



Thread Lock Checker

by

Software Verify

Copyright © 2009-2025 Software Verify Limited

Thread Lock Checker

Thread Error Detection for Windows Operating Systems

by Software Verify Limited

Welcome to the Thread Lock Checker software tool. Thread Lock Checker is software tool that scans source code looking for specific uses of critical section locking objects. Once you have found these specific uses you can decide if they are correct for the context in which they are found.

We hope you will find this document useful.

Thread Lock Checker Help

Copyright © 2009-2025 Software Verify 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: February 2025 in United Kingdom.

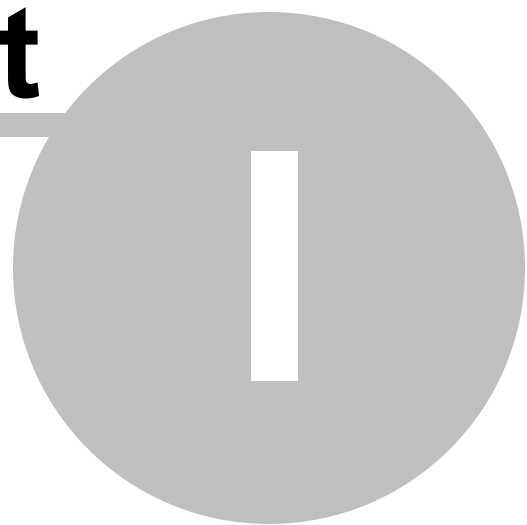
Table of Contents

Foreword	1
Part I How to get Thread Lock Checker	2
Part II What does Thread Lock Checker do?	4
Part III Default Lock Types	8
Part IV Menu	10
1 File	11
2 Edit	11
3 Software Updates	12
4 Help	15
Part V The user interface	17
Part VI Modifying the settings	20
1 Detect	21
2 Lock Type	23
3 File Filters	24
4 Line Pragma	25
Part VII How to use Thread Lock Checker	27
Part VIII Command Line Interface	30
1 Alphabetic Reference	31
2 Usage Reference	34
3 Command Line Examples	37
Index	0

Foreword

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

Part



1 How to get Thread Lock Checker

Thread Lock Checker is free for commercial use. Thread Lock Checker can be downloaded for Software Verify's website at <https://www.softwareverify.com/product/thread-lock-checker/>.

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

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

Whilst Thread Lock Checker is free for commercial use, Thread Lock Checker is copyrighted software and is not in the public domain.

You are free to use the software at your own risk.

You are not allowed to distribute the software in any form, or to sell the software, or to host the software on a website.

Contact

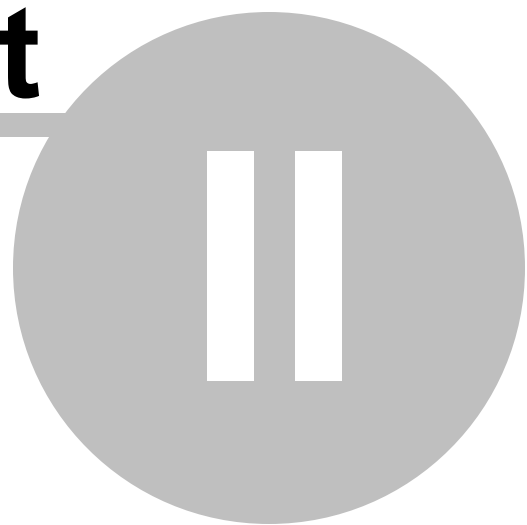
Contact Software Verify at:

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

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

Visit our blog to read our articles on debugging techniques and tools.
Follow us on twitter to keep track of the latest software tools and updates.

Part



2 What does Thread Lock Checker do?

Thread Lock Checker scans your source code looking for common errors when locking critical sections. Before describing what Thread Lock Checker detects we will present a brief outline of how critical sections and locks can be used.

Note: For simplicity in this help file, we always use CSingleLock for the example lock type, even though Thread Lock Checker can detect many more lock types than that.

Critical Sections and locks

Critical sections are used to provide synchronised access to data so that only one part of a program can modify the data at a time and no one can read the data while it is being modified. To make the management of locking and unlocking critical sections easy Microsoft introduced the concept of the single lock and multi lock, represented by the CSingleLock and CMultiLock MFC classes. Other third party libraries also have similar constructs (we'll cover those later).

A really important and useful thing about using CSingleLock and CMultiLock to control locks on the stack is that if an exception is thrown the locks are automatically cleaned up by the exception handling chain and all locks are automatically released, preventing the likelihood of an exception causing a deadlock due to a lock being left locked during an exception handling sequence.

A feature of the CSingleLock and CMultiLock classes is that their constructor constructs the lock and by default initialises it to be unlocked. Class methods Lock() and Unlock() are provided to lock the object. If the object goes out of scope (or is deleted if it is on the heap) when it is locked, the object automatically unlocks itself.

Thus you could have several different ways of expressing the same concept:

1. Start with an unlocked lock, do some work, lock the lock, access protected code, unlock the lock, do some work

```
void doWork1()
{
    CSingleLock    lock(&critSect);           // start unlocked

    ...

    lock.Lock();

    ... // access protected resource

    lock.Unlock();

    ...
}
```

2. Start with an unlocked lock on the heap, do some work, lock the lock, access protected code, unlock the lock, delete the lock, do some work

```
void doWork2()
{
```

```
    CSingleLock    *lock;

    lock = new CSingleLock(&critSect);          // start unlocked

    ...

    lock.Lock();

    ... // access protected resource

    lock->Unlock();
    delete lock;

    ...
}
```

3. Start with an unlocked lock on the heap, do some work, lock the lock, access protected code, delete the lock (implicitly unlocking the lock), do some work

```
void doWork3()
{
    CSingleLock    *lock;

    lock = new CSingleLock(&critSect);          // start unlocked

    ...

    lock.Lock();

    ... // access protected resource

    delete lock; // implicitly unlock the lock during the delete

    ...
}
```

4. do some work, create a local scope, create a locked lock, access protected code, leave local scope implicitly unlocking and destroying the lock, do some work

```
void doWork4()
{
    ...

    {
        CSingleLock    lock(&critSect, TRUE);          // start locked

        ... // access protected resource

        // lock is unlocked at the end of the local scope
    }

    ...
}
```

For the most part using CSingleLock and CMultiLock with the locking parameter set to its default value (FALSE) is not very useful. This leaves you with a lock that by default is not locked and implies that you will be using the Lock() and Unlock() methods to manage if the lock is locked or unlocked.

The real value in the CSingleLock and CMultiLock classes is that you can pass TRUE to the constructor so that the lock is created in the locked state and just let the lock destruction unlock the lock implicitly. Using this method you can ensure that you never leave locks unlocked and never forget to lock a lock. This is demonstrated in doWork4() above.

Thread Lock Checker

Thread Lock Checker scans your source code looking for common errors when locking critical sections using classes like CSingleLock and CMultiLock.

There are four valid ways to initialise a CSingleLock object using its constructor:

```
CSingleLock    lock(&sect);                // default unlocked

CSingleLock    lock(&sect, FALSE);          // unlocked

CSingleLock    lock(&sect, TRUE);           // locked

CSingleLock    lock(&sect, lockStatus);     // lockStatus decides if locked or not
```

Of the methods shown above, only one is guaranteed to lock and protect the resource. The first two do not lock and protect the resource. The last method depends on the variable to lock and protect the resource. Thread Lock Checker detects all of these conditions. By default the locked state (3rd in the list above) is not reported for your attention.

There are also many invalid ways to initialise a CSingleLock object that will be accepted by the compiler. The following is four examples of forgetting to specify the variable name. This results in a lock that is created then immediately destroyed. Even if it gets locked it will be destroyed (and thus unlocked) before it can be used to protect the critical resource.

```
CSingleLock(&sect);                        // default unlocked

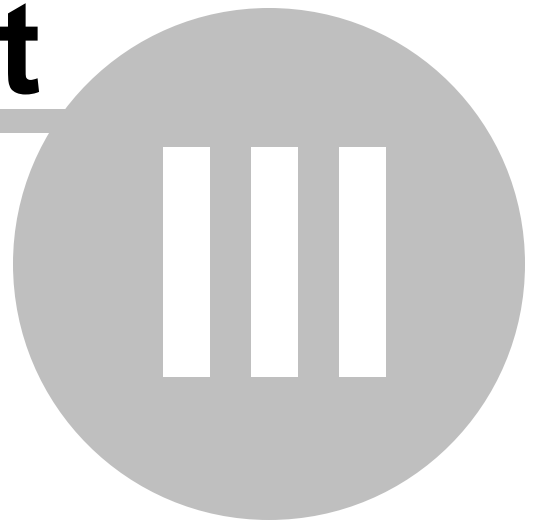
CSingleLock(&sect, FALSE);                 // unlocked

CSingleLock(&sect, TRUE);                  // locked

CSingleLock(&sect, lockStatus);            // lockStatus decides if locked or not
```

Thread Lock Checker will report all four of these invalid ways of creating a CSingleLock for your attention.

Part



3 Default Lock Types

The default settings will cause Thread Lock Checker to search for the following types:

Standard Library

`std::unique_lock<std::mutex>`
`std::unique_guard<std::mutex>`

MFC

`CSingleLock`
`CMultiLock`

Qt

`QMutexLocker`
`QReadLocker`
`QWriteLocker`

ACE

`ACE_Guard<ACE_Lock>`
`ACE_Guard<ACE_Adaptive_Lock>`
`ACE_Guard<ACE_Mutex>`

MSJ/MSDN

`COptexSingleLock`

Software Verify

`stubSingleLock`
`stubMultiLock`

What if I've got my own lock types I need to check for?

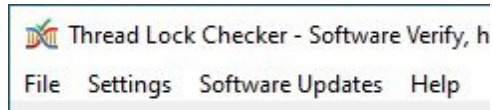
If you've got your own lock types you can specify these using the settings dialog or the command line.

Part

IV

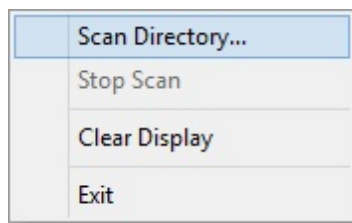
4 Menu

The main menu contains four menus, File, Edit, Software Updates and Help.

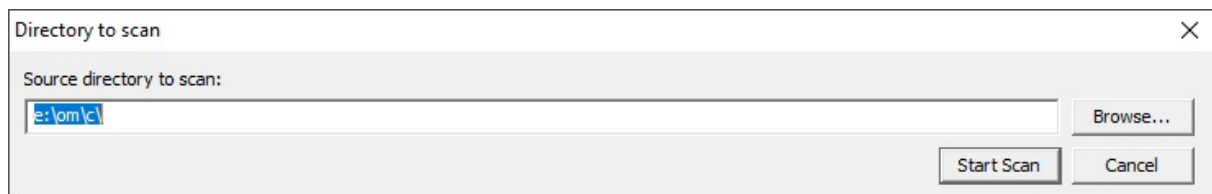


4.1 File

The File menu controls starting and stopping scans for thread lock errors, clearing the display and exiting the program.



File menu → **Scan Directory...** → displays a directory chooser then starts scanning that directory for thread locking errors.



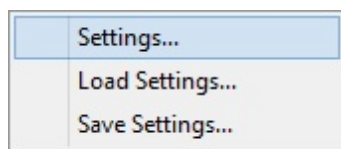
File menu → **Stop scan** → stops any scan that is in progress

File menu → **Clear Display** → clear all results from a previous scan and to remove any source code from the source code editing window.

File menu → **Exit** → closes Thread Lock Checker

4.2 Edit

The Edit menu controls editing settings, loading settings and saving settings.



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

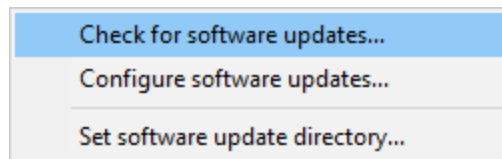
Edit menu ➞ **Load Settings...** ➞ loads previously saved settings.

Edit menu ➞ **Save Settings...** ➞ saves settings.

4.3 Software Updates


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

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

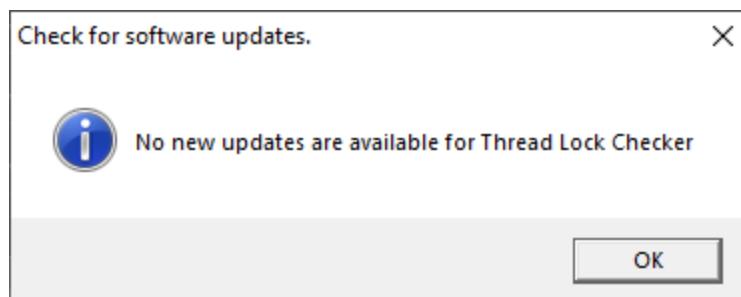


 **Software Updates** menu ➞ **Check for software updates** ➞ checks for updates and shows the software update dialog if any exist

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

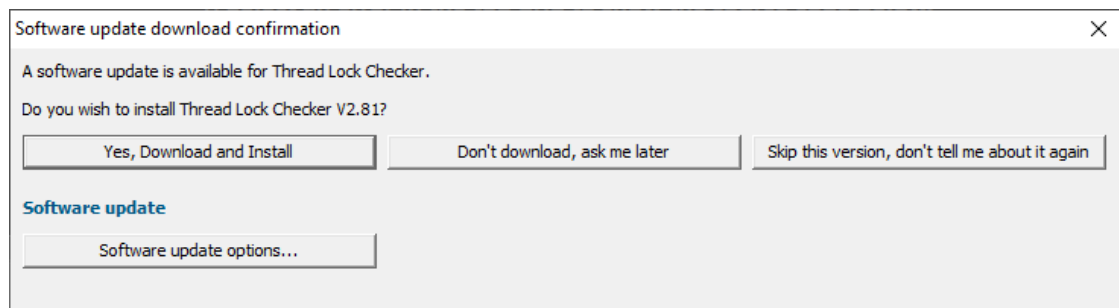
 Before updating the software, close the help manual, and end any active session by closing target programs.

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

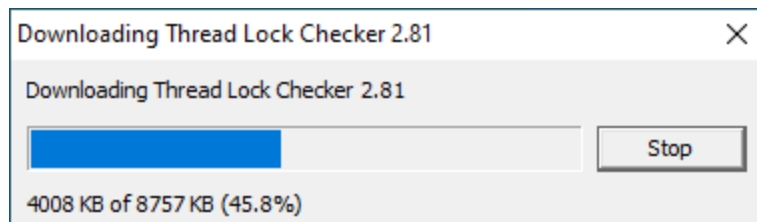


Software Update dialog

If a software update is available for Thread Lock Checker you'll see the software update dialog.



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



Once the update has downloaded, Thread Lock Checker will close, run the installer, and restart.

You can stop the download at any time, if necessary.

- **Don't download...** ➡ Doesn't download, but you'll be prompted for it again next time you start Thread Lock Checker
- **Skip this version...** ➡ Doesn't download the update and doesn't bother you again until there's an even newer update
- **Software update options...** ➡ edit the software update schedule

Problems downloading or installing?

If for whatever reason, automatic download and installation fails to complete:

- Download the latest installer manually from the software verify website.


Make some checks for possible scenarios where files may be locked by Thread Lock Checker as follows:

- Ensure Thread Lock Checker and its help manual is also closed
- Ensure any error dialogs from the previous installation are closed

You should now be ready to run the new version.

Software update schedule

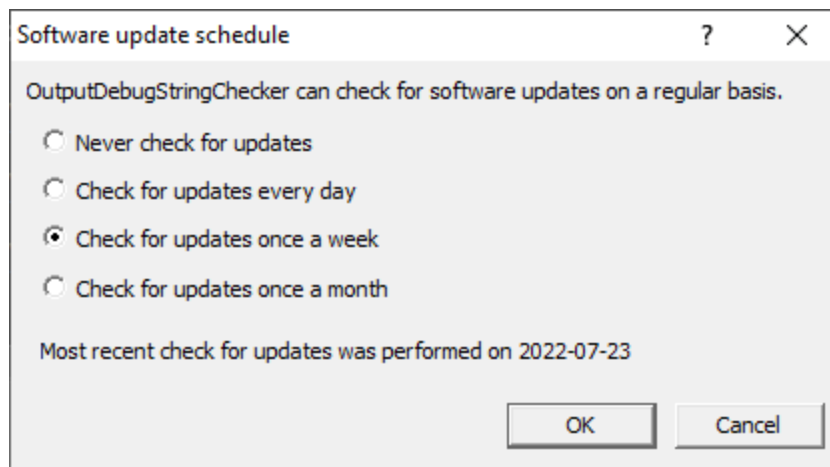
Thread Lock Checker can automatically check to see if a new version of Thread Lock Checker is available for downloading.

 **Software Updates** menu → **Configure software updates** → shows the software update schedule dialog

The update options are:

- never check for updates
- check daily (the default)
- check weekly
- check monthly

The most recent check for updates is shown at the bottom.



Software update directory

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

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


 **Software Updates** menu → **Set software update directory** → shows the Software update download directory dialog




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

Example reasons for invalid directories include:

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

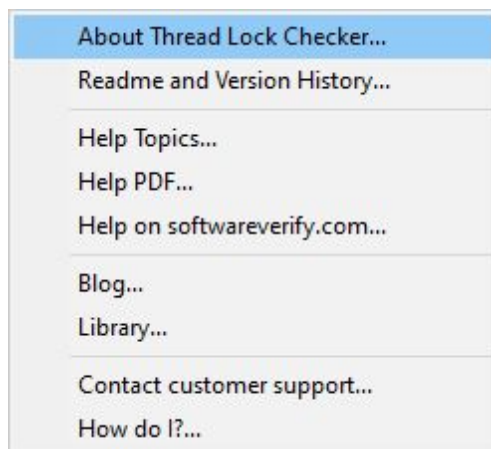
 When modifying the download directory, you should ensure the directory will continue to be valid. Updates may no longer occur if the download location is later invalidated.

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


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

4.4 Help

The Help menu controls displaying this help document and displaying information about Thread Lock Checker.



Help menu  **About Thread Lock Checker...**  displays information about Thread Lock Checker.



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

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

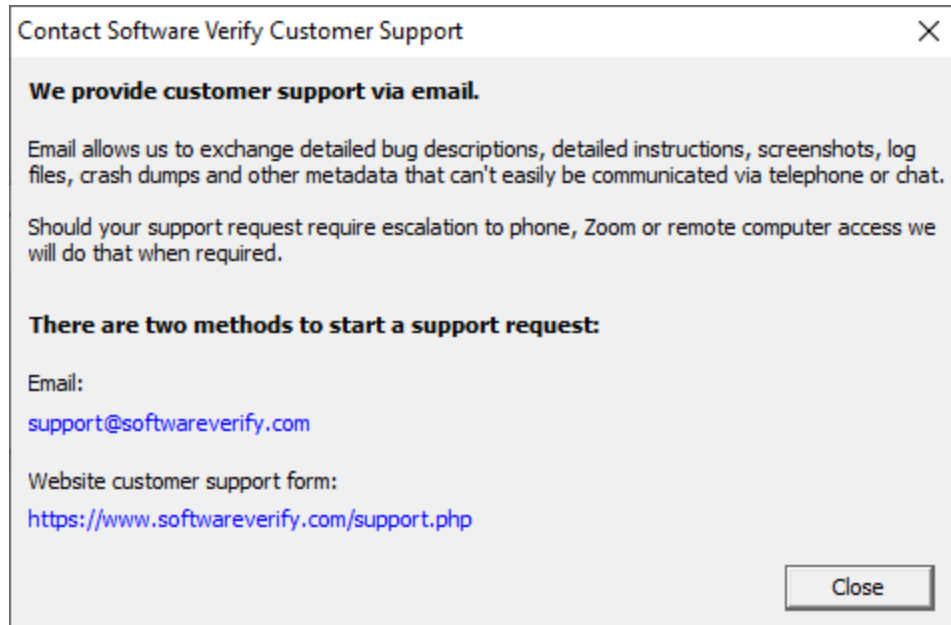
Help menu  **Help PDF...**  displays this help file in PDF format.

Help menu  **Help on softwareverify.com...**  display the Software Verify documentation web page where you can view online documentation or download compiled HTML Help and PDF help documents.

Help menu  **Blog...**  display the Software Verify blog.

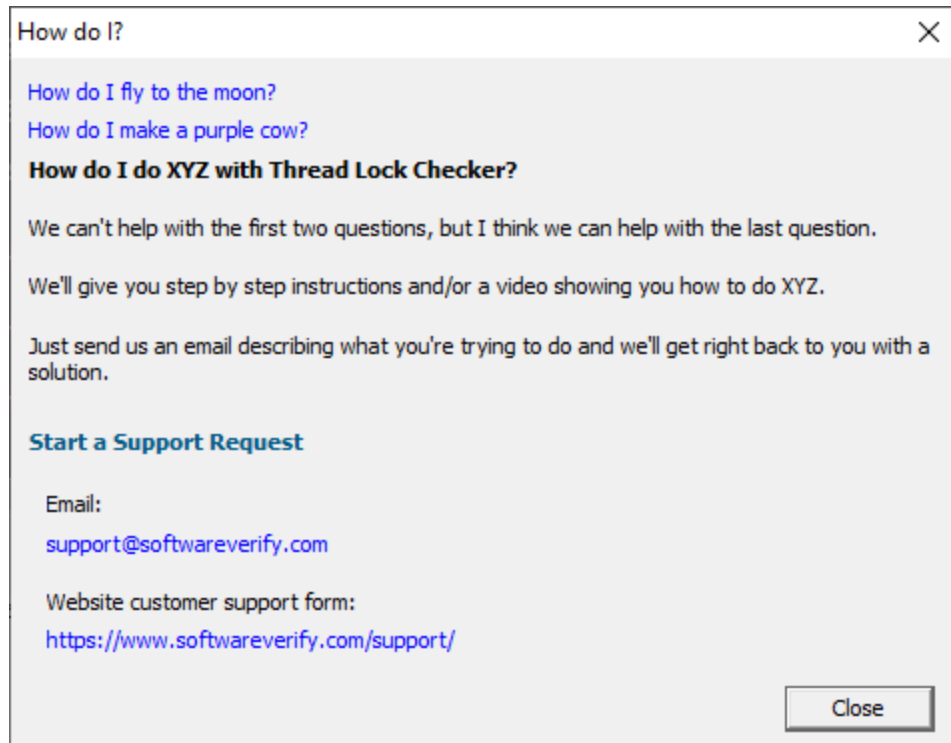
Help menu  **Library...**  display the Software Verify library - our best blog articles grouped by related topics.

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

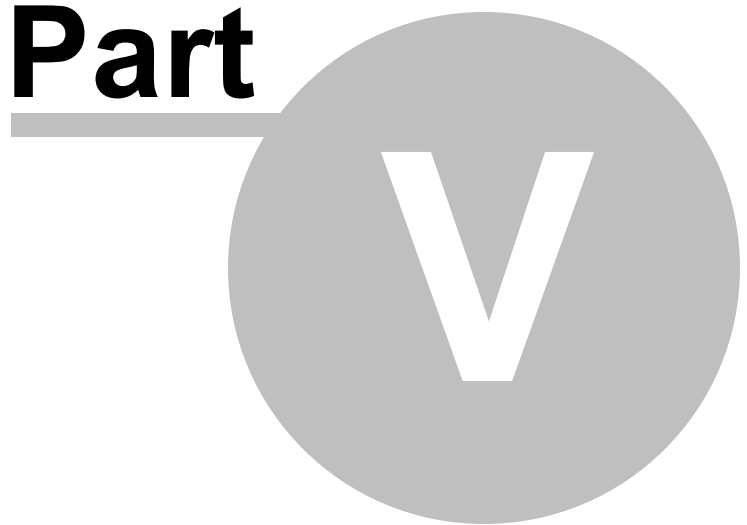


Click a link to contact customer support.

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



Part



5 The user interface

The Thread Lock Checker user interface is split into three sections:

- Menu
- Error report grid
- Source code viewer

Error Report Grid

The error report grid displays one error/warning per line.

Each line contains the type of lock that was checked, the reason for the error/warning, the line number and the filename of the error.

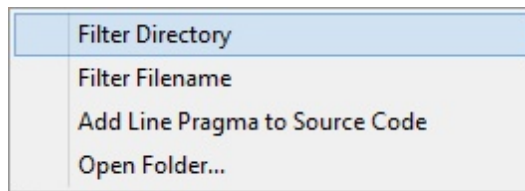
The status of each line is colour coded, with red text for lines that are most likely a coding mistake.

You can sort the data by clicking on any column header. Click again to reverse the sort direction.

Click on any line to display the source code in the lower window. Double click to edit the source code in Visual Studio.

Lock	Reason	Function	Line Number	Filename
CSingleLock	No lock variable	void exampleClassWithLockingErrors::setDataError1(const CString&str)	54	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	No lock variable	void exampleClassWithLockingErrors::setDataError2(const CString&str)	67	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	No lock variable	void exampleClassWithLockingErrors::setDataError3(const CString&str)	80	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError5(const CString&str)	102	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError7(const CString&str)	125	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Unknown unlocked	void exampleClassWithLockingErrors::setDataError8(const CString&str)	137	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Default unlocked	void exampleClassWithLockingErrors::setDataError9(const CString&str)	149	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError12(const CString&str)	187	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError14(const CString&str)	212	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError16(const CString&str)	237	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError18(const CString&str)	262	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError20(const CString&str)	289	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError21(const CString&str)	302	e:\om\c\threadLockChecker\tlcExample\exampleClassWithLockingErrors.cpp

A context menu is available on the error report grid.



The context menu has four options:

- Two options for filtering by directory and by filename. Any values set by these filters can be edited on the File Filters settings.

- An option to insert a line pragma at the end of the selected line
- An option to open an explorer window at the directory for the selected line.

Source Code Viewer

The source code viewer displays the source code with the line of interested highlighted in green. Click on any line in the error report grid to view the source code in the source code viewer. If you would rather edit the source code using Visual Studio, double click the line in the grid.



```
void exampleClassWithLockingErrors::setDataError9(const CString &str)
140 }
141
142 //-----
143 // Example correct API usage
144 // will be reported for default unlocked
145 //-----
146
147 void exampleClassWithLockingErrors::setDataError9(const CString &str)
148 {
149     CSingleLock lock(&dataSect);
150
151     data = str;
152 }
153
154 //-----
155 // Example correct API usage
156 // will be reported for using variable.
157 // This example tests more advanced parsing
158 //-----
159
```

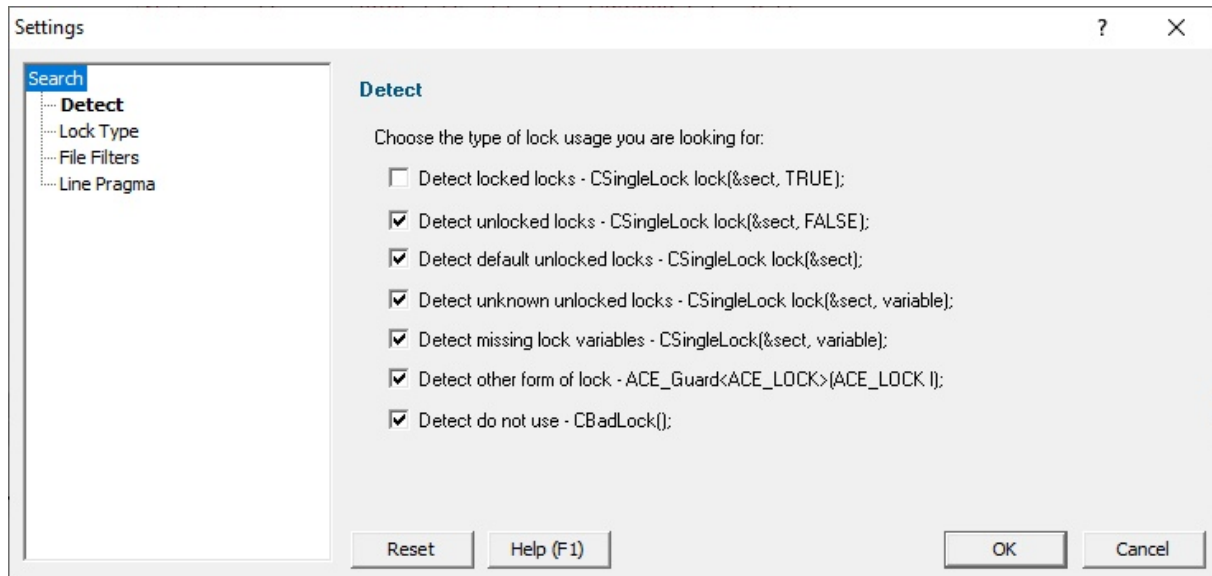
Part

VI

6 Modifying the settings

The settings dialog allows you to control all Thread Lock Checker settings.

The settings are grouped into logical choices making it easier to manage the settings.

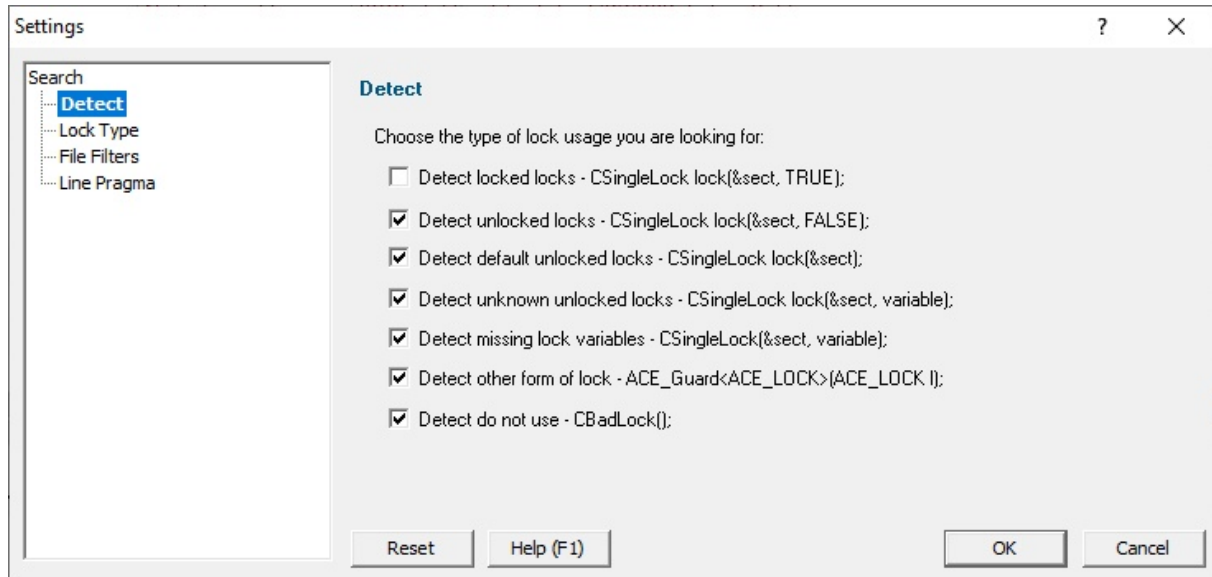


Reset

Click **Reset** to reset all settings on all pages of the settings dialog.

6.1 Detect

The Detect page of the settings dialog allows you to control which types of error you are interested in.



Detect lock usage

Thread Lock Checker allows you to detect a variety of different lock usage patterns. Depending on how you use your lock classes you will be interested in different groups of the patterns. Select the checkbox next to the types of usage you want Thread Lock Checker to report.

Detect locked locks

Detect lock usage of the form

```
CSingleLock lock(&sect, TRUE);
```

Detect unlocked locks

Detect lock usage of the form

```
CSingleLock lock(&sect, FALSE);
```

Detect default unlocked locks

Detect lock usage of the form

```
CSingleLock lock(&sect);
```

Detect unknown unlocked locks

Detect lock usage of the form

```
CSingleLock lock(&sect, variable);
```

Detect missing lock variables

Detect lock usage of the form

```
CSingleLock(&sect, variable);
```

Detect other form of lock

Detect an alternate usage of the lock

```
ACE_Guard<ACE_LOCK>(ACE_LOCK 1);
```

Detect do not use

Detect lock that should not be used

```
CBadlock();
```

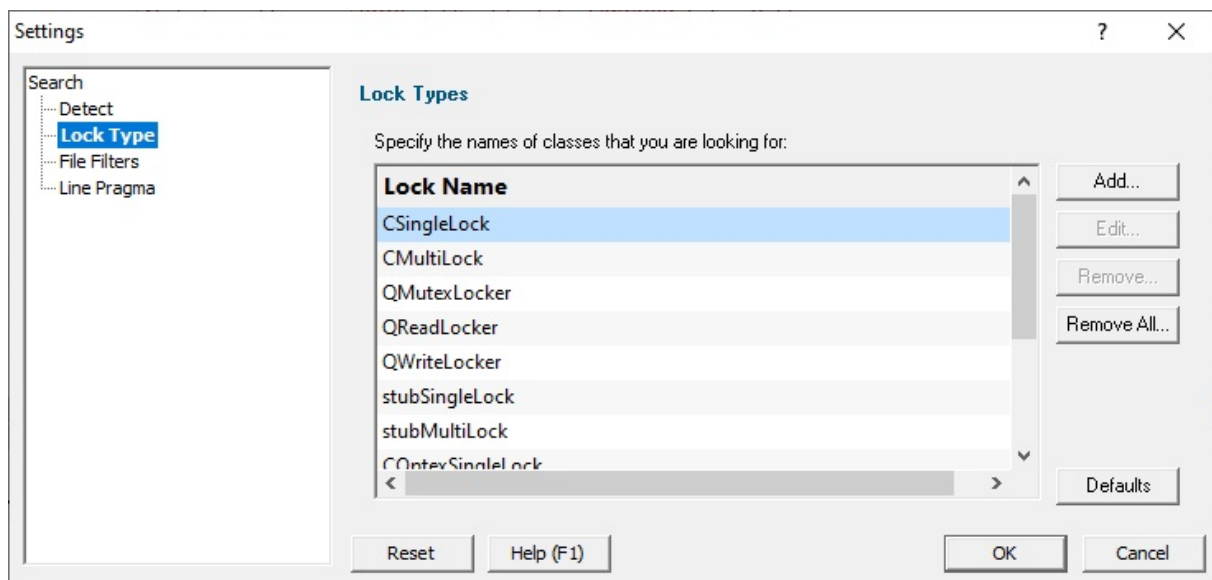
Reset

Click **Reset** to reset all settings on all pages of the settings dialog.

6.2 Lock Type

The Lock Type page of the settings dialog allows you to specify lock types Thread Lock Checker will search for.

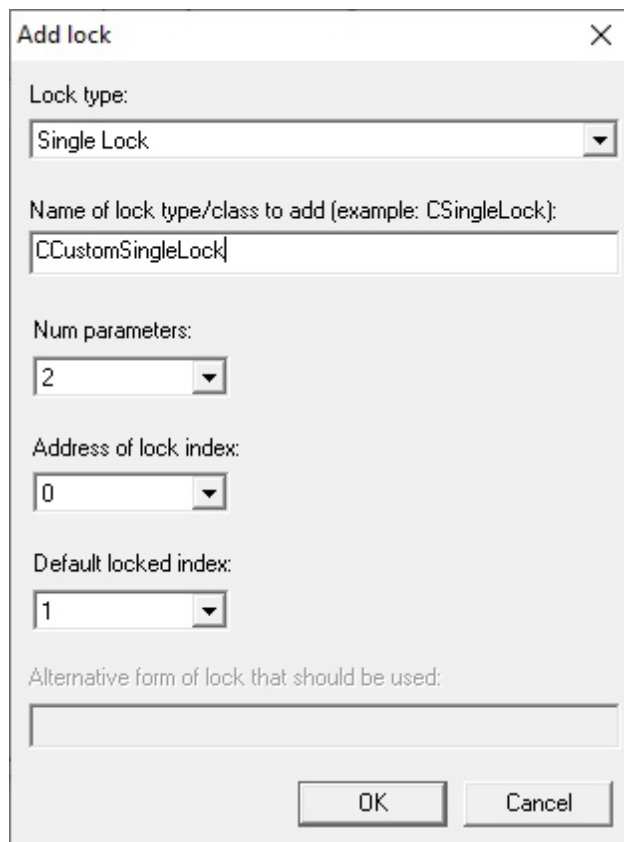
The default list of lock types that Thread Lock Checker will scan for are listed here.



You can add and remove lock classes that should be scanned.

Add

Click **Add** to add a new lock class.

The image shows a Windows-style dialog box titled "Add lock" with a close button (X) in the top right corner. The dialog contains several fields: "Lock type:" with a dropdown menu showing "Single Lock"; "Name of lock type/class to add (example: CSingleLock):" with a text input field containing "CCustomSingleLock"; "Num parameters:" with a dropdown menu showing "2"; "Address of lock index:" with a dropdown menu showing "0"; "Default locked index:" with a dropdown menu showing "1"; and "Alternative form of lock that should be used:" with an empty text input field. At the bottom right are "OK" and "Cancel" buttons.

Add lock

Lock type:
Single Lock

Name of lock type/class to add (example: CSingleLock):
CCustomSingleLock

Num parameters:
2

Address of lock index:
0

Default locked index:
1

Alternative form of lock that should be used:

OK Cancel

The Add Lock dialog box will be displayed. Type the name of the new class and click OK.

Remove

Click **Remove** to remove the selected lock classes.

Remove All

Click **Remove All** to remove all lock classes.

Defaults

Click **Defaults** to restore the default lock classes.

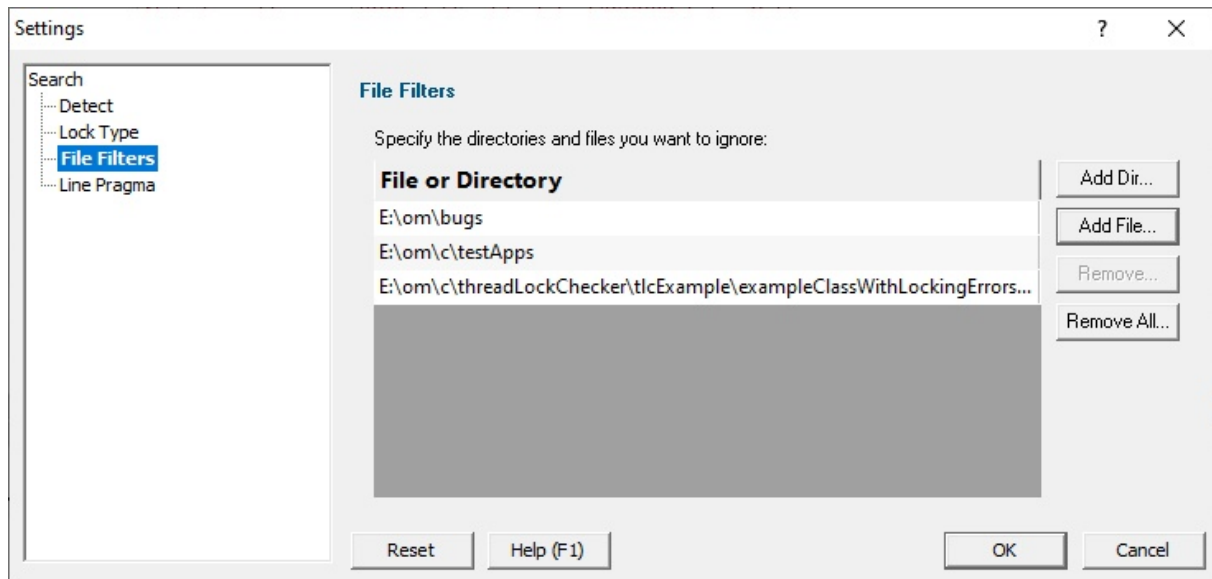
Reset

Click **Reset** to reset all settings on all pages of the settings dialog.

6.3 File Filters

The File Filters page of the settings dialog allows you to specify directories and files that Thread Lock Checker will ignore when searching for thread lock errors.

This allows you to ignore directories files where you deliberately have special case usage of these lock types (unit tests, development work, custom use cases) that would trigger warning results in Thread Lock Checker that you don't want.

**Add Dir...**

Click **Add Dir...** to add a directory. The Microsoft directory chooser will be displayed.

Add File...

Click **Add File...** to add a file. The Microsoft file chooser will be displayed.

Remove

Click **Remove** to remove the selected directories and files.

Remove All

Click **Remove All** to remove all directories and files.

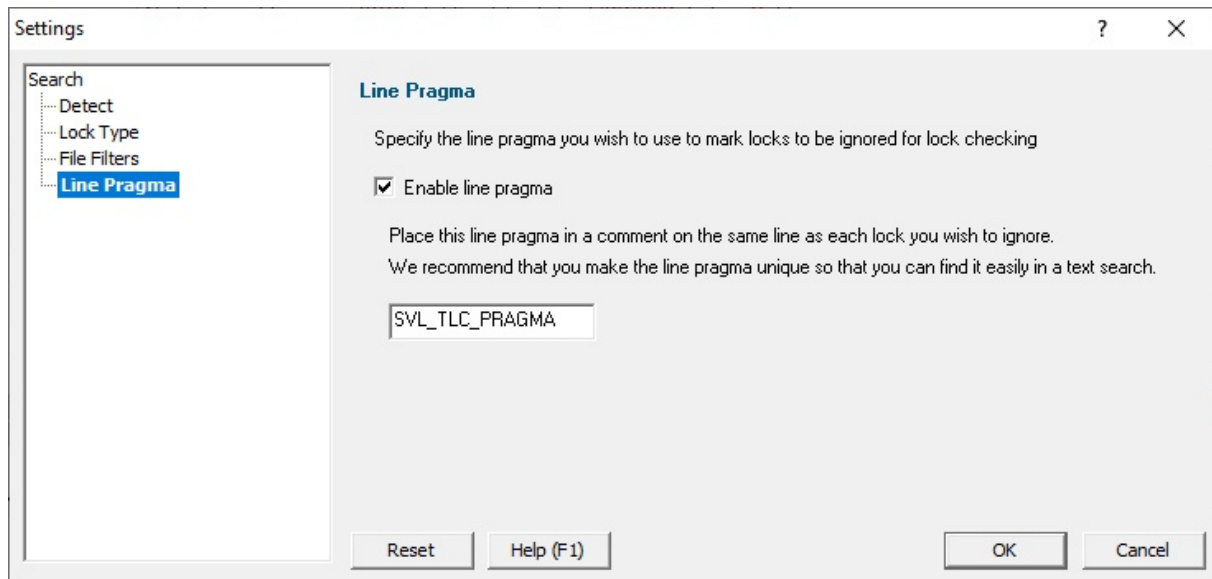
Reset

Click **Reset** to reset all settings on all pages of the settings dialog.

6.4 Line Pragma

The Line Pragma page of the settings dialog allows you to specify a line pragma that can be used to mark individual locks to be ignored when checking for thread locking errors.

This allows you identify special lock use cases in your source code to avoid FALSE positives from Thread Lock Checker.



When line pragmas are enabled any line containing the specified line pragma in a comment (single line comment or multi-line comment) will be ignored when checking for thread locking errors.

If the line pragma is enabled, the line pragma cannot be empty or whitespace. We recommend a value that is unique across your codebase so that you can find all line pragmas easily if you need to search for them.

Example usage of the pragma in code:

```
{
    CSingleLock    lock(&critSect, FALSE);    // SVL_TLC_PRAGMA, this lock is intentionally unlock
    ...
}
```

Part

VII

7 How to use Thread Lock Checker

To use Thread Lock Checker follow these basic steps:

1. If you want to change the type of lock usage to scan for, or to change the type of lock classes to scan for, **Edit > Settings...** to open the settings dialog.
2. **File > Scan directory...** to display the directory chooser.



You can type the directory path or use the Microsoft directory chooser by clicking **Browse...**

When you are happy with your choice of directory click **Start Scan** to start the process of finding locking errors.

3. If at any time you want to stop the scan **File > Stop scan**.
4. Any lock errors are shown in the grid.

Lock	Reason	Function	Line Number	Filename
CSingleLock	No lock variable	void exampleClassWithLockingErrors::setDataError1(const CString&str)	54	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	No lock variable	void exampleClassWithLockingErrors::setDataError2(const CString&str)	67	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	No lock variable	void exampleClassWithLockingErrors::setDataError3(const CString&str)	80	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError5(const CString&str)	102	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError7(const CString&str)	125	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Unknown unlocked	void exampleClassWithLockingErrors::setDataError8(const CString&str)	137	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Default unlocked	void exampleClassWithLockingErrors::setDataError9(const CString&str)	149	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError12(const CString&str)	187	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError14(const CString&str)	212	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError16(const CString&str)	237	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError18(const CString&str)	262	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError20(const CString&str)	289	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp
CSingleLock	Unlocked	void exampleClassWithLockingErrors::setDataError21(const CString&str)	302	e:\om\c\threadLockChecker\Example\exampleClassWithLockingErrors.cpp

5. Select an error shown in the grid to view the source code in the source code window.


```
void exampleClassWithLockingErrors::setDataError9(const CString &str)
{
    //-----
    // Example correct API usage
    // will be reported for default unlocked
    //-----
    void exampleClassWithLockingErrors::setDataError9(const CString &str)
    {
        CSingleLock lock(&dataSect);
        data = str;
    }
    //-----
    // Example correct API usage
    // will be reported for using variable.
    // This example tests more advanced parsing
    //-----
}
```

Double click on the error to edit the source code in Visual Studio.

6. If you want to clear the results, **File > Clear**.
7. If you want to filter a location by directory, right click on the entry in the list and choose **Filter Directory**.
8. If you want to filter a location by filename, right click on the entry in the list and choose **Filter Filename**.
9. If you want to mark a lock to be ignored for lock checking, right click on the entry in the list and choose **Add Line Pragma to Source Code**.
10. If you want to go to the source directory, right click on the entry in the list and choose **Open Folder...**

Part



8 Command Line Interface

Thread Lock Checker can be used from the command line as well as with the GUI.

User interface

If command line arguments are detected the user interface is not displayed.

Error reporting

When running from the command line any errors are written to the output file specified on the command line.

Return code

The process return code is 0 for no errors found.

If there are threading errors found the return code is a positive integer describing the number of errors found.

8.1 Alphabetic Reference

/addFileFilter

Adds a filename or directory to the list of file filters Thread Lock Checker will examine when checking for thread locking errors.

```
/addFileFilter filename  
/addFileFilter directory
```

Example: /addFileFilter e:\om\c\threadLockChecker\tlcExample\tlcExample.cpp

Example: /addFileFilter e:\om\c\threadLockChecker\tlcExample\

/addLockType

Adds a lock type to the list of lock types Thread Lock Checker will check.

```
/addLockType type
```

Example: ./addLockType CSingleLock

/detectDefaultUnlocked

Thread Lock Checker will detect locks configured as `CSingleLock lock(§);`

This looks like it's locked, but it's actually not locked (the default value for the second parameter is FALSE).

```
/detectDefaultUnlocked:On|Off
```

/detectLocked

Thread Lock Checker will detect locks configured as `CSingleLock lock(§, TRUE);`

This lock is locked. That's usually desirable behaviour. This checking option is normally turned off as you don't normally want to detect correct behaviour.

`/detectLocked:On|Off`

Example: `/detectLocked:On`

Example: `/detectLocked:Off`

`/detectNoLock`

Thread Lock Checker will detect locks configured as `CSingleLock(, variable);`

This last case is a compile error. As such it is unlikely to get to the point where you'll need to catch this before a commit to your version control system. But we detect such errors for completeness of error detection.

`/detectNoLock:On|Off`

`/detectNoLockVariable`

Thread Lock Checker will detect locks configured as `CSingleLock(§, variable);`

Note, the variable 'lock' in the other examples is missing.

`/detectNoLockVariable:On|Off`

`/detectUnknownUnlocked`

Thread Lock Checker will detect locks configured as `CSingleLock lock(§, variable);`

In this case we don't know if variable is TRUE or FALSE, so we bring this to your attention for manual checking.

`/detectUnknownUnlocked:On|Off`

`/detectUnlocked`

Thread Lock Checker will detect locks configured as `CSingleLock lock(§, FALSE);`

This lock isn't locked. Is this really what you want?

`/detectUnlocked:On|Off`

`/dir`

Specifies the directory hierarchy that will be scanned by Thread Lock Checker.

`/dir directory`

Example: `/dir e:\om\c\`

`//linePragma`

Specifies the line pragma to use if line pragmas are enabled.

`//linePragma pragma`

Example: `//linePragma SVL_TLC_PRAGMA`

Example usage of the pragma in code:

```
CSingleLock lock(&critSect, FALSE);           // SVL_TLC_PRAGMA, this
lock is intentionally unlocked to start with
```

/linePragmaEnable

Enables or disables the ignoring of locks if a line pragma is found in a comment on the same line as the lock declaration.

If you use this option to enable line pragmas you must use **/linePragma** to specify a line pragma to use. If you don't specify a line pragma this option does nothing.

/linePragmaEnable:On|Off

Example: /linePragmaEnable:On

Example: /linePragmaEnable:Off

/loadSettings

Specifies a file to load that contains settings that will override the current settings for this command line search.

/loadSettings filename

Example: /loadSettings e:\tests\test1\testSettings.tlc

/output

Specifies a filename to write the results of the command line search.

If no errors are found any file with this name will be deleted.

If errors are found a text file will be written containing error information. Error information will be listed one error per line.

/output filename

Example: /output e:\tests\test1\testResults.txt

/removeAllFileFilters

Removes all file filters from the list of file filters Thread Lock Checker will use to check which files and directories to ignore when checking for thread locking errors.

This option should be used before any use of /addFileFilter.

/removeAllFileFilters

/removeAllLockTypes

Removes all lock types from the list of lock types Thread Lock Checker will check.

This option should be used before any use of `/addLockType`.

`/removeAllLockTypes`

/reset

Reset all settings to the default values for this command line search.

You should put this first on your command line to create a known default state prior to setting values using the command line.

`/reset`

/turnAllDetectOff

Turns off all detect options.

You should put this before any `/detectXXXXX` options if you wish to start from a known (nothing detected) state when setting detect options.

`/turnAllDetectOff`

8.2 Usage Reference

Search

/dir

Specifies the directory hierarchy that will be scanned by Thread Lock Checker.

`/dir directory`

Example: `/dir e:\om\c\`

Detect

All `/detectXXXX` options are followed by `:On` or `:Off` to enable or disable the option

/detectLocked

Thread Lock Checker will detect locks configured as `CSingleLock lock(§, TRUE);`

This lock is locked. That's usually desirable behaviour. This checking option is normally turned off as you don't normally want to detect correct behaviour.

`/detectLocked:On|Off`

Example: `/detectLocked:On`

Example: `/detectLocked:Off`

/detectUnlocked

Thread Lock Checker will detect locks configured as `CSingleLock lock(§, FALSE);`

This lock isn't locked. Is this really what you want?

`/detectUnlocked:On|Off`

/detectDefaultUnlocked

Thread Lock Checker will detect locks configured as `CSingleLock lock(§);`

This looks like it's locked, but it's actually not locked (the default value for the second parameter is FALSE).

`/detectDefaultUnlocked:On|Off`

/detectUnknownUnlocked

Thread Lock Checker will detect locks configured as `CSingleLock lock(§, variable);`

In this case we don't know if variable is TRUE or FALSE, so we bring this to your attention for manual checking.

`/detectUnknownUnlocked:On|Off`

/detectNoLockVariable

Thread Lock Checker will detect locks configured as `CSingleLock(§, variable);`

Note, the variable 'lock' in the other examples is missing.

`/detectNoLockVariable:On|Off`

/detectNoLock

Thread Lock Checker will detect locks configured as `CSingleLock(, variable);`

This last case is a compile error. As such it is unlikely to get to the point where you'll need to catch this before a commit to your version control system. But we detect such errors for completeness of error detection.

`/detectNoLock:On|Off`

/turnAllDetectOff

Turns off all detect options.

You should put this before any `/detectXXXXX` options if you wish to start from a known (nothing detected) state when setting detect options.

`/turnAllDetectOff`

Lock Types

/addLockType

Adds a lock type to the list of lock types Thread Lock Checker will check.

`/addLockType type`

Example: `./addLockType CSingleLock`

/removeAllLockTypes

Removes all lock types from the list of lock types Thread Lock Checker will check.

This option should be used before any use of `/addLockType`.

`/removeAllLockTypes`

File Filters

/addFileFilter

Adds a filename or directory to the list of file filters Thread Lock Checker will examine when checking for thread locking errors.

`/addFileFilter filename`
`/addFileFilter directory`

Example: `/addFileFilter e:\om\c\threadLockChecker\tlcExample\tlcExample.cpp`

Example: `/addFileFilter e:\om\c\threadLockChecker\tlcExample\`

/removeAllFileFilters

Removes all file filters from the list of file filters Thread Lock Checker will use to check which files and directories to ignore when checking for thread locking errors.

This option should be used before any use of `/addFileFilter`.

`/removeAllFileFilters`

Line Pragma

/linePragmaEnable

Enables or disables the ignoring of locks if a line pragma is found in a comment on the same line as the lock declaration.

If you use this option to enable line pragmas you must use **/linePragma** to specify a line pragma to use. If you don't specify a line pragma this option does nothing.

`/linePragmaEnable:On|Off`

Example: `/linePragmaEnable:On`

Example: `/linePragmaEnable:Off`

/linePragma

Specifies the line pragma to use if line pragmas are enabled.

/linePragma pragma

Example: /linePragma SVL_TLC_PRAGMA

Example usage of the pragma in code:

```
CSingleLock  lock(&critSect, FALSE);           // SVL_TLC_PRAGMA, this
lock is intentionally unlocked to start with
```

Miscellaneous

/loadSettings

Specifies a file to load that contains settings that will override the current settings for this command line search.

/loadSettings filename

Example: /loadSettings e:\tests\test1\testSettings.tlc

/output

Specifies a filename to write the results of the command line search.

If no errors are found any file with this name will be deleted.

If errors are found a text file will be written containing error information. Error information will be listed one error per line.

/output filename

Example: /output e:\tests\test1\testResults.txt

/reset

Reset all settings to the default values for this command line search.

You should put this first on your command line to create a known default state prior to setting values using the command line.

/reset

8.3 Command Line Examples

This section provides some example command lines. For each example we'll break it down, argument by argument so that you can see how the command line works.

In all these examples the executable to run is **threadLockChecker.exe**. You will need to provide the full path to threadLockChecker.exe, or add the path to threadLockChecker install directory to your \$PATH.

Example 1

threadLockChecker.exe /dir e:\om\c\ /output e:\test.txt

/dir e:\om\c

Scan the directory e:\om\c\

/output e:\test.txt

Write the output to e:\test.txt. If there are no errors there will be no output file.

Example 2

threadLockChecker.exe /dir e:\om\c\ /output e:\test.txt /removeAllLockTypes /addLockType CSingleLock

/dir e:\om\c

Scan the directory e:\om\c\

/output e:\test.txt

Write the output to e:\test.txt. If there are no errors there will be no output file.

/removeAllLockTypes

Remove all existing lock type definitions.

/addLockType CSingleLock

Add one lock type definition so that Thread Lock Checker is checking for just the type **CSingleLock**

Example 3

threadLockChecker.exe /loadSettings e:\tests\test1\settings.tlc /output e:\test.txt

/loadSettings e:\tests\test1\settings.tlc

Load settings from the file e:\tests\test1\settings.tlc.

/output e:\test.txt

Write the output to e:\test.txt. If there are no errors there will be no output file.

Example 4

threadLockChecker.exe /loadSettings e:\tests\test1\settings.tlc /output e:\test.txt /dir e:\om\c

/loadSettings e:\tests\test1\settings.tlc

Load settings from the file e:\tests\test1\settings.tlc.

/output e:\test.txt

Write the output to e:\test.txt. If there are no errors there will be no output file.

/dir e:\om\c

Scan the directory e:\om\c\

