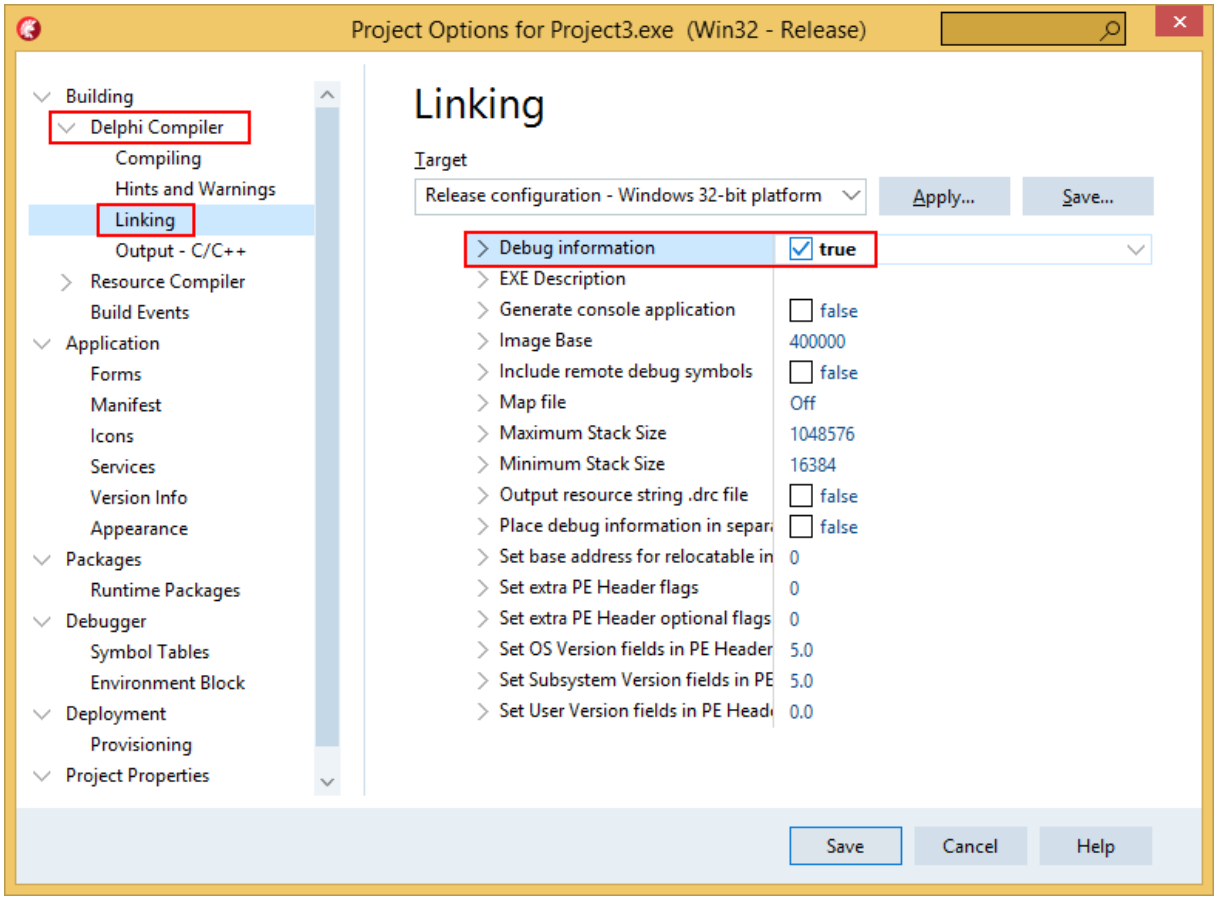
Change your project settings to target 32 bit builds.



Compiler Settings



Linker Settings

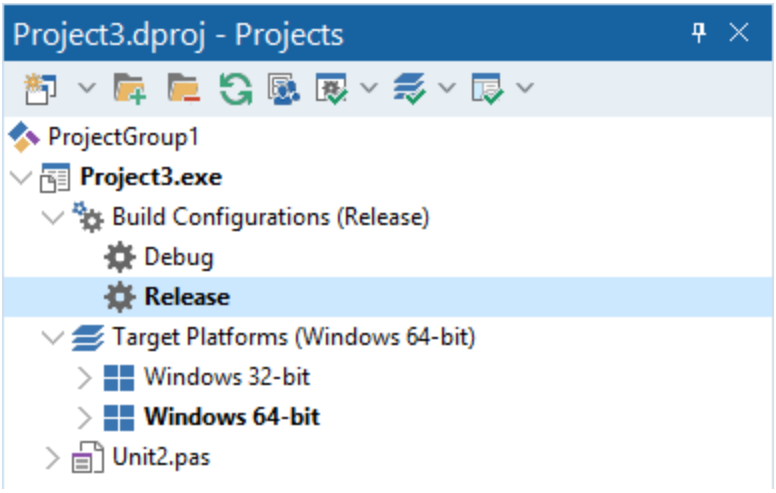


When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

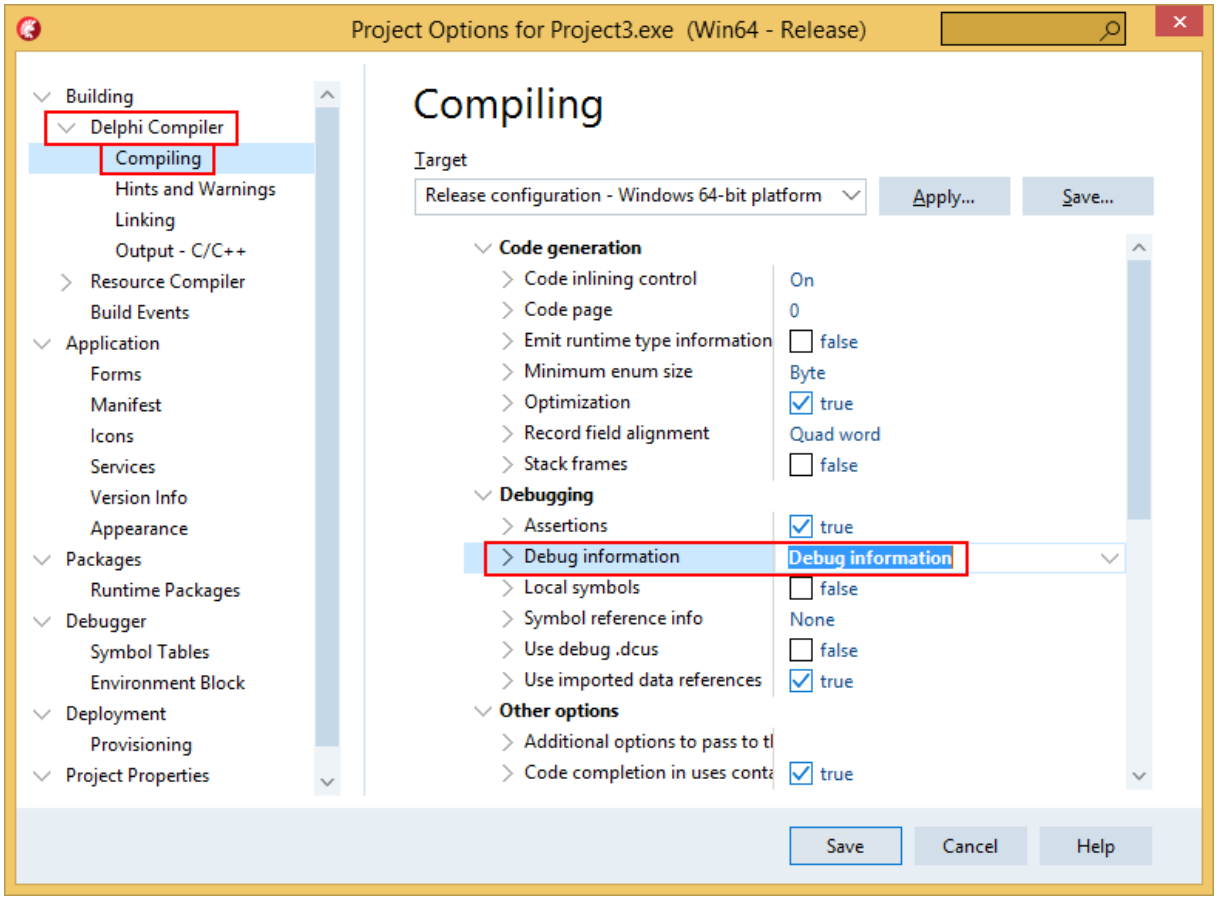
64 bit Delphi

Project Configuration

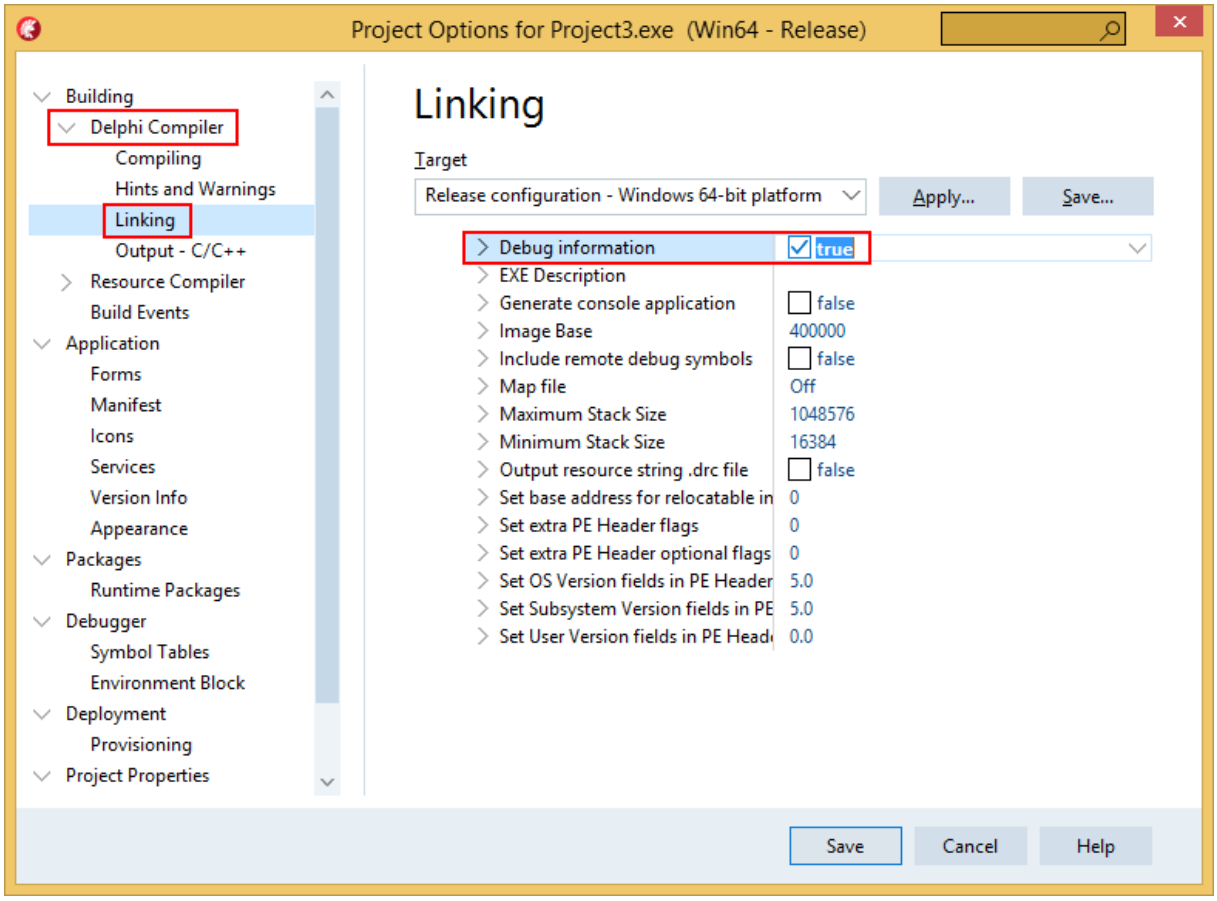
Change your project settings to target 64 bit builds.



Compiler Settings



Linker Settings



When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

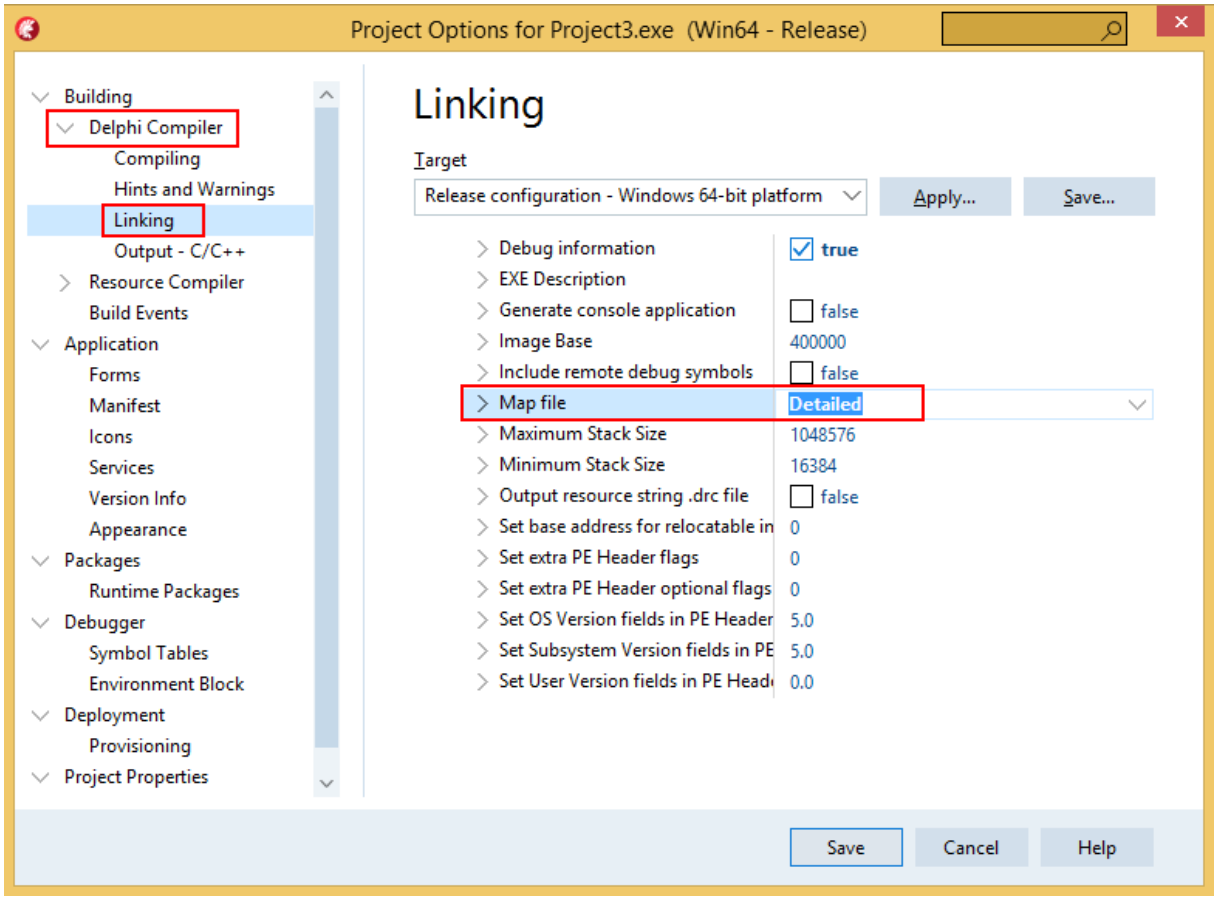
MAP files

MAP files are not generated by default. You need to enable the option to generate a detailed map file.

The method is the same for 32 bit and 64 bit Delphi.

Select the project configuration as shown in the Debugging Information section above, then modify the Delphi Compiler, Linking settings.

Linker Settings



When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

Debugging Information or MAP files?

If you can create both debugging information and MAP files which should I use?

Coverage Validator uses this information to provide symbols, filenames and line numbers.

For the purposes of instrumenting your modules (EXE / DLL / etc) this information is used to identify functions and to identify line numbers.

For this purpose ***it does*** matter whether you use Debugging Information or MAP files.

Debugging Information

TDS debugging information appears to be accurate, in that they reflect the correct location of functions and line numbers in the module they represent.

Some additional data is present in the last symbol in any given source file. Our symbol reader handles this and removes the unwanted information.

MAP files

MAP file information does not appear accurate. It is good enough for resolving addresses into symbols, filenames and line numbers for creating callstacks and crash addresses, but it is not good enough for placing hooks at the correct place for every line in the module. Some modules get instrumented perfectly, while others fail for no apparent reason. Given the lack of information in a MAP file we can only assume that some of the data identified as lines indicating code are in fact lines indicating data in the code segment. Instrumenting data is not going to work - you're corrupting the data. This would explain why instrumenting these modules with MAP file information doesn't work.

Our recommendation

Although in some circumstances working with MAP file data from Delphi will work, we strongly recommend that you use TDS debugging information.

11.4 MingW, gcc, g++

The following compiler options are available if you are using MingW, gcc or g++.

-g
This is the default debug format. This will normally choose the DWARF symbol format.

-gdwarf
The DWARF symbol format.

-gstabs
The STABS symbol format.

-gCoff
The COFF symbol format. This does create a lot of unnecessary symbols, making symbol parsing slower.

11.5 Dev C++

Dev C++ uses the gcc and g++ compilers.

The following compiler options are available if you are using gcc or g++.

-g
This is the default debug format. This will normally choose the DWARF symbol format.

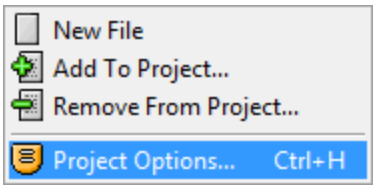
-gdwarf
The DWARF symbol format.

-gstabs
The STABS symbol format.

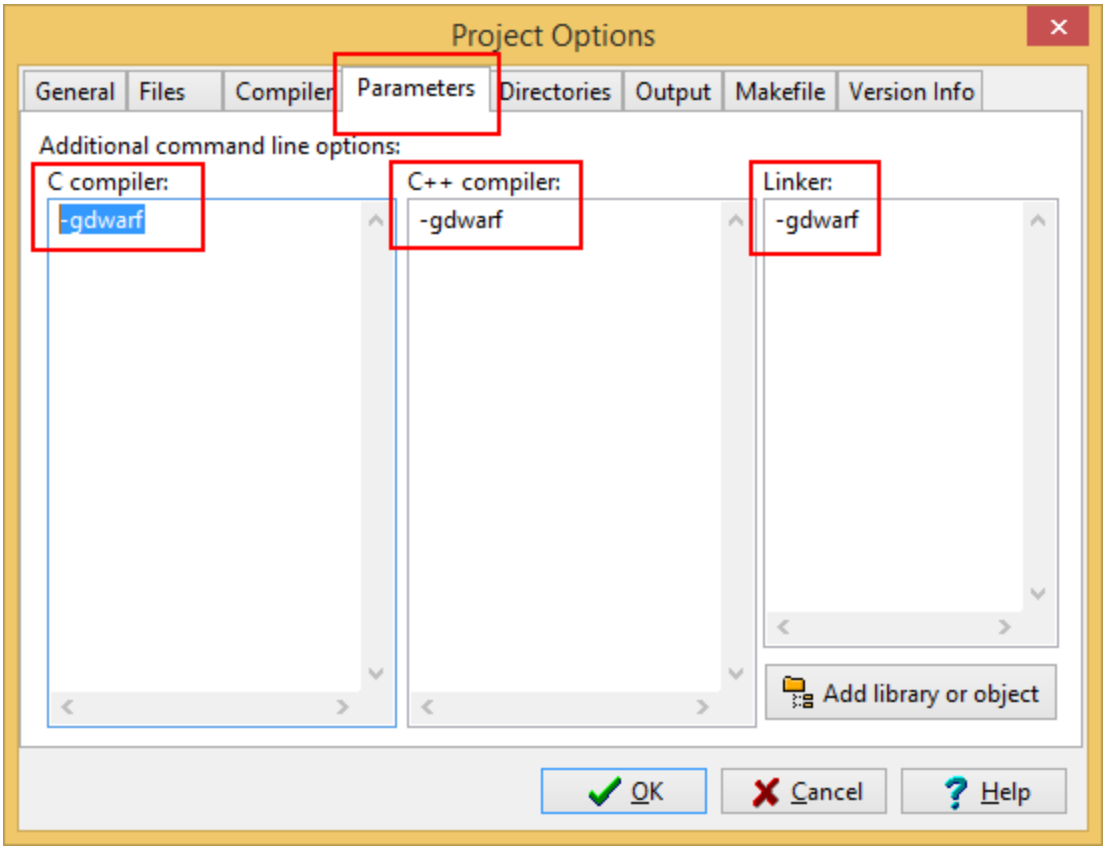
-gCoff

The COFF symbol format. This does create a lot of unnecessary symbols, making symbol parsing slower.

You can edit the compiler and linker options by choosing **Project Options...** from the **Project** menu.



Compiler and Linker options



When you have edited the project options you need to rebuild the software for the options to take effect and create the debug information.

11.6 Salford Software FORTRAN 95

Salford FORTRAN95 symbolic information is embedded in the .exe/.dll as COFF (Common Object File Format) information, with some proprietary extensions to Salford Software (which they have kindly shared with us).

Please consult the documentation for Salford FORTRAN95 to include debug information (including filenames and line numbers) in the COFF information.

Add the /DEBUG option to the compiler options.

If you still have problems, please contact us giving as much detail as possible, including what you've tried.

11.7 Metrowerks

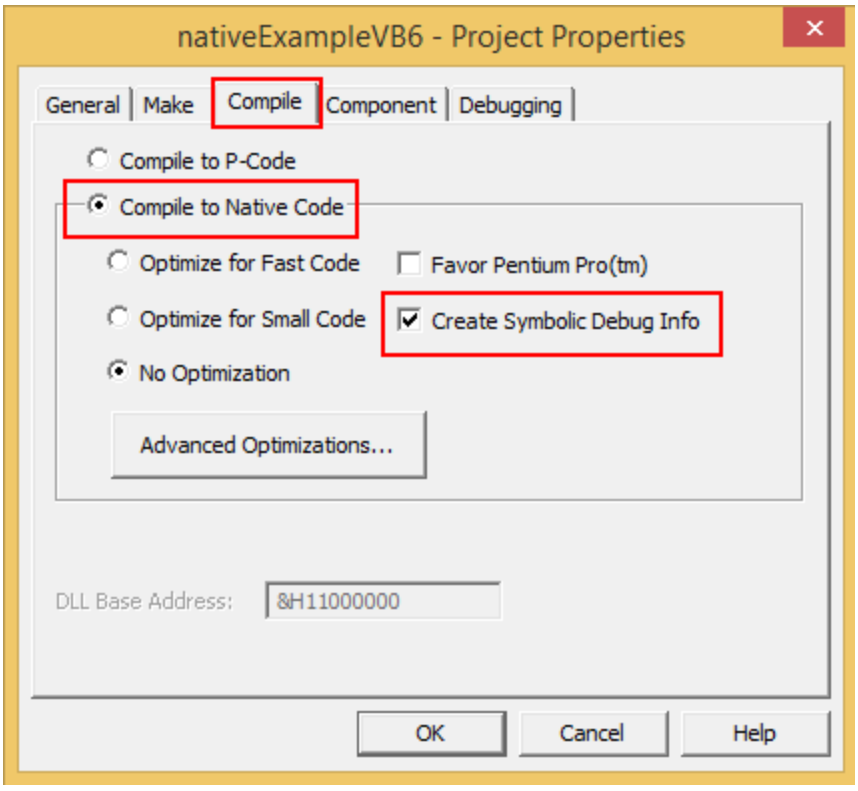
Metrowerks symbolic information is embedded in the .exe/.dll as CodeView information.

Please consult the documentation for CodeWarrior in order to include debug information (including filenames and line numbers) in the CodeView information.

If you still have problems, please contact us giving as much detail as possible, including what you've tried.

11.8 Visual Basic 6

To get debug symbols for Visual Basic you need to open the **Properties** dialog box from the **Project** menu (you'll find it at the bottom of the menu).



When you have changed your project properties you need to build the application.

Go to the **File** menu and choose **Make <projectname.exe>**.

Part XII

12 Frequently Asked Questions

Here's a brief description about the type of question included in each of the following sections:

- General questions

  How Coverage Validator works and how to do a few of the more common tasks.

- Unexpected results

  Missing or unhooked data and not finding the data you expected.

- Crashes and error reports

  Your program crashes with Coverage Validator or Coverage Validator itself has a problem.

- Debug symbols and DbgHelp

  Symbols not loading, troubleshooting search paths for DbgHelp.dll, and finding or installing different versions.

- System and environment

  Setting up power users, and file extensions used by Coverage Validator.

12.1 General Questions

Does Coverage 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

If you are launching your application with any option other than CreateProcess you are effectively using CreateRemoteThread to inject into the application you have just started running using CreateProcess.

The Inject and Wait for Application to Start functionality also use CreateRemoteThread to inject into an application.

For the reasons below, injection using CreateRemoteThread does not always work.

Common reasons for injection failure

- A missing DLL in your application

Check your application is complete.

- The target application is a .NET application or .NET service

  Check your application or service is not written using .NET technology.

- A missing DLL in Coverage Validator

  Check Coverage 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 Coverage Validator

  If the application being injected into is a service it is recommended that the service and Coverage 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 clear the symbol cache?

Flush the symbol cache files:

- **Settings** Menu › **Edit Settings** › **Hook safety** › **Clean Instrumentation Cache** › **Scan and delete symbol cache files** › **Close** › **OK**

You may also want to disable the on-disk cache of PDB file symbols:

- **Settings** Menu › **Edit Settings** › **Hook safety** › deselect **Cache instrumentation data** › **OK**

⊟ I have an idea for a feature, can it be added to Coverage Validator?

We have tried to add as many features to Coverage Validator that we thought would be useful to our users.

In fact, every feature in Coverage 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 Coverage Validator. But no Quake, FlightSim or Flappy Bird Easter eggs though, sorry!

Please contact us to let us know your thoughts.

12.2 Unexpected results

Some lines are not coloured, why?

You may notice in the source code views that some lines are not coloured to indicate visited, not visited or hook failure. There's a couple of possible reasons for this:

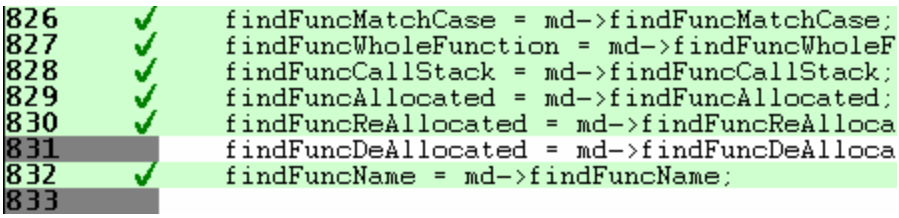
- The source code has not been compiled due to conditional compilation using compiler pragmas or #if, #ifdef, #ifndef statements

Or, if using map files with line information:

- The compiler map file did not include object code addresses for the lines that are not coloured

  When this happens, Coverage Validator has no way of accurately determining which object code corresponds to the source code, and so can't hook the object code.

For example, here's a few lines of code:



This shows that line 831 has not been hooked, whilst all those around it have been hooked (and in the example shown, all have been visited). If we now examine the part of the MAP file for the appropriate executable.

```
821 0001:000936e8   822 0001:000936f2   824 0001:00093700   826 0001:00093713
827 0001:00093720   828 0001:0009372c   829 0001:00093738   830 0001:00093745
832 0001:0009374b   834 0001:00093772   866 0001:00093778   867 0001:000938a1
```

There's no entry for line 831, which is why Coverage Validator couldn't provide a hook to verify if the line was executed.

At present we don't know why the compiler map file omits information for source code lines that are clearly part of the executable image, and which are identified in the PDB debugging information.

When you find this happening, change the line hooking options so that map files are only used when PDB files can't be found, i.e. use the **Use MAP when no PDB** option.

Why are some lines not hooked?

Coverage Validator instruments lines in your application by inserting code to recognise the execution of the start of every line in your application.

Before inserting the code, checks are made to ensure that it is safe to re-write the function lines.

If it is not safe to re-write the lines, they can't be instrumented.

The following items can prevent the function from being hooked:

- Function too short to hook. The function must be at least 5 bytes in length
- The code for the line is too short to hook. The code for the line must be at least 5 bytes in length
- Function cannot be disassembled
- Instruction sequence cannot be hooked

You can improve the likelihood of your function being hooked by enabling the check boxes on the Hook Control settings and increasing the instrumentation level.

➔ See also the instrumentation logging which details reasons why some code is not instrumented

Why are some functions not hooked?

Coverage Validator instruments functions in your application by re-writing the prologue and epilogue of each function in your application, inserting code to monitor code coverage.

Before inserting the code, checks are made to ensure that it is safe to re-write the function prologue and epilogue.

If it is not safe to re-write the function epilogue and prologue the function cannot be instrumented.

The following items can prevent the function from being hooked:

- Function too short to hook
- Function has multiple exits
- Function has jumps into epilogue
- Function has jumps into prologue
- Function cannot be disassembled
- Instruction sequence cannot be hooked

You can improve the likelihood of your function being hooked by enabling the check boxes on the Hook Control settings and increasing the instrumentation level.

➡ See also the instrumentation logging which details reasons why some code is not instrumented

12.3 Crashes and error reports

The program I'm trying to monitor keeps crashing, why?

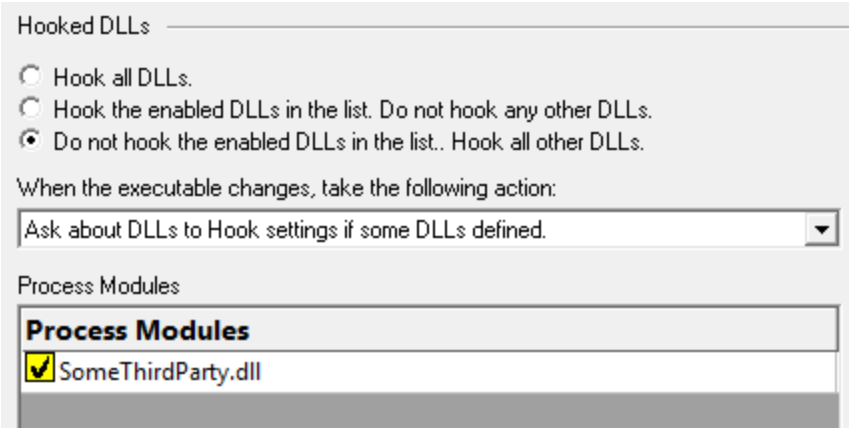
The following assumes your crash is one that only happens when using Coverage Validator.

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 Coverage 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.



- **Third party DLLs are using global hooks**

  A global hook DLL from a third party vendor could be adversely affecting Coverage 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 Coverage 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 Coverage Validator**

It happens. We've tried to make Coverage Validator as robust as possible, but bugs and new scenarios do occur.

First, ensure that the crash never happens if you are not using Coverage 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.

Coverage Validator gives an Unrecoverable Error?

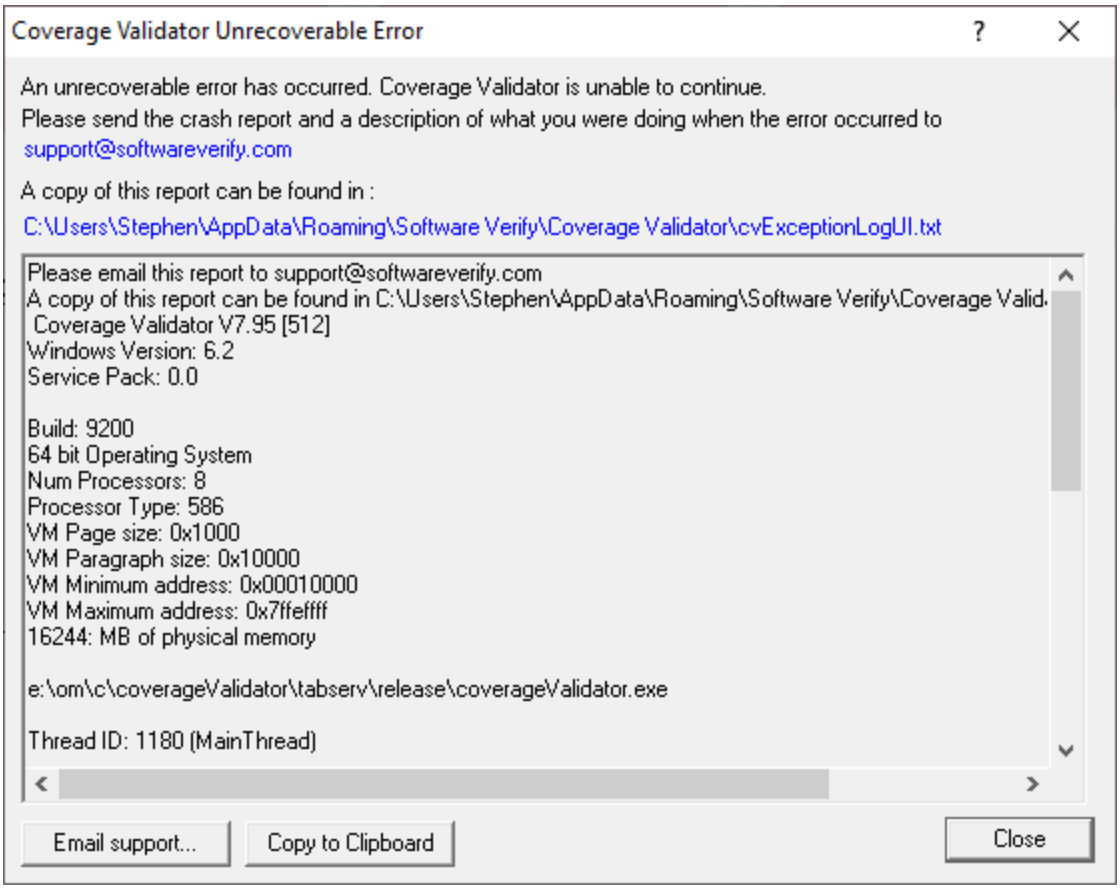
The Coverage Validator Unrecoverable Error dialog is displayed when an unexpected internal error means Coverage Validator cannot continue to execute.

A stack trace and register dump is shown and you can **Copy to Clipboard** so that the data can be sent to us with a description of the activities that caused the error.

We'll aim to fix any problems in as timely manner as possible.

The data shown in the dialog is also written to `c:\users\<username>\AppData\Roaming\Software Verify\Coverage Validator\cvExceptionLogUI.txt`

The picture below shows a stack overflow exception report (we created the crash for this topic).



cvExceptionLogUI.txt

cvExceptionLogUI.txt details a crash in the Coverage Validator user interface.

cvExceptionLog.txt

cvExceptionLog.txt details a crash in the target program that Coverage Validator was monitoring. The crash may be in the target program or in Coverage Validator's monitoring code.

What is in svlCVExceptionReport.txt?

In the event of a crash, the file `c:\users\<username>\AppData\Roaming\Software Verify\Coverage Validator\cvExceptionLogUI.txt` contains information that identifies where Coverage 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 when a crash occurred.

The file contains only the data for the most *recent* exception.

12.4 Debug symbols and DbgHelp

Why does Coverage Validator fail to load my symbols?

In a few cases Coverage 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.

Coverage 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 Coverage 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 Coverage Validator.

If the version used by Visual Studio is higher, it's possible Microsoft changed the PDB file format, making the symbols unreadable by Coverage Validator.

To fix this:

- Copy the DbgHelp.dll from Visual Studio to the Coverage Validator installation directory
- Remove any DbgHelp.dll from your application directory

  When Coverage Validator launches an application it copies Coverage Validator's DbgHelp.dll to the directory of the executable.

  This ensure 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.

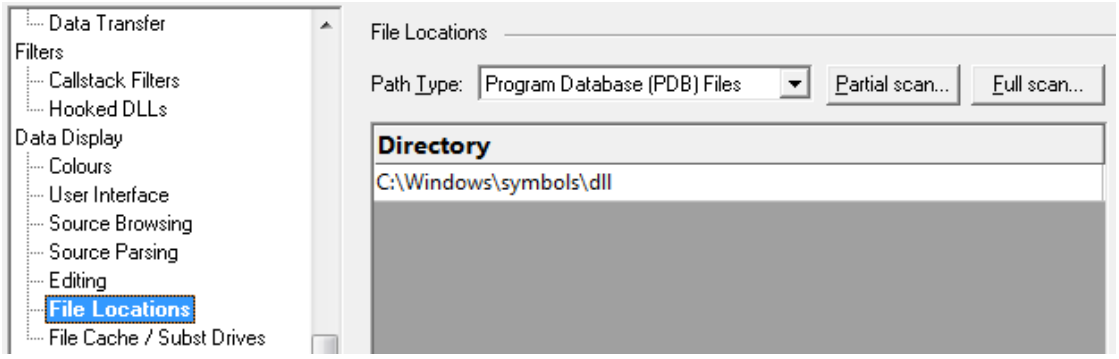
Visual Studio 2005 (8.0) and later versions

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 Files on the File Locations tab:



You may need to restart Coverage 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.

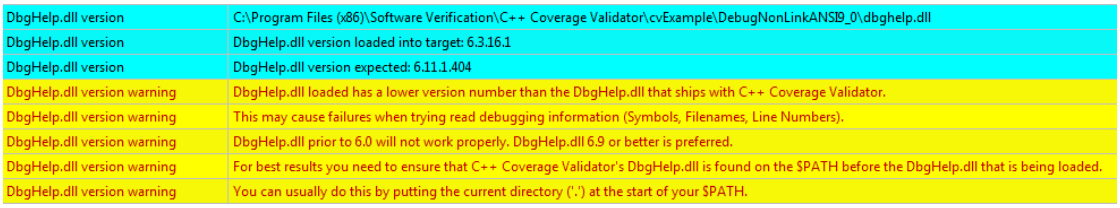
The `-gCoff` option is also supported, but this does create a lot of unnecessary symbols, making symbol parsing slower.

Troubleshooting DbgHelp.dll

Coverage Validator uses the Microsoft Debugging DLL, DbgHelp.dll, copying the correct private version to your application's directory as your program is started.

However, there are cases where your application can be started independently, and you must ensure that your application uses the correct DbgHelp.dll.

Diagnostic error messages appear on the Diagnostics tab as in the example below detailing which version of DbgHelp.dll was expected and what was actually loaded.



If you see any DbgHelp warning dialogs, or get diagnostic errors, ensure the correct DbgHelp.dll is used by:

- **Copy (don't move) DbgHelp.dll**

  from: the Coverage Validator install directory

  to: the location of the application being tested (the same directory as the .exe).

  Rerun your test.

- **Try updating the versions of DbgHelp.dll** in:

  `c:\windows\system32`

  and

  `c:\windows\system32\dllcache`

  Accept any Windows permission warnings if you try to do this.

Rerun your test.

If you *still* continue to have problems, please drop us a line via our support email.

How do I examine (and fix) the DbgHelp symbol search path?

It can be 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 Coverage 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 colourseets, and the messages are the exact same output from the DbgHelp.dll debugging stream.

Examples of examining the diagnostics

Below we show three examples using nativeExample.exe and nativeExample.pdb from our example application.

- **Correct symbol file found**

  DbgHelp first searches in various places looking for nativeExample.pdb

DbgHelp Search Info	DBGHELP: C:\WINDOWS\symbols\dll\nativeExample.pdb - file not found
DbgHelp Search Info	DBGHELP: C:\WINDOWS\symbols\dll\exe\nativeExample.pdb - file not found
DbgHelp Search Info	DBGHELP: C:\WINDOWS\symbols\dll\symbols\exe\nativeExample.pdb - file not found

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

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

DbgHelp Search Info	DBGHELP: nativeExample - private symbols & lines
DbgHelp Search Info	C:\Program Files (x86)\Software Verify\Coverage Validator x86\examples\nativeExample\ReleaseNonLinkANSI10_0\nativeExample.pdb
Loaded symbols	Loaded PDB symbols for:C:\Program Files (x86)\Software Verify\Coverage Validator x86\examples\nativeExample\ReleaseNonLinkANSI10_0\nativeExample.exe

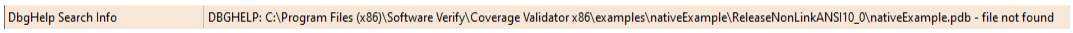
DbgHelp loads private symbols and lines, (the alternative being that DbgHelp loads public symbols).

Outcome:
Success. Symbols are loaded.

- **Missing symbol file**

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

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



  nativeExample.pdb never gets found on the search path.

  SymSrv might then look for additional locations for nativeExample.pdb, but has no luck.

  DbgHelp might find some COFF symbols in the executable, however these don't contain filename or line number information.

  Finally all options are exhausted.

  Outcome:
  Failure. The PDB file could not be found. Some default symbols are loaded but are not of much use.

  Resolution:
  Check the File Locations PDB paths to ensure that all the possible paths for PDB files are listed.

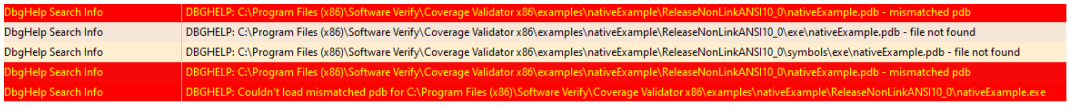
- **Incorrect symbol file**

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

  This time, nativeExample.pdb *does* get found in the same directory as the .exe file of the target program.

  DbgHelp tries to load the symbols but fails - the checksum inside the PDB file does not match the module.

  This might be because the symbols are for a different build of the software, or it's an incorrectly named PDB file belonging to another program.



Finally all options are exhausted.

DbgHelp Search Info	DBGHELP: nativeExample - no symbols loaded

Outcome:
Failure. A PDB file was found, but it was not the right one.

Resolutions:
Double check the PDB is the correct one for the build you are running.
When copying builds from another machine (or from a build server), make sure to copy the correct PDB as well.
Check the File Locations PDB paths to ensure that all the possible paths for PDB files are listed.
Check the order of those PDB paths in case there are multiple paths resulting in the wrong PDB being found first.

How can I create a map file with line numbers

If you don't have the ability to use .PDB files for debug information , you may be able to use .MAP files with line information.

The following is only applicable to Debug builds. Map files for Release builds can't have line number data.

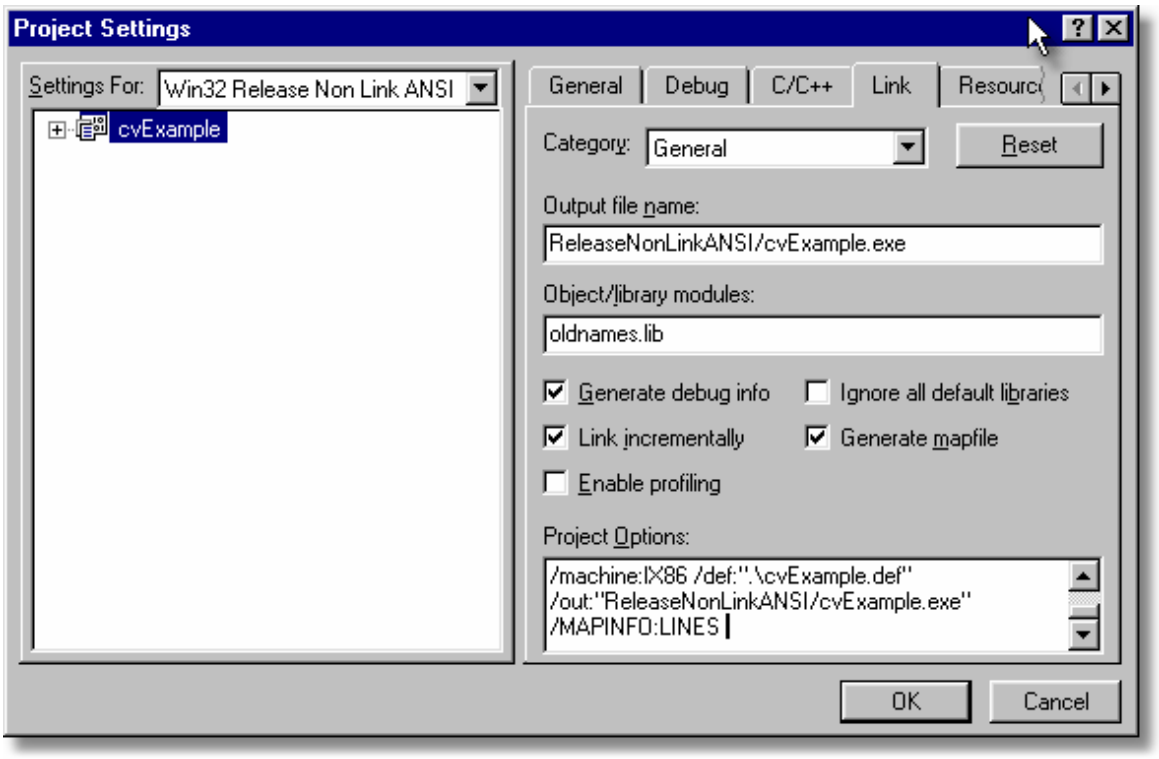
Microsoft discontinued support for including line information in .MAP files with Visual Studio 8.0 (2005). There is no easy workaround to this.

To select the `/MAPINFO:LINES` option for Visual Studio 6.0 use the following steps. If you are using Visual Studio 7.0, 7.1 (i.e .NET 2002 or 2003) the project settings user interface is slightly different, but the basic principle remains the same.

In Visual Studio 6:

Project Menu › **Settings...** › Select project › Shows project settings

The example image below shows project **nativeExample**.



- **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 Coverage Validator will not recognise the MAP as valid. The solution to this problem is to rebuild the application.

➜ See also, the map file timestamp threshold setting.

12.5 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 Coverage 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 Coverage Validator.

For example, if Coverage Validator is installed in:

```
C:\Program Files (x86)\Software Verify\Coverage Validator
```

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

```
C:\Program Files (x86)\Software Verify\Coverage Validator\stub
C:\Program Files (x86)\Software Verify\Coverage Validator\stublib
```

Compiler errors

If you include `stublib.h`, your project must have included `windows.h` first, (or see below for an alternative).

If you fail to include `windows.h` then `stublib.h` will refer to some none-existent datatypes, causing compiler errors similar to the ones shown below.

Here's an example program that will not compile:

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

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

⊟ See the compiler errors from the above code

```
--------------------Configuration: testMV_allEnum - Win32
Debug--------------------
Compiling...
testMV_allEnum.cpp
c:\program files\software verification\coverage validator\stub\allenum.h(70) :
error C2146: syntax error : missing ';' before identifier 'lRequest'
c:\program files\software verification\coverage validator\stub\allenum.h(70) :
error C2501: 'LONG' : missing storage-class or type specifiers
c:\program files\software verification\coverage validator\stub\allenum.h(70) :
error C2501: 'lRequest' : missing storage-class or type specifiers
c:\program files\software verification\coverage validator\stub\allenum.h(71) :
error C2146: syntax error : missing ';' before identifier 'reserved3'
c:\program files\software verification\coverage validator\stub\allenum.h(71) :
error C2501: 'DWORD' : missing storage-class or type specifiers
c:\program files\software verification\coverage validator\stub\allenum.h(71) :
error C2501: 'reserved3' : missing storage-class or type specifiers
c:\program files\software verification\coverage validator\stub\allenum.h(73) :
error C2143: syntax error : missing ';' before '*'
c:\program files\software verification\coverage validator\stub\allenum.h(73) :
error C2501: 'BYTE' : missing storage-class or type specifiers
c:\program files\software verification\coverage 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 **svlCVStubService.lib / svlCVStubService_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_CVSTUB_SERVICE_H
#define _SVL_CVSTUB_SERVICE_H
```

```
#include "svlServiceError.h"

// IMPORTANT.
// If you use svlCVStub_LoadCoverageValidator() to load svlCoverageValidatorStub.dl
// application, you must also use svlCVStub_UnloadCoverageValidator() to unload the
// your application being closed down. Failure to do so will almost certainly resul
// It does not matter how the application is closed down, you must ensure that you
// svlCVStub_UnloadCoverageValidator() to unload the DLL if you have loaded it.
//
// The DLL prepares itself in different ways and shuts itself down differently depe
// it is:-
// a) Directly linked to the application for use with the API or injected with Cove
//    When the DLL is used in this manner to DLL expects to oversee and manage the
//    shutdown.
// b) Loaded by using svlCVStub_LoadCoverageValidator().
//    When the DLL is used in this manner to DLL expects to be removed prior to app
//    and the behaviour of the DLL is undefined once you enter the program shutdown
//
//     This difference in behaviour is intentional and is done to allow the use of
//     services.

#ifdef __cplusplus
extern "C" {
#endif

SVL_SERVICE_ERROR svlCVStub_LoadCoverageValidator(serviceCallback_FUNC callback,
                                                  void                 *userParam);

SVL_SERVICE_ERROR svlCVStub_UnloadCoverageValidator();

#ifdef __cplusplus
}
#endif

#endif
```

See the implementation file

```
#include "svlCVStubService.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 svlCVStub_LoadCoverageValidator(serviceCallback_FUNC callback,
                                                  void                *userParam)
{
   SVL_SERVICE_ERROR   errCode = SVL_OK;

   if (hModule == NULL)
   {
      hModule = LoadLibraryW(L"svlCoverageValidatorStub.dll");     // change this t
      if (hModule != NULL)
      {
         // DLL loaded, set the service callback function

         SETCALLBACK_FUNC   setCallbackFunc;

         setCallbackFunc = (SETCALLBACK_FUNC)GetProcAddress(hModule, "apiSetService
         if (setCallbackFunc != NULL)
         {
            (*setCallbackFunc)(callback, userParam);
         }

         // now start the profiler

         PROC *p;

         p = GetProcAddress(hModule, "startProfiler");
         if (p != NULL)
             (*p)();
```

```
            // now turn on provision of symbols by the stub

            ENABLE_STUB_SYMBOL_FUNC   enableSymbolFunc;

            enableSymbolFunc = (ENABLE_STUB_SYMBOL_FUNC)GetProcAddress(hModule, "apiEn
            if (enableSymbolFunc != NULL)
            {
               (*enableSymbolFunc)();
            }
            else
            {
               errCode = SVL_FAILED_TO_ENABLE_STUB_SYMBOLS;
            }
         }
         else
         {
            errCode = SVL_LOAD_FAILED;
         }
      }
      else
      {
         errCode = SVL_ALREADY_LOADED;
      }

      return errCode;
}

//-NAME-----------------------------------
//.DESCRIPTION.........................
//.PARAMETERS..........................
//.RETURN.CODES........................
//---------------------------------------

typedef void (*UNLOAD_FUNC)();
typedef HANDLE (*GET_STUB_HEAP_FUNC)();

SVL_SERVICE_ERROR svlCVStub_UnloadCoverageValidator()
{
   SVL_SERVICE_ERROR   errCode = SVL_OK;

   if (hModule != NULL)
   {
      // get the stub heap before we shut down the DLL

      HANDLE         hStubHeap = NULL;
      GET_STUB_HEAP_FUNC   getHeapFunc;

      getHeapFunc = (GET_STUB_HEAP_FUNC)GetProcAddress(hModule, "apiGetInternalMVst
      if (getHeapFunc != NULL)
      {
         hStubHeap = (*getHeapFunc)();
      }

      // get the unload stub function
```

```
        UNLOAD_FUNC   unloadFunc;

        unloadFunc = (UNLOAD_FUNC)GetProcAddress(hModule, "apiShutdownCoverageValidat
        if (unloadFunc != NULL)
        {
           (*unloadFunc)();

           // get the function

           HMODULE   hModule;

           hModule = GetModuleHandleW(L"svlCoverageValidatorStub.dll");
           if (hModule != NULL)
           {
              // unload the stub

              FreeLibrary(hModule);

              // destroy the stub's heap (which was still in use whilst FreeLibrary()

              if (hStubHeap != NULL)
                 HeapDestroy(hStubHeap);
              else
              {
                 if (errCode == SVL_OK)
                    errCode = SVL_FAIL_TO_CLEANUP_INTERNAL_HEAP;
              }
           }
           else
           {
              errCode = SVL_FAIL_MODULE_HANDLE;
           }
        }
        else
        {
           errCode = SVL_FAIL_UNLOAD;
        }

        hModule = NULL;
     }
     else
        errCode = SVL_NOT_LOADED;

   return errCode;
}
```

12.6 System and environment

How do I create a Power User on Windows XP?

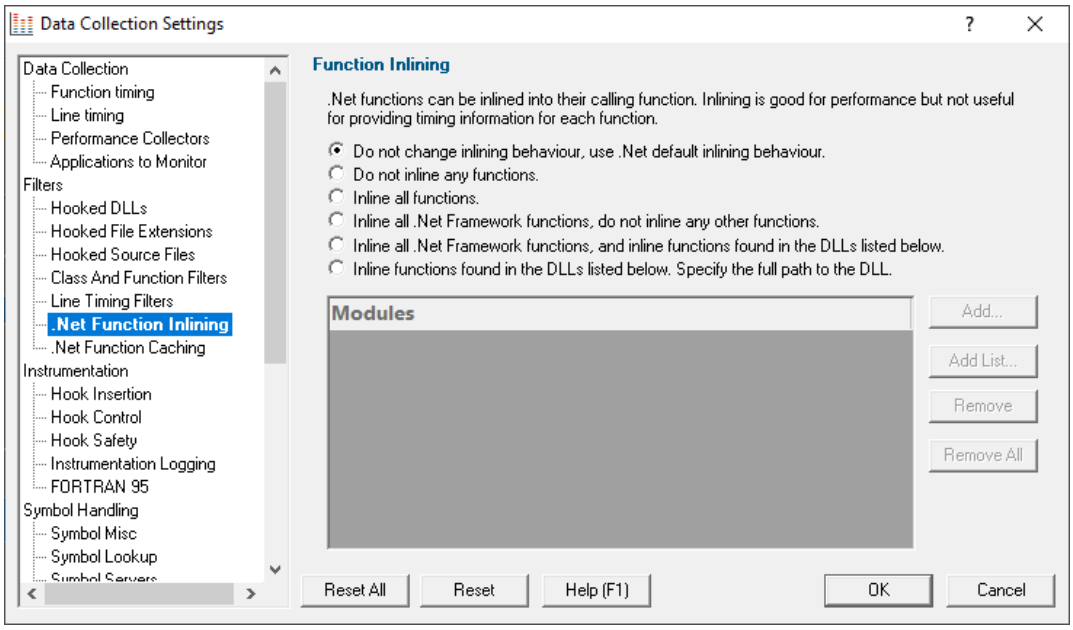
Windows 2000 and Windows XP Pro allow Power User accounts that stop short of full Administrator permissions.

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**



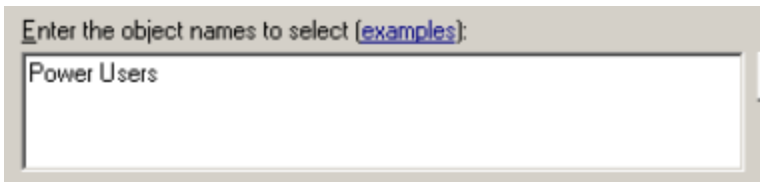
- On the right, select and **Right click** on **'Test User'** › **Properties**

  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 Coverage Validator use?

Most configuration data is stored in the registry, but some information is file-based such as settings, coverage, hook and filter data.

Coverage Validator uses the following extensions:

Session Export and Session Save

- **html** HTML export files
- **xml** XML export files

Settings, Filters, Hooks and Coverage

- **cvm** Session files for 32 bit or 64 bit Coverage Validator
- **cvm_x64**
- **cvs** Settings for 32 bit or 64 bit
- **cvs_x64**
- **cvx** Hooked DLLs
- **cvxc** Class and function filters
- **cvxfl** File locations
- **cvxft** Source file filters

Program Launch, Extensions

- **dll** Extension DLLs
- **exe** Program files

Part XIII

13 Installing Floating Licensing

How to install floating licences

Floating licences float globally. Your team members in an office on the other side of the world can share a floating licence with you.

If you have floating licences install the software on all machines in your business unit that wish to use the software.

For an overview of how floating licences work, please read this.

Floating licence server

The floating licence server is managed by Software Verify.

No server to setup, no licences to misconfigure. All the things that are bad about floating licences, we've removed all that.

If you need to acquire a licences or release a licence, see the Floating Licences tab.

Floating licence help

If you have problems with the floating licences please contact support@softwareverify.com

If you need to purchase additional floating licences for a new floating licence please visit https://www.softwareverify.com/purchasing/.

If you need to purchase additional floating licences to add to an existing floating licence please contact sales@softwareverify.com.

Part

14 Copyright notices

14.1 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 -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- K -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- W -

- X -