

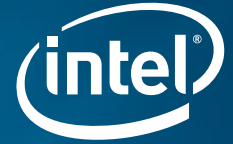


# Resolve Resource Leaks in Your Applications

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® Parallel Inspector 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 Parallel Inspector 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® Parallel Inspector 2011 is a serial and multithreading error-checking analysis tool for Microsoft Visual Studio\* C/C++ developers. Intel Parallel Inspector tracks 26 different types of resources, helping you identify places in your code where resources are allocated but never released. Intel Parallel Inspector also detects challenging memory leaks and corruption errors as well as threading data races and deadlock errors. 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 Parallel Inspector to identify and fix resource leak errors in your programs before they start causing problems.

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

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

You can use Intel Parallel Inspector 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 Parallel Inspector 2011 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.*

#### Install Parallel Inspector 2011

1. [Download](#) an evaluation copy of Intel Parallel Studio.
2. Click parallel\_studio\_setup.exe to install Intel Parallel Studio.

#### 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](#)

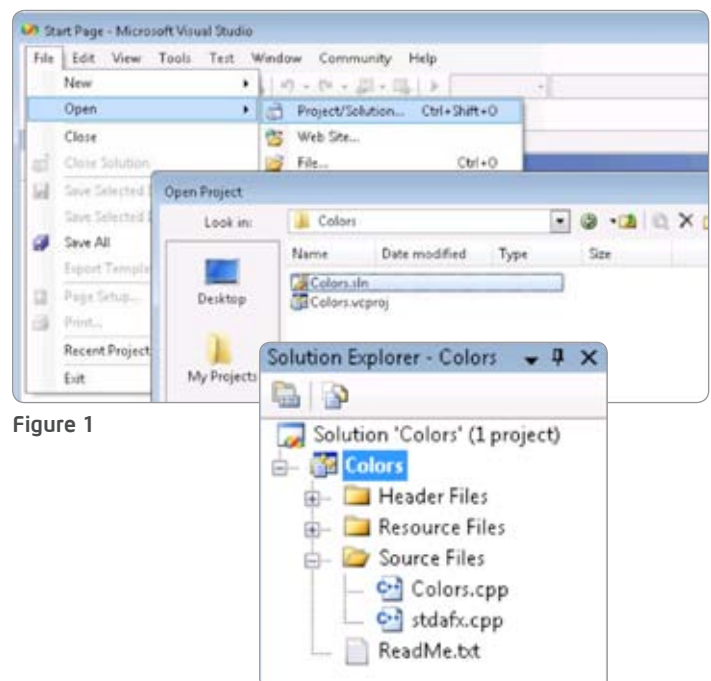


Figure 1

Figure 2

# Resolve Resource Leaks in Your Applications



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.

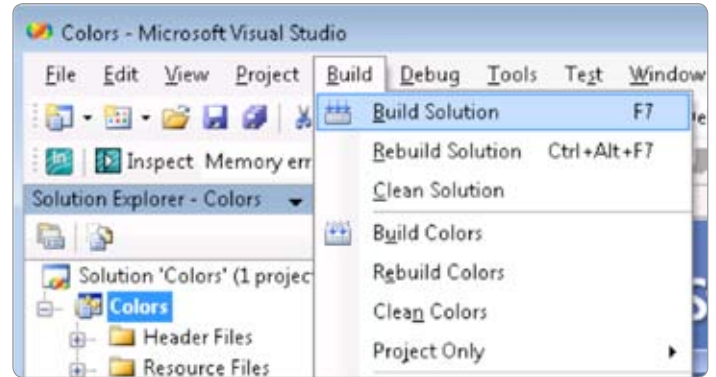


Figure 3

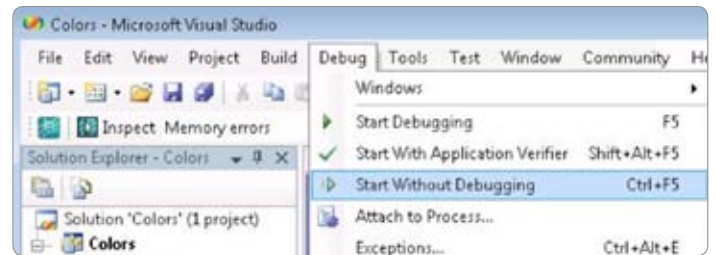


Figure 4

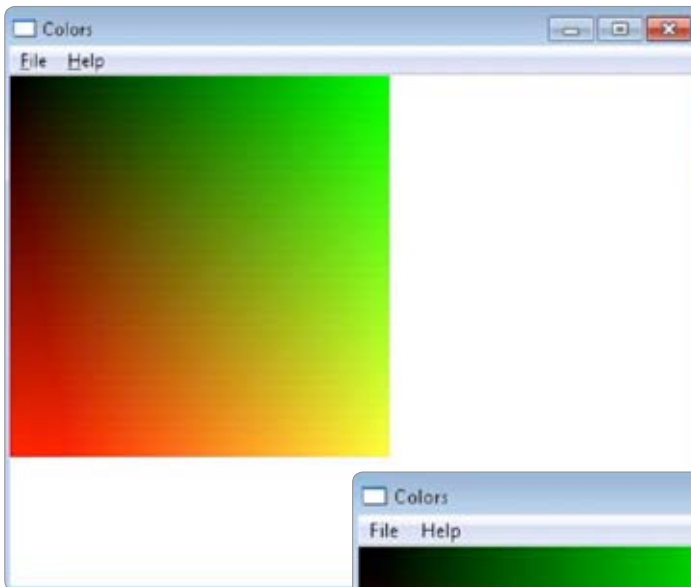


Figure 5

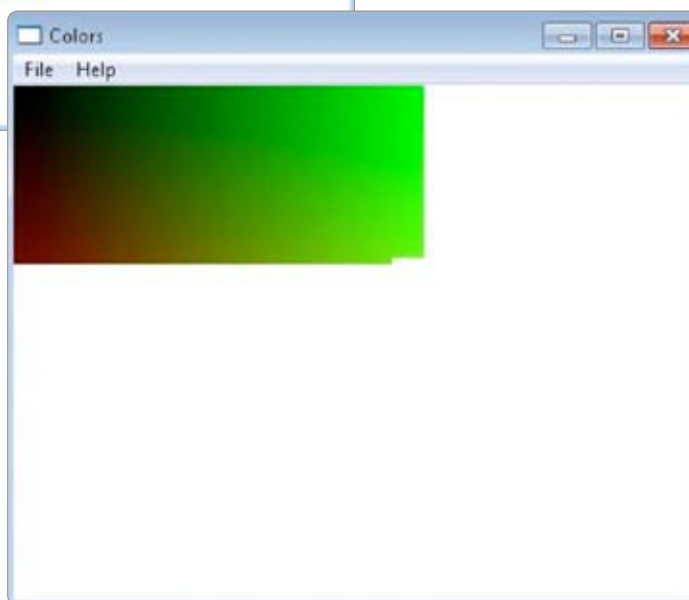


Figure 6

# Resolve Resource Leaks in Your Applications



## Configure and run analysis

Now let's run Intel Parallel Inspector to find the error. Click the **Configure Analysis** button on the Intel Parallel Inspector toolbar. **Figure 7**

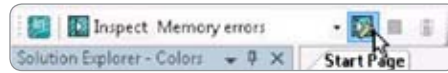


Figure 7

Set the dial to the **Does my target leak memory? level (mi1)**. Resource leak detection is performed in conjunction with every level of memory checking. The mi1 level has the least overhead and is the best choice if you are only interested in checking for resource leaks **Figure 8**.

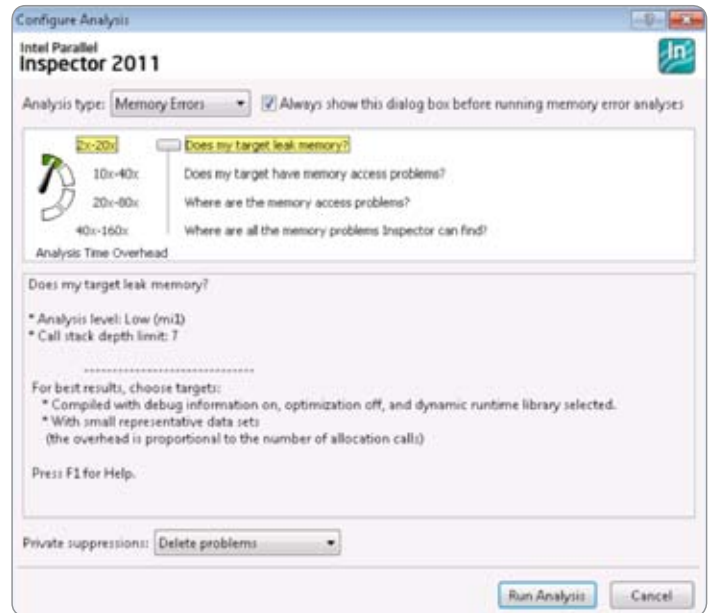


Figure 8

Finally, click the **Run Analysis** 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 Parallel Inspector will take a short time to perform its final analysis. While this analysis is being conducted, you will see the window identified in **Figure 9** in your Microsoft Visual Studio document area.

When Intel Parallel Inspector has finished its analysis, you will see the window identified in **Figure 10**.

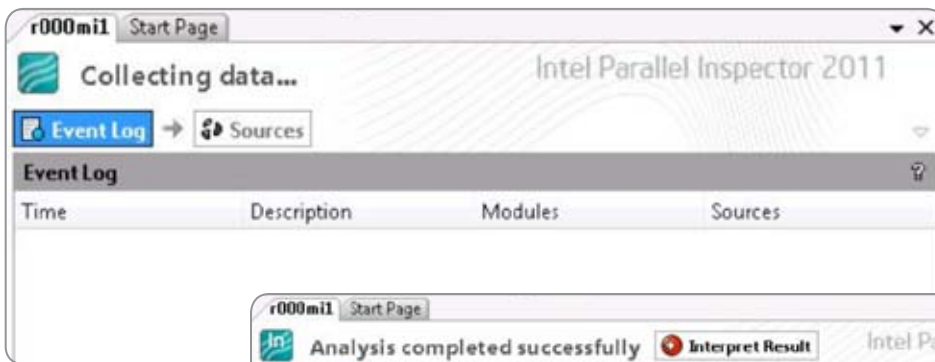


Figure 9

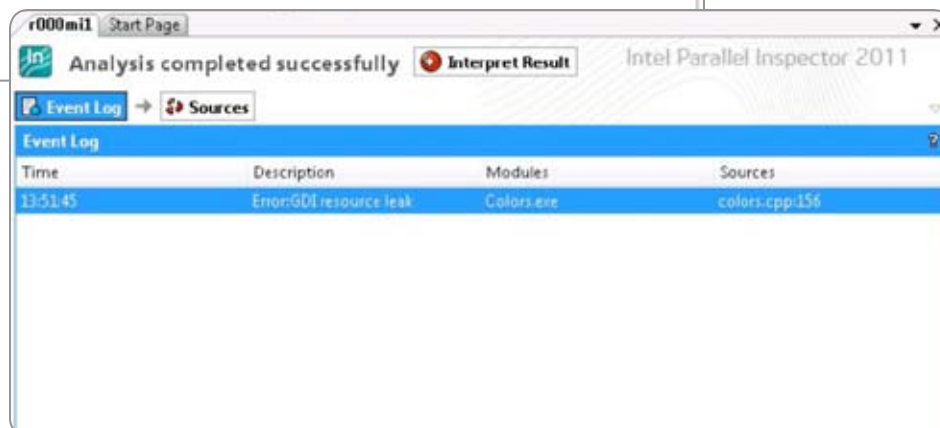


Figure 10

# Resolve Resource Leaks in Your Applications



## Interpret the results

Click on the **Interpret Result** button. This will take you to the **Overview** view.

This output, as seen in **Figure 11**, 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 **Problem Sets** 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 12**

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

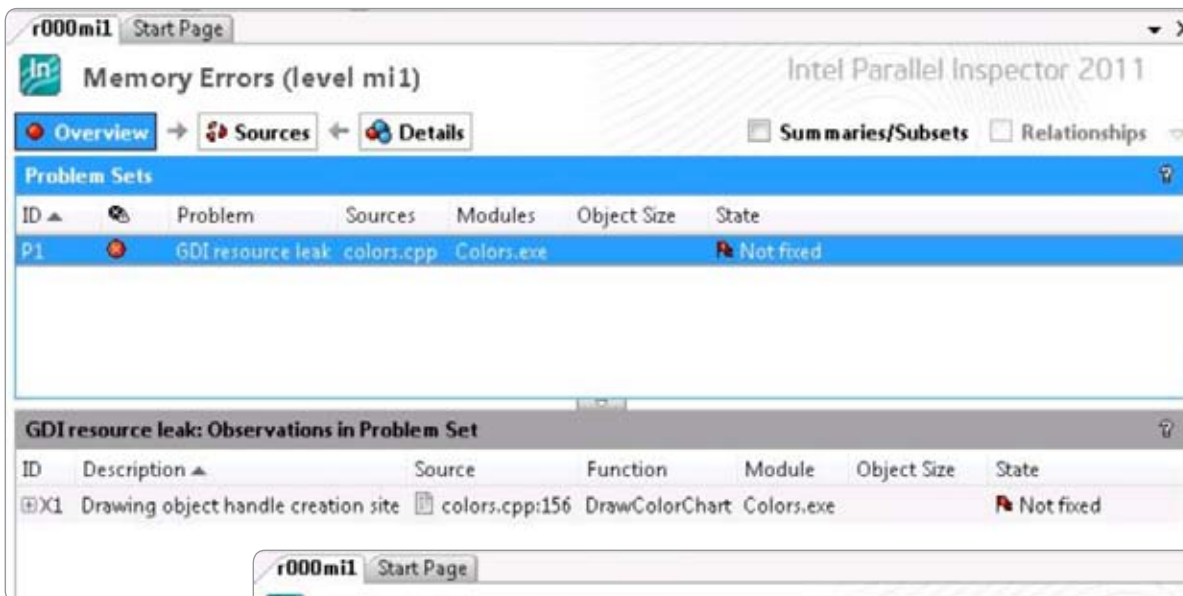


Figure 11

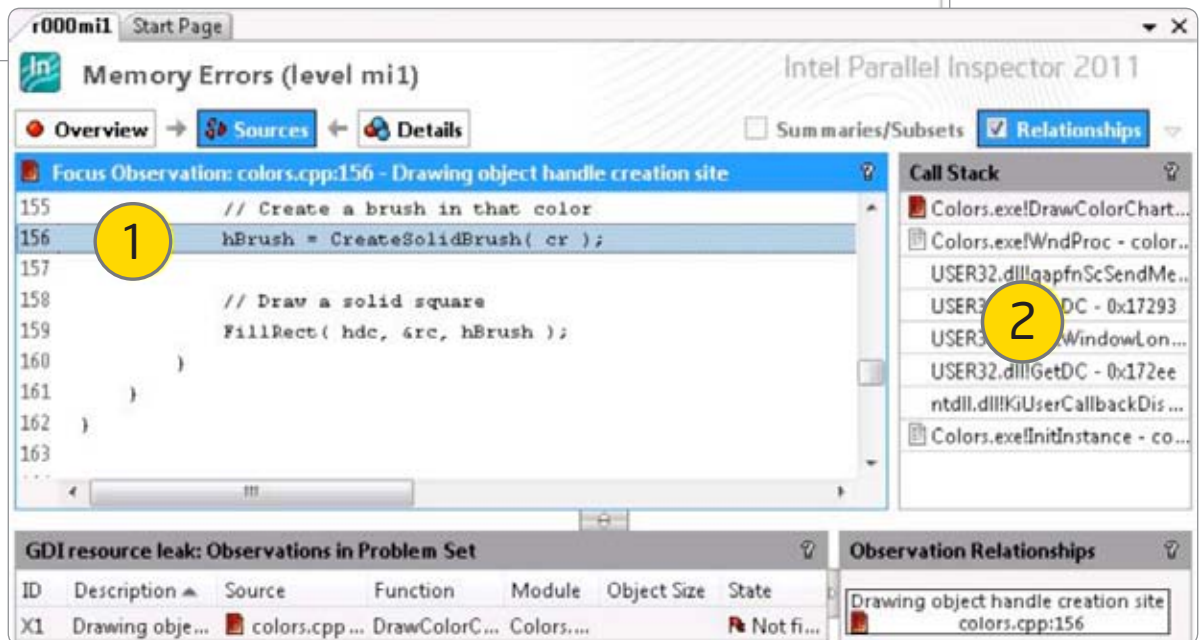
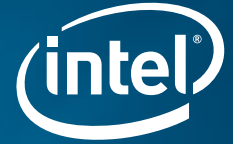


Figure 12

# Resolve Resource Leaks in Your Applications



For the purpose of reporting resource leaks, Intel Parallel Inspector 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. **Figure 12**

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 problem in the lower-left pane and select Explain Problem and Intel Parallel Inspector will give you a description of the resource leak, including the correct function to call to release the resource. **Figure 14**

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.

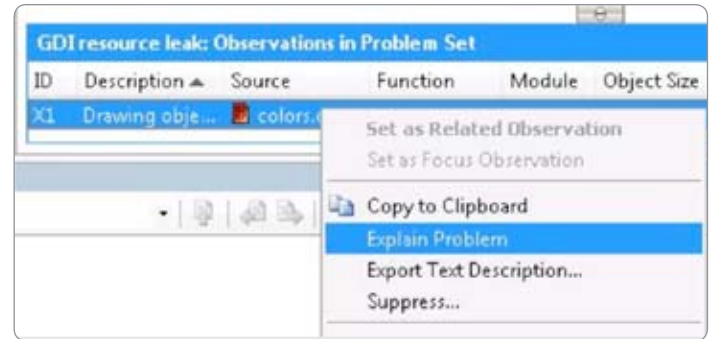


Figure 13

## 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 Parallel Inspector 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 Parallel Inspector on your application can help you find and fix resource leak errors early in the development lifecycle.

# Resolve Resource Leaks in Your Applications



## The Path to Parallelism

We are here to help developers write correct, high-performing code that will take advantage of both today's and tomorrow's processing power. Learn more about parallelism and Intel Parallel Studio from our experts.

### Related links

- [Intel® Software Network Forums](#)
- [Intel® Software Products Knowledge Base](#)
- [Intel® Software Network Blogs](#)
- [Intel® Parallel Studio Website](#)
- [Intel® Threading Building Blocks Website](#)
- [Parallelism blogs, papers, and videos](#)
- [Free, On-Demand Programming](#)

### Check out additional evaluation guides:

- [Optimize for performance](#)
- [Eliminate memory errors](#)
- [Prepare serial apps for multicore](#)
- [Model parallelization of serial apps](#)
- [Apply robust parallelism](#)
- [Simplify the path to parallelism](#)

# Resolve Resource Leaks in Your Applications



## Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options.” Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101