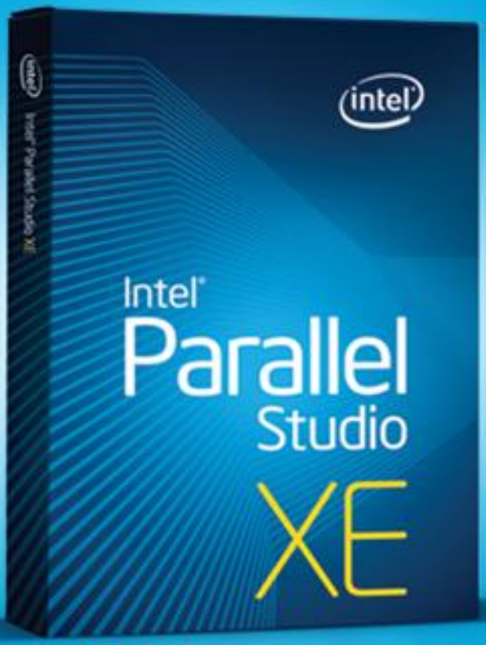
Resolve Resource Leaks in Your Applications

with Intel® Parallel Studio XE



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.

Introduction

After you check your program for threading and memory errors and it is clean, if you still have an intermittent failure that you cannot quite track down, it could be caused by a resource leak. Most resource leaks are benign. You allocate a handle and keep it around for the life of your program, and then when your program exits, the resources are automatically released.

You can usually get away with that, but eventually resources such as a file here and a hatched brush there add up. Some resource types, such as GDI brushes, are in limited supply but all resources consume some amount of system memory. The result may be sluggish system performance or unexpected system API failure.

Intel® Inspector XE is a serial and multithreading error-checking analysis tool which 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 easy, comprehensive developer-productivity tool pinpoints errors and provides guidance to help ensure application reliability and quality.

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 2005*, 2008*, and 2010*. These tutorials contain instructions and screens for the Microsoft Visual Studio 2005 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

Estimated completion time: 15-30 minutes

1 Download an evaluation copy of Intel Parallel Studio XE.

2. Install Intel Parallel Studio XE by clicking on the **parallel_studio_xe_2011_setup.exe** (can take 15 to 30 minutes depending on your system).

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* 2005.

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. Figure 1

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

Figure 1

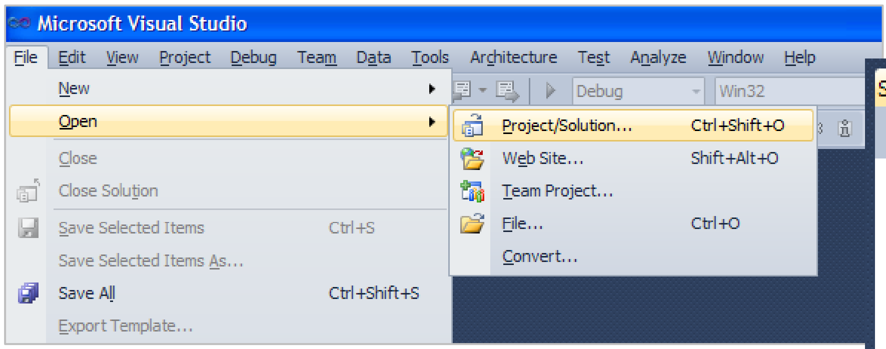
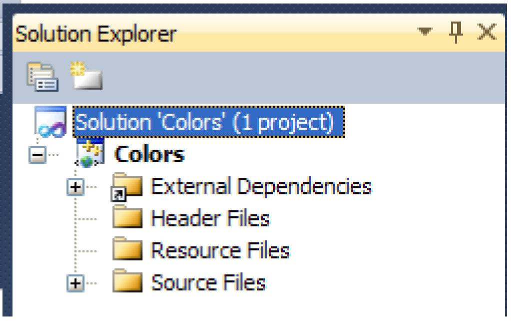

Microsoft Visual Studio
File Edit View Project Debug Team Data Tools Architecture Test Analyze Window Help
New
Open
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Export Template...
Debug
Win32
Project/Solution... Ctrl+Shift+O
Web Site... Shift+Alt+O
Team Project...
File... Ctrl+O
Convert...

Figure 2



2. Build the application using **Build** > **Build Solution**. Figure 3
3. Run the application using **Debug** > **Start Without Debugging**. Figure 4.

The application should appear as shown in 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.

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.

Figure 3

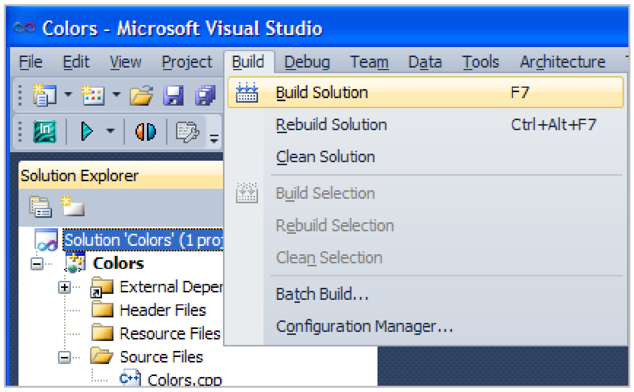


Figure 4

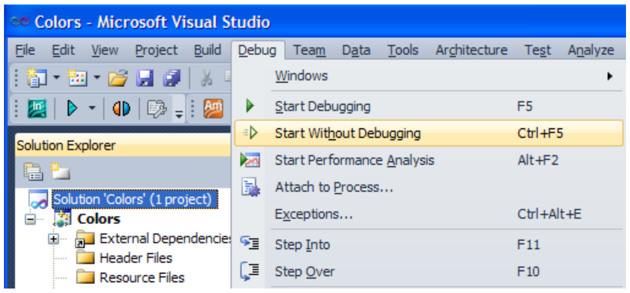

Colors - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze
Windows
Start Debugging F5
Start Without Debugging Ctrl+F5
Start Performance Analysis Alt+F2
Attach to Process...
Exceptions... Ctrl+Alt+E
Step Into F11
Step Over F10
Solution Explorer
Solution 'Colors' (1 project)
Colors
External Dependencie
Header Files
Resource Files

Figure 5

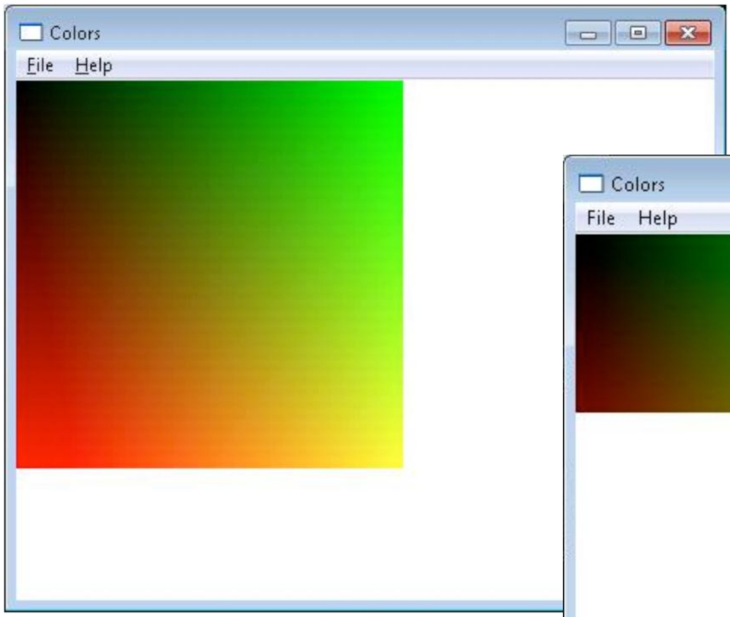
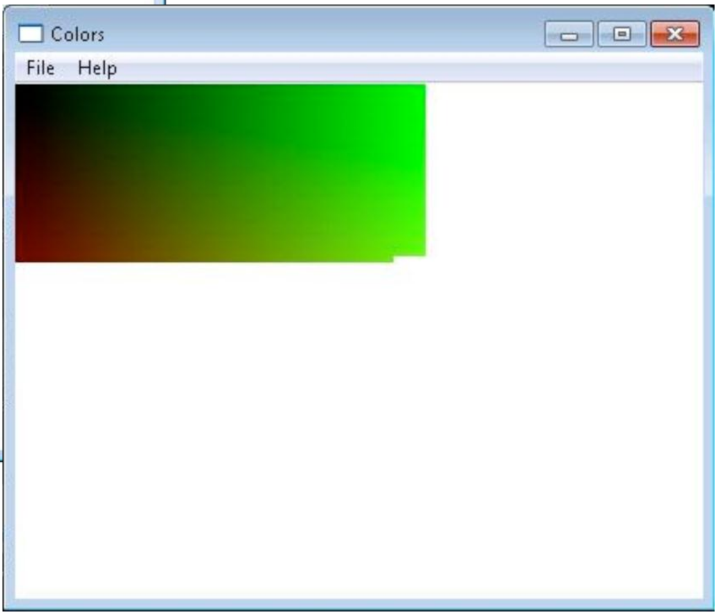


Figure 6



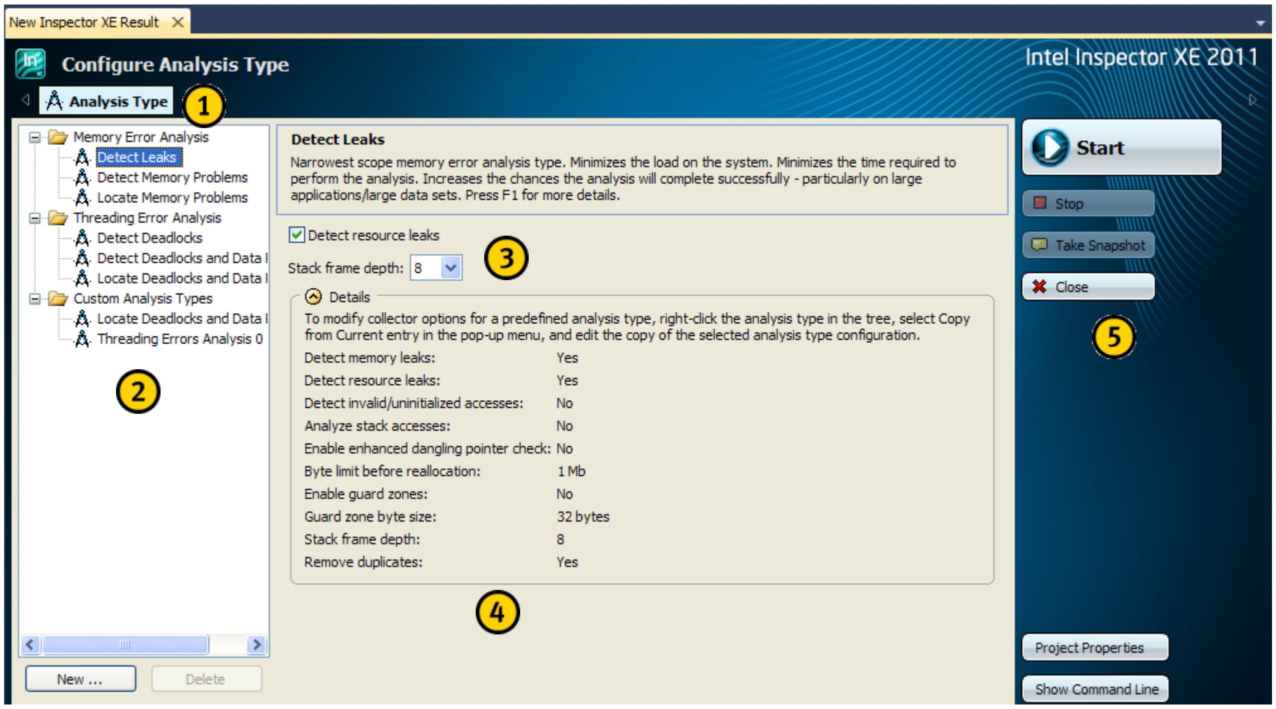
Configure and run analysis

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 2011** > **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



Detect Leaks

Narrowest scope memory error analysis type. Minimizes the load on the system. Minimizes the time required to perform the analysis. Increases the chances the analysis will complete successfully - particularly on large applications/large data sets. Press F1 for more details.

☑ Detect resource leaks

Stack frame depth: 8

Details

To modify collector options for a predefined analysis type, right-click the analysis type in the tree, select Copy from Current entry in the pop-up menu, and edit the copy of the selected analysis type configuration.

Detect memory leaks:	Yes
Detect resource leaks:	Yes
Detect invalid/uninitialized accesses:	No
Analyze stack accesses:	No
Enable enhanced dangling pointer check:	No
Byte limit before reallocation:	1 Mb
Enable guard zones:	No
Guard zone byte size:	32 bytes
Stack frame depth:	8
Remove duplicates:	Yes

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 **New** 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.

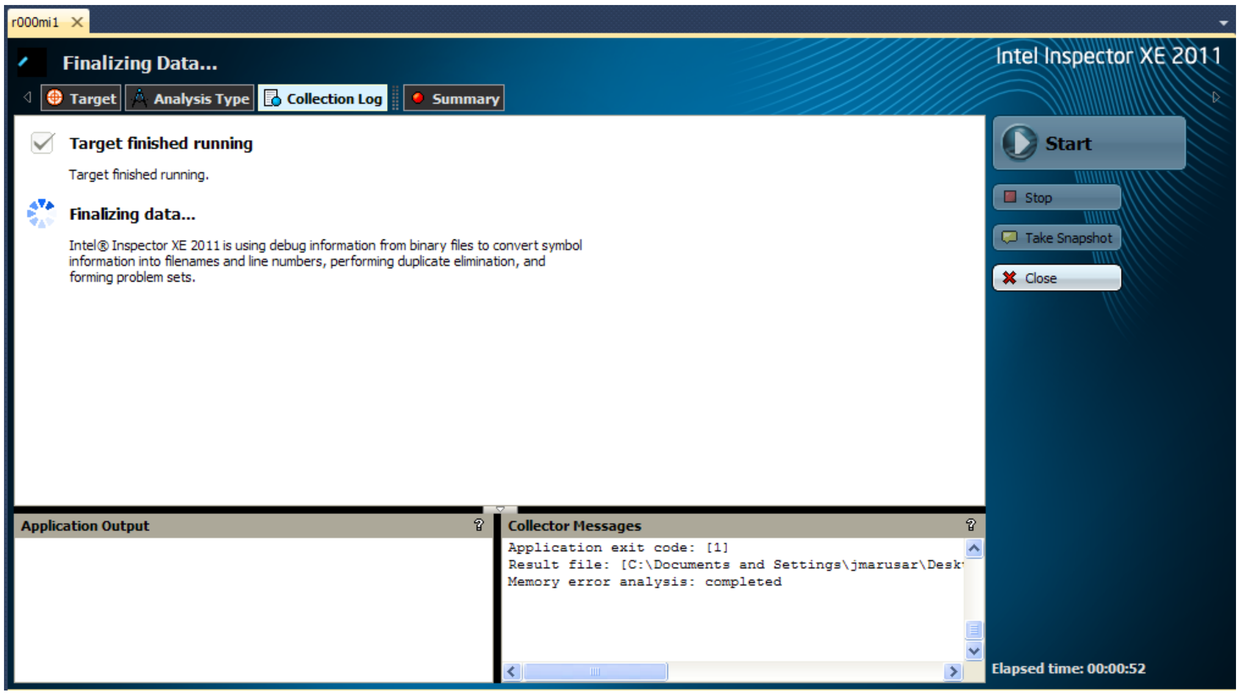
4 The **Details** region shows all current analysis type settings. Try choosing a different preset analysis type or checkbox/drop-down list value to see the impact on the **Details** region.

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 5. 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.

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

Figure 8



Target finished running

Target finished running.

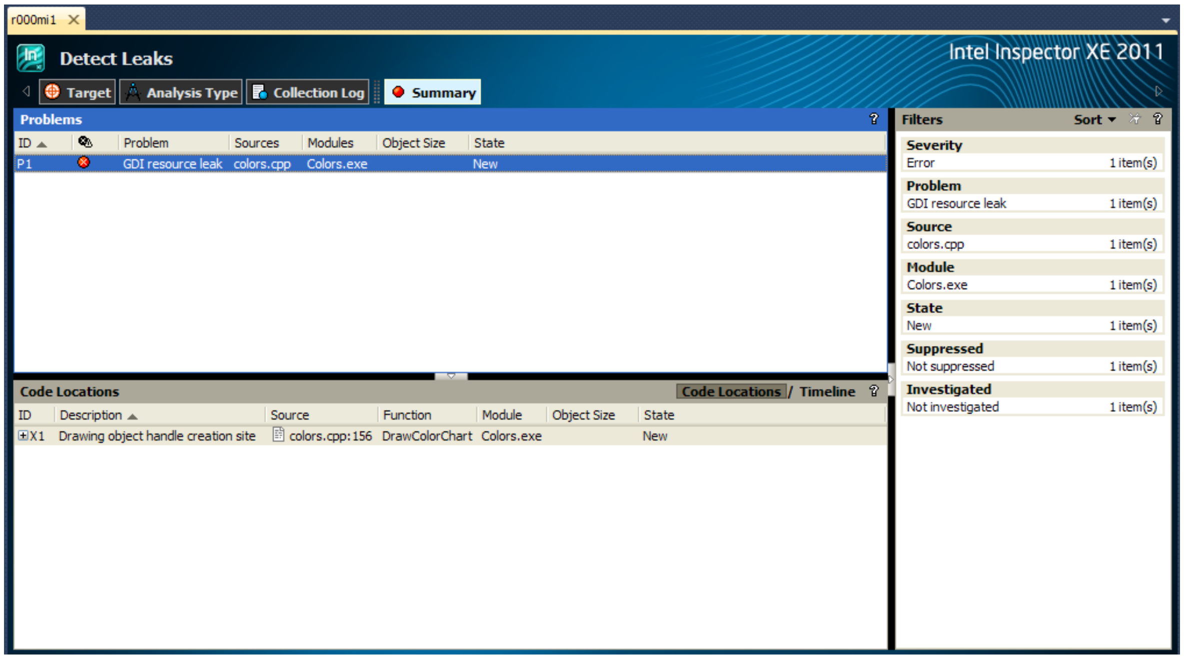
Finalizing data...

Intel® Inspector XE 2011 is using debug information from binary files to convert symbol information into filenames and line numbers, performing duplicate elimination, and forming problem sets.

```
Application exit code: [1]
Result file: [C:\Documents and Settings\jmarusar\Desk
Memory error analysis: completed
```

Elapsed time: 00:00:52

Figure 9



ID		Problem	Sources	Modules	Object Size	State
P1	●	GDI resource leak	colors.cpp	Colors.exe		New

ID	Description	Source	Function	Module	Object Size	State
X1	Drawing object handle creation site	colors.cpp:156	DrawColorChart	Colors.exe		New

Filters	
Severity	
Error	1 item(s)
Problem	
GDI resource leak	1 item(s)
Source	
colors.cpp	1 item(s)
Module	
Colors.exe	1 item(s)
State	
New	1 item(s)
Suppressed	
Not suppressed	1 item(s)
Investigated	
Not investigated	1 item(s)

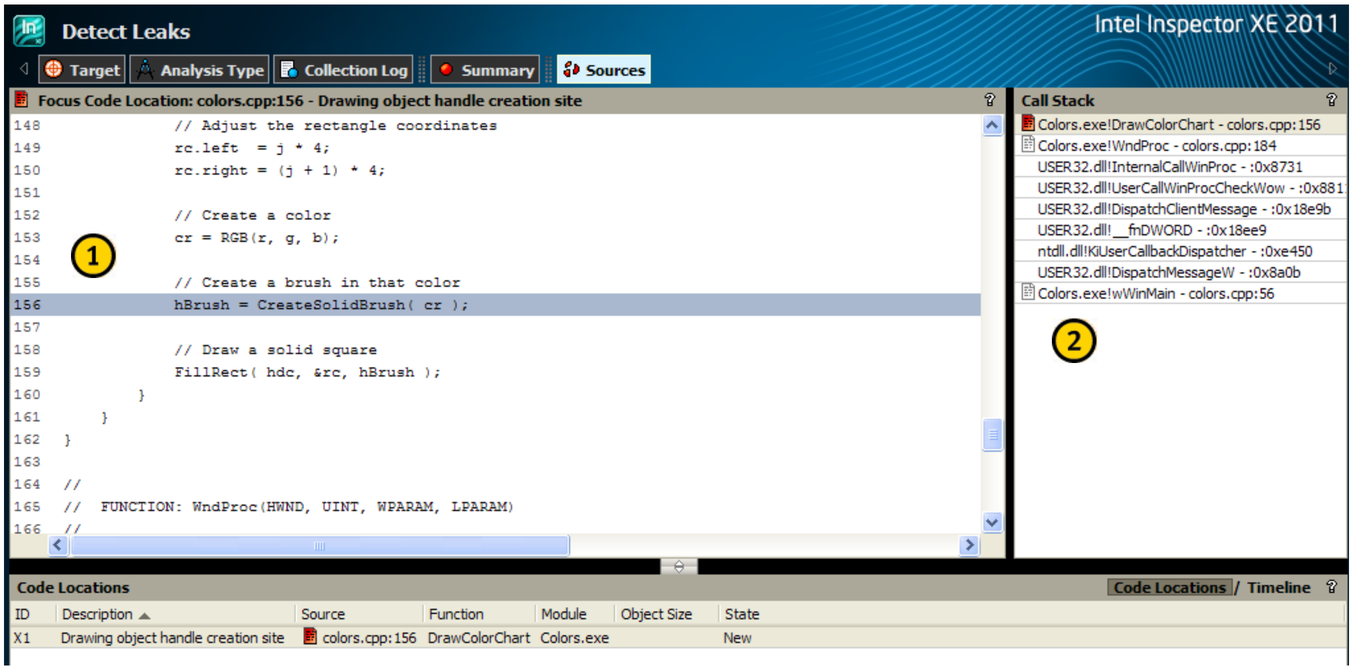
Interpret the results

This output, as seen in Figure 9, shows a single GDI resource leak. The lower pane informs you that this is a drawing object handle leak in the DrawColorChart function on line 156 of the colors.cpp file. Double click on this problem in the Problems list to go to the Sources view.

This view shows you (1) the application source code where the leaked handle was created, and (2) the call stack that led to the error. Figure 10

By examining the source code, you can see that a GDI brush is created on line 156 and never released.

Figure 10



Focus Code Location: colors.cpp:156 - Drawing object handle creation site

```
148             // Adjust the rectangle coordinates
149             rc.left  = j * 4;
150             rc.right = (j + 1) * 4;
151
152             // Create a color
153             cr = RGB(r, g, b);
154
155             // Create a brush in that color
156             hBrush = CreateSolidBrush( cr );
157
158             // Draw a solid square
159             FillRect( hdc, &rc, hBrush );
160         }
161     }
162 }
163
164 //
165 //  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
166 //
```

Call Stack

```
Colors.exe!DrawColorChart - colors.cpp:156
Colors.exe!WndProc - colors.cpp:184
USER32.dll!InternalCallWinProc - :0x8731
USER32.dll!UserCallWinProcCheckWow - :0x881
USER32.dll!DispatchClientMessage - :0x18e9b
USER32.dll!__fnDWORD - :0x18ee9
ntdll.dll!KiUserCallbackDispatcher - :0xe450
USER32.dll!DispatchMessageW - :0x8a0b
Colors.exe!wWinMain - colors.cpp:56
```

Code Locations

ID	Description ▲	Source	Function	Module	Object Size	State
X1	Drawing object handle creation site	colors.cpp:156	DrawColorChart	Colors.exe		New

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.

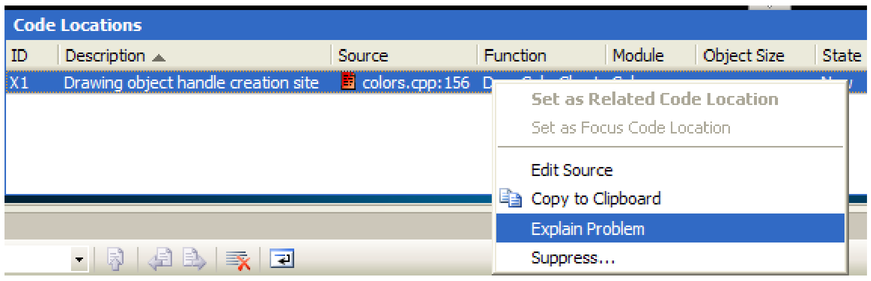
In this case, however, the handle is not used outside of the immediate scope in which it is created, and so it can be deleted immediately after it is used. If you are unsure of the correct function to use to release this resource, you may right click on the code location in the lower-left pane and select 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

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.

Figure 11



ID	Description ▲	Source	Function	Module	Object Size	State
X1	Drawing object handle creation site	colors.cpp:156	D			

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 ensure all your code is inspected.

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

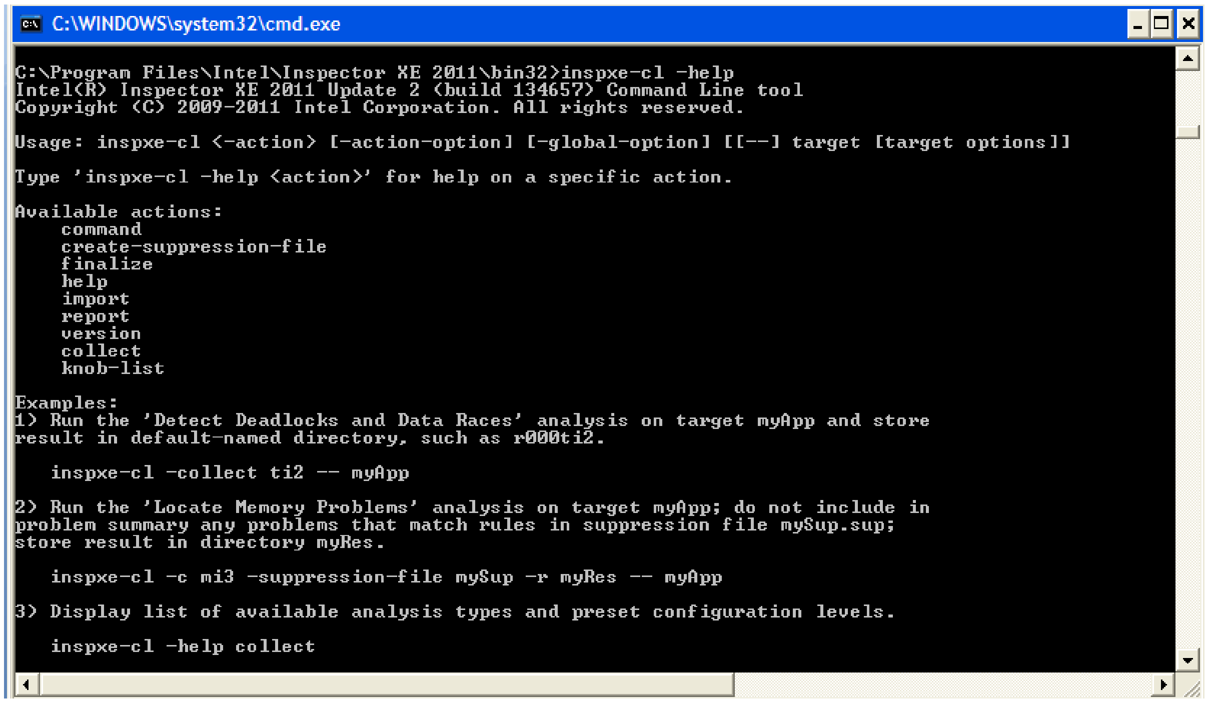
As you can see, Intel Inspector XE has to execute your code path to find errors in it. Thus, run Intel Inspector XE on multiple versions of your code, on different workloads that stress different code paths, as well as on corner cases. Furthermore, given the inherent time dilation that comes with code-inspection 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

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

```
> c:\Program Files\Intel\Inspector XE 2011\bin32\inspxe-cl -help
```

Figure 12



```
C:\Program Files\Intel\Inspector XE 2011\bin32>inspxe-cl -help
Intel(R) Inspector XE 2011 Update 2 (build 134657) Command Line tool
Copyright (C) 2009-2011 Intel Corporation. All rights reserved.

Usage: inspxe-cl <-action> [-action-option] [-global-option] [[--] target [target options]]

Type 'inspxe-cl -help <action>' for help on a specific action.

Available actions:
    command
    create-suppression-file
    finalize
    help
    import
    report
    version
    collect
    knob-list

Examples:
1) Run the 'Detect Deadlocks and Data Races' analysis on target myApp and store
result in default-named directory, such as r000ti2.

   inspxe-cl -collect ti2 -- myApp

2) Run the 'Locate Memory Problems' analysis on target myApp; do not include in
problem summary any problems that match rules in suppression file mySup.sup;
store result in directory myRes.

   inspxe-cl -c mi3 -suppression-file mySup -r myRes -- myApp

3) Display list of available analysis types and preset configuration levels.

   inspxe-cl -help collect
```

Additional Resources

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

Intel Parallel Studio XE product 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