

Lua Bug Validator

by

Software Verification

Copyright Object Media Limited (c) 2005-2006

Lua Bug Validator

Application flow tracing for Lua

by Software Verification

Welcome to the Lua Bug Validator software tool. Lua Bug Validator is an application flow tracer. A flow tracer is an application that monitors everything that another application does.

Lua Bug Validator monitors each function call, function return, line visit and exception. At each step the function arguments and local variables are stored allowing you to view this data at a later time, perhaps whilst investigating a bug or crash.

We hope you will find this document useful.

Lua Bug Validator Help

Copyright © 2005-2007 Software Verification Limited

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

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

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

Printed: June 2008 in United Kingdom.

Table of Contents

Foreword	0
Part I Overview	4
1 Introducing Lua Bug Validator	4
2 Why Lua Bug Validator?	4
3 What do you need to run Lua Bug Validator?	5
4 How to buy Lua Bug Validator	6
5 How does Lua Bug Validator work?	6
6 What does Lua Bug Validator do?	6
7 User Permissions	7
Part II Getting Started	10
1 Quick Start	10
Part III The User Interface	13
1 Menu Reference	13
2 Toolbar Reference	15
3 The status bar	15
4 The main display	17
Icons	17
Execution History	17
Diagnostic	20
5 User Interface Mode	22
6 Lua Runtime	23
7 Settings	25
Data Collection	27
Tracing Filters	27
Display Filters	29
Colours	31
Source Code Browsing	32
Editing	33
File Locations	35
Inter-Process Communication	37
UI Global Hook DLLs	38
8 Loading and Saving Settings	41
Loading settings	41
Saving settings	42
9 User Permissions Warnings	42
10 Managers	43
Session Manager	43
11 Tools	44
Loaded Classes	44

Colour coded source code editor	45
Refresh	47
12 Loading, Saving, Exporting, Closing	47
Loading	48
Saving	48
Exporting	49
XML Export Tags.....	52
Close Session	53
13 Starting your target program	53
Novice and Intermediate	53
Launching the program.....	54
Re-Launching the program.....	56
Expert	57
Launching the program.....	57
Re-Launching the program.....	59
14 Stopping your target program	59
15 Environment Variables	60
16 Closing Lua Bug Validator	61
17 Help	61
Tip of the day	62
About Lua Bug Validator	62
Help Topics	62
Tutorial	62
Readme	62
Part IV Frequently Asked Questions	64
1 I have an idea for a feature, can it be added to Lua Bug Validator?	64
2 What file extensions does Lua Validator use for itself?	64
3 Lua Bug Validator Unrecoverable Error	64
4 What is in svLBVExceptionReport?	65
5 Installing DbgHelp.dll	66
6 How do I create a Power User on Windows XP?	67
Index	69

Part



1 Overview

Welcome to the Lua Bug Validator help manual. This section provides a brief overview of the capabilities of Lua Bug Validator.

You will need to use a version of Lua that uses a DLL to hold the Lua core.

An example of this is the Luacheia distribution of Lua

<http://luacheia.lua-users.org/>.

1.1 Introducing Lua Bug Validator

Lua Bug Validator is an automatic execution history tracer for Windows NT® 4.0 and above, running on the Intel i386 (and compatible) family of processors.

Lua Bug Validator automatically detects which lines of your program have been executed, and records the execution history of the application. Lua Bug Validator places a very low overhead on the performance of your application and does not require the target program to be recompiled or relinked. Lua Bug Validator is intended to be used in software development labs to analyse software development bugs and crashes.

The user interface is split into two separate sections, each section dedicated to a different task. The two main sections are:

- Execution History

Display the execution history of the application, showing line by line execution history of the application. Selecting a specific line displays the appropriate source code in a syntax coloured window. Another window displays the function parameters and method variables for the function.

- Diagnostic

Diagnostic information collected by the stub. Information about lines that could not be hooked is displayed here.

1.2 Why Lua Bug Validator?

Lua Bug Validator allows you to record the execution history of an application and to decode the execution history logs recorded at customer sites using Lua Bug Validator Client. When you are trying to analyse "customer only" bugs or bugs that crash without throwing exceptions or dropping into the debugger, Lua Bug Validator is a valuable tool.

Sessions can be saved and reloaded in Lua Bug Validator for later analysis. Sessions saved using Lua Bug Validator Client can be loaded and decoded so that the cause of a crash at a customer site can be analysed. Sessions can be exported to HTML or XML for providing reports that are appropriate for the management team, quality assurance team, software engineering teams.

Reliable

Lua Bug Validator has been created with the following criteria in mind.

1) Lua Bug Validator must have no adverse effect on the program's behaviour.

Any hooks Lua Bug Validator places into the target program's code must not affect the registers or the condition code flags of the program. The program must behave in the same way when being inspected by Lua Bug Validator as when the program is running without being inspected by Lua Bug Validator.

2) Lua Bug Validator 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 Lua Bug Validator is installed on, we have configured every hook, so that they can be enabled or disabled, and/or installed or not installed.

3) Lua Bug Validator must be capable of having as little impact on the target programs performance as possible.

To do this we allow you to enable and disable as many or as few function hooks as you wish.

4) Lua Bug Validator's user interface must be independent of the target program.

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

- If the Lua Bug Validator user interface crashes, your target program will not crash.
- If the target program crashes the Lua Bug Validator user interface will not crash - you will still have data to work with.
- If the target program is stopped in the debugger, Lua Bug Validator's user interface will continue to work.

5) Flexibility

Where there are multiple ways of presenting the data, the user should be given a choice over how that display works. Not all users like the same choices, so providing some choice over the display is always better than forcing all users to use the same settings.

1.3 What do you need to run Lua Bug Validator?

- Lua runtime

Lua Bug Validator requires your Lua application to be run using Lua 5.0 or later.

You will need to use a version of Lua that uses a DLL to hold the Lua core.

An example of this is the Luacheia distribution of Lua <http://luacheia.lua-users.org/>.

- User Privileges

Typically **Administrator** and **Power User** user types have the appropriate privileges. User Permissions are discussed in more detail.

- Registry Access Privileges

Lua Bug Validator requires read access and write access to **HKEY_CURRENT_USER\Software\SoftwareVerification\LuaBugValidator**.

When working with services, Lua Bug Validator requires read access and write access to **HKEY_USERS\DEFAULT\Software\SoftwareVerification\LuaBugValidator**.

If read access and write access to

HKEY_USERS\DEFAULT\Software\SoftwareVerification\LuaBugValidator is not allowed, Lua Bug Validator will use default settings when working with services (thus any user selections will not

apply). In addition, error messages will be displayed when Lua Bug Validator tries to access this registry key. These error messages can be suppressed if they are not desired.

- Operating System

Lua Bug Validator requires Windows NT® 4.0 or better.

We recommend that the minimum service pack levels are used.

- Windows NT 4.0, Service Pack 6
- Windows 2000, Service Pack 2
- Windows XP, Service Pack 1.

1.4 How to buy Lua Bug Validator

Lua Bug Validator can be purchased online from Software Verification Limited.

Software Verification Limited
PO Box 123
Ely
Cams
CB6 2WQ

email: sales@softwareverify.com
web: <http://www.softwareverify.com>

1.5 How does Lua Bug Validator work?

Lua Bug Validator is a multi-part program. One part of the program (known as "the stub") is injected or linked into the target program and communicates with the Lua Bug Validator user interface.

The stub is typically injected into the target program when the Lua runtime starts executing. Communication between the stub and the user interface is done via named pipes. There is no human readable data format for the communication between the two parts of the program. Both the stub and the user interface are multithreaded. If required the stub can be linked into the program so that it does not need to be injected into the program.

As your program executes the hooks that were inserted track the program's execution and write it to a buffer that the user interface reads.

1.6 What does Lua Bug Validator do?

Lua Bug Validator provides functionality to track an application's execution history for Lua programs running on Windows NT® 4.0, Windows 2000® and Windows XP.

The execution history is displayed as a list, with the source code corresponding to any particular line shown in a window to the right. Any function parameters, method variables and return values are displayed. Powerful filtering options are provided for limiting the data collected and also limiting the display of the collected data.

1.7 User Permissions

This section details the privileges a user requires to successfully run Lua Bug Validator. Debugging tools such as Lua Bug Validator are intrusive tools - they require specific privileges not normally granted to typical applications.

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

User Privileges

Installation:

Administrator privileges are required.

General Use:

An ordinary user account has the required privileges.

Registry Access Privileges

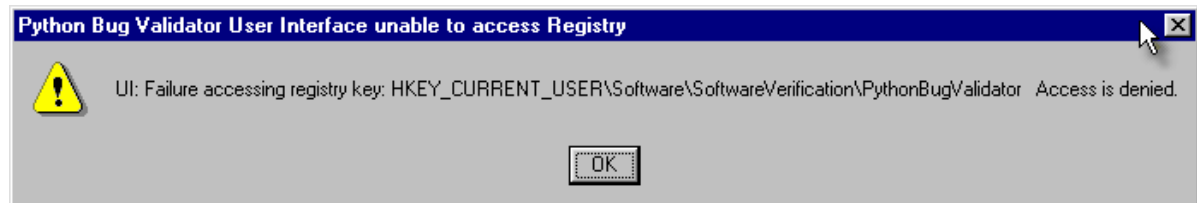
Lua Bug Validator requires read access and write access to **HKEY_CURRENT_USER\Software\SoftwareVerification\PythonBugValidator**.

You can modify the registry access permissions using the **regedt32.exe** tool Security menu. Ask your administrator to modify your registry access permissions if you cannot do this yourself.

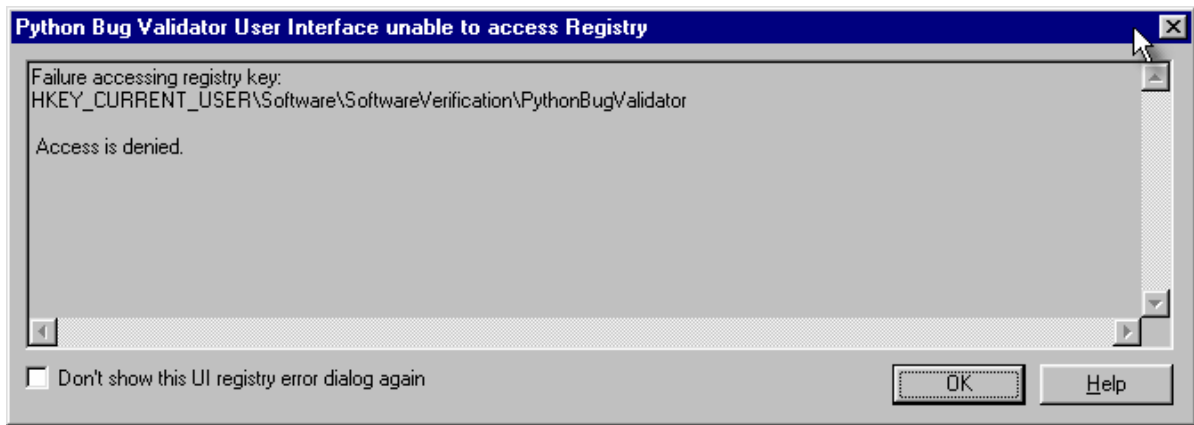
Error Notifications

When Lua Bug Validator fails to gain access for read or write to the registry a message box is displayed. The message indicates the name of the registry key that failed and the failure reason.

This message box is displayed as a simple message box during early startup and late closedown of Lua Bug Validator.

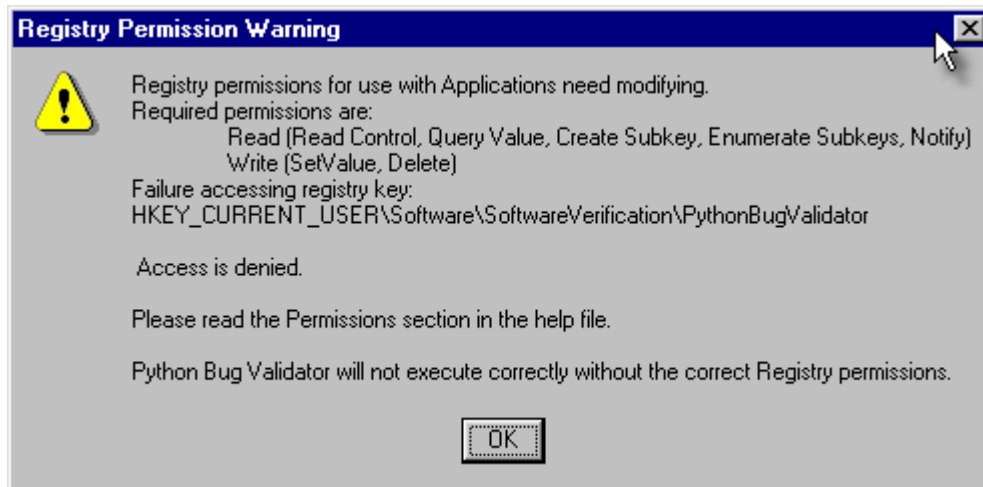


This message box is displayed when Lua Bug Validator is not starting up or closing down.



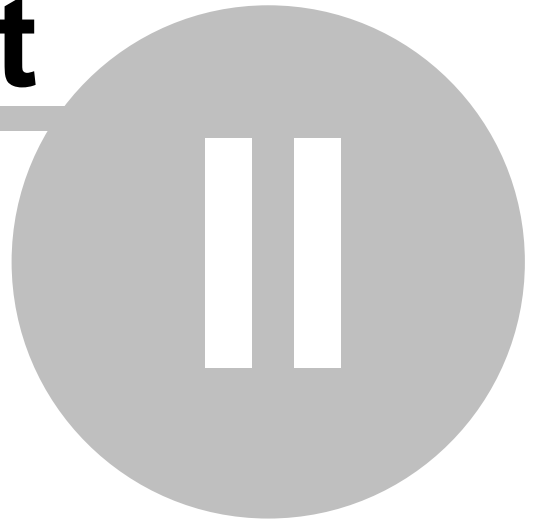
Detailed Registry Access Error Messages

The following detailed registry access error messages are also displayed when failing to gain access to the registry.



You may also want to read this topic relating to creating Power User accounts for Windows XP.

Part



2 Getting Started

If you have never used Lua Bug Validator before you have probably acquired Lua Bug Validator because you wish to analyse the execution behaviour of your application. As such you may want to 'dive in' and monitor code execution immediately. If you choose to read the manual first you will find out more about the product and how to use it to its full advantage.

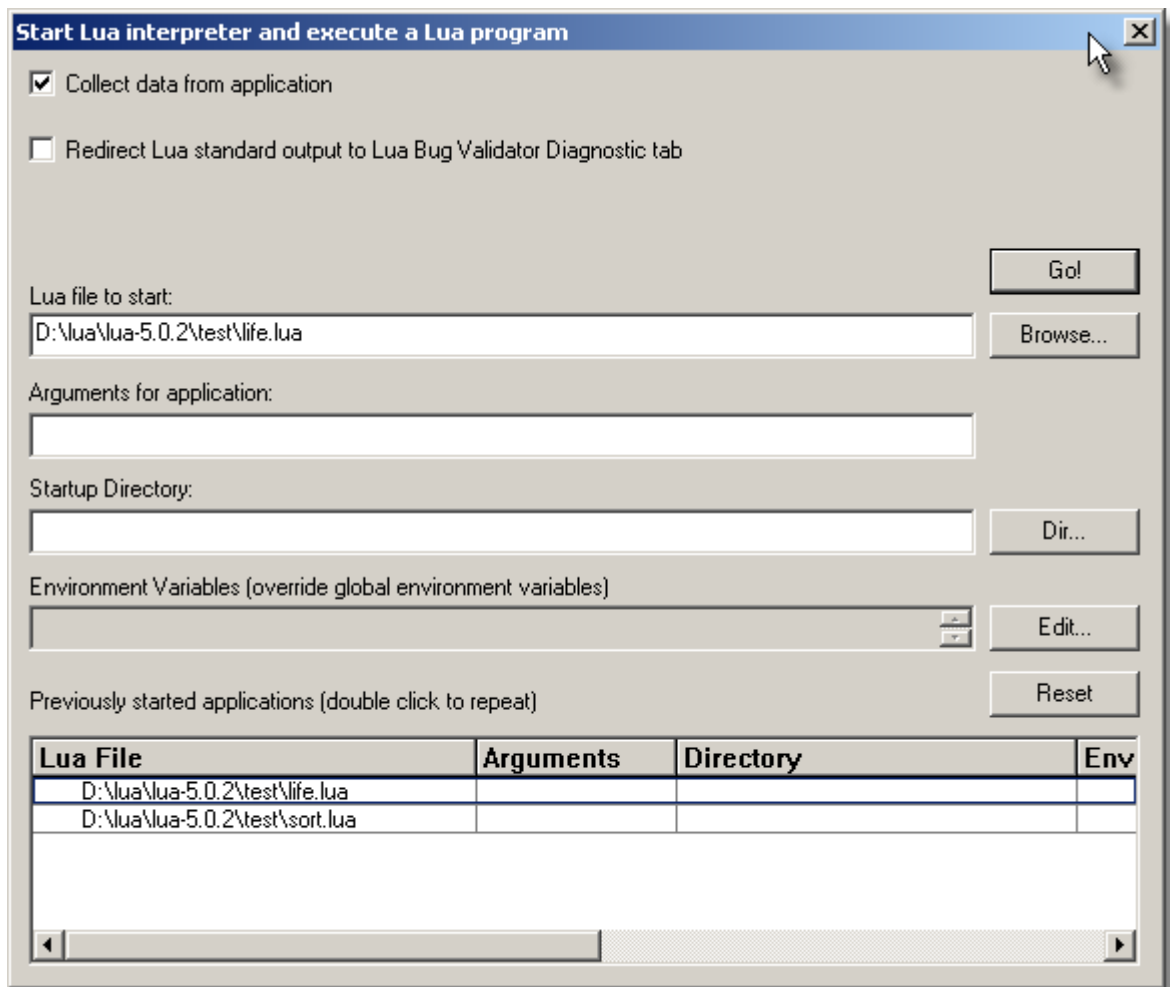
For those that wish to 'dive in' this section is for you. For those that do not wish to 'dive in' please skip to the next chapter.

2.1 Quick Start

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



The launch program dialog will be displayed. If you have just installed the software you will be shown the launch wizard. If you have switched to Dialog user interface mode you will be shown the launch dialog (shown below).



The picture above shows the launch program dialog. If you wish to know how this dialog works click [here](#).

- 1) Click on the **Start** button to use a file browser to choose the program to launch. The program will be launched automatically.
- 2) Lua Bug Validator will start the target program and inject the stub into the target program. A progress dialog will be displayed whilst the stub is being injected into the target program. The progress dialog lets you know what task it is performing during the injection sequence. When the stub is correctly installed in the target program the stub will establish communications with Lua Bug Validator.
- 3) Data will be collected by Lua Bug Validator Client until the target program exits.
- 4) When the target program exits, Lua Bug Validator closes the session. The data collection icons on the session toolbar are disabled (the toolbar image will look like the image shown at the top of this section).

Part



3 The User Interface

Lua Bug Validator is a two part program. The part that the user interacts with is the user interface. The other part of Lua Bug Validator is known as the stub. The stub's function is to install and control the data hooks inserted into the target program. The user never interacts directly with the stub, so the stub will not be described in detail. The user influences how the stub behaves via the user interface. This section describes the various functions of the user interface so that you can get the most from using Lua Bug Validator.

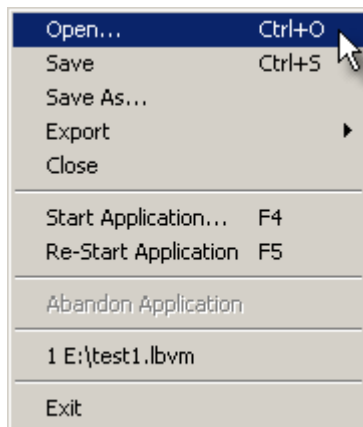
Typical usage of Lua Bug Validator is:

- Start the target program
- Collect the flow trace of the program.
- Close the program

3.1 Menu Reference

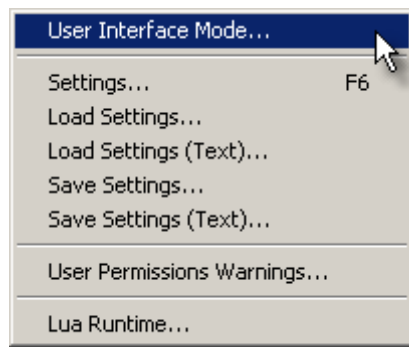
This section lists the various menus in Lua Bug Validator and provides links to the appropriate section of the help manual.

File



- Open
- Save
- Save As
- Export
- Close
- Start Application
- Re-Start Application
- Abandon Application
- Exit

Configure



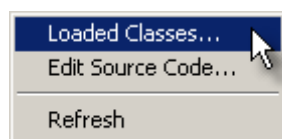
- User Interface Mode
- Settings
- Load Settings
- Load Settings (Text)
- Save Settings
- Save Settings (Text)
- User Permissions Warnings
- Lua Runtime

Managers



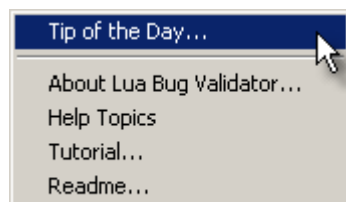
- Session Manager

Tools



- Loaded Classes
- Edit Source Code
- Refresh

Help



- Tip of the Day

- About Lua Bug Validator
- Help Topics
- Tutorial
- Readme

3.2 Toolbar Reference

This section lists the various toolbars in Lua Bug Validator and provides links to the appropriate section of the help manual. The icons are described in sequence from left to right in the toolbar.

Standard



- Load session
- Save session
- Help
- Context sensitive help

Session



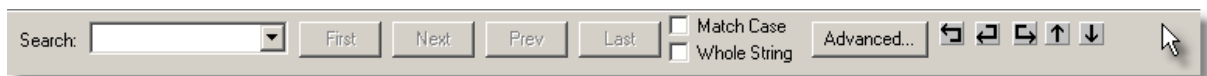
- Data collection settings
- Launch application
- Re-Launch application
- Stop application

Tools



- Refresh view

Search



3.3 The status bar

Lua Bug Validator provides various items of status data on the status bar at the bottom of the main window. You can use the data item counts to give a very crude indicator of how data is being

collected by the stub and sent to Lua Bug Validator. The number of data items pending processing indicates how much has yet to be processed.

The status bar displays, from left to right, the following items

- Command description.
- Number of data items processed.
- Target program name, and description.

Command description

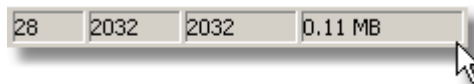
The command description displays **Ready** when awaiting a command. When the mouse moves over a menu item a description of the menu item is displayed in the command description area.



Data collection item counts

The data collection item counts are displayed in the second, third, fourth and fifth fields on the status bar. The fields are:

- Number of data items sent from stub that have been processed.
- Number of data items in shared data area.
- Total number of data items that have been present shared data area.
- Size of data collected, specified in MB.



Data collection status

The data collection status is displayed in the sixth field on the status bar.



Target program name

The target program name displays **No active session** when there is no session running, terminating or loaded.

When a program is running, terminating or has finished running, the name of the program is displayed with a status in parentheses. The status is one of Starting, Running, Terminating or Ready.



3.4 The main display





The main display of Lua Bug Validator consists of two tabbed windows. The tabbed windows are:

- Execution History
- Diagnostic

3.4.1 Icons

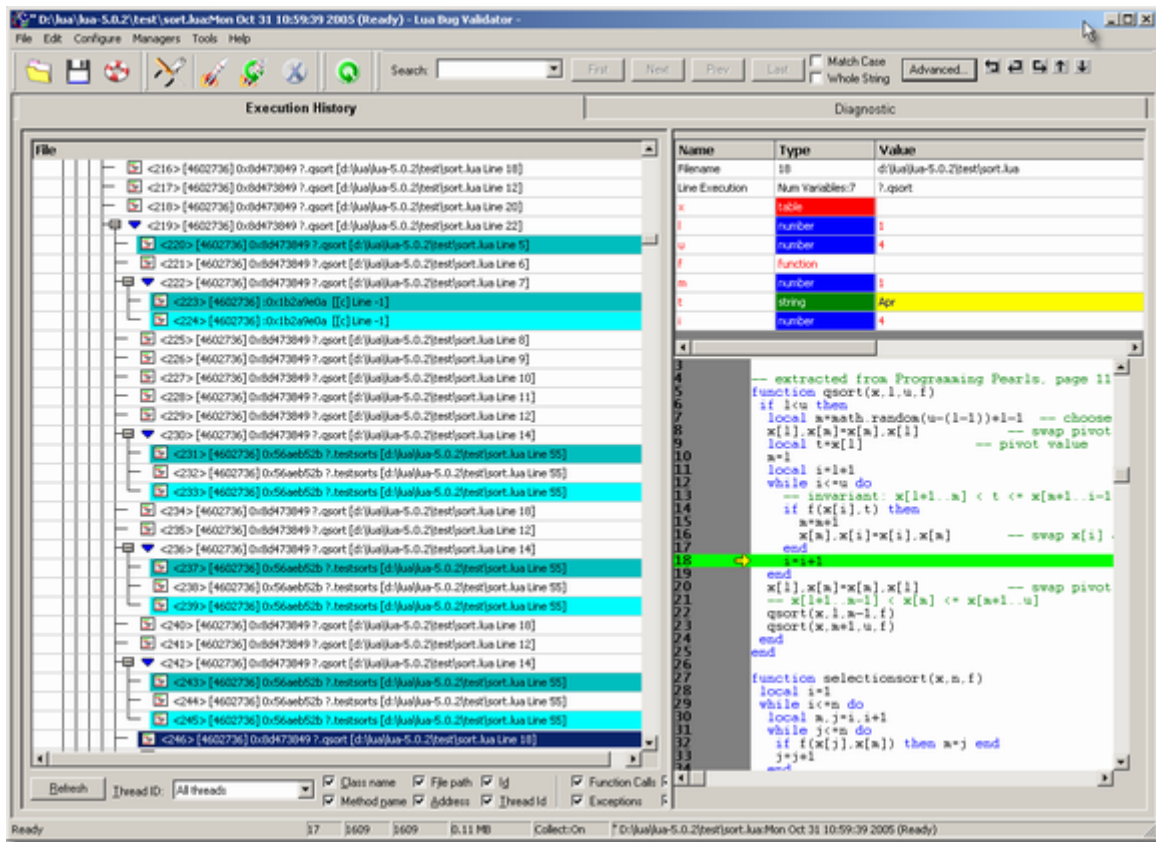
Some of the displays include an icon on the left border of the scrolled list/tree to indicate the type of data that is present on that line. The icons are shown below, with an explanation.

General

-  Option enabled.
-  Option Disabled.
-  Source code line indicator.
-  Source code.

3.4.2 Execution History

The **Execution History** tab on the tabbed window displays summary information about file visit counts. This is known as the **execution** view.

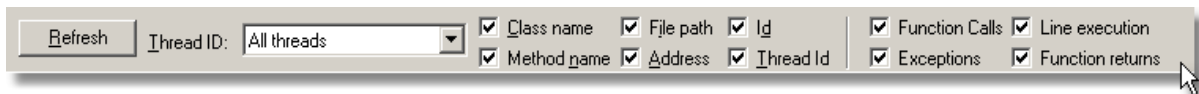


The execution history display is split into two panes. The left hand pane is a tree control showing the execution history lines, the right hand pane displays the data and source code for a particular line. The tree control when expanded shows a small code fragment for the particular line. The data for a given line consists of file and line number information, function parameters, method variables, return values and exception information as appropriate.

Each line displays the thread ID, the class name, method name the program counter, the source file name and the line number for the source file. When multiple threads are displayed, each time the thread ID changes, the background colour for the lines is changed, so that changing execution context is easy to see on the display.

Controls

The **file and line** controls are shown below.



Refresh

To refresh the display click the **Refresh** button.

Thread ID

The Thread ID combo box lists the ID of each thread. You can show the execution history for an individual thread by selecting the thread ID. To show the execution history for all threads, choose **All threads**. When displaying the execution history for all threads, the execution history shows the order in which each thread swaps in and out of the thread scheduling, this is highlighted by a change in the background colour for each thread.

Show class name

To have the class name shown in the execution history, select the **Class name** check box.

Show method name

To have the full path to the DLL shown in the execution history, select the **Method name** check box.

Show address

To show the code address in the execution history, select the **Address** check box.

Show file path

To show the encoded file name in the execution history, select the **File path** check box.

Id

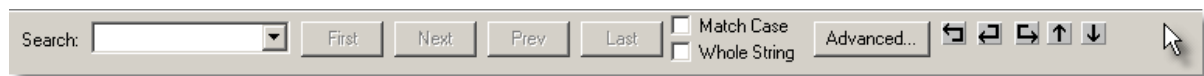
To show the history id in the execution history, select the **Id** check box.

Thread Id

To show the thread id in the execution history, select the **Thread Id** check box.

Search

The search toolbar can be used to search for data in the execution history. Data can also be automatically highlighted by using the **Advanced...** button.

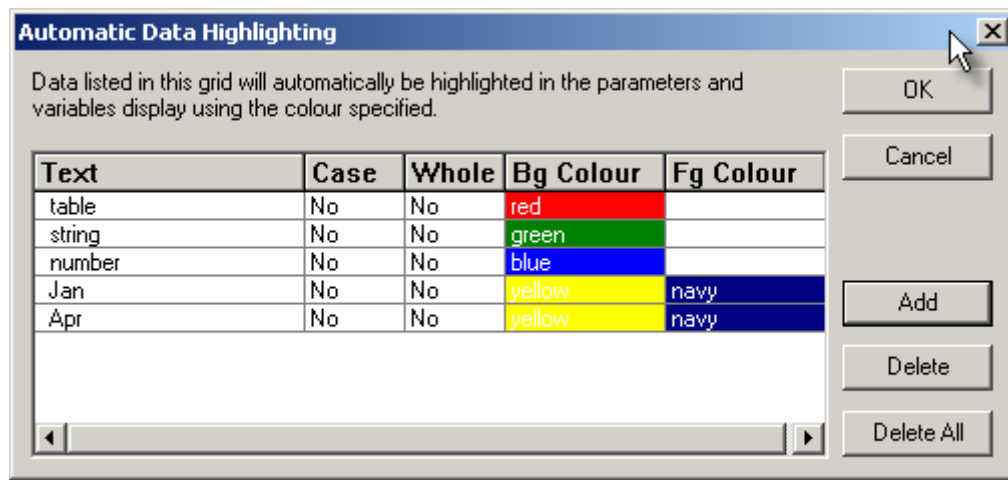


To search for data in the execution history the **Search** field and the **First**, **Next**, **Prev**, **Last** buttons are used. Previous search texts are remembered in the combo box for easy reuse. The history for search texts is 30 items.

The **Match Case** and **Whole String** check boxes affect the pattern matching of the search. Selecting the **Match Case** check box makes text searches case sensitive. Deselecting the **Match Case** check box makes text searches case insensitive. Selecting the **Whole String** check box makes text searches match the entire string. Deselecting the **Whole String** check box makes text searches match the partial strings.

Advanced

The **Advanced...** button displays the Auto Highlight dialog.



Data matching the search criteria for each item in the auto highlight dialog is highlighted in the data display above the source code window.

Name	Type	Value
Filename	18	d:\lua\lua-5.0.2\test\sort.lua
Line Execution	Num Variables:7	?.qsort
x	table	
l	number	1
u	number	4
f	function	
m	number	1
t	string	Apr
i	number	4

The columns in the Automatic Data Highlighting dialog are as follows:

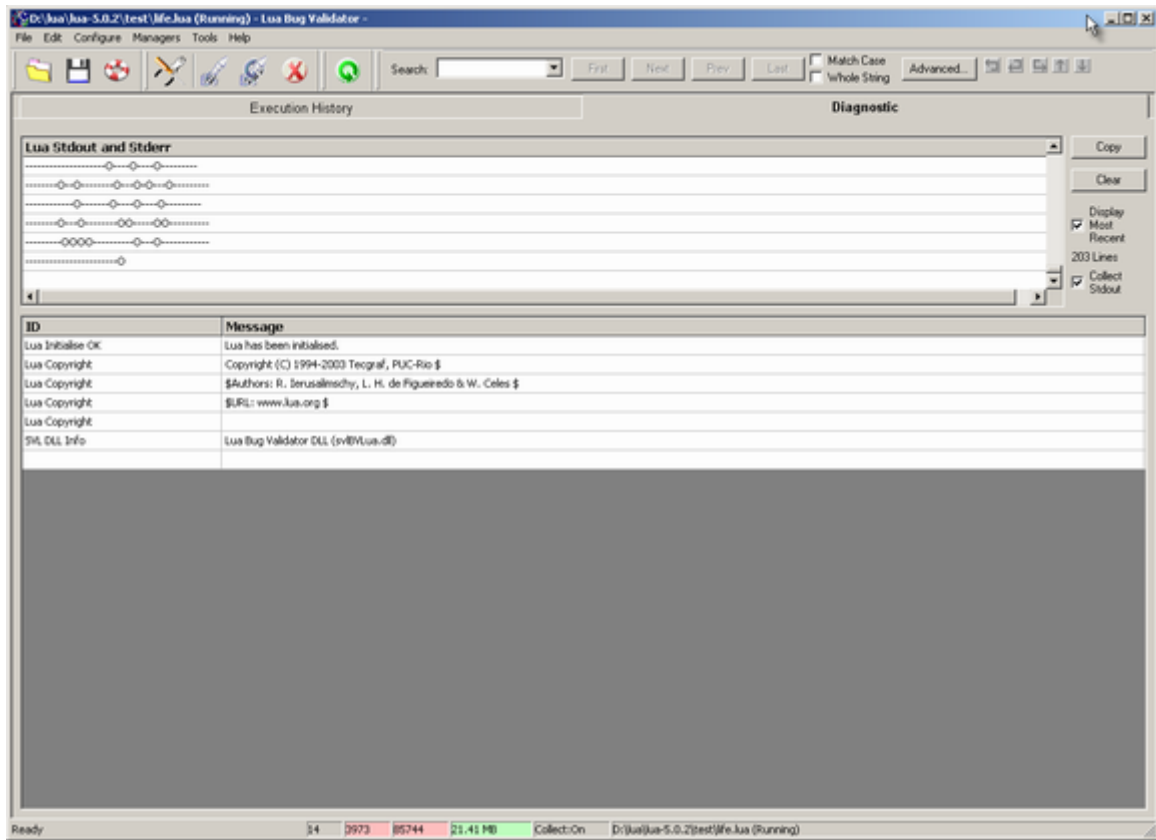
- **Text.** The text to search for.
- **Case.** Yes if the search is case sensitive. No if the search is case insensitive.
- **Whole.** Yes if the text is the entire string. No if the text is a partial match.
- **Bg Colour.** The background colour for any matching data display.
- **Fg Colour.** The foreground colour for any matching data display.

The controls are:

- **Add.** Click **Add** to add an item. Click on each field to edit the contents. The **text** field is an edit box. The other fields are combo boxes.
- **Delete.** Select one or more items, and click **Delete** to delete the items.
- **Delete All.** Click **Delete All** to delete all items.

3.4.3 Diagnostic

The **Diagnostic** tab on the tabbed window displays all diagnostic information collected from the stub. This is known as the diagnostic view.

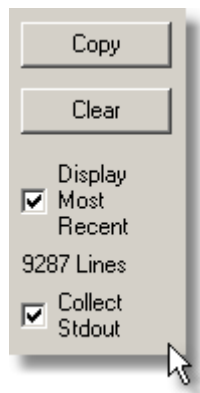


When Lua Bug Validator's stub is loaded into a target program, the stub send diagnostic information to Lua Bug Validator so that, if needed, the diagnostic information can be inspected for unusual behaviour. Many of the diagnostic messages have a text component explaining the reason for the message.

When Lua Bug Validator is operating correctly, you will see some of these messages, but not all of these diagnostic messages.

Stdout and Stderr

If the option to redirect stdout and stderr to Lua Bug Validator was selected on the launch dialog any data from stdout and stderr will be displayed in the topmost list on the diagnostic tab. Some controls are provided for managing this list.



Copy

Click the **Copy** button to copy the selected contents of the list to the clipboard. If there is no selection the entire list is copied.

Clear

Click the **Clear** button to clear the list.

Display Most Recent

Select the **Display Most Recent** checkbox to ensure the most recent entries in the list are displayed as they are added to the list.

Collect Stdout

When the **Collect Stdout** option is selected data from stdout and stderr is added to the list. When the **Collect Stdout** option is deselected data from stdout and stderr is added to the list. Please note that you must also have enabled redirection of stdout and stderr when you launched your application.

3.5 User Interface Mode

The user interface provides two modes. These are Wizard and Dialog. The mode controls the way in which data is presented to you when you are setting options that control Lua Bug Validator and when you are launching an application.

Wizard

In Wizard mode, the following interfaces are provided as wizard interfaces.

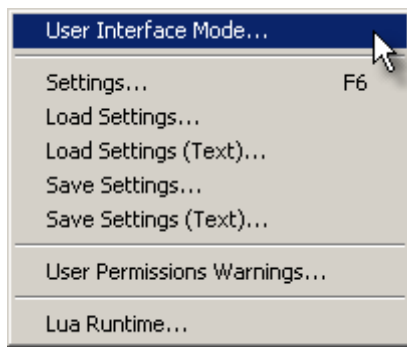
- Launch application
- Data collection and Data settings dialog. This dialog provides a simplified interface to control commonly used settings of Lua Bug Validator. All other settings are set to the default value.

Dialog

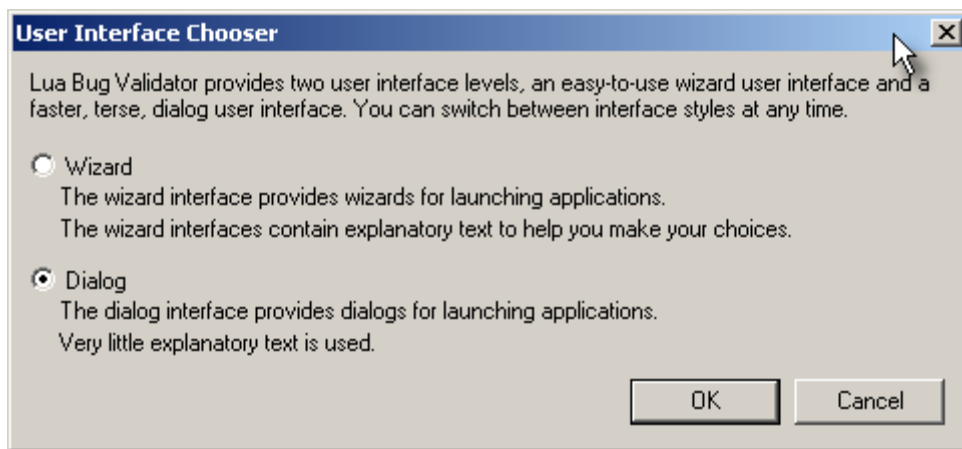
In Dialog mode, there are no wizards. The emphasis with Dialog mode is making quick selection of options without lots of explanatory text.

Setting the user interface mode

To set the user interface mode, choose **User Interface Mode...** from the **Configure** menu.



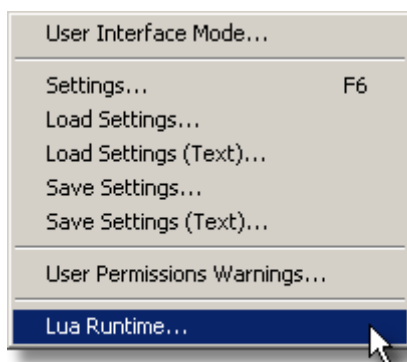
The User Interface Chooser dialog is displayed.



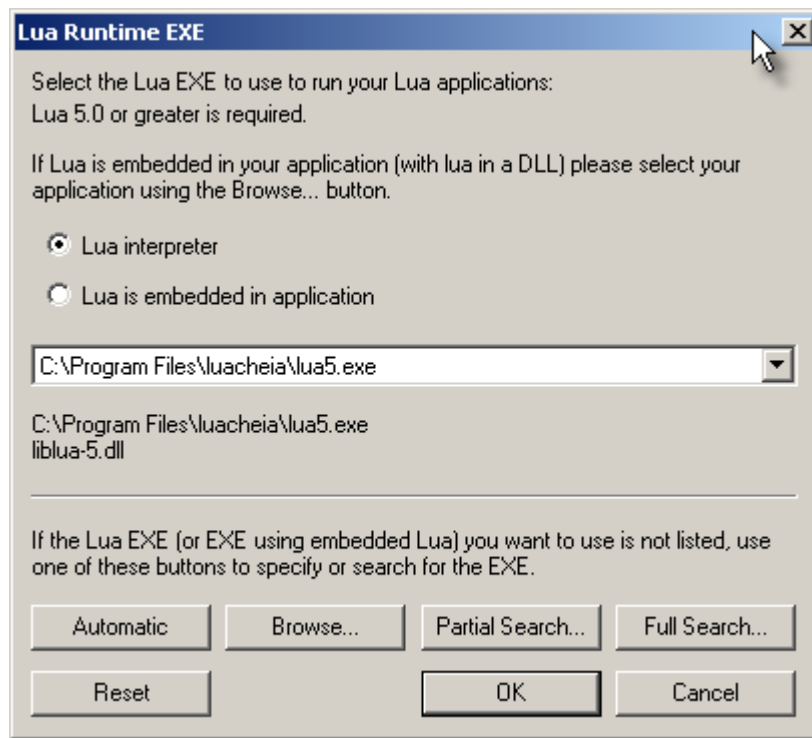
Select the mode you want to use and click OK.

3.6 Lua Runtime

To set the Lua Runtime choose **Lua Runtime...** on the **Configure** menu.



The Lua Runtime dialog is displayed.

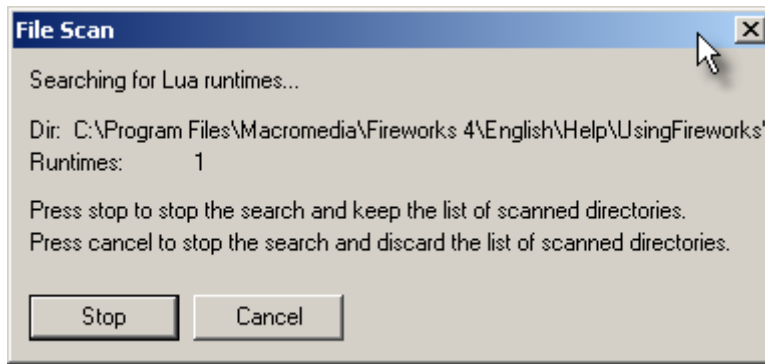


Select the **Lua interpreter** checkbox if you are using the Lua interpreter to run your application. Select the **Lua is embedded in application** checkbox if your application has Lua embedded in it.

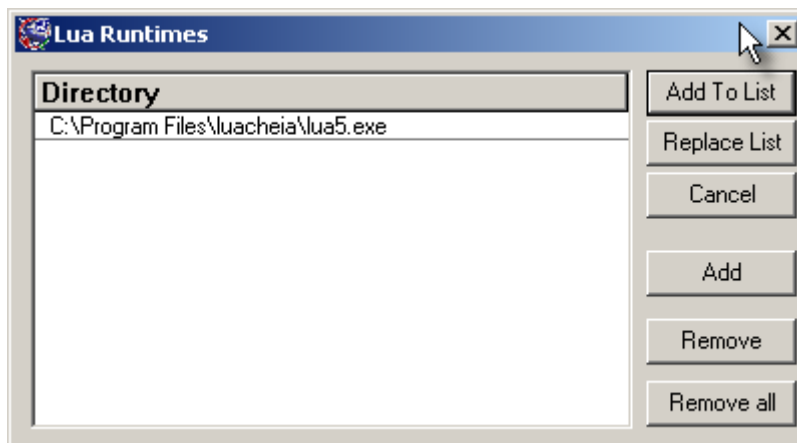
Select the EXE to use from the combo box. If there are no EXEs, or you have just installed a new version of Lua you have four options.

- **Automatic.** Clicking the Automatic button searches the registry and all paths that are mentioned in any environment variable for Lua EXEs.
- **Browse.** Clicking the **Browse...** button allows you to select the appropriate Lua EXE using the standard Microsoft File Browser.
- **Partial Search.** Clicking the **Partial Search...** button displays performs a search on a specified directory tree for installed Lua EXEs. The Microsoft Directory browser dialog is used to specify the directory to search from.
- **Full Search.** Clicking the **Full Search...** button displays searches your computer's disk drives for installed Lua EXEs.

During a Partial Search or Full Search a progress dialog is displayed. The **Cancel** button cancels the search. The **Stop** button stops the search and keeps the results.



When a Partial Search or Full Search is completed the results are displayed. The **Add To List** button adds the results to any existing Lua Runtime specifications. The **Replace List** button replaces any existing Lua Runtime specifications. The **Cancel** button cancels the dialog without affecting the existing Lua Runtime specifications. The **Add** button allows you to add to the list of Lua Runtimes. The **Remove** button allows you to remove any selected Lua Runtimes from the list. The **Remove All** button removes all Lua Runtimes.

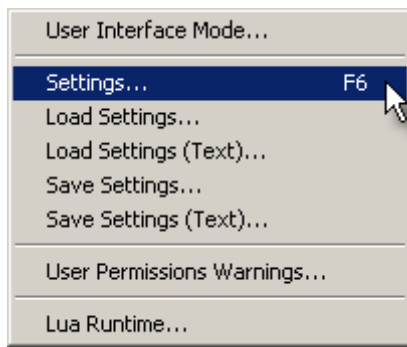


- **Add To List.** Choose **Add to List** to add the files displayed in the list to the list of Lua Runtimes.
- **Replace List.** Choose **Replace List** to replace the Lua Runtimes with the files from the file scan.
- **Cancel.** Choose **Cancel** to discard the list of files.
- **Add.** Choose **Add** to add a file to the list of files.
- **Remove.** Select a file on the list and click **Remove** to remove the file from the list.
- **Remove all.** Choose **Remove all** to remove all files from the list.

3.7 Settings

Lua Bug Validator allows a fine degree of control of which data is displayed. This control is provided by some global settings which affect all displays on Lua Bug Validator. In addition to the global settings each of the main display windows has its own local controls. This section describes the global settings available from the display settings dialog.

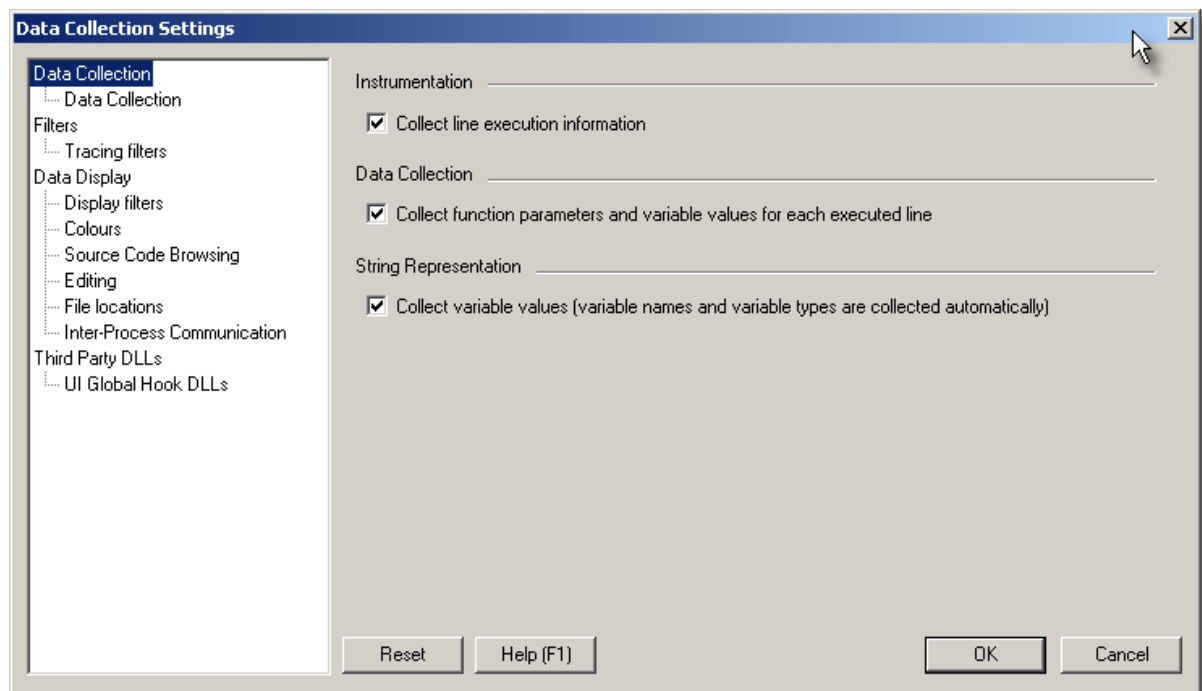
To view the display settings dialog, choose **Settings...** on the **Configure** Menu,



or click on the display settings icon on the session toolbar.



The settings dialog is displayed.

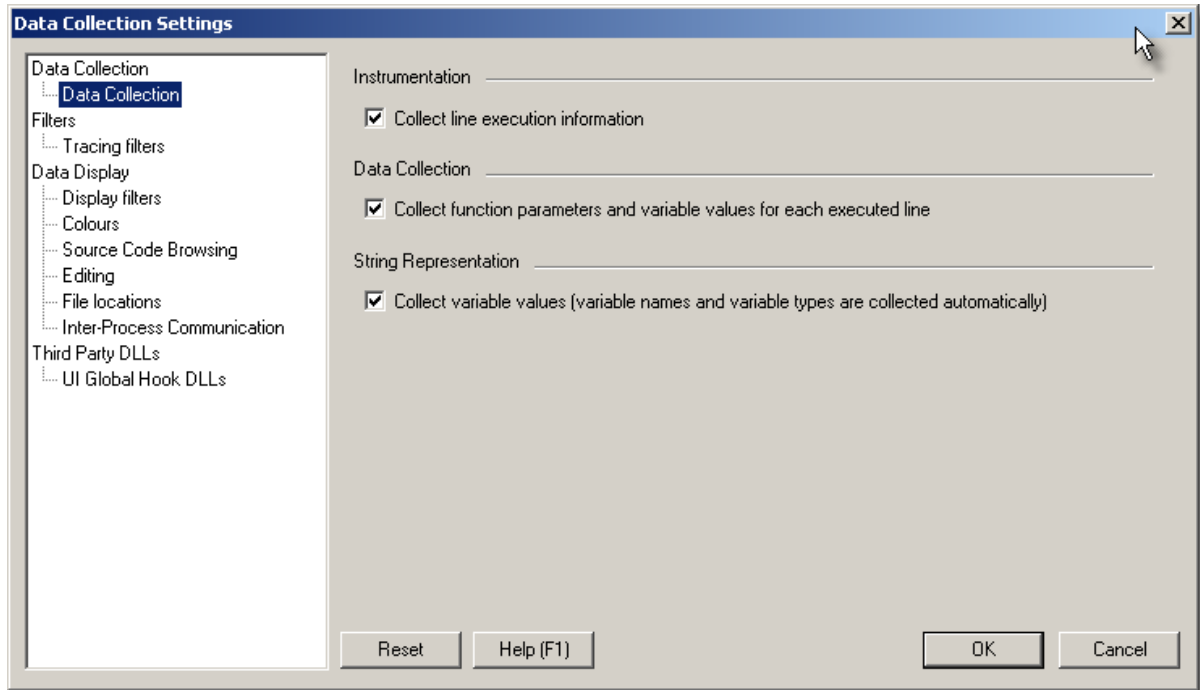


Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator Client, not just the settings visible on the current tab of the dialog.

3.7.1 Data Collection

The **Instrumentation** tab allows you to configure the type of data collected by Lua Bug Validator.



Lua Bug Validator collects execution information related to function calls, function returns and exceptions automatically. You can also record information about individual line execution by selecting the **Collect line execution information** check box.

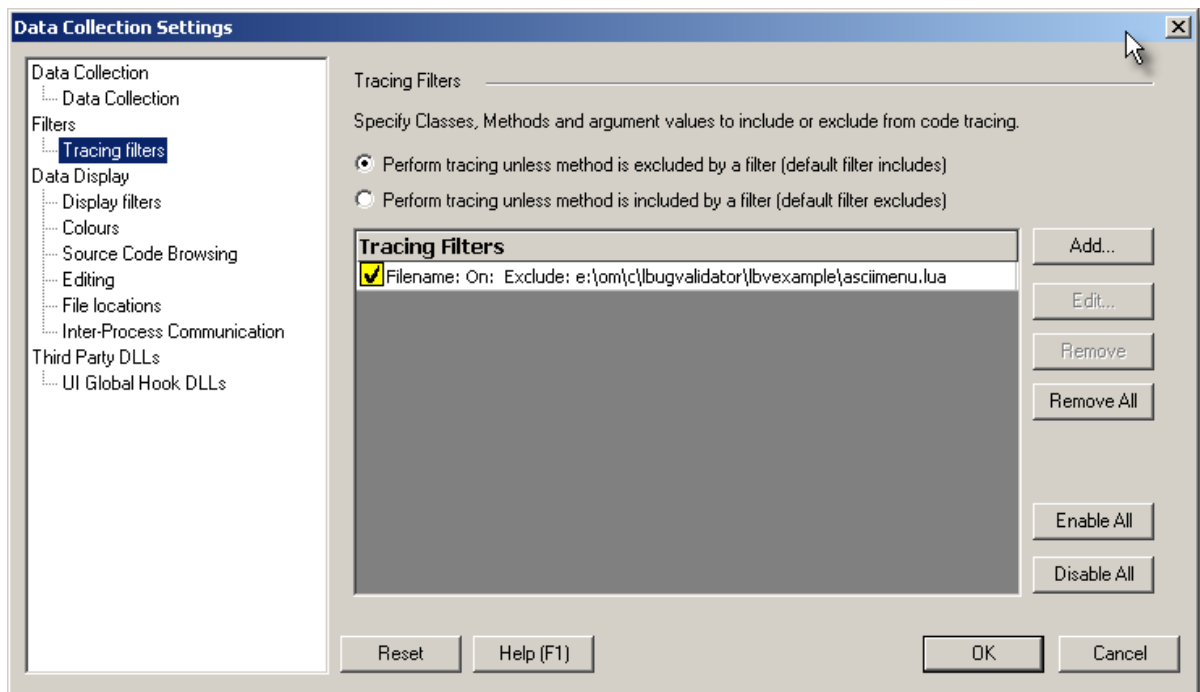
Collecting information about function parameters, method variables, return values and exceptions can be useful. However collecting this information takes additional execution time and storing this information uses more space. For this reason Lua Bug Validator allows you to choose which data to collect. Select the appropriate check boxes to enable collecting of function parameters and method variables.

Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator Client, not just the settings visible on the current tab of the dialog.

3.7.2 Tracing Filters

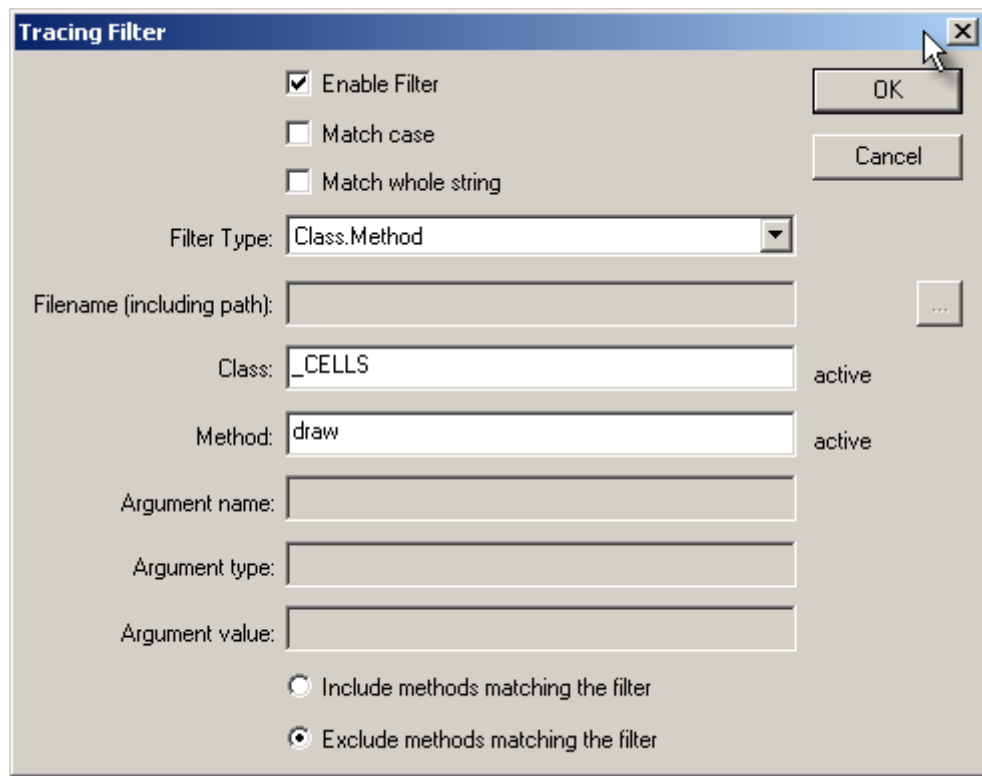
The **Tracing Filters** tab allows you to configure filters to restrict the collection of data from the application being monitored.



The tracing filters work by specifying a data match which will include or exclude data in the execution history log. When more than one filter matches an item of data, the data is added to the log if the number of including filters is greater than or equal to the number of excluding filters. Data can be matched by filename, class, method, class and method, argument type, argument name, argument value (where argument means any parameter, variable or return value).

The inclusion and exclusion concept allows you to exclude data from one part of your application unless another criteria for inclusion is met.

- **Add....** To add a new filter, click the **Add...** button. The Tracing Filter dialog is displayed.
- **Edit....** To edit an existing filter, select the filter and click the **Edit...** button. The Tracing Filter dialog is displayed.
- **Remove.** Select the items to remove and click the **Remove** button.
- **Remove All.** Click the **Remove All** button to remove all tracing filters.
- **Enable All.** Click the **Enable All** button to enable all tracing filters.
- **Disable All.** Click the **Disable All** button to disable all tracing filters.



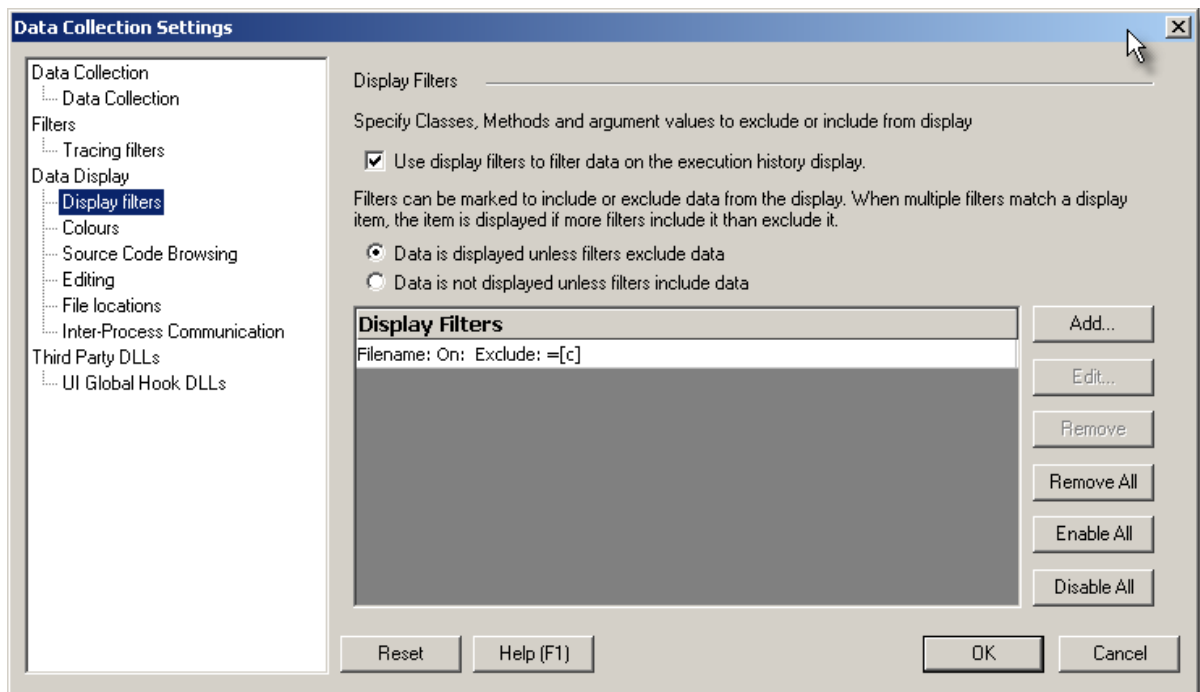
Select the filter type by using the **Filter Type** combobox. The appropriate edit fields are enabled. Each field supports the use of the '*' wildcard to match any sequence of characters. Enter the string(s) to filter. Choose how the data matching is to be performed by selecting or deselecting the **Match case** check box and **Match whole string** check box. Finally choose if the filter is going to include data or exclude data from the display.

Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator Client, not just the settings visible on the current tab of the dialog.

3.7.3 Display Filters

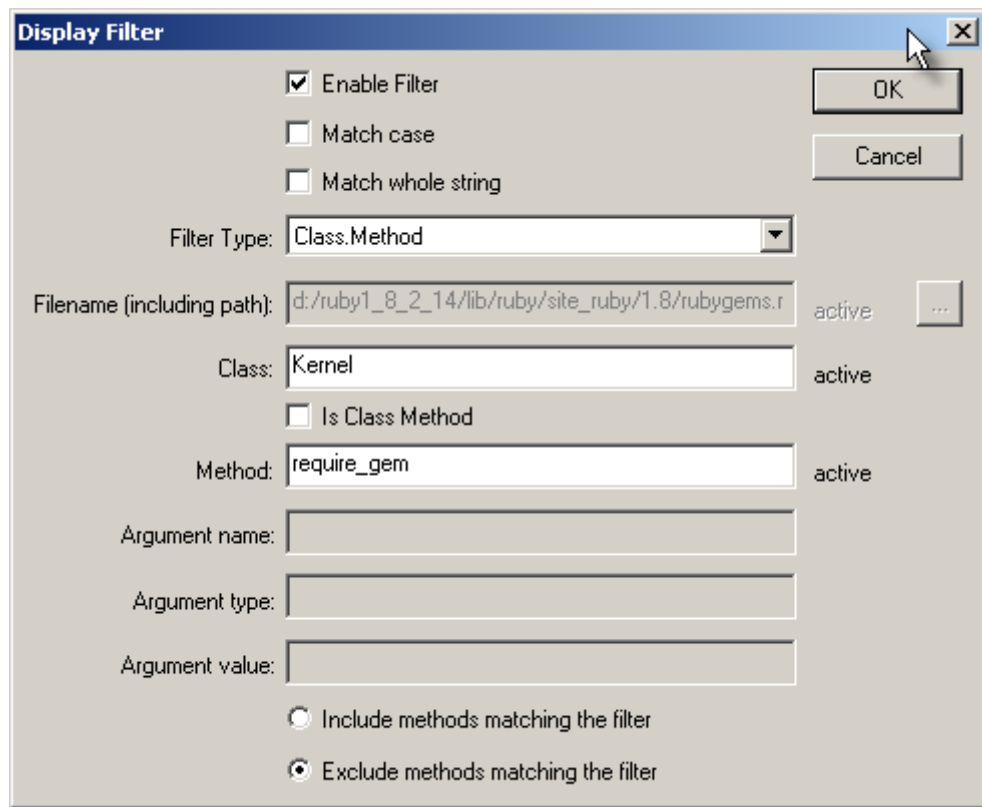
The **Display Filters** tab allows you to configure filters to restrict the display of data on the execution history view.



The display filters work by specifying a data match which will include or exclude a data item from the display. When more than one filter matches an item of data, the data is displayed if the number of including filters is greater than or equal to the number of excluding filters. Data can be matched by filename, class, method, class and method, argument type, argument name, argument value (where argument means any parameter, variable or return value).

The inclusion and exclusion concept allows you to exclude data from one part of your application unless another criteria for display is met.

- **Add....** To add a new filter, click the **Add...** button. The Display Filter dialog is displayed.
- **Edit....** To edit an existing filter, select the filter and click the **Edit...** button. The Display Filter dialog is displayed.
- **Remove.** Select the items to remove and click the **Remove** button.
- **Remove All.** Click the **Remove All** button to remove all display filters.
- **Enable All.** Click the **Enable All** button to enable all display filters.
- **Disable All.** Click the **Disable All** button to disable all display filters.



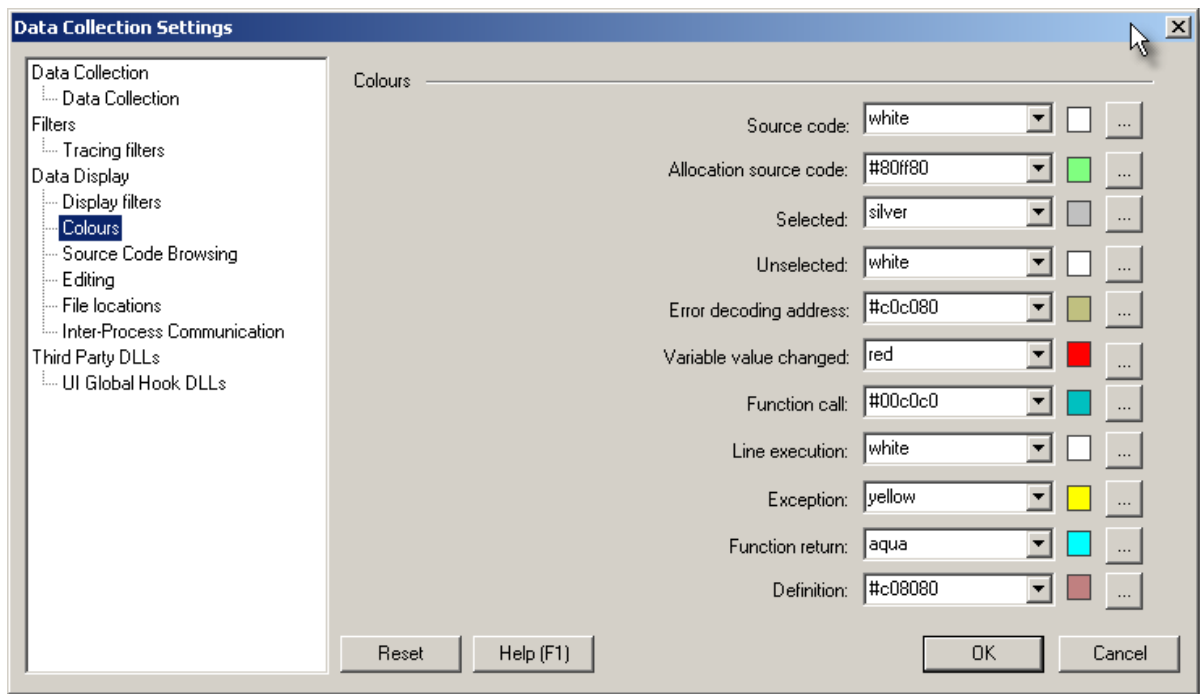
Select the filter type using the **Filter Type** combobox. The appropriate edit fields are enabled. Each field supports the use of the ****** wildcard to match any sequence of characters. Enter the string(s) to filter. Choose how the data matching is to be performed by selecting or deselecting the **Match case** check box and **Match whole string** check box. Finally choose if the filter is going to include data or exclude data from the display.

Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator Client, not just the settings visible on the current tab of the dialog.

3.7.4 Colours

The **Colours** tab allows you to specify the colours that will be used to represent each type of data item collected by Lua Bug Validator.



For each colour there are two ways of specifying the colour to use.

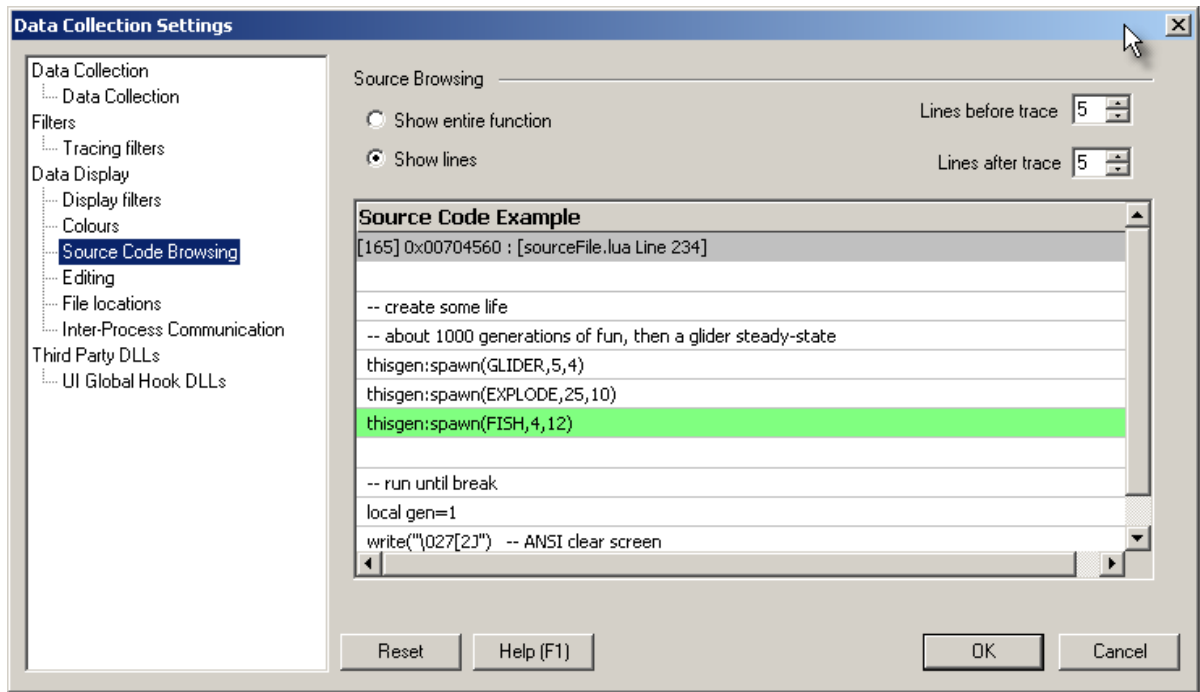
- Select a named colour from the combo box.
- Click on the button labeled ... to edit the colour using the standard Microsoft® colour choosing dialog.

Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator, not just the settings visible on the current tab (Colours) of the dialog.

3.7.5 Source Code Browsing

The **Source Code Browsing** tab allows you control how source code is displayed.



Source Browsing

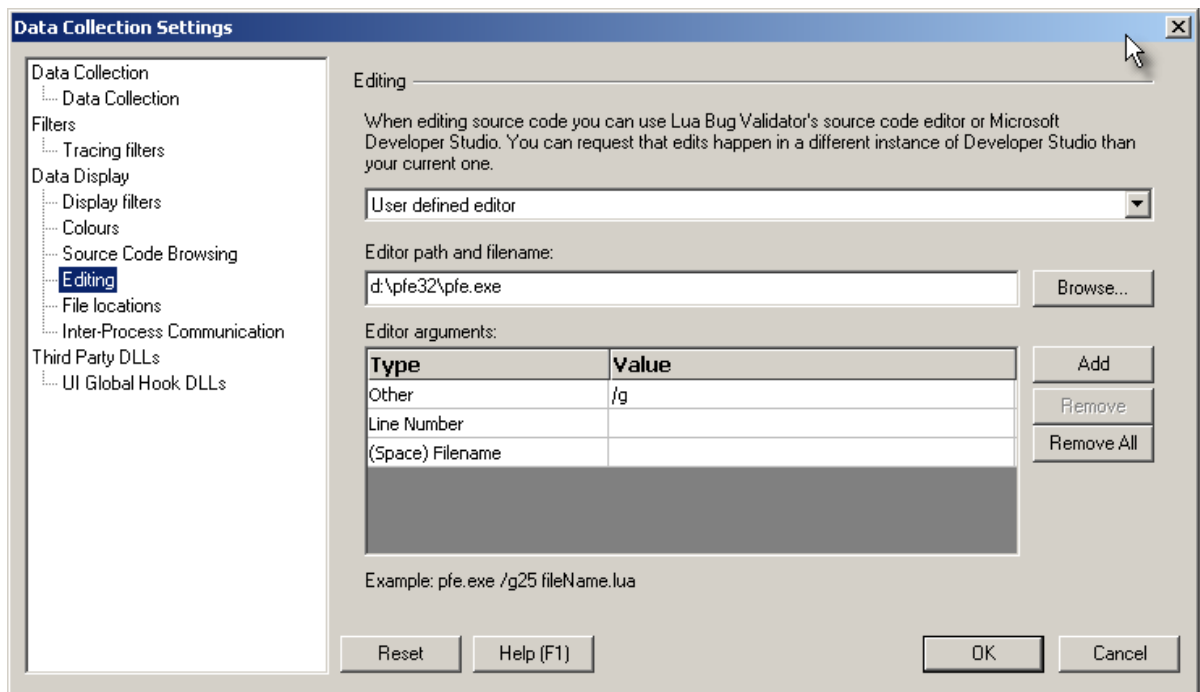
When expanding the callstack to view the source code in a view, Lua Bug Validator can show the entire function containing the executed line or just a fragment of the surrounding code.

Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator Client, not just the settings visible on the current tab (Files) of the dialog.

3.7.6 Editing

The **Editing** tab allows you to configure the editing options for Lua Bug Validator.



Editing

When editing source code from within Lua Bug Validator you can do the editing using Lua Bug Validator's syntax coloured source code editor, using Microsoft® Developer Studio® 6.0, or using a custom editor definition. Select the appropriate entry in the combo box.

When using Microsoft® Developer Studio® to edit files, you can choose to edit source code using a currently open instance of Developer Studio® (probably the same one you are using to develop your application), or to open a new instance of Developer Studio®.

Custom Editor

To specify the editor to use, type the path to the editor in the edit field **Edit path and filename**, or use the **Browse...** button to use Microsoft's file dialog to specify the editor.

The picture above shows an example configuring the Programmers File Editor (Pfe32.exe) to edit a file and position the cursor at the appropriate line using the /g command line switch.

If you specify no arguments, the editor will be passed the filename to edit. If the editor requires command line switches to specify the filename and/or the line number and/or any optional arguments, you must specify the arguments in the list of **Editor arguments**. Arguments are appended to the editor name in the order shown in the list. An example command line is shown below the list for the file `fileName.lua`, line 25.

There are six types of argument:

- **(Space) Filename**. This appends a space followed by the filename.
- **Filename**. This appends the filename.
- **(Space) Line Number**. This appends a space followed by the line number.
- **Line Number**. This appends the line number.
- **Space**. This appends a space.
- **Other**. This appends the text typed in the **Value** column of the list.

Add

To add an argument type (and value for type "Other"), click the **Add** button. Use the combo box in the list field to choose the type of argument. If the argument type is "Other", type the text in the **Value** column.

Remove

To remove an argument, select the argument in the list and click the **Remove** button. Alternatively the keyboard 'delete' key can be used.

Remove All

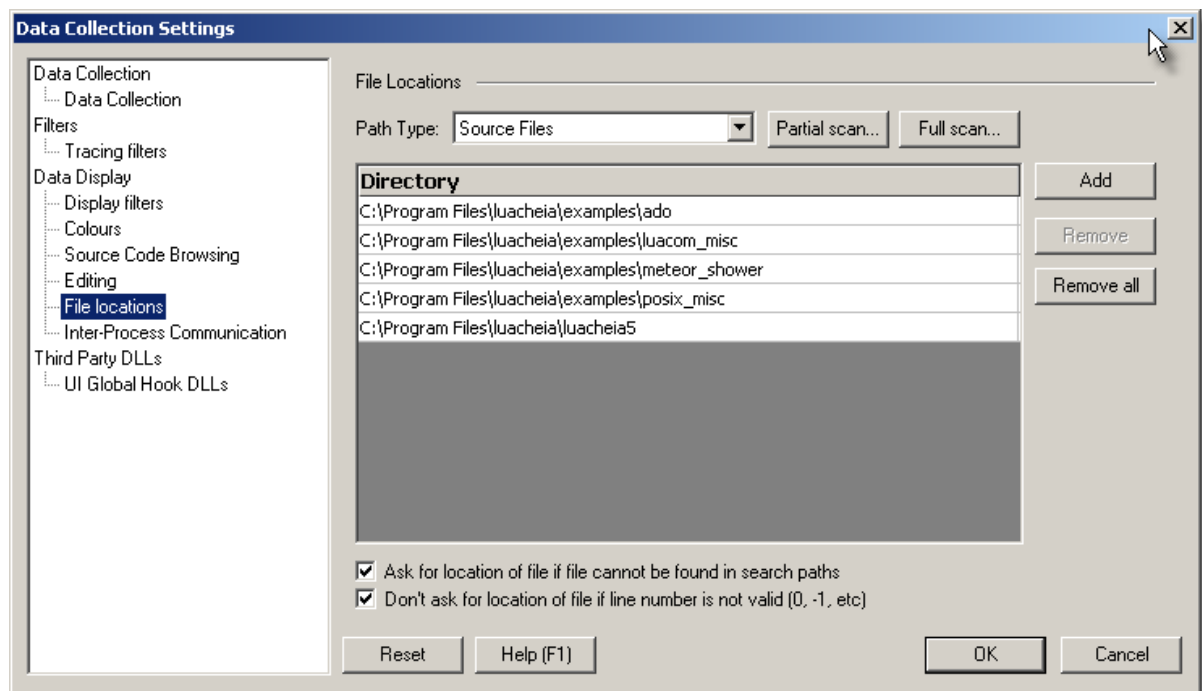
To remove all arguments, click the **Remove All** button.

Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator, not just the settings visible on the current tab (Editing) of the dialog.

3.7.7 File Locations

The **File Locations** tab allows you to specify which directories Lua Bug Validator Client should look in for source code files and third party source code files. This is required as some of the file information is only the filename and not the directory name. When this happens Lua Bug Validator Client scans the directories that it knows about and tries to find the file in these directories. When Lua Bug Validator Client fails to find a file it will ask the user to specify where the file is.



In the picture above we can see that the user has specified that the source files for the target program are in one directory.

Path Type

The path type combo box allows you to choose which category of file you are specifying paths for. It is possible to specify paths for the following types of file:

- Source code files.
- Third party source code files.

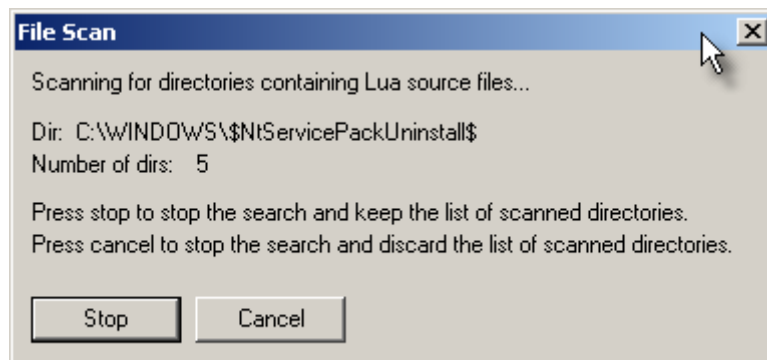
It is not necessary to specify any paths, and you do not need to specify paths for all types of files. It is recommended that directories for source code files and for third party source code files are specified as it will help Lua Bug Validator Client provide source code browsing without user intervention.

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

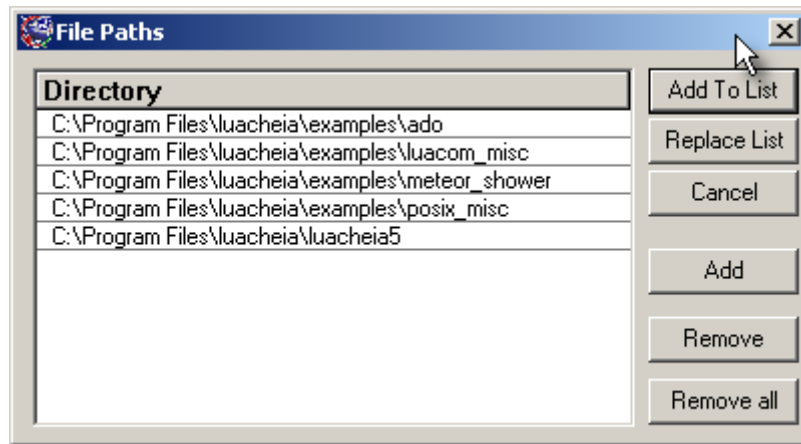
- Add a file. Click the **Add** button. Type the name of the directory into the scrolled list.
- Remove a file. Select the directory in the scrolled list to be removed. Click the **Remove** button.
- Remove all directories. Click the **Remove all** button to remove all directories from the scrolled list.
- Scan all disk drives on the computer for the files. To do a complete scan of all the disk drives on the computer, click on the **Full Scan...** button. A progress dialog indicating which files are being processed is displayed. Once the scan is complete you can add the files to the files in the list, or replace the files in the list with the new list of files from the scan.
- Scan part of a disk drive on the computer for the files. To do a partial scan of part of a disk drive, click on the **Partial Scan...** button. A directory browser dialog is displayed. Choose the disk drive and directory you want to scan. When you have chosen the drive and directory to scan a progress dialog indicating which files are being processed is displayed. Once the scan is complete you can add the files to the files in the list, or replace the files in the list with the new list of files from the scan.

File Scan

The file scan dialog is shown below. To stop the search for files and keep the list of files the file scan has found, click **Stop**. To stop the search and discard the list of files the file scan has found, click **Cancel**.



When the file scan is complete, the list of files that has been found is displayed in the File Paths dialog.



- **Add To List.** Choose **Add to List** to add the files displayed in the list to the list of files on the **Files** tab of the display settings dialog.
- **Replace List.** Choose **Replace List** to replace the files displayed on the **Files** tab of the display settings dialog with the files from the file scan.
- **Cancel.** Choose **Cancel** to discard the list of files.
- **Add.** Choose **Add** to add a file to the list of files.
- **Remove.** Select a file on the list and click **Remove** to remove the file from the list.
- **Remove all.** Choose **Remove all** to remove all files from the list.

Source Code Display

When displaying and/or editing source code, Lua Bug Validator will search for the source code file in the paths specified using the File Locations dialog. When a file cannot be found, Lua Bug Validator can ask the user to specify the file location, or Lua Bug Validator can be instructed to ignore the file. To control this functionality, select or deselect the **Ask for location of file if file cannot be found in search paths** check box.

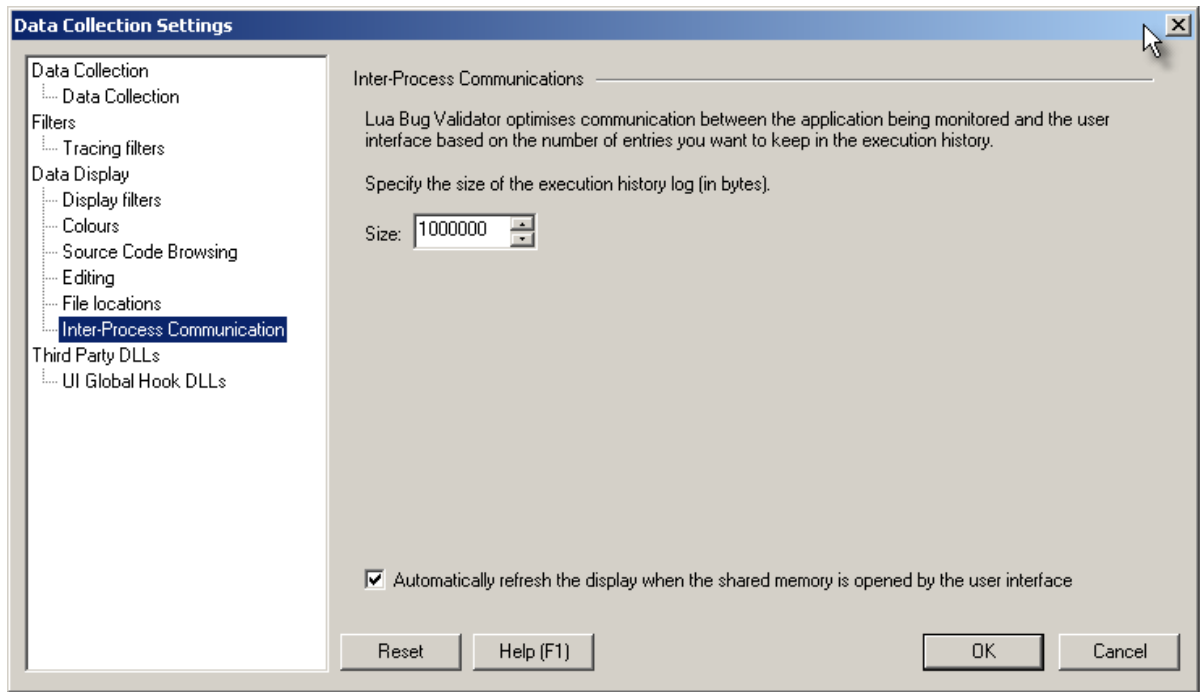
Additionally, if the line number is not known (0, -1, etc), Lua Bug Validator can be instructed to ignore the file. To control this functionality, select or deselect the **Don't ask for location if line number is not valid** check box.

Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator Client, not just the settings visible on the current tab (Files) of the dialog.

3.7.8 Inter-Process Communication

The **Inter-Process Communication** tab allows you to configure the size of the buffer used to communicate with the user interface.



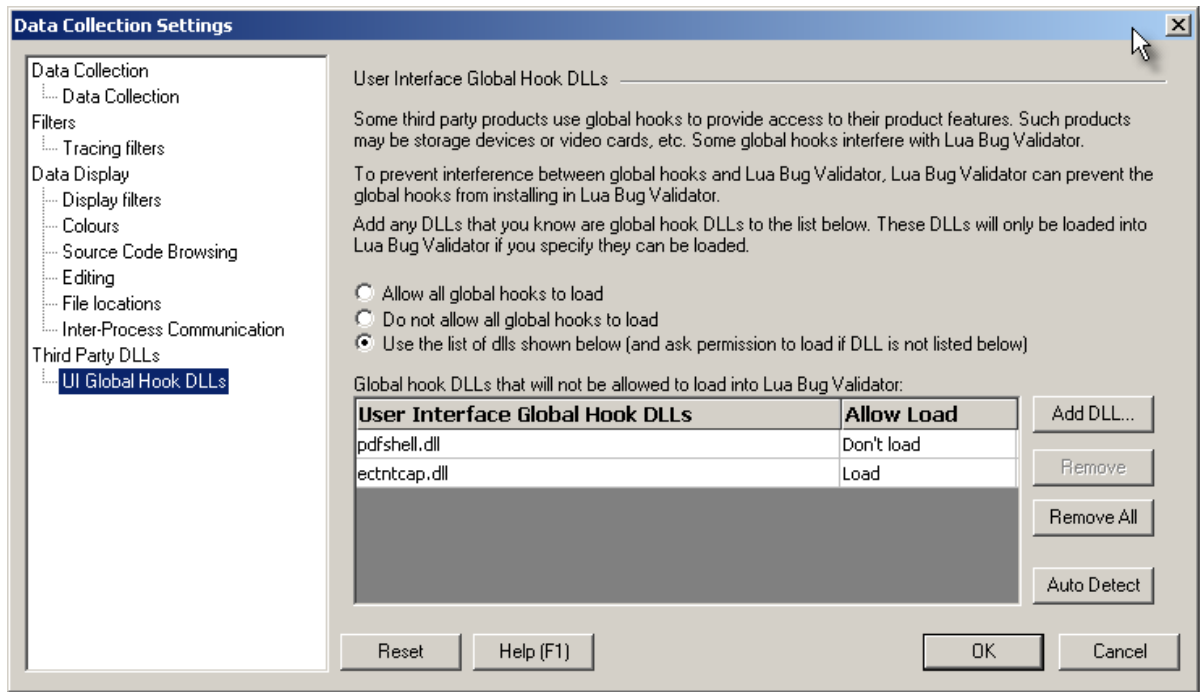
Lua Bug Validator Client communicates the execution history to the user interface via a buffer. Larger numbers for the number of entries mean that you will have more information in the execution history.

Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator Client, not just the settings visible on the current tab of the dialog.

3.7.9 UI Global Hook DLLs

The **User Interface Global Hook DLLs** tab allows you to detect global hook DLLs and to specify 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 which also uses a global hook via the IMGHOOK.DLL. The IMGHOOK.DLL in this instance is used so that when the browse for files and browse for folders user interfaces are used they correctly display all the storage devices on the computer, including the zip drive and any special options for the drive.

The IMGHOOK.DLL mentioned does not interfere with the correct operation of Lua Bug Validator. Some hook DLLs do interfere with the correct operation of Lua Bug Validator. The User Interface Global Hook DLLs tab allows you to automatically detect and/or to specify DLLs that should be treated as global hook DLLs. Any DLL that is specified as a global hook DLL will fail to load into Lua Bug Validator when loaded via LoadLibrary() or LoadLibraryEx().

Behaviour

- **Allow all global hooks to load.**

Select this option to allow all global hook DLLs to load into Lua Bug Validator.

- **Do not allow all global hooks to load.**

Select this option to prevent any global hook DLLs from loading into Lua Bug Validator.

- **Use the list of dlls shown.**

Select this option to specify which DLLs to allow to load and to specify which DLLs should not load. Any DLLs not result in the user being asked for permission to load a DLL.

Add DLL...

To add a DLL to the list of global hook DLLs, click the **Add DLL...** button. The standard Microsoft file chooser dialog is displayed. Choose a dll to add to the list of global hook DLLs.

Remove

To remove a DLL from the list of Global Hook DLLs, select the DLL on the list and click the **Remove** button.

Remove All

To remove all DLLs from the list of Global Hook DLLs, click the **Remove All** button.

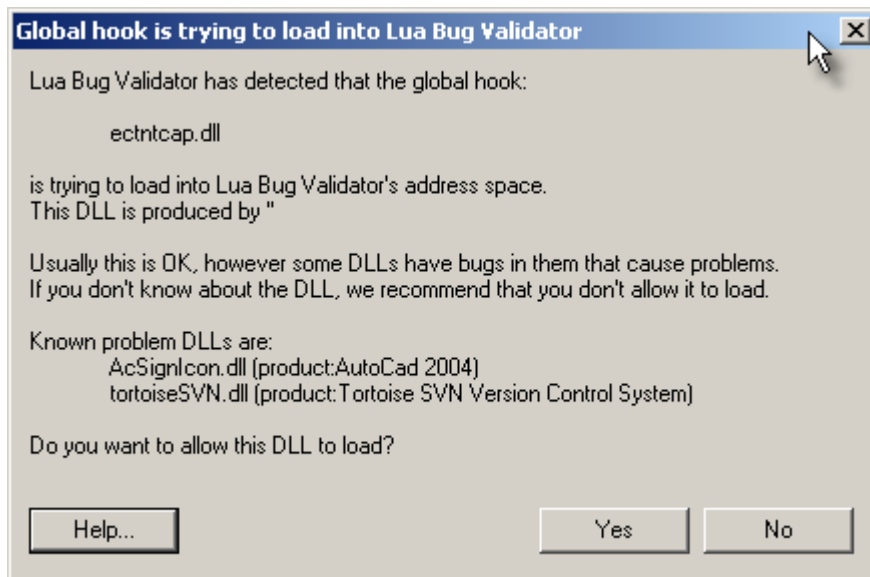
Auto Detect

To automatically detect DLLs which may be global hook DLLs, click the **Auto Detect** button. Lua Bug Validator will identify any DLLs in the current process (Lua Bug Validator) that are not DLLs that Lua Bug Validator uses. Such DLLs are likely to be global hook DLLs.

If you wish Lua Bug Validator to unload any global hook DLLs that get loaded into the target process before Lua Bug Validator can prevent them from being loaded, select the **Unload already loaded global hooks** check box.

Global Hook Warning Dialog

When the global hook behaviour is set to "**Use the list of dlls shown**", and a global hook is loaded that is not on the list of known global hooks, the user is presented with a dialog informing them of the event. The user can then allow the global hook to load, or deny the global hook from loading. A sample dialog is shown below.



- **Help.**
Displays this help page.
- **Yes.**
Allows the DLL to load. This is recorded in the list of DLLs, so that this question will not be asked again.
- **No.**
Does not allow the DLL to load. This is recorded in the list of DLLs, so that this question will not be asked again.

Reset

The display settings dialog has a button labeled **Reset** at the bottom left of the dialog. This button resets all display related settings in Lua Bug Validator, not just the settings visible on the current tab of the dialog.

3.8 Loading and Saving Settings

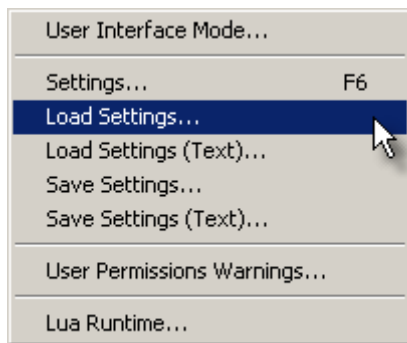
Lua Bug Validator allows the settings to be loaded and saved to data files as well as the current settings being stored in the registry.

3.8.1 Loading settings

Lua Bug Validator saves most of its settings in the Windows Registry. However Lua Bug Validator also allows users to specify a specific set of settings, of the user's choice, each time a Lua Bug Validator is executed from the command line. The settings for each test are specified by the `-settings` command line argument.

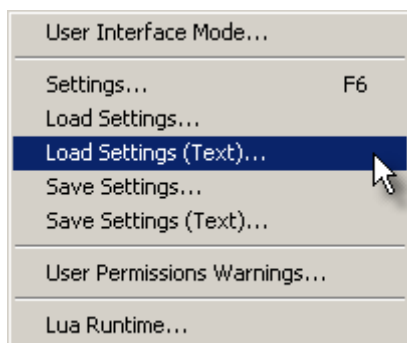
The `-settings` command line argument specifies a file to load the settings from. To facilitate the creation and modification of settings in a file Lua Bug Validator provides the ability to load and save settings.

To load a previously saved settings file, choose **Load Settings...** on the **Configure** menu.



A file selection dialog is displayed. Select the file to load the settings.

To load a previously saved settings file that was saved in ASCII text format, choose **Load Settings (Text)...** on the **Configure** menu.



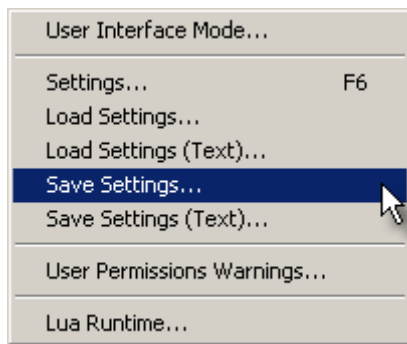
A file selection dialog is displayed. Select the file to load the settings.

3.8.2 Saving settings

Lua Bug Validator saves most of its settings in the Windows Registry. However Lua Bug Validator also allows users to specify a specific set of settings, of the user's choice, each time a Lua Bug Validator is executed from the command line. The settings for each test are specified by the `-settings` command line argument.

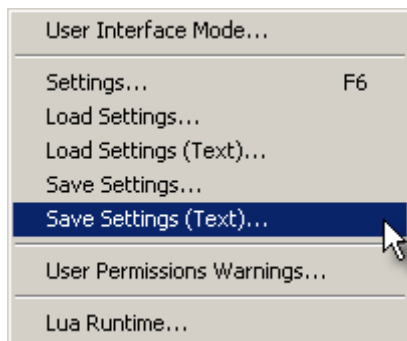
The `-settings` command line argument specifies a file to load the settings from. To facilitate the creation and modification of settings in a file Lua Bug Validator provides the ability to load and save settings.

To save settings to a file, choose **Save Settings...** on the **Configure** menu.



A file selection dialog is displayed. Select the file to save the settings.

To save settings to a file in ASCII text format, choose **Save Settings (Text)...** on the **Configure** menu.

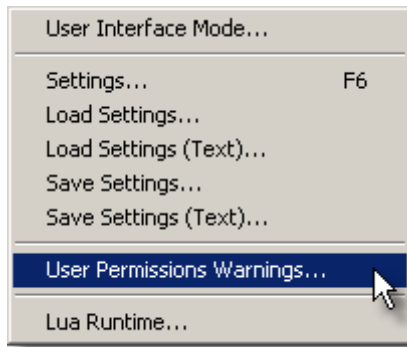


A file selection dialog is displayed. Select the file to save the settings.

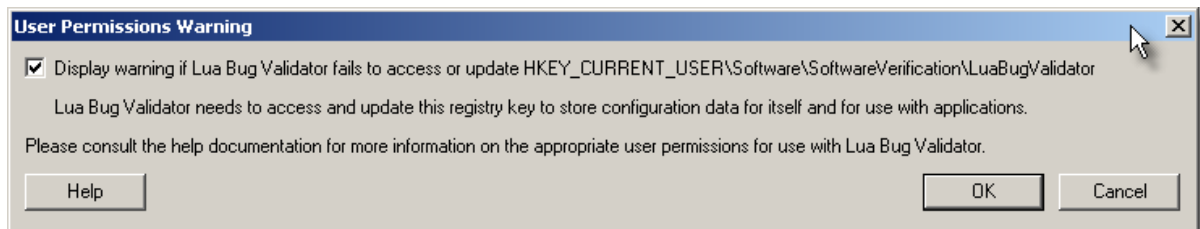
3.9 User Permissions Warnings

Lua Bug Validator displays warning dialogs when errors occur accessing the Registry and/or obtaining debugging privileges. These warnings are enabled by default, but can be enabled or disabled as desired. The User Permissions Warnings dialog is used to enable or disable these warnings.

To display the User Permissions Warnings dialog, choose **User Permissions Warnings...** on the **Configure** menu.



The User Permissions Warning dialog is displayed.



Select or deselect the check boxes appropriate to the warnings you wish to receive and click OK to accept the changes. The Help button displays the User Permissions help topic.

You may also want to read this topic relating to creating Power User accounts for Windows XP.

3.10 Managers

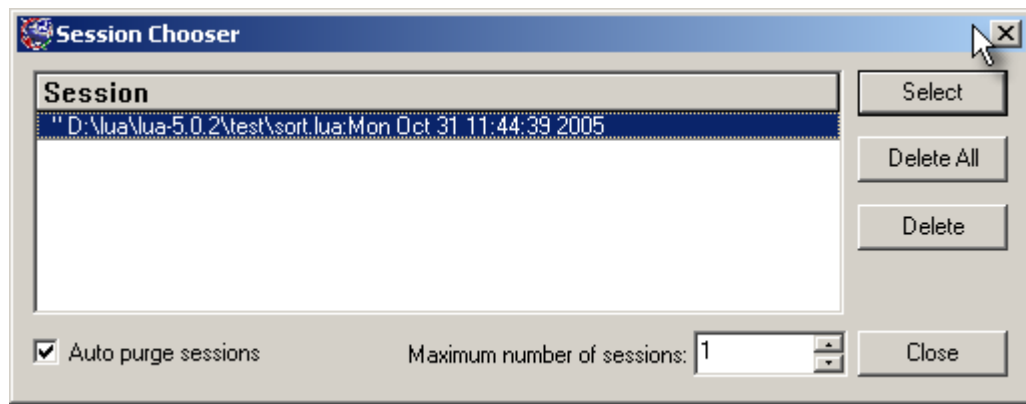
Lua Bug Validator provides some user interfaces to allow you to manage or inspect data collected by Lua Bug Validator.

3.10.1 Session Manager

Lua Bug Validator provides a session manager to allow you to manage the sessions that Lua Bug Validator is controlling. To display the session manager choose **Session Manager...** on the **Managers** menu.



The session manager is displayed. The current session is highlighted in the scrolled list.



The scrolled list named **Session** displays all the loaded sessions. Each time a session is started or loaded it is added to the session manager and Lua Bug Validator treats that session as the current session being worked.

Each session has a name that is created in the following way:
Name of the executable program: Weekday Month Day Hour:Minute:Second Year

Select

To make a specific session the current session, select the session in the scrolled list and click the **Select** button.

Delete

To delete a session, select the session in the scrolled list and click the **Delete** button. If you delete the current session whilst an application is running, the application will remain running. You will not be able to inject Lua Bug Validator into that instance of the program a second time.

Delete All

To delete all sessions, click the **Delete All** button.

Close

To close the manager, click the **Close** button.

Limiting the number of sessions

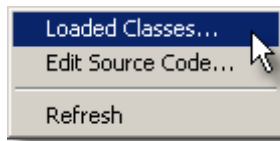
To automatically limit the number of sessions held in memory at any time, select the **Auto purge sessions** check box. Then type the maximum number of session to be held in memory into the **Maximum number of sessions** field, or use the spinner control to select the number.

3.11 Tools

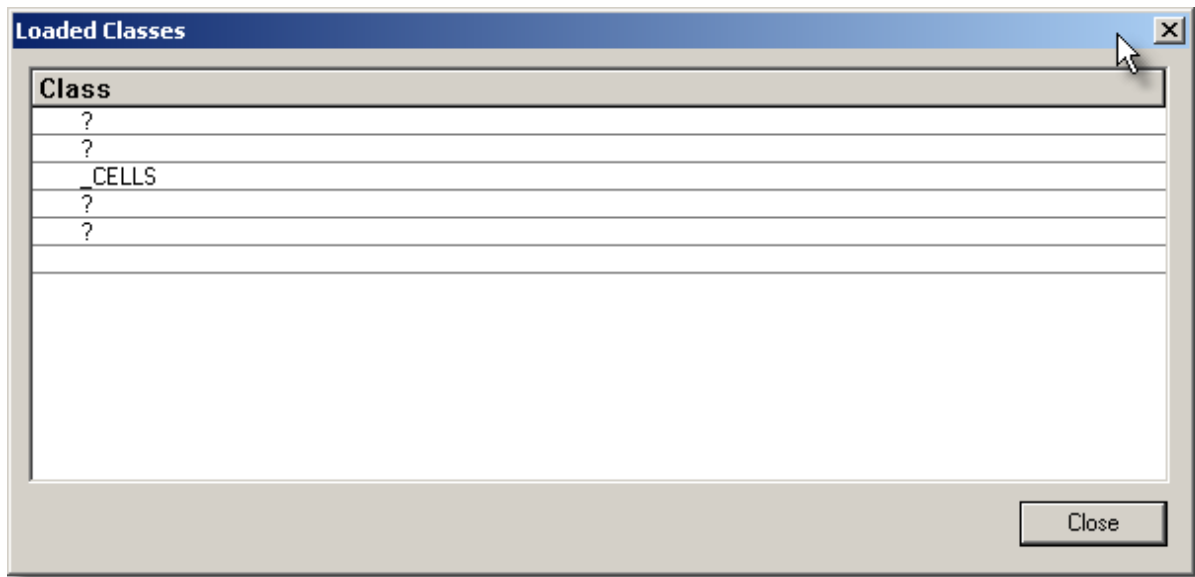
This section of the help file describes the tools Lua Bug Validator provides.

3.11.1 Loaded Classes

To display the Loaded Classes dialog, choose **Loaded Classes...** on the **Tools** menu.



The Loaded Modules dialog is displayed.



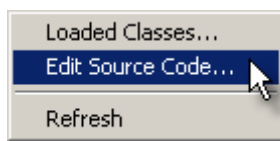
The dialog displays the name of each loaded class.

3.11.2 Colour coded source code editor

Lua Bug Validator provides a choice of editing solutions. When a source file is to be edited, the source file can be edited using Microsoft® Visual Studio® or the source file can be edited using Lua Bug Validator's syntax colouring source code editor. To configure which editor use as the default click [here](#).

The source code editor can be started in one of three ways:

- Double clicking on a source code fragment displayed in a view.
- Choosing **Edit Source Code...** from a popup menu display in a view.
- Choosing **Edit Source Code...** on the **Tools** menu.



The picture below shows the source code editor when displaying a file from the summary or file and line tabs. The lines in yellow have been visited. These lines have a green tick next to them indicating that they have been successfully hooked. Lines that could not be hooked have a red cross displayed

next to them. Lines that have not been visited are displayed in a different colour. You can see that the line where the for/next loop starts has an arrow displayed with the green tick. The arrow indicates the source code line of interest when the source code editor was displayed.

As can be seen, Lua Bug Validator has successfully hooked every line in this function, including the lines that comprise the for/next loop.

```

d:\lua\lua-5.0.2\test\sort.lua - Edit Source Code
File Edit Formatting
i=i+1
end
end

function show(m,x)
io.write(m,"\n\t")
local i=1
while x[i] do
io.write(x[i])
i=i+1
if x[i] then io.write(",") end
end
io.write("\n")
end

function testsorts(x)
local n=1
→ while x[n] do n=n+1 end; n=n-1 -- cour
show("original",x)
qsort(x,1,n,function (x,y) return x<y end;
show("after quicksort",x)
selectionsort(x,n,function (x,y) return x<
show("after reverse selection sort",x)
qsort(x,1,n,function (x,y) return x<y end;
show("after quicksort again",x)
end

-- array to be sorted
x={"Jan","Feb","Mar","Apr","May","Jun","Ju
testsorts(x)

```

The editor supports the following operations:

File

Load. Load a file into the editor.

Save. Save a file.

Save As... Save the file with a different name than its original name.

Exit. Finish editing the file.

Edit

Undo. Unlimited undo of editing changes.

Redo. Unlimited redo of editing changes.

Cut. Cut selected text from the document and place on the clipboard.

Copy. Copy selected text from the document and place on the clipboard.

Paste. Paste text from the clipboard into the document.

Select All. Select all text in the document.

Delete. Delete the selected text.

Find... Find a specific string in the document.

Replace... Find and replace a specific string in the document.

Add Bookmark... Add a bookmark to remember the current text position. This is not the same type of bookmark as a Lua Bug Validator bookmark.

Goto Bookmark... Goto to a previously defined bookmark. This is not the same type of bookmark as a Lua Bug Validator bookmark.

Goto Line... Goto a specific line in the document.

Formatting

Convert tabs to spaces. Convert all tabs in the document into spaces.

Convert spaces to tabs. Convert all spaces in the document into tabs, where there are enough spaces.

Use Colour. Use the colour coding scheme defined by the colour syntax.

Colour Syntax None, C, C++. Choose the colour encoding scheme.

Edit Colour Syntax... Edit the colour encoding scheme.

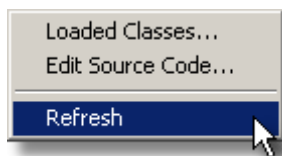
Options... Edit options.

Settings... Edit settings.

Wrap Width... Width of the text for wrapping.

3.11.3 Refresh

To refresh the data displayed on the current tabbed window, choose **Refresh** on the **Tools** menu,



or click the **Refresh** icon on the session toolbar.



3.12 Loading, Saving, Exporting, Closing

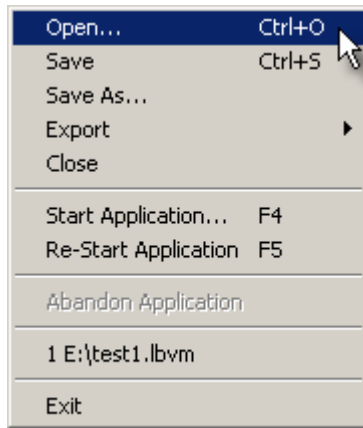
Lua Bug Validator allows sessions to be saved so that you can send the session to a colleague, examine the session at a later date, compare the session with another session, or create baseline sessions for use in regression tests. To complement the save capability sessions can be loaded.

Sessions can be exported in HTML and XML formats.

When you have finished working with a session, a session can be closed. The session can be reopened later if so desired.

3.12.1 Loading

To load a session choose **Open...** on the **File** menu,



or click on the open icon on the standard toolbar.

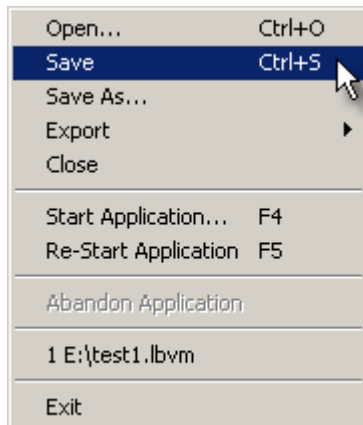


A file selection dialog will be displayed. Select the file you wish to load and click OK.

3.12.2 Saving

Save

To save a session choose **Save** on the **File** menu,



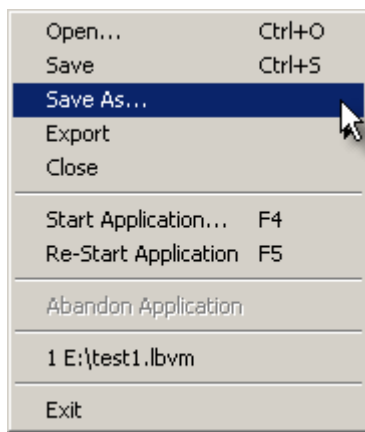
or click on the save icon on the standard toolbar.



The session will be saved using the current file name. If the session has not been saved the same process as for Save As will be used to specify the data to save and the name to save the file.

Save As

To save a session using a different name to the current sessions choose **Save As...** on the **File** menu,

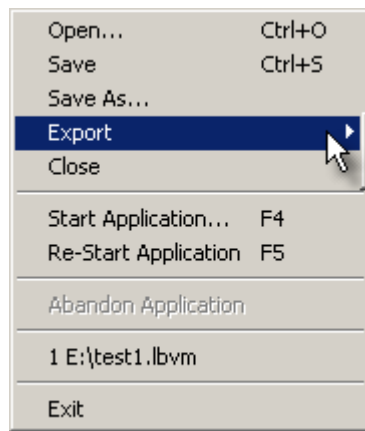


The standard Microsoft file save dialog will be displayed. Choose a name for the file and click Save. The session will be saved using the file name you specified.

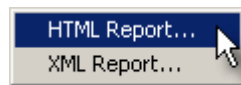
3.12.3 Exporting

Lua Bug Validator sessions can be exported in HTML and XML formats. There is no facility to import sessions. If you wish to save a session for later use with Lua Bug Validator, Lua Bug Validator's load and save options provide a more compact representation of the data.

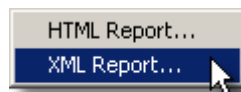
To export a session choose the **Export** submenu on the **File** menu.



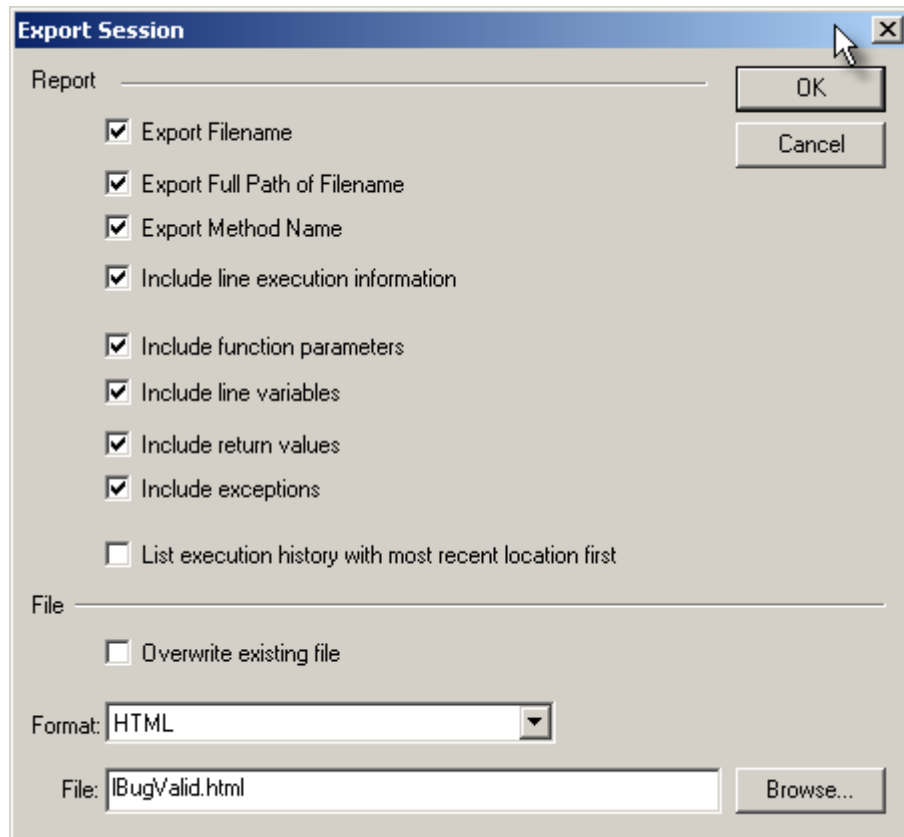
On the **Export** submenu, choose **HTML Report...**,



or **XML Report...** as appropriate.



The session export dialog is be displayed.



Scope

Select which data you want to export.

- Filename information.
- Method information.
- Order of data exported.

If HTML export is chosen, the export is in the form of a table. If you require a specific HTML format for session exports, we recommend exporting an XML report and using that to generate the HTML report to your requirements.

File

To specify the filename to save the session, type the filename in the **File** field, or click the **Browse...** button to use a file specification dialog.

If you would like to be warned when you are about to overwrite an existing file, deselect the **Overwrite existing file** check box. If you would not like to be warned when you are about to overwrite an existing file, select the **Overwrite existing file** check box.

Format

The file format for a session export can be HTML or XML. The appropriate format is chosen to correspond to the menu option used to display the export session dialog. The file format can be changed using the **Format** combo box.

3.12.3.1 XML Export Tags

This section describes which XML tags are used to export the session data from a Lua Bug Validator session.

The start of an exported XML file lists a few details about Lua Bug Validator:

<VALIDATORVERSION>Version of Lua Bug Validator**</VALIDATORVERSION>**

<VALIDATORDATE>Date Lua Bug Validator was built**</VALIDATORDATE>**

<VALIDATORTIME>Time Lua Bug Validator was built**</VALIDATORTIME>**

<TITLE>Name of executable**</TITLE>**

Execution history is listed in the following tag pairs.

<EXECUTIONDATA></EXECUTIONDATA>

The tags found inside the above tag pairs are shown below. Note that all hexadecimal numbers are written with leading zeros and a leading 0x.

<THREAD>Thread id**</THREAD>**

<FILENAME>Filename of source file**</FILENAME>**

<LINE>Decimal line number**</LINE>**

<METHOD_ID>Hexadecimal method id**</METHOD_ID>**

<METHODNAME>Method name**</METHODNAME>**

Function parameters are listed in the following tag pairs.

<FUNCTION_PARAMETERS>

<NUMBER>number of parameters**</NUMBER>**

<PARAM>

<NAME>name**</NAME>**

<TYPE>type**</TYPE>**

<VALUE>value**</VALUE>**

</PARAM>

</FUNCTION_PARAMETERS>

<EXCEPTION>

<NAME>name**</NAME>**

<TYPE>type**</TYPE>**

<VALUE>value**</VALUE>**

</EXCEPTION>

<LINE_VARIABLES>

<NUMBER>number of variables**</NUMBER>**

<VARIABLE>

<NAME>name**</NAME>**

<TYPE>type**</TYPE>**

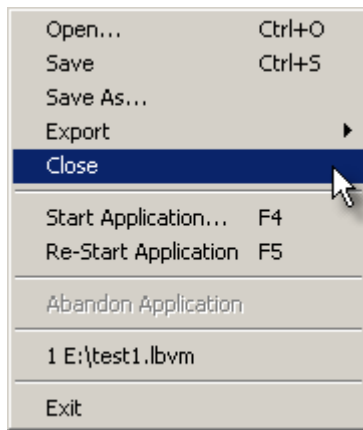
<VALUE>value**</VALUE>**

```
</VARIABLE>
</LINE_VARIABLES>

<RETURN_VALUE>
  <NAME>name</NAME>
  <TYPE>type</TYPE>
  <VALUE>value</VALUE>
</RETURN_VALUE>
```

3.12.4 Close Session

To close a session choose **Close** on the **File** menu.



The session will be closed and all windows will be reset to their original state. If you wish to reopen or delete the session, you can do so using the session manager.

3.13 Starting your target program

Lua Bug Validator provides a choice of 3 methods to start a target program and have Lua Bug Validator collect data from the target program about the program's execution.

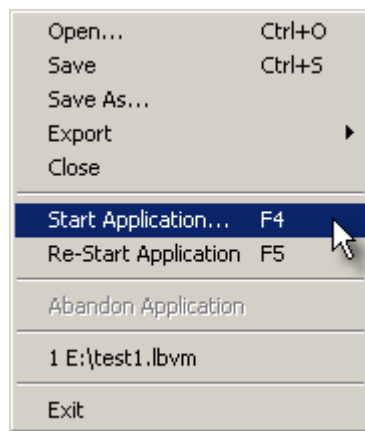
- You can start a program in a specified directory and with as many command line arguments as you want.
- You can inject Lua Bug Validator into an already running program.
- You can have Lua Bug Validator wait until a specific program starts to run before attaching to that program. This is useful for when you want Lua Bug Validator to attach to a program that is started as an OLE server, for example.

3.13.1 Novice and Intermediate

This section describes the novice mode and intermediate mode user interfaces for launching applications, attaching to running processes and for waiting for applications to start.

3.13.1.1 Launching the program

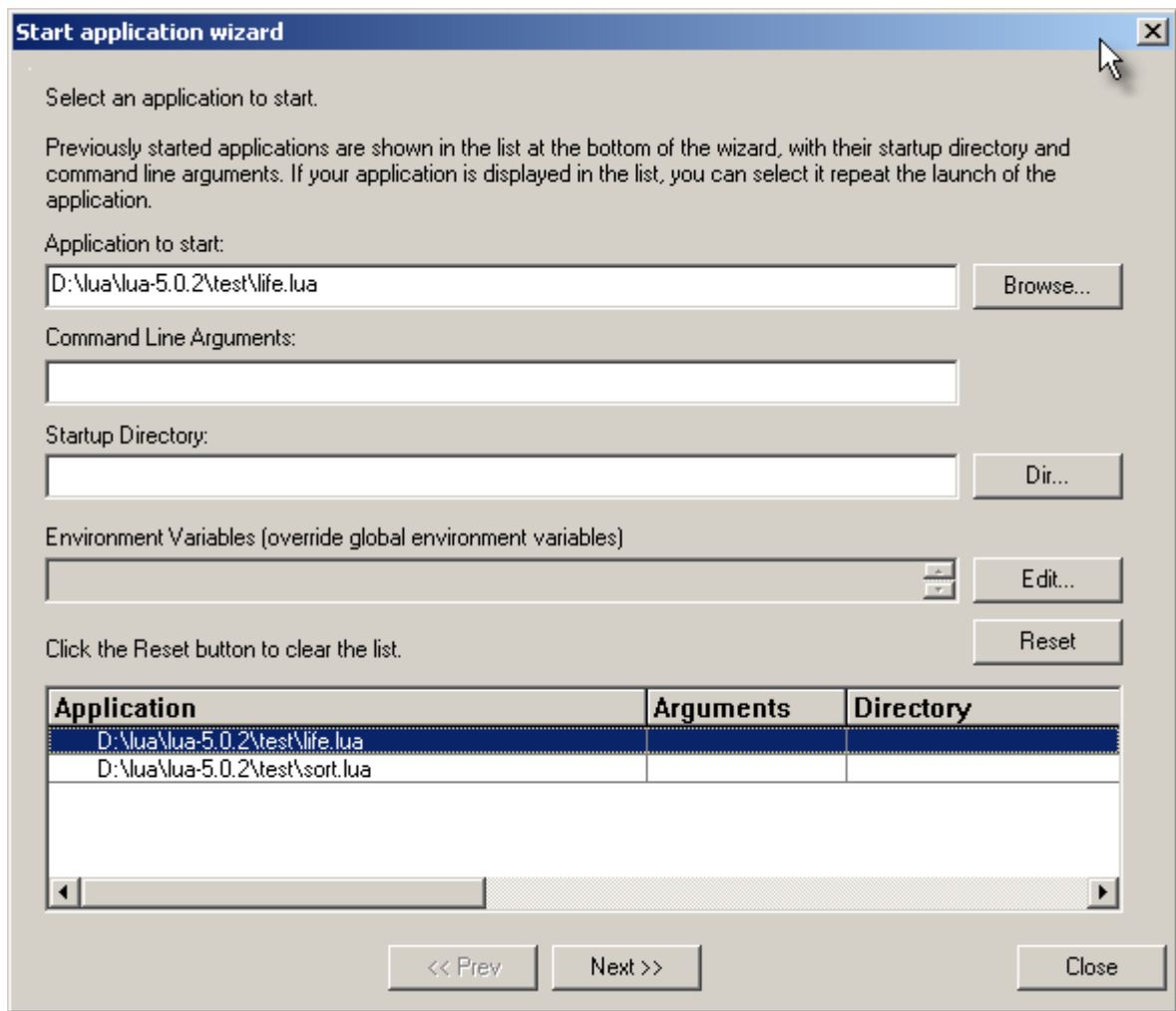
To start your program choose **Start Application...** on the **File** menu,



or click on the launch icon on the session toolbar.



The launch program wizard will be displayed.

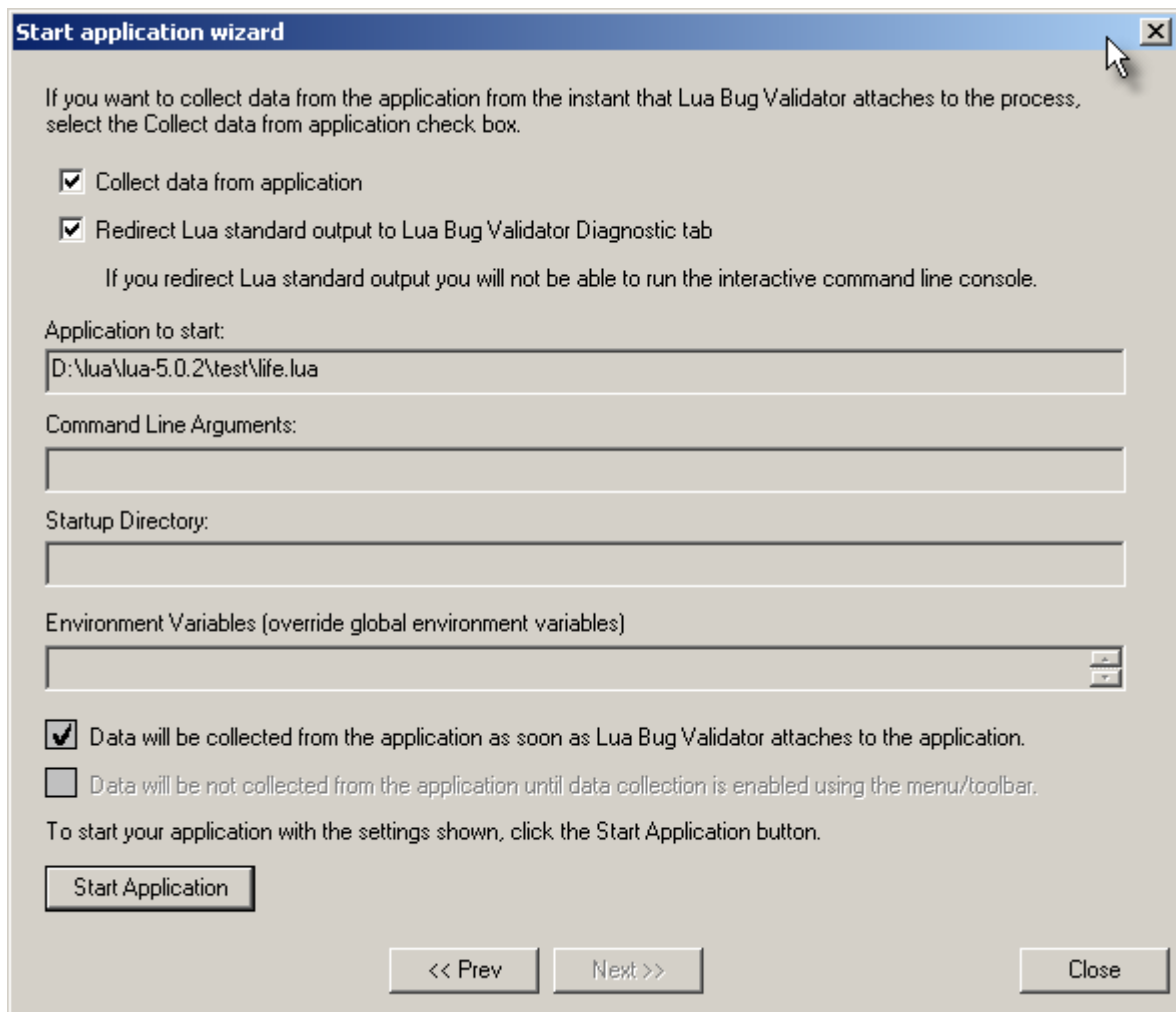


The picture shows some already launched programs shown in the scrollable list.

Detailed program start (1)

- 1) Type in the program name in the **Application** field, or click the **Browse...** button to use a file browser to choose the program to launch.
- 2) Type in the program arguments in the **Arguments** field.
- 3) Type in the program directory in the **Startup Directory** field, or click the **Dir...** button to use a directory browser to choose the directory.
- 4) Click the **Edit...** button to display the Environment Variables dialog to allow you to define any environment variables for the application.
- 5) Click on the **Next** button to move to the next page of the wizard.

If your application has Lua embedded in it, the Application to Start field will be disabled.



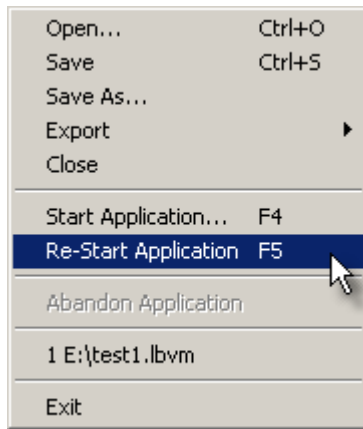
If you want Lua Bug Validator to collect data from the moment the application is launched, select the **Collect Data from application** check box. If you deselect the **Collect Data from application** check box, data will not be collected.

Starting your program

The last page of the wizard displays a summary of the options you have chosen. If you are satisfied with the options, click the **Start Application** button to start your application and attach Lua Bug Validator to the application.

3.13.1.2 Re-Launching the program

To re-start your program choose **Re-Start Application...** on the **File** menu,



or click on the launch icon on the session toolbar.



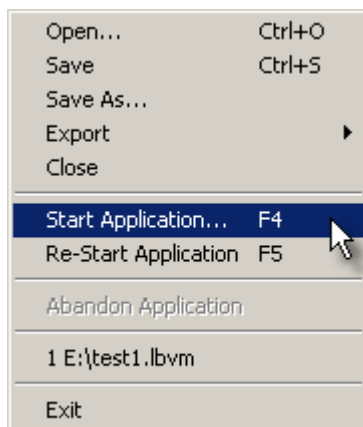
The application that was most recently launched is started.

3.13.2 Expert

This section describes the expert mode user interfaces for launching applications, attaching to running processes and for waiting for applications to start.

3.13.2.1 Launching the program

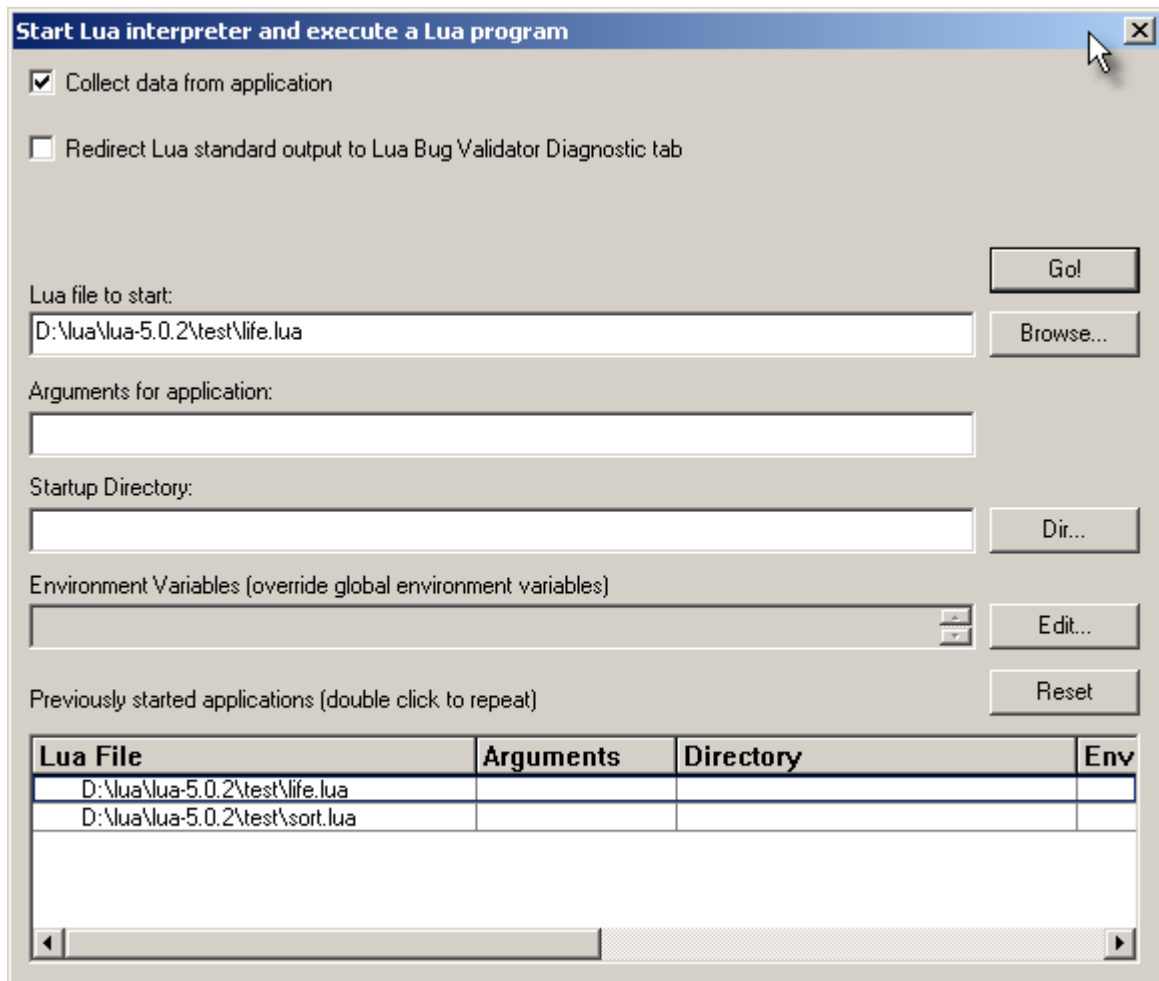
To start your program choose **Start Application...** on the **File** menu,



or click on the launch icon on the session toolbar.



The launch program dialog will be displayed.



The picture shows some already launched programs shown in the scrollable list. You can relaunch these programs by double clicking their entry in the list. Depending on what you want to do with your program you have a choice of launching the program.

Detailed program start (1)

- 1) Type in the program name in the **Application** field, or click the **Browse...** button to use a file browser to choose the program to launch.
- 2) Type in the program arguments in the **Arguments** field.
- 3) Type in the program directory in the **Startup Directory** field, or click the **Dir...** button to use a directory browser to choose the directory.
- 4) Click the **Edit...** button to display the Environment Variables dialog to allow you to define any environment variables for the application.
- 5) Click on the **Go!** button to launch the program.

Detailed program start (2)

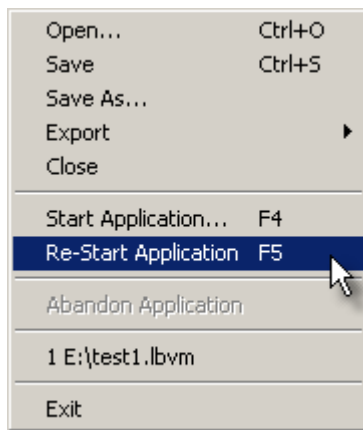
- 1) Click on the program shown in the scrolled list that you want to start. The details for the program will appear in the **Application**, **Arguments** and **Startup Directory** fields.
- 2) Modify any of the **Application**, **Arguments** and **Startup Directory** fields as appropriate.
- 3) Click on the **Go!** button to launch the program.

If you want Lua Bug Validator to collect data from the moment the application is launched, select the **Collect Data from application** check box. If you deselect the **Collect Data from application** check box, data will not be collected.

If your application has Lua embedded in it, the Application to Start field will be disabled.

3.13.2.2 Re-Launching the program

To re-start your program choose **Re-Start Application...** on the **File** menu,



or click on the launch icon on the session toolbar.

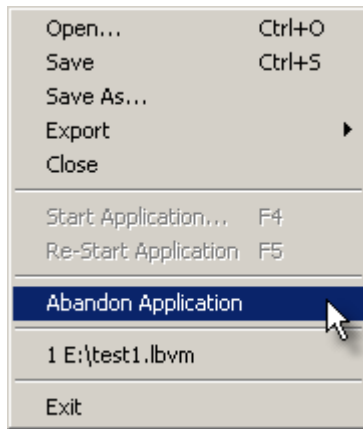


The application that was most recently launched is started.

3.14 Stopping your target program

Sometimes you will start a run of your target program and realise that you made a mistake, forgot to do something or need a different option enabled and need to restart the test from the beginning. You can stop the target program using Task Manager, using the debugger, or using Lua Bug Validator.

Lua Bug Validator provides you with a simple way of stopping the target program. Choose **Stop Application** on the **File** Menu,



or click on the large red X on the session toolbar.

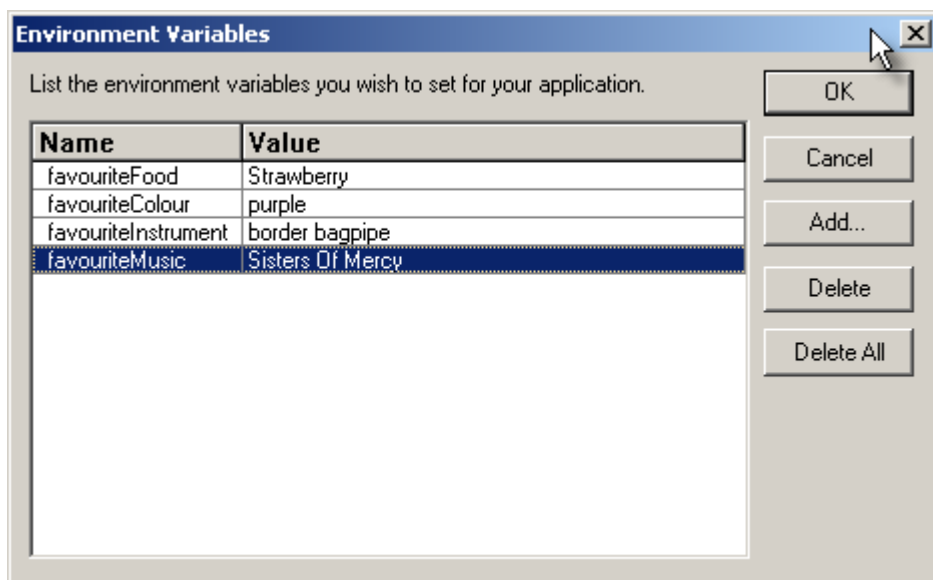


The target program is instructed to end using **ExitProcess()** from inside the stub. When Lua Bug Validator has ended the target program, Lua Bug Validator also abandons the current session. The session can still be accessed from Session Manager.

Using Lua Bug Validator to stop the target program is usually quicker than using Task Manager or the debugger because the session is thrown away for you, rather than you having to terminate the current session.

3.15 Environment Variables

When launching an application Lua Bug Validator allow you to specify the environment values to pass to the application. This is done via the Environment Variables dialog.



The image shows the Environment Variables dialog with four variables defined. This would be equivalent to the following text in a batch file:

```
set favouriteFood=Strawberry
set favouriteColour=Purple
set favouriteMusic=Sisters Of Mercy
set favouriteInstrument=Border Bagpipe
```

Add...

To add an environment variable, click **Add...** then enter the name and value in the appropriate fields. Double click on a field in the grid to edit a field.

Delete

To delete an environment variable, select it in the list and click **Delete**.

Delete All

To delete all environment variable definitions, click **Delete All**.

OK

To dismiss the dialog and accept the changes click **OK**.

Cancel

To dismiss the dialog without accepting any changes click **Cancel**.

3.16 Closing Lua Bug Validator

To finish working with Lua Bug Validator, choose **Exit** on the **File** menu.



3.17 Help

Lua Bug Validator provides a help system to help you get the most from Lua Bug Validator.

3.17.1 Tip of the day

The tip of the day dialog provides helpful tips on how to get the most from Lua Bug Validator.

3.17.2 About Lua Bug Validator

Information about this version of Lua Bug Validator, copyright notices and user information.

3.17.3 Help Topics

Display the online help for Lua Bug Validator.

PDF Help

PDF versions of the help file are available from <http://www.softwareverify.com/helpPDFs.html>.

3.17.4 Tutorial

Display the **tutorial.html** file in the default web browser.

The tutorial contains information that will help you learn how to use Lua Bug Validator. As the tutorial is improved based on user feedback we will make updates available on the website so that you do not need to download a Lua Bug Validator update just to get access to the latest tutorial.

Tutorials are available from <http://www.softwareverify.com/tutorials.html>.

After downloading the tutorial zip file unzip into the installation directory for Lua Bug Validator.

3.17.5 Readme

Display the **readme.html** file in the default web browser.

The readme file contains all the latest information about Lua Bug Validator including the change history for bug fixes and feature improvements.

Part



4 Frequently Asked Questions

This section lists the commonly asked questions about Lua Bug Validator.

4.1 I have an idea for a feature, can it be added to Lua Bug Validator?

We have tried to add as many features to Lua Bug Validator that we thought would be useful to the potential users of Lua Bug Validator. In fact every feature in Lua Bug Validator has been used to solve problems and bugs on consulting work carried out for our customers and on internal projects at Software Verification Limited. We know the features we have put in the product are useful.

However it is possible we have overlooked a feature that you may find very useful. We will be pleased to consider all ideas for new features to Lua Bug Validator. Please contact us at the address provided on the contact page.

4.2 What file extensions does Lua Validator use for itself?

Lua Bug Validator stores most of the configuration data it needs in the registry. However some data, such as the data hooks, coverage data and filter data is stored in files. This section describes the file extensions used by Lua Bug Validator so that you can recognise such files. The file data structure is not described as it may change from version to version of the product.

Settings, Filters, Flow Trace

lbvs Settings
lbvm Recorded Session containing flow trace.

Program Launch, Extensions

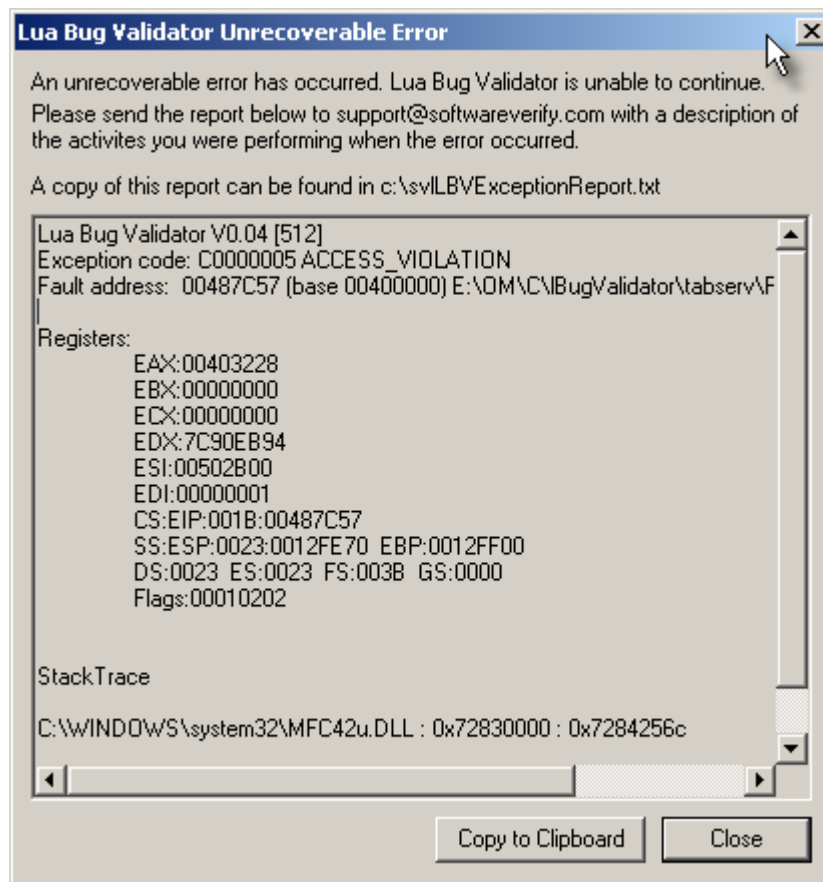
lua Program files

4.3 Lua Bug Validator Unrecoverable Error

The Lua Bug Validator Unrecoverable Error dialog is displayed when an internal error has occurred that Lua Bug Validator did not expect. Lua Bug Validator cannot continue to execute. A stack trace and register dump is shown so that the data can be sent with a description of the activities that caused the error to support@softwareverify.com. The data shown in the dialog is also written to **c:\sv\LBVExceptionReport.txt**.

The picture below shows the exception report for an access violation exception. The error shown below is artificial and was deliberately caused to allow this picture to be taken.

If you see this dialog, please copy the data and send it with a description of what you were doing with Lua Bug Validator to support@softwareverify.com so that we can fix the bug that caused this error.

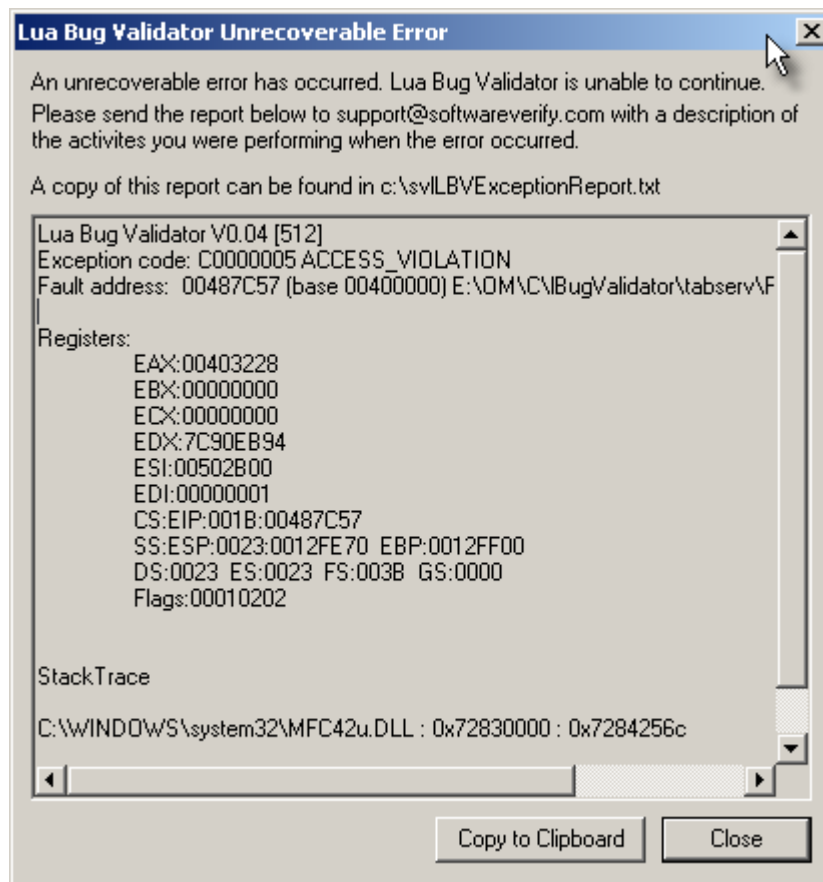


4.4 What is in svILBVExceptionReport?

The file `c:\svILBVExceptionReport.txt` contains information that identifies where Lua Bug Validator was executing when Lua Bug Validator crashed. The `c:\svILBVExceptionReport.txt` contains the same information that is displayed in the exception report dialog when a crash occurred. The file `c:\svILBVExceptionReport.txt` contains the data for the most recent exception.

The picture below shows the exception report for an access violation exception. The error shown below is artificial and was deliberately caused to allow this picture to be taken.

If you see this dialog, please copy the data and send it with a description of what you were doing with Lua Bug Validator to support@softwareverify.com so that we can fix the bug that caused this error.



4.5 Installing DbgHelp.dll

Windows File Protection and DbgHelp.dll

Microsoft constantly upgrade DbgHelp.dll to provide an improved, richer debugging API. Software Verification use DbgHelp.dll to provide debugging information about the application under test. Some of the APIs we use are only available in more recent versions of DbgHelp.dll.

On Windows 2000 and Windows XP, Windows File Protection comes into play. How straightforward the installation, depends on your machine and how it is setup. If your machine does not have the backup operating system files on it, ready for a reinstall, then you will not have problems. If your machine does have the backup operating system files on it, you will encounter problems.

Windows NT

DbgHelp.dll is not installed by default on Windows NT. Windows NT does not have Windows File Protection.

Consequently, installing DbgHelp.dll on Windows NT is straightforward.

Windows 2000

On Windows 2000 machines, the current version of DbgHelp.dll needs to be upgraded as it does not have all the APIs required by our debugging tools. As such the comments on this page may apply to you if you are installing our software tools on a Windows 2000 machine.

Windows XP

On Windows XP machines, the current version of DbgHelp.dll does not need to be upgraded. At present installations of our software on Windows XP machines is straightforward - DbgHelp.dll does not need to be upgraded.

Installation

We have modified our license program to check for the correct version of DbgHelp.dll and to deliberately overwrite the `c:\windows\system32\dllCache\dbgHelp.dll` copy of dbghelp.dll so that Windows File Protection cannot restore `c:\windows\system32\dbgHelp.dll`.

When Windows notices that both the files have changed, it asks you for the installation media (CD/DVD) so that it can restore the files. You can choose to cancel the restoration so that the more recent version of DbgHelp.dll is used. Unfortunately when your machine has the backup installation files already on the hard disk, no installation media is needed and Windows silently restores the files without asking you if this is what is desired.

We have modified our products so that for systems where the above situation happens, the correct version of DbgHelp.dll will always be used, as long as your application is not stored in the Windows System directory. Even if a manual inspection of c:\windows\system32\dbgHelp.dll shows that c:\windows\system32\dbgHelp.dll has not been upgraded, our tools will still use the correct dbghelp.dll version installed with the product.

Microsoft DbgHelp.dll downloads

Updating DbgHelp.dll is an obvious thing to do. However, even Microsoft's own Debugging tools download fails to upgrade DbgHelp.dll.

Microsoft's most recent Debugging tools download is available from <http://www.microsoft.com/ddk/debugging/installx86.asp>.

This is a rare occasion where the technology designed to prevent "DLL Hell", actually causes it by preventing you from upgrading DbgHelp.dll in the system directory (where it is most likely to be loaded from for any given application). DbgHelp.dll is completely backwards compatible, so we are confused as to why Windows File Protection prevents this file from being upgraded.

4.6 How do I create a Power User on Windows XP?

Windows XP provides Power User accounts but does not make it easy to create a user with Power User privileges. To create a user with Power User privileges do the following:

- Create a **Limited User** account (we will call it "Test Limited User").
- Open **Control Panel** and set to Classic View.
- Open **Administrative Tools**.
- Open **Computer Management**.
- In the left hand pane expand **Local Users and Groups**.

- In the left hand pane select **Users**.
- In the right hand pane select the user account you created above ("Test Limited User").
- Right click on "Test Limited User" and choose **Properties** from the context menu.
- Select the **Member Of** tab.
- Click **Add**. A dialog box will be displayed. In the bottom edit box type **Power Users**. Click OK.
- Select the **Users** entry. Click **Remove**. Click OK.
- Close the **Computer Management** window.

Your Test Limited User is now a member of the Power Users group. Now is a good time to rename the account to something more appropriate.

Index

- D -

Data Collection 57
Detailed program start 57
Detailed program start (2) 57
Diagnostic 20

- E -

Exit 61

- F -

Frequently Asked Questions 64

- Q -

Quick program start 57

- R -

Refresh 47
Repeat program start 57
Reset 25

- S -

start a target program 53
stop 59