Getting Started Tutorial: Analyzing Memory Errors

Intel® Inspector XE 2011 for Linux® OS

C++ Sample Application Code

Document Number: 326594-001

World Wide Web: http://developer.intel.com

Legal Information
Contents

Legal Information........................................................................................................5
Overview..................................................................................................................7

Chapter 1: Navigation Quick Start

Chapter 2: Analyzing Memory Errors
  Build Application and Create New Project.........................................................13
  Configure Analysis..............................................................................................17
  Run Analysis........................................................................................................19
  Choose Problem Set...........................................................................................20
  Interpret Result Data..........................................................................................21
  Resolve Issue......................................................................................................23
  Resolve Next Issue............................................................................................24
  Rebuild and Rerun Analysis................................................................................26

Chapter 3: Summary

Chapter 4: Key Terms
Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/


*Other names and brands may be claimed as the property of others.

Microsoft, Windows, Visual Studio, Visual C++, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.

Copyright (C) 2010-2011, Intel Corporation. All rights reserved.
### Overview

Discover how to find and fix memory errors using the Intel(R) Inspector XE and the `tachyon_insp_xe` C++ sample application.

| About This Tutorial | This tutorial demonstrates an end-to-end workflow you can ultimately apply to your own applications:  
|                     | • From building an application to produce an optimal inspection result  
|                     | • To inspecting an application to find memory errors  
|                     | • To editing application code to fix the memory errors  
|                     | • To rebuilding and reinspecting the application |

| Estimated Duration  | 10-15 minutes. |

| Learning Objectives | After you complete this tutorial, you should be able to:  
|                    | • List, in order, the steps to find and fix memory errors using the Intel Inspector XE.  
|                    | • Define key Intel Inspector XE terms, such as analysis, result, problem set, problem, and code location.  
|                    | • Identify compiler/linker options that produce the most accurate, complete analysis results.  
|                    | • Explain how data set size impacts application execution time and analysis speed.  
|                    | • Run memory error analyses.  
|                    | • Influence analysis scope and running time.  
|                    | • Access help for the Intel Inspector XE command-line interface.  
|                    | • Navigate among windows in the Intel Inspector XE results.  
|                    | • Display a prioritized to-do list for fixing errors.  
|                    | • Access help for fixing specific errors.  
|                    | • Access source code to fix errors. |

| More Resources | The concepts and procedures in this tutorial apply regardless of programming language; however, a similar tutorial using a sample application in another programming language may be available at http://software.intel.com/en-us/articles/intel-software-product-tutorials/. This site also offers tutorials for all the Intel(R) Parallel Studio XE products and a printable version (PDF) of tutorials.  
|                | In addition, you can find more resources at http://software.intel.com/en-us/articles/intel-parallel-studio-xe/. |
Intel(R) Inspector XE is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows* and Linux* operating systems. You can also use the Intel Inspector XE to visualize and manage static security analysis results created by Intel(R) compilers in various suite products.

**Intel Inspector XE Access**

1. In a terminal session, type the following `source` command to set up your environment: `source <install-dir>/inspxe-vars.sh.`

   **NOTE** The default installation directory is `/opt/intel/inspector_xe_2011/`.

2. Type `inspxe-gui`.

**Intel Inspector XE GUI**
The menu, toolbar, and **Project Navigator** offer different ways to perform many of the same functions.

Use the menu to create, configure, and open projects; create, import, open, and compare results; set various options; and open the Intel Inspector XE *Getting Started Tutorials* and *Help*.

Use the toolbar to open the Intel Inspector XE *Getting Started Tutorials*; create, configure, and open projects; create, open, and compare results; and open the **Project Navigator**.

Use the **Project Navigator**:
- **Tree** to see a hierarchical view of your projects and results based on the directory where the opened project resides.
- **Context menus** to perform functions available from the menu and toolbar plus delete or rename a selected project or result, close all opened results, and copy various directory paths to the system clipboard.

Use result tabs to view and manage result data.

**Intel Inspector XE Result Tabs**

1. Use result tab names to distinguish among results.
2. Click buttons on the navigation toolbar to change window views.
3. Use window panes to view and manage result data.

4. Click buttons to display help pages that describe how to use window panes.

5. Drag window pane borders to resize window panes.

6. Click controls to show/hide window panes.

7. Use title bars to identify window panes.

8. Data column headers - Drag to reposition the data column; drag the left or right border to resize the data column; click to sort results in ascending or descending order by column data.

9. Right-click data in window panes to display context menus that provide access to key capabilities.
There are many ways to take advantage of the power and flexibility of the Intel(R) Inspector XE. The following workflow, which shows how to find and fix memory errors in serial or parallel programs, is one way to help maximize your productivity as quickly as possible.

---

**Step 1: Prepare for analysis**
Build an application to inspect for memory errors and create a new project.

**Step 2: Find errors**
- Configure a memory error analysis.
- Run the memory error analysis on the application.

**Step 3: Fix errors**
- Choose a problem set in the analysis result.
- Interpret the result data.
- Resolve the issue.
- Resolve the next issue.

**Step 4: Check your work**
Rebuild the application and rerun the memory error analysis.

---

**Build Application and Create New Project**

To create an application the Intel Inspector XE can inspect for memory errors:
- Get software tools.
• Verify optimal compiler/linker options.
• Verify optimal data set size.
• Build the application.
• Verify the application runs outside the Intel Inspector XE.
• Open the Intel Inspector XE GUI.
• Create a new project.

Get Software Tools
tachyon_insp_xe

You need the following tools to try tutorial steps yourself using the tachyon_insp_xe sample application:

• Intel Inspector XE installation package (.tgz file and license information)
• .tgz file extraction utility, such as tar
• Supported compiler (see Release Notes for more information)
• Editor

Acquire Intel Inspector XE

If you do not already have access to the Intel Inspector XE, you can download an evaluation copy from http://software.intel.com/en-us/articles/intel-software-evaluation-center/.

Install and Set Up Intel Inspector XE Sample Applications

1. Copy the tachyon_insp_xe.tgz file from the samples/<locale>/C++/ directory to a writable directory or share on your system. The default installation path is /opt/intel/inspector_xe_2011/.
2. Extract the sample from the .tgz file to create the tachyon_insp_xe directory.
3. Ensure you have set the EDITOR or VISUAL environment variable to your text editor.

• Samples are non-deterministic. Your screens may vary from the screen captures shown throughout this tutorial.
• Samples are designed only to illustrate the Intel Inspector XE features; they do not represent best practices for creating code.

Verify Optimal Compiler/Linker Settings

You can use the Intel Inspector XE to analyze:

• Memory errors in debug and release modes of binaries - the Intel Inspector XE can analyze native code in native binaries and in mixed native/managed binaries.
• Threading errors in debug and release modes of binaries - the Intel Inspector XE can analyze native and managed code in native/managed/mixed binaries.

Applications compiled/linked in debug mode using the following options produce the most accurate, complete results.

<table>
<thead>
<tr>
<th>Compiler/Linker Options</th>
<th>Correct C/C++ Setting</th>
<th>Correct Fortran Setting</th>
<th>Impact If Not Set Correctly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug information</td>
<td>Enabled (-g)</td>
<td>Enabled (-debug or -g)</td>
<td>Missing file/line information</td>
</tr>
<tr>
<td>Optimization</td>
<td>Disabled (-O0)</td>
<td>Disabled (-O0)</td>
<td>Incorrect file/line information</td>
</tr>
</tbody>
</table>
**Compiler/Linker Options**

<table>
<thead>
<tr>
<th>Dynamic runtime library</th>
<th>Correct C/C++ Setting</th>
<th>Correct Fortran Setting</th>
<th>Impact If Not Set Correctly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selected (-shared-intel for Intel(R) compilers; default or -Bdynamic for GNU compilers)</td>
<td>Selected (-shared-intel)</td>
<td>False positives or missing code locations</td>
<td></td>
</tr>
</tbody>
</table>

| Basic runtime error checks | Disabled (do not use -fmudflap) | Disabled (-check: [no]bounds) | False positives |

**Verify Optimal Data Set Size**

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

For example, it takes longer to process a 1000x1000 pixel image than a 100x100 pixel image. A possible reason for the longer processing time: You may 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 may control analysis cost without sacrificing completeness by removing this kind of redundancy from your data set.

Instead of choosing large, repetitive data sets, choose small, representative data sets. Data sets with runs in the seconds time range are ideal. Create additional data sets to ensure all your code is inspected.

**Build the Application**

1. In a terminal session, change directory to the tachyon_insp_xe directory.
2. Type make.

**Verify the Application Runs Outside the Intel Inspector XE**

1. In the same terminal session, type ./tachyon.find_and_fix_memory_errors to execute the sample application.
2. Check for a display similar to the following:
Notice the application output window is empty. The cause: Memory errors. This is how the application output should look after various memory errors are resolved:

Open the Intel Inspector XE GUI

1. In another terminal session, type the following `source` command to set up your environment: `source <install-dir>/inspxe-vars.sh.

2. Type `inspxe-gui`.

Create a New Project

1. Choose `File > New > Project...` to display a dialog box similar to the following:

2. In the **Project name** field, type `memory`. Then click the **Create project** button to create a `config.inspxeproj` file in the `-/intel/inspxe/projects/memory/` directory (default location) and display a dialog box similar to the following:
3. Click the Browse... button next to the Application field and select the tachyon_insp_xe/tachyon.find_and_fix_memory_errors application. Notice the Intel Inspector XE autofills the project Working directory field for you. Then click the OK button to display a memory project is open window.

**Key Terms**

False positive

**Configure Analysis**

The Intel Inspector XE offers a range of preset memory analysis types to help you control analysis scope and cost. The analysis type with the narrowest scope minimizes the load on the system and the time and resources required to perform the analysis; however, it detects the narrowest set of errors and provides minimal details. The analysis type with the widest scope maximizes the load on the system and the time and resources required to perform the analysis; however, it detects the widest set of errors and provides context and the maximum amount of detail for those errors.

To configure a memory error analysis, choose a memory analysis type.
Choose Memory Error Analysis Type

1. To display an Analysis Type window similar to the following: Choose File > New > Analysis...

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

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.

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.
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.

2. After you finish experimenting, choose the **Detect Memory Problems** analysis type.

**Key Terms**

*Analysis*

**Run Analysis**

To find memory errors that may need fixing, run a memory error analysis.

**Run Memory Error Analysis**

Click the **Start** button on the **Analysis Type** window and the Intel Inspector XE:

- Executes the `tachyon.find_and_fix_memory_errors` application.
- Identifies memory errors that may need handling.
- Collects the result in a directory in the `intel/inspxe/projects/memory/` directory.
- Finalizes the result (convert symbol information into filenames and line numbers, performs duplicate elimination, and forms problem sets).

During analysis, the Intel Inspector XE displays a **Collection Log** window similar to the following:
The result name appears in the tab. Here, the name of the result (and the name of the result directory in the ~/intel/inspxe/projects/memory/ directory) is r000mi2, where

- \( r \) = constant
- \( 000 \) = next available number
- \( mi \) = memory error analysis type
- \( 2 \) = preset analysis type of medium scope

**NOTE** Intel Inspector XE also offers a pointer to the result in the Project Navigator.

The Collection Log pane shows analysis milestones.

Notice you can start to manage results before analysis (collection and finalization) is complete by clicking the Summary button; however, this tutorial does not cover handling issues before analysis is complete.

**NOTE** This tutorial explains how to run an analysis from the Intel Inspector XE graphical user interface (GUI). You can also use the Intel Inspector XE command-line interface (inspxe-cl command) to run an analysis.

The Summary window automatically displays after analysis completes successfully.

**Key Terms**

- Analysis
- Collection
- Finalization

**Choose Problem Set**

To start exploring a detected memory error:

- Understand window panes.
- Choose a problem set.

**Understand Summary Window Panes**
Think of the **Summary** window as the starting point for managing result data. It groups code locations into problem sets and then prioritizes the problem sets by severity and size.

Think of the **Problems** pane as a *to-do* list. Start at the top and work your way down.

The **Code Locations** pane shows all the code locations in all the problems in the selected problem set. By default, the Intel Inspector XE selects the first problem set for you.

### Choose a Problem Set

Double-click the data row for the **Mismatched allocation/deallocation** problem set to display the **Sources** window, which provides more visibility into the cause of the error.

### Key Terms
- Code location
- Problem
- Problem set
- Result

### Interpret Result Data

To determine the cause of the detected memory error:

- Interpret window panes and icons.
Interpret Sources Window Panes and Icons

Like the pane on the Summary window, the Code Locations pane shows all the code locations in the single problem in the Mismatched allocation/deallocation problem set. The Allocation site code location represents the location and associated call stack from which the memory block was allocated. The Mismatched deallocation site code location represents the location and associated call stack attempting the deallocation.

The Related Code Location pane shows the source code in the find_and_fix_memory_errors.cpp source file surrounding the Allocation site code location. Notice the code contains a new allocator. Also notice the icon in the pane title matches the icon on the Allocation site code location data row in the Code Locations pane. The source code corresponding to the Allocation site code location is highlighted.

The Focus Code Location pane shows the source code in the find_and_fix_memory_errors.cpp source file surrounding the Mismatched deallocation site code location. Notice the code contains a free() deallocator instead of a delete deallocator. Also notice the icon in the pane title matches the icon on the Mismatched deallocation site code location data row in the Code Locations pane. The source code corresponding to the Mismatched deallocation site code location is highlighted.
<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image" alt="Icon" /></td>
<td>This code location is the focus code location. Intel Inspector XE chose it for you when you double-clicked the <strong>Mismatched allocation/deallocation</strong> problem set on the <strong>Summary</strong> window. Its source code is currently displayed in the <strong>Focus Code Location</strong> pane.</td>
</tr>
<tr>
<td><img src="Image" alt="Icon" /></td>
<td>This code location is related to the focus code location. Its source code is currently displayed in the <strong>Related Code Location</strong> pane.</td>
</tr>
</tbody>
</table>

#### Access More Information on Interpreting and Resolving Problems

1. Right-click any code location in the **Code Locations** pane.
2. Choose **Explain Problem** to display the Intel Inspector XE Help information for the **Uninitialized memory access** problem type.

#### Key Terms
- Code location
- Problem
- Problem set
- Related code location

#### Resolve Issue

To fix the detected memory error:
- Access an editor directly from the Intel Inspector XE.
- Change the source code.

#### Access Editor

Double-click the highlighted code in the **Focus Code Location** pane to open the **find_and_fix_memory_errors.cpp** source file in an editor:

```c++
for (int y = r.begin(); y != r.end(); ++y) {
    drawing_area * drawing = new drawing_area(startx, starty, 1);
    for (int x = startx; x < stopx; x++) {
        color_t c = render_one_pixel(x, y, local_mbox, serial, startx, stopx, starty, stopy);
        drawing->put_pixel(c);
    }
    delete drawing; //Memory Error: use delete instead of free
    if(!video->next_frame()) return;
    //free(local_mbox);
}
void * thread_trace(thr_parms * parms)
<xe/src/find_and_fix_memory_errors/find_and_fix_memory_errors.cpp" 212L, 7041C
```
Change the Source Code

1. Comment `free(drawing);` and uncomment `//delete drawing;`:
2. Save your edits and return to the Sources window.

**NOTE** The Sources window data is unchanged because it is a snapshot of the source code at the time of analysis.

3. Click the Summary button to display the Summary window.

**Key Terms**

Code location

**Resolve Next Issue**

To fix another detected memory error:

- Choose another problem set.
- Interpret the result data.
- Fix the memory error.

**Choose Another Problem Set**

In the Problems pane on the Summary window, double-click the data row for a Memory leak problem set to display the Sources window:
Interpret Result Data

A **Memory leak** problem occurs when a block of memory is allocated and never released.

The **Allocation site** code location represents the location from which the memory block was allocated.

Scroll the source code to line 180:

Notice the `free()` call that releases the memory allocated by the `malloc()` call is commented out. This is the cause of the **Memory leak** error.

**Fix the Memory Error**

1. Double-click near line 180 in the **Focus Code Location** pane to open the `find_and_fix_memory_errors.cpp` source file in your editor:
2. Uncomment `//free(local_mbox);`.

3. Save your edits and return to the **Sources** window.

**Key Terms**
- Code location
- Problem
- Problem set

**Rebuild and Rerun Analysis**

To check if your edits fixed the memory errors:
- Rebuild the application with your edited source code.
- Rerun the analysis.

**Rebuild the Application**

In another terminal session, type `make` in the `tachyon_insp_xe/` directory.

**Rerun the Analysis**

To run another analysis of the same analysis type: In the Intel Inspector XE GUI, choose **File > New > Memory Error Analysis / Detect Memory Problems**.

The **Summary** window automatically displays after analysis (both collection and finalization) completes successfully:
Notice the Intel Inspector XE:

- Created a new result tab.
- No longer detects the **Mismatched allocation/deallocation** and **Memory leak** problems.

**Key Terms**

**Analysis**
## Summary

This tutorial demonstrated an end-to-end workflow you can ultimately apply to your own applications.

<table>
<thead>
<tr>
<th>Step</th>
<th>Tutorial Recap</th>
<th>Key Tutorial Take-aways</th>
</tr>
</thead>
</table>
| 1. Prepare for analysis     | You built and ensured the application runs on your system outside the Intel Inspector XE, and created a project to hold analysis results. | • Applications compiled/linked in debug mode using the following options produce the most accurate, complete results: `-g`, `-O0`, `-shared-intel` for Intel(R) compilers, or default or `-Bdynamic` for GNU compilers, and no `-fmadflap`.  
• Use small, representative data sets to control analysis cost without sacrificing completeness. Data sets with runs in the seconds time range are ideal. Create additional data sets to ensure all your code is inspected. |
| 2. Find errors              | You chose an analysis type and ran an analysis. During analysis, the Intel Inspector XE:  
• Ran the application, identified errors that may need handling, and collected a result.  
• Added a pointer to the result in the Project Navigator (standalone GUI). | • Intel Inspector XE offers preset analysis types to help you control analysis scope and cost. Widening analysis scope maximizes the load on the system, and the time and resources required to perform the analysis.  
• Run error analyses from the File menu, toolbar, or command line using the `inspxe-cl` command. |
| 3. Fix errors               | You explored detected problems, interpreted the result data, accessed an editor directly from the Intel Inspector XE, and changed source code. | • A code location is a fact the Intel Inspector XE observes at a source code location. A problem is a small group of closely related code locations that indicate an error in the target. A problem set is a larger group of more loosely related code locations that could share a common solution.  
• Think of the Problems pane on the Summary window as a to-do list: Start at the top and work your way down.  
• Double-click a code location or problem set on the Summary window to navigate to the Sources window. Click the Summary button on the Sources window to return to the Summary window.  
• Right-click a code location or problem set to display a context menu, then choose Explain Problem to access more information on interpreting and resolving the problem. |
<table>
<thead>
<tr>
<th>Step</th>
<th>Tutorial Recap</th>
<th>Key Tutorial Take-aways</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. Check your work</td>
<td>You recompiled, relinked, and reinspected the application.</td>
<td>• Double-click a code location on the Sources window to open an editor.</td>
</tr>
</tbody>
</table>

**Next step:** Prepare your own application(s) for analysis. Then use the Intel Inspector XE to find and fix errors.
The following terms are used throughout this tutorial.

**analysis**: A process during which the Intel Inspector XE performs collection and finalization.

**code location**: A fact the Intel Inspector XE observes at a source code location, such as a *write code location*. Sometimes called an *observation*. A focus code location is a source code location with relationships you choose to explore. A related code location is a source code location with a relationship to a focus code location and possibly other code locations.

**collection**: A process during which the Intel Inspector XE executes an application, identifies issues that may need handling, and collects those issues in a result.

**false positive**: A reported error that is not an error.

**finalization**: A process during which the Intel Inspector XE uses debug information from binary files to convert symbol information into filenames and line numbers, performs duplicate elimination, and forms problem sets.

**problem**: A small group of closely related code locations that indicate an error in an application, such as a *data race* problem.

**problem set**: A larger group of more loosely related code locations that could share a common solution, such as a problem set resulting from deallocating an object too early during program execution. You can view problem sets only after analysis is complete.

**project**: A compiled application, collection of configurable attributes for the compiled application, and a container for results and suppression rules.

**result**: A collection of issues that may need handling.

**target**: An application the Intel Inspector XE inspects for errors.