Resolve Resource Leaks to Improve Stability

Introduction

A resource leak refers to a type of resource consumption in which the program cannot release resources it has acquired. Typically the result of a bug, common resource issues, such as memory leaks, often only cause problems in very specific situations or after extensive use of an application.

Intel® Inspector XE is a dynamic analysis tool for serial and parallel applications, bringing together both memory and threading error-checking capabilities. It detects resource leaks after a single occurrence, enabling you to locate errors even when they don't appear in a reproducible application failure. By running Intel Inspector XE on your application, you can pinpoint and fix resource leaks before they become a problem. Intel® Inspector XE can track 26 different types of resources, helping you identify places in your code where resources are allocated but never released. It is available for both Linux* and Windows* including integration with Microsoft* Visual Studio*. It supports applications created with the C/C++, C#, .NET, and Fortran languages.

This guide will show you how to use Intel Inspector XE to identify and fix resource leak errors in your programs before they start causing problems.

Step by Step: Identify, Analyze, and Resolve Resource Errors

You can use Intel Inspector XE to identify, analyze, and resolve resource errors in serial or parallel programs by performing a series of steps in your workflow. This tutorial guides you through these workflow steps while using a sample program named "Colors."

NOTE: Intel Inspector XE integrates into Microsoft Visual Studio 2010* and above. These tutorials contain instructions and screens for the Microsoft Visual Studio 2010 development environment (IDE). To use a different IDE, replace the menu items with the related menu items for your IDE.

Step 1. Install and Set Up Intel® Parallel Studio XE 2015

Install and Set Up Intel Parallel Studio XE

1. Download and install an evaluation copy of the Intel Parallel Studio XE.

Install the Colors sample application

1. Download the colors-conf.zip sample file to your local machine. This is a C++ GUI application created with Microsoft Visual Studio* 2010.
2. Extract the files from the **colors-conf.zip** file to a writable directory or share on your system.

Run the sample application

1. Open the sample in Microsoft Visual Studio. Go to **File > Open > Project/Solution** and open the **colors_conf\vc8\colors.sln** solution file. Allow the solution to be converted if required. Figure 1

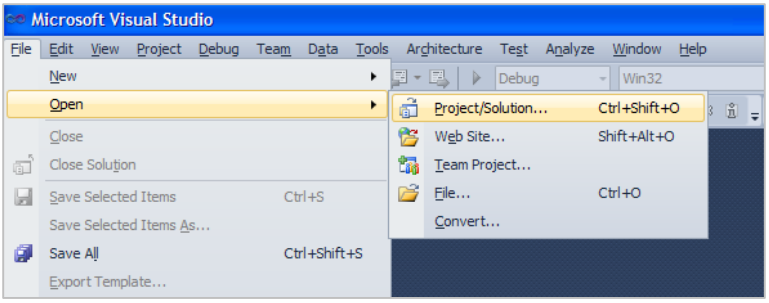

Figure 1

This will display the colors solution in the Solution Explorer pane. Figure 2

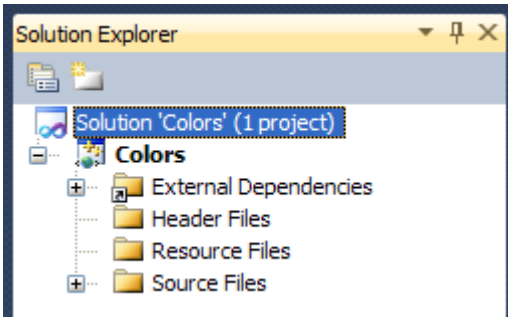

Figure 2

2. Build the application using **Build > Build Solution**. Figure 3

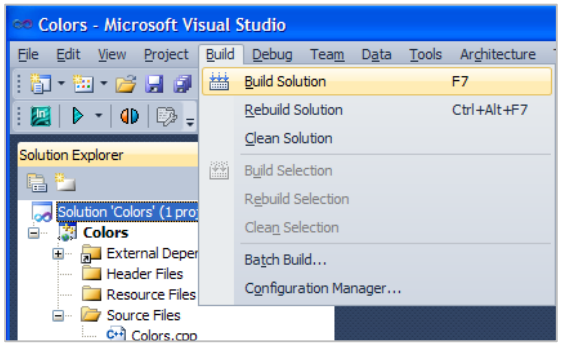

Figure 3

3. Run the application using Debug > Start Without Debugging. Figure 4.

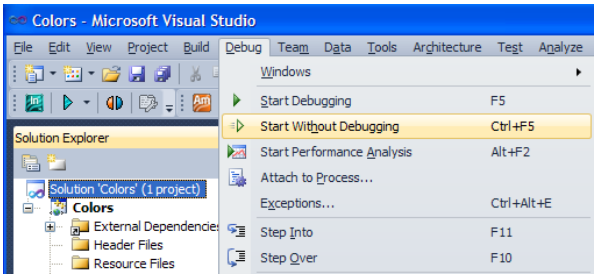

Figure 4

The application should appear as shown in Figure 5.

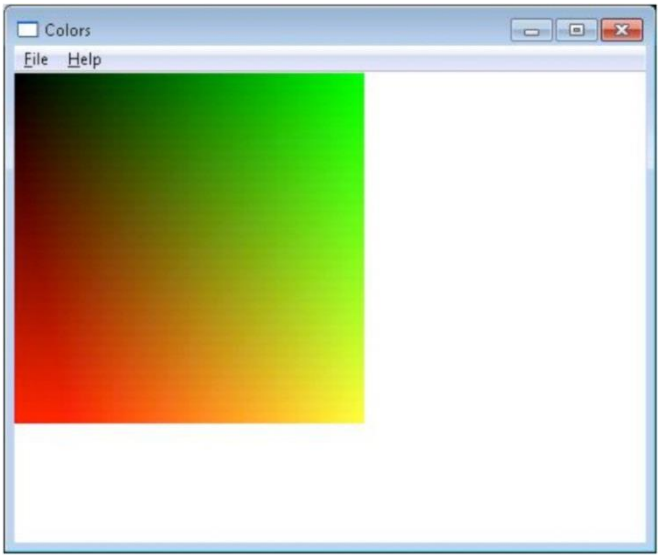

Figure 5

Next, try resizing the window twice. After the first resize, the application will look normal, but after the second resize you should see a failure that looks like the one illustrated in Figure 6.

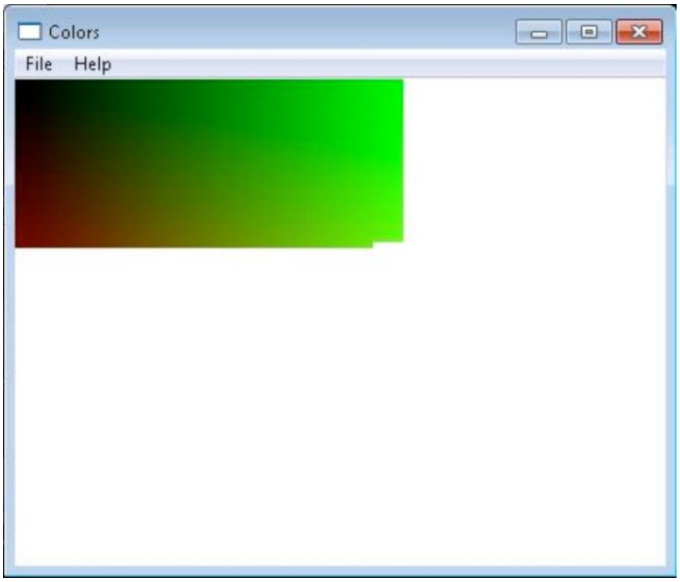

Figure 6

If you resize the window again, the colors will disappear completely, and the window control buttons won't be redrawn correctly. These problems are caused by a resource leak in the program.

Configure and run analysis

Choose a preset configuration to influence memory error analysis scope and running time.

To configure a memory error analysis:

1. From the Microsoft Visual Studio menu, choose **Tools > Intel Inspector XE > New Analysis...** to display an Analysis Type window.
2. Choose the **Detect Leaks** analysis type to display a window similar to the following. Figure 7

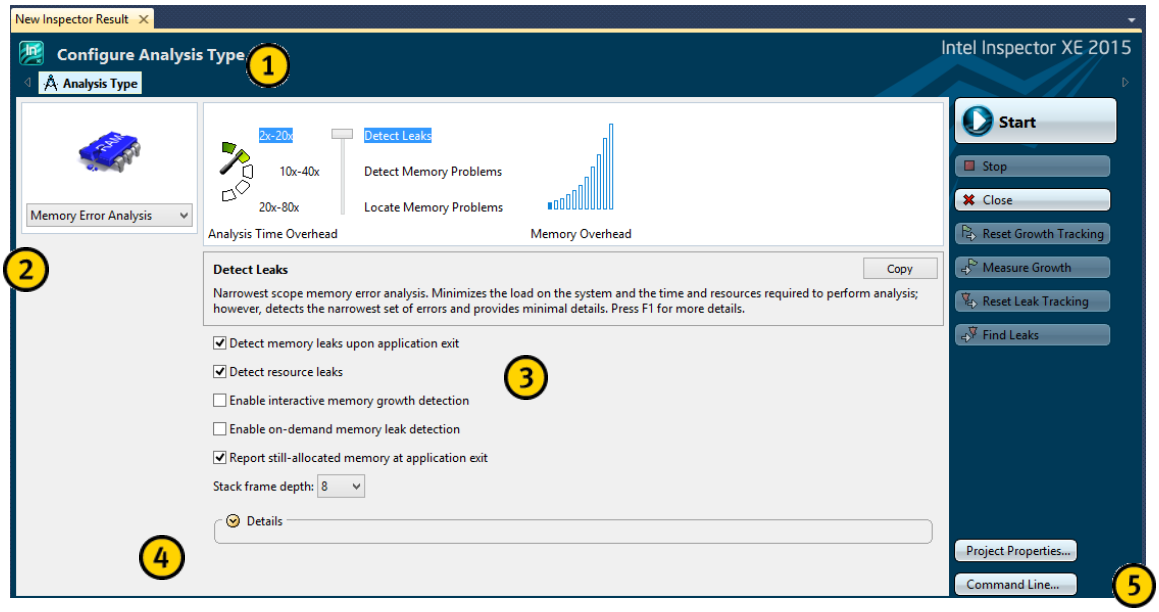

Figure 7

1. Use the Navigation toolbar to navigate among the Intel Inspector XE windows. The buttons on the toolbar vary depending on the displayed window.

2. The Analysis Type tree shows available preset analysis types.

   This tutorial covers memory error analysis types, which you can use to search for these kinds of errors: GDI resource leak, incorrect memcpy call, invalid deallocation, kernel resource leak, invalid memory access, invalid partial memory access, memory leak, mismatched allocation/deallocation, missing allocation, uninitialized memory access, and uninitialized partial memory access.

   Use threading error analysis types to search for these kinds of errors: Data race, deadlock, lock hierarchy violation, and cross-thread stack access.

   You can also use the **Copy** button to create custom analysis types from existing analysis types.

3. Use the checkbox(es) and drop-down list(s) to fine-tune some, but not all, analysis type settings. If you need to fine-tune more analysis type settings, choose another preset analysis type or create a custom analysis type.

4. The Details region shows all current analysis type settings. Try choosing a different preset analysis type or checkbox/drop-down list value to see the impact on the Details region.

5. Use the Command toolbar to control analysis runs and perform other functions. For example, use the Project Properties button to display the Project Properties dialog box, where you can change the default result directory location, set parameters to potentially speed up analysis, and perform other project configuration functions.

3. Finally, click the Start button to start analyzing the application. You should see the same results as shown in Figure 8. As you did before, resize the window a couple of times to cause the erroneous display, and then close the application. Intel Inspector XE will take a short time to perform its final analysis. While this analysis is being conducted, you will see the window identified in Figure 8 in your Microsoft Visual Studio document area.

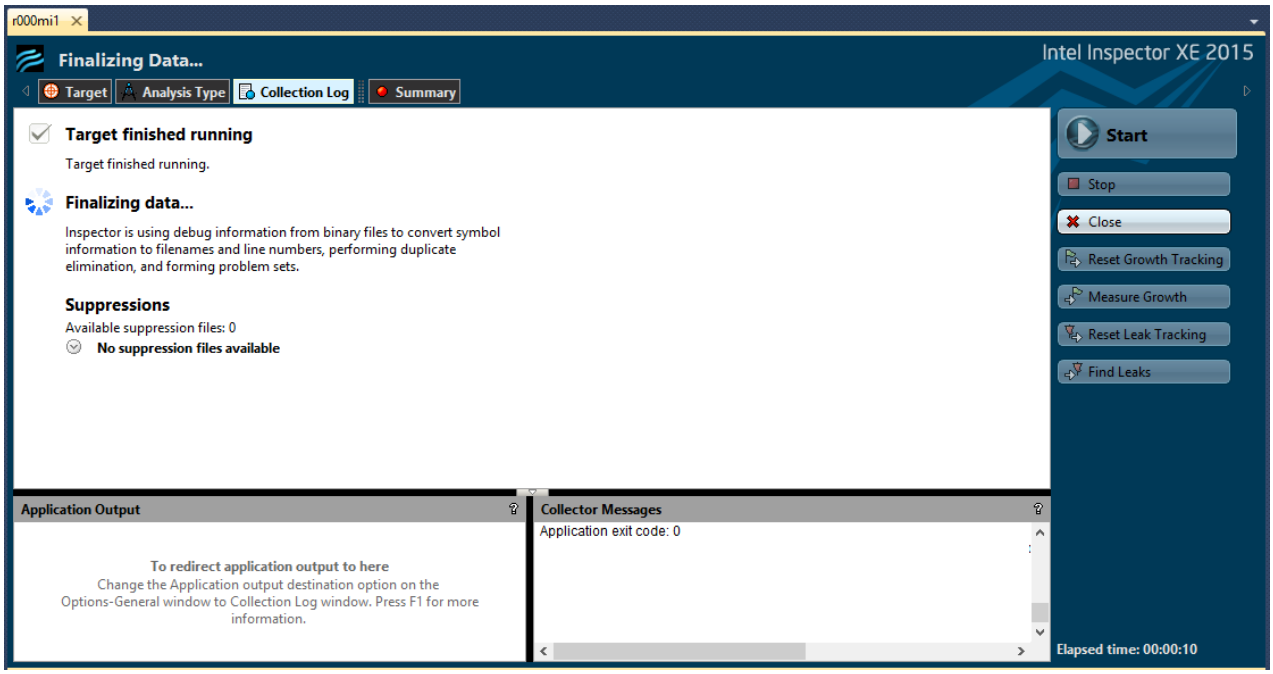

Figure 8

When Intel Inspector XE has finished its analysis, you will see the window identified in Figure 9

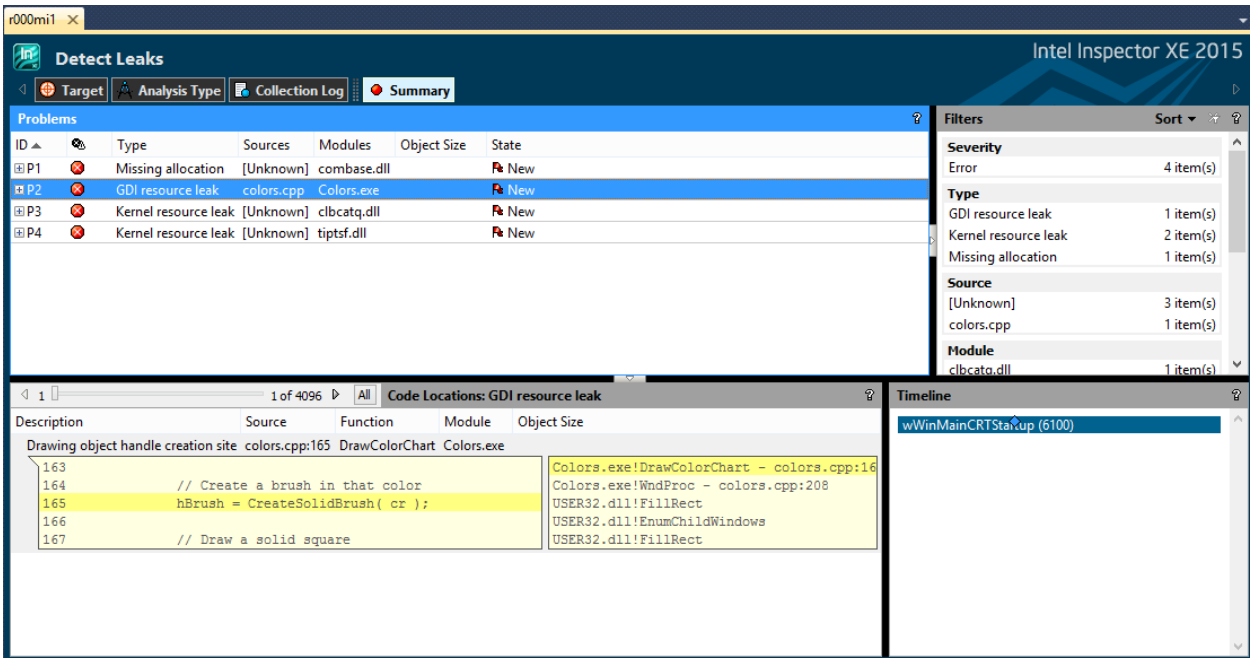

Figure 9

Interpret the results

This output, as seen in Figure 9, shows a GDI resource leak. The lower pane informs you that this is a drawing object handle leak in the DrawColorChart function on line 165 of the colors.cpp file. Double click on this problem in the Problems list to go to the Sources view.

This view shows you the application source code where the leaked handle was created, and the call stack that led to the error. Figure 10

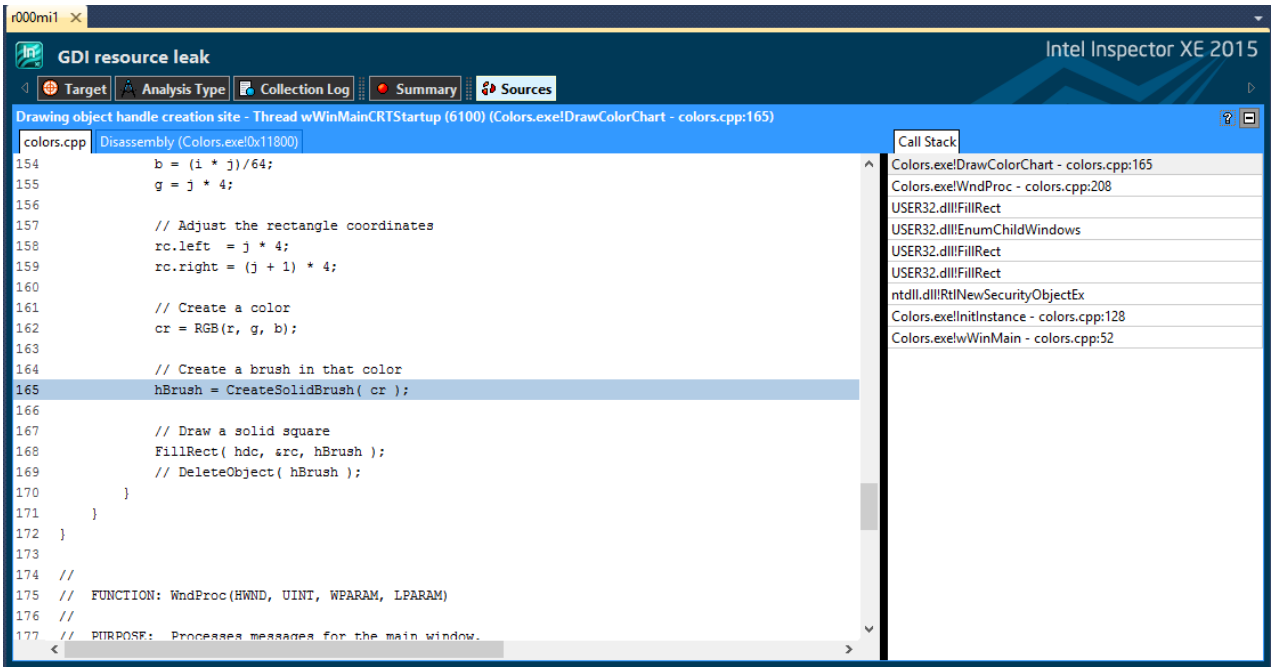

Figure 10

By examining the source code, you can see that a GDI brush is created on line 165 and never released.

For the purpose of reporting resource leaks, Intel Inspector XE does not distinguish between resources for which the program has kept a handle but has never released and resources for which the program has lost all handles. Before fixing the problem, you must look at your source code and determine the necessary lifespan of the resource that is being allocated.

In this case, however, the handle is not used outside of the immediate scope in which it is created, and so it can be deleted immediately after it is used. If you are unsure of the correct function to use to release this resource, you may right click on the code location in the lower-left pane and select **Explain Problem** and Intel Inspector XE will give you a description of the resource leak, including the correct function to call to release the resource. Figure 11

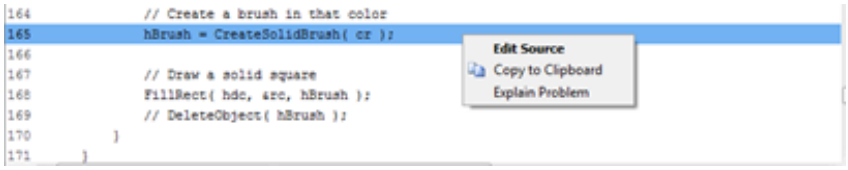

Figure 11

When you are ready to fix the problem, double click the line of interest in the Sources view, and the source file will open for editing at the line you selected. To resolve the problem in the sample application, add the following code below the call to `FillRect()`:

```
DeleteObject( hBrush );
```

Now, rebuild the solution and run it again. This time, you should be able to resize the window repeatedly without any malfunctions.

Success

In this example, we had a resource leak error that was producing visible and easily reproducible results. However, resource leaks are not always this obvious. Frequently, a resource leak only causes problems in very specific situations or after extensive use of the application.

Intel Inspector XE detects resource leaks after a single occurrence. Therefore, it is able to help you locate errors even when they don't appear in a reproducible application failure. Periodically running Intel Inspector XE on your application can help you find and fix resource leak errors early in the development lifecycle.

Tips for Larger/Complex Applications

Key Concept: Choosing Small, Representative Data Sets

When you run an analysis, Intel Inspector XE executes the target against a data set. Data set size has a direct impact on target execution time and analysis speed.

For example, it takes longer to process a 1000x1000 pixel image than a 100x100 pixel image. One possible reason could be that you have loops with an iteration space of 1…1000 for the larger image, but only 1…100 for the smaller image. The exact same code paths may be executed in both cases. The difference is the number of times these code paths are repeated.

You can control analysis cost, without sacrificing completeness, by removing this kind of redundancy from your target. Instead of choosing large, repetitive data sets, choose small, representative data sets. Data sets with runs in the time range of seconds are ideal. You can always create additional data sets

To get help on inspxe-cl, use the –help command line option.

Managing Threading Errors

Intel Inspector XE can also identify, analyze, and resolve threading errors, such as latent data races and deadlocks in parallel programs. Subtle errors can manifest intermittently and non-deterministically, making them extremely hard to find, reproduce, and fix.

Using the Command-line to Automate Testing

As you can see, Intel Inspector XE has to execute your code path to find errors in it. Thus, run Intel Inspector XE on multiple versions of your code, on different workloads that stress different code paths, as well as on corner cases. Furthermore, given the inherent time dilation that comes with code-inspection tools, it would be more efficient to run these tests overnight or as part of your regression testing suite and have the computer do the work for you; you just examine the results of multiple tests in the morning.

The Intel Inspector XE command-line version is called inspxe-cl, and is available by opening a command window (Start > Run, type in "cmd" and press OK) and typing in the path leading to where you installed Intel Inspector XE. Figure 12

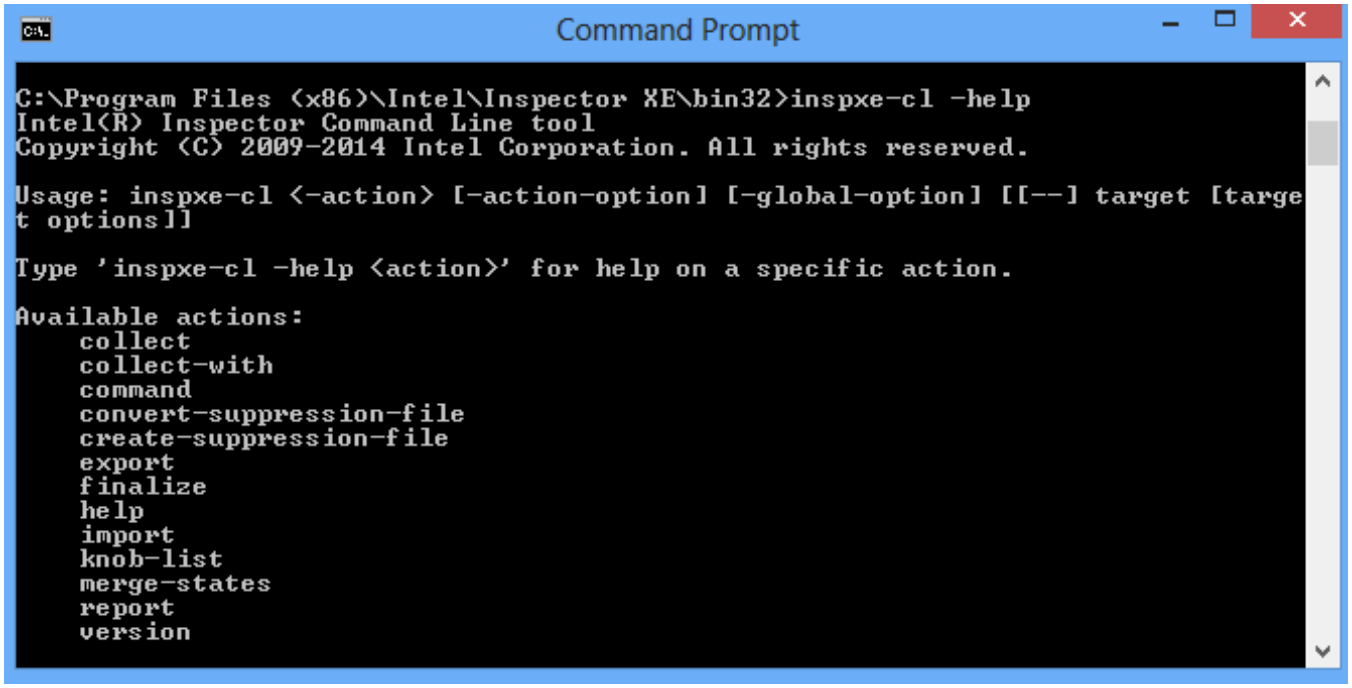

Figure 12

Additional Resources

Learning Lab – Technical videos, whitepapers, webinar replays and more.

Intel Parallel Studio XE page – How to videos, getting started guides, documentation, product details, support and more.

Evaluation Guide Portal – Additional evaluation guides that show how to use various powerful capabilities.

Intel® Software Network Forums – A community for developers.

Intel® Software Products Knowledge Base – Access to information about products and licensing,

Download a free 30 day evaluation

Purchase Options: Language Specific Suites

Intel® Parallel Studio XE comes in three editions based on your development needs. Single language (C++ or Fortran) versions are available in the Composer and Professional editions.

- **Composer Edition** includes compilers, performance libraries, and parallel models made to build fast parallel code.
- **Professional Edition** includes everything in the Composer edition. It adds performance profiler, threading design/prototyping, and memory & thread debugger to design, build, debug and tune fast parallel code.
- **Cluster Edition** includes everything in the Professional edition. It adds a MPI cluster communications library, along with MPI error checking and tuning to design, build, debug and tune fast parallel code that includes MPI.

	Intel® Parallel Studio XE Composer Edition[1]	Intel® Parallel Studio XE Professional Edition[1]	Intel® Parallel Studio XE Cluster Edition
Intel® C++ Compiler	√	√	√
Intel® Fortran Compiler	√	√	√
Intel® Threading Building Blocks (C++ only)	√	√	√
Intel® Integrated Performance Primitives (C++ only)	√	√	√
Intel® Math Kernel Library	√	√	√
Intel® Cilk™ Plus (C++ only)	√	√	√
Intel® OpenMP*	√	√	√
Rogue Wave IMSL* Library[2] (Fortran only)	Bundled and Add-on	Add-on	Add-on
Intel® Advisor XE		√	√
Intel® Inspector XE		√	√
Intel® VTune™ Amplifier XE[3]		√	√
Intel® MPI Library[3]			√
Intel® Trace Analyzer and Collector			√
Operating System (Development Environment)	Windows* (Visual Studio*) Linux* (GNU) OS X*[4] (XCode*)	Windows (Visual Studio) Linux (GNU)	Windows (Visual Studio) Linux (GNU)

Notes:
1. Available with a single language (C++ or Fortran) or both languages.
2. Available as an add-on to any Windows Fortran* suite or bundled with a version of the Composer Edition.
3. Available bundled in a suite or standalone
4. Available as single language suites on OS X.

Learn more about Intel Parallel Studio XE

- Click or enter the link below: http://intel.ly/parallel-studio-xe
- Or scan the QR code on the left

Download a free 30-day evaluation

- Click or enter the link below: http://intel.ly/sw-tools-eval
- Click on 'Product Suites' link

Notices

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Intel_Resolve_Resource_Leaks_Eval_Guide/Rev-081714