



# Getting Started Tutorial: Analyzing Memory Errors

Intel® Inspector XE 2013 for Linux\* OS

Fortran Sample Application Code

Document Number: 327649-002US

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

Legal Information



# Contents

<b>Legal Information.....</b>	<b>5</b>
<b>Overview.....</b>	<b>7</b>

## **Chapter 1: Navigation Quick Start**

## **Chapter 2: Analyzing Memory Errors**

Build Application and Create New Project.....	13
Configure Analysis.....	16
Run Analysis.....	18
Choose Problem.....	20
Interpret Result Data.....	21
Resolve Issue.....	23
Investigate Problem Using Interactive Debugging.....	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.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

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/](http://www.intel.com/products/processor_number/)

BlueMoon, BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, E-GOLD, Flexpipe, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel CoFluent, Intel Core, Intel Inside, Intel Insider, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel vPro, Intel Xeon Phi, Intel XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Pentium, Pentium Inside, Puma, skool, the skool logo, SMARTi, Sound Mark, Stay With It, The Creators Project, The Journey Inside, Thunderbolt, Ultrabook, vPro Inside, VTune, Xeon, Xeon Inside, X-GOLD, XMM, X-PMU and XPOSYS are trademarks of Intel Corporation in the U.S. and/or other countries.

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

Copyright © 2009-2012, Intel Corporation. All rights reserved.



# Overview

---



Discover how to find and fix memory errors using the Intel® Inspector XE and the `nqueens_fortran` Fortran sample application.

<b>About This Tutorial</b>	<p>This tutorial demonstrates an end-to-end workflow you can ultimately apply to your own applications:</p> <ol style="list-style-type: none"><li>1. Build an application to produce an optimal inspection result.</li><li>2. Inspect an application to find memory errors.</li><li>3. Edit application code to fix the memory errors.</li><li>4. Investigate memory errors using interactive debugging.</li><li>5. Rebuild and reinspect the application.</li></ol>
<b>Estimated Duration</b>	10-15 minutes.
<b>Learning Objectives</b>	<p>After you complete this tutorial, you should be able to:</p> <ul style="list-style-type: none"><li>• List the steps to find and fix memory errors using the Intel Inspector XE.</li><li>• Define key Intel Inspector XE terms.</li><li>• Identify compiler/linker options that produce the most accurate and complete analysis results.</li><li>• Run memory error analyses.</li><li>• Influence analysis scope and running time.</li><li>• Navigate among windows in the Intel Inspector XE results.</li><li>• Display a prioritized <i>to-do</i> list for fixing errors.</li><li>• Access help for fixing specific errors.</li><li>• Access source code to fix errors.</li><li>• Launch an interactive debugging session to investigate problems more deeply.</li></ul>
<b>More Resources</b>	<p>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 <a href="http://software.intel.com/en-us/articles/intel-software-product-tutorials/">http://software.intel.com/en-us/articles/intel-software-product-tutorials/</a>. This site also offers tutorials for other Intel® products and a printable version (PDF) of tutorials.</p> <p>In addition, you can find more resources at <a href="http://software.intel.com/en-us/articles/intel-parallel-studio-xe/">http://software.intel.com/en-us/articles/intel-parallel-studio-xe/</a>.</p>





# Navigation Quick Start



Intel® 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 analysis results created by Intel® compilers in various suite products.

## Intel Inspector XE Access

- Do one of the following:
  - Type one of the following `source` commands to set up your environment:
    - `source <install-dir>/inspxe-vars.sh`
    - `source <install-dir>/inspxe-vars.csh`
  - Add `<install-dir>/bin32` or `<install-dir>/bin64` to your path.



**NOTE** The default installation directory is `/opt/intel/inspector_xe_2013/`.

- Type `inspxe-gui`.

## Intel Inspector XE GUI

The screenshot displays the Intel Inspector XE 2013 GUI. The main window is titled "Detect Deadlocks and Data Races". The "Problems" table lists detected issues:

ID	Problem	Sources	Modules	State
P1	Data race	nqueens_threading.f90	threading_issues.exe	Not fixed
B	Data race	nqueens_threading.f...	threading_issues.exe	Not fixed
	Data race	nqueens_threading.f...	threading_issues.exe	Not fixed
	Data race	nqueens_threading.f...	threading_issues.exe	Not fixed
	Data race	nqueens_threading.f...	threading_issues.exe	Not fixed
P2	Data race	nqueens_threading.f90	threading_issues.exe	Not fixed
	Data race	nqueens_threading.f...	threading_issues.exe	Not fixed
	Data race	nqueens_threading.f...	threading_issues.exe	Not fixed

The "Filters" panel on the right shows the following counts:

- Severity: Error (2 item(s))
- Problem: Data race (2 item(s))
- Source: nqueens\_threading.f90 (2 item(s))
- Module: threading\_issues.exe (2 item(s))
- State: Not fixed (2 item(s))
- Suppressed: 0

The "Code Locations" pane at the bottom shows the source code for the selected problem:

```

Description    Source          Function          Module
Read           nqueens_threading.f90  NQUEENS_ip_SETQUE...  threading_issues.e...
133  ! vertical attacks
134  if (lcl_queens(i) == col) r
135  !if (queens(i) == col) retu
136  ! diagonal attacks
137  if (abs(lcl_queens(i)-col)

Write          nqueens_threading.f90  NQUEENS_ip_SETQUE...  threading_issues.e...
141  ! column is ok, set the queen
142  lcl_queens(row) = col
143  !queens(row) = col
144
145  if (row == size) then
  
```

The "Timeline" pane on the right shows the execution flow, including "main (2912) (2912)" and "OMP Worker Thread #1 (3884) (3884)".

**A1 - A3**

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

**A1**

Use the menu to create projects and dynamic analysis results, import result archive files and results from other Intel® error-detection products, open projects and results, compare results, configure projects, set various options, and access the *Getting Started* page and *Help*.

**A2**

Use the toolbar to open the *Getting Started* page; create, configure, and open projects; create dynamic analysis results; and open and compare results.

**A3**

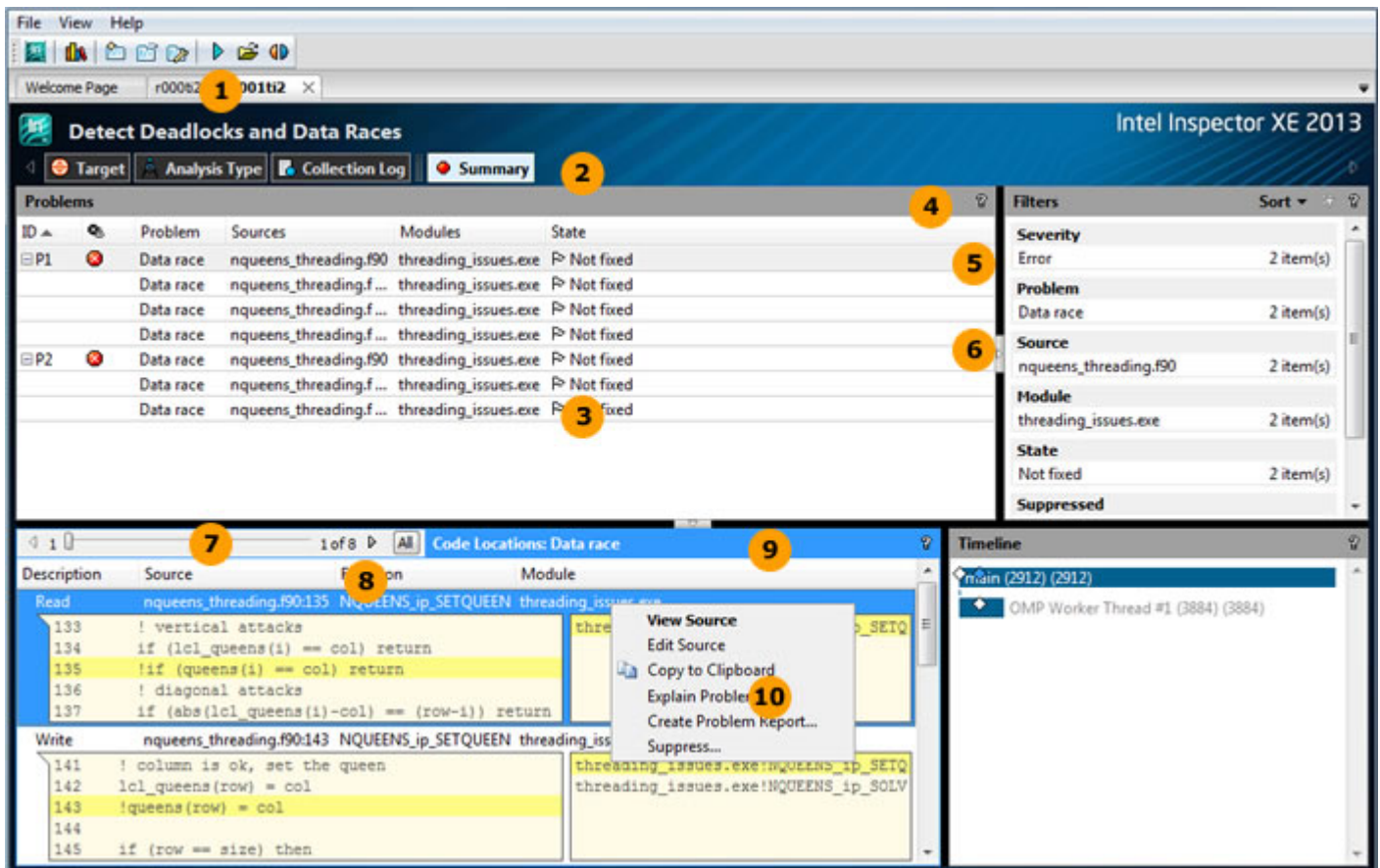
Use the **Project Navigator**:

- **Tree** to see a hierarchical view of your projects and results based on the directory where the opened project resides.
- **Context menus** (right-click to open) 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.

**B**



Use result tabs to view and manage result data.

## Intel Inspector XE Result Tabs



**1**

Use result tab names to distinguish among results.

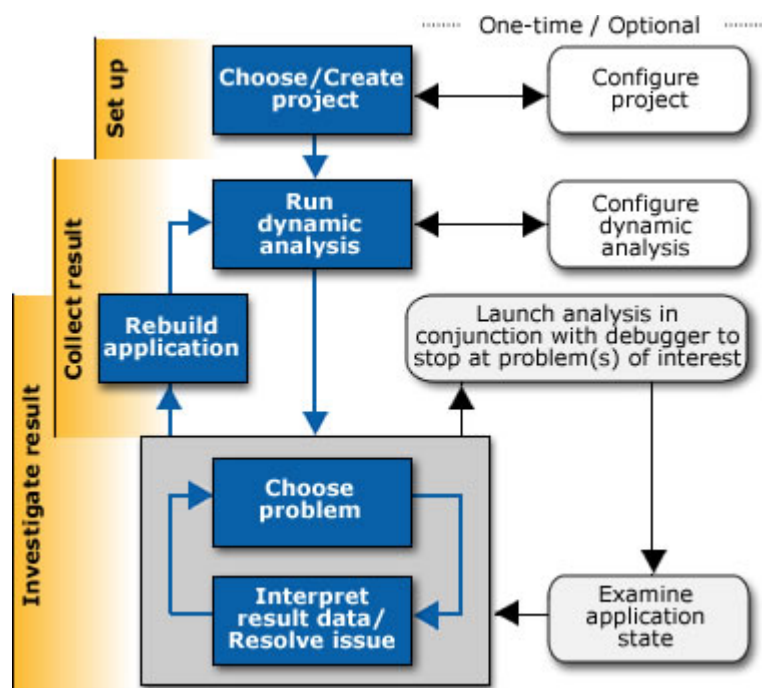
- 2 Click buttons on the navigation toolbar to change window views.
- 3 Use window panes to view and manage result data.
- 4 Click  buttons to display help pages that describe how to use window panes.
- 5 Drag window pane borders to resize window panes.
- 6 Click  controls to show/hide window panes.
- 7 Click window pane data controls to adjust result data within the pane (and possibly in adjacent 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 Use title bars to identify window panes.
- 10 Right-click data in window panes to display context menus that provide access to key capabilities.



# Analyzing Memory Errors



There are many ways to take advantage of the power and flexibility of the Intel 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.



<b>Step 1: Unpack and set up sample for analysis</b>	Build an application to inspect for memory errors and create a new project.
<b>Step 2: Collect result</b>	<ul style="list-style-type: none"> <li>Configure a memory error analysis.</li> <li>Run the memory error analysis on the application.</li> </ul>
<b>Step 3: Investigate result</b>	<ul style="list-style-type: none"> <li>Choose a problem set in the analysis result.</li> <li>Interpret the result data.</li> <li>Resolve the issue.</li> <li>Investigate another problem using interactive debugging.</li> </ul>
<b>Step 4: Check your work</b>	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 settings.
- 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

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

- Intel Inspector XE installation package (.tgz file and license information)
- .tgz file extraction utility
- Supported compiler (see *Release Notes* for more information)
- Editor

### Acquire the 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 the Intel Inspector XE Sample Applications

1. Copy the `nqueens_fortran.tgz` file from the `<install-dir>/samples/<locale>/Fortran/` directory to a writable directory or share on your system. The default `<install-dir>` is `/opt/intel/inspector_xe_2013/`.
2. Extract the sample from the .tgz file to create the `nqueens_fortran` 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 on both debug and release modes of C++ and Fortran binaries containing native code; however, applications compiled/linked in debug mode using the following settings produce the most accurate and complete analysis results.

Compiler/Linker Property	Correct C/C++ Setting	Correct Fortran Setting	Impact If Not Set Correctly
Debug information	Enabled (-g)	Enabled (-debug or -g)	Missing file/line information
Optimization	Disabled (-O0)	Disabled (-O0)	Incorrect file/line information
Dynamic runtime library	Selected (-shared-intel for Intel(R) compilers; default or -Bdynamic for GNU compilers)	Selected (-shared-intel)	False positives or missing code locations

Compiler/Linker Property	Correct C/C++ Setting	Correct Fortran Setting	Impact If Not Set Correctly
Basic runtime error checks	Disabled (do not use -fmudflap)	None (-check:none)	False positives

## Build the Application

1. In a terminal session, change directory to the `nqueens_fortran/` directory.
2. Type `make nqueens_memory_debug` to build a debug version of the `nqueens_memory` sample application.

## Verify the Application Runs Outside the Intel Inspector XE

1. In the same terminal session, type `./nqueens_memory_debug` to execute the sample application.
2. Check for non-deterministic application output similar to the following:

```

Usages: nqueens_memory[ debug] boardSize
Using default size of 10
Starting nqueens solver for size 10 with 8 thread(s)
Number of solutions: 724
correct result!
Calculations took 116 ms.

```

## Open the Intel Inspector XE GUI

1. In another terminal session, do one of the following:
  - Type one of the following `source` commands to set up your environment:
    - `source <install-dir>/inspxe-vars.sh`
    - `source <install-dir>/inspxe-vars.csh`
  - Add `<install-dir>/bin32` or `<install-dir>/bin64` to your path.

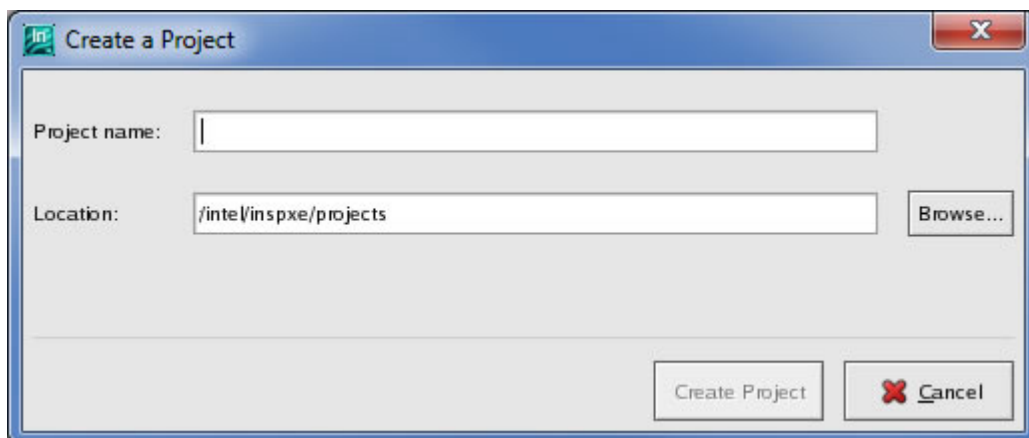


**NOTE** The default installation directory is `/opt/intel/inspector_xe_2013/`.

2. Type `inspxe-gui`.

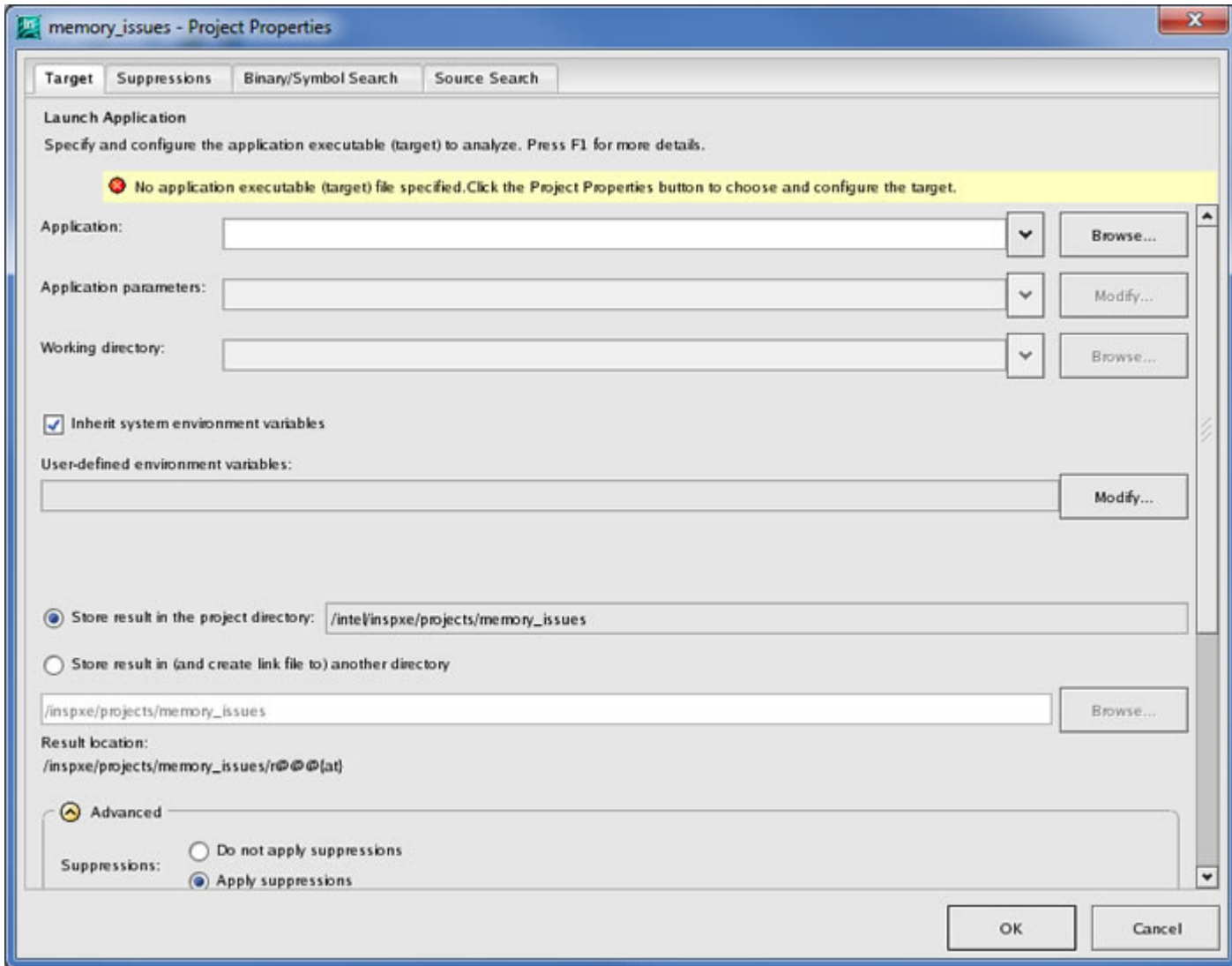
## Create a New Project

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





2. In the **Project name** field, type `memory_issues`. Then click the **Create project** button to create a `config.inspxproj` file in the `~/intel/inspxe/projects/memory_issues/` 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 `nqueens_fortran/nqueens_memory_debug` application. Notice the Intel Inspector XE autofills the project **Working directory** field for you. Then click the **OK** button to return to the Welcome page, where the name of the opened project displays in the title bar and in the **Project Navigator** pane. (If necessary, choose **View > Project Navigator** to display the **Project Navigator**.)

### Key Terms

False positive

## Configure Analysis



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

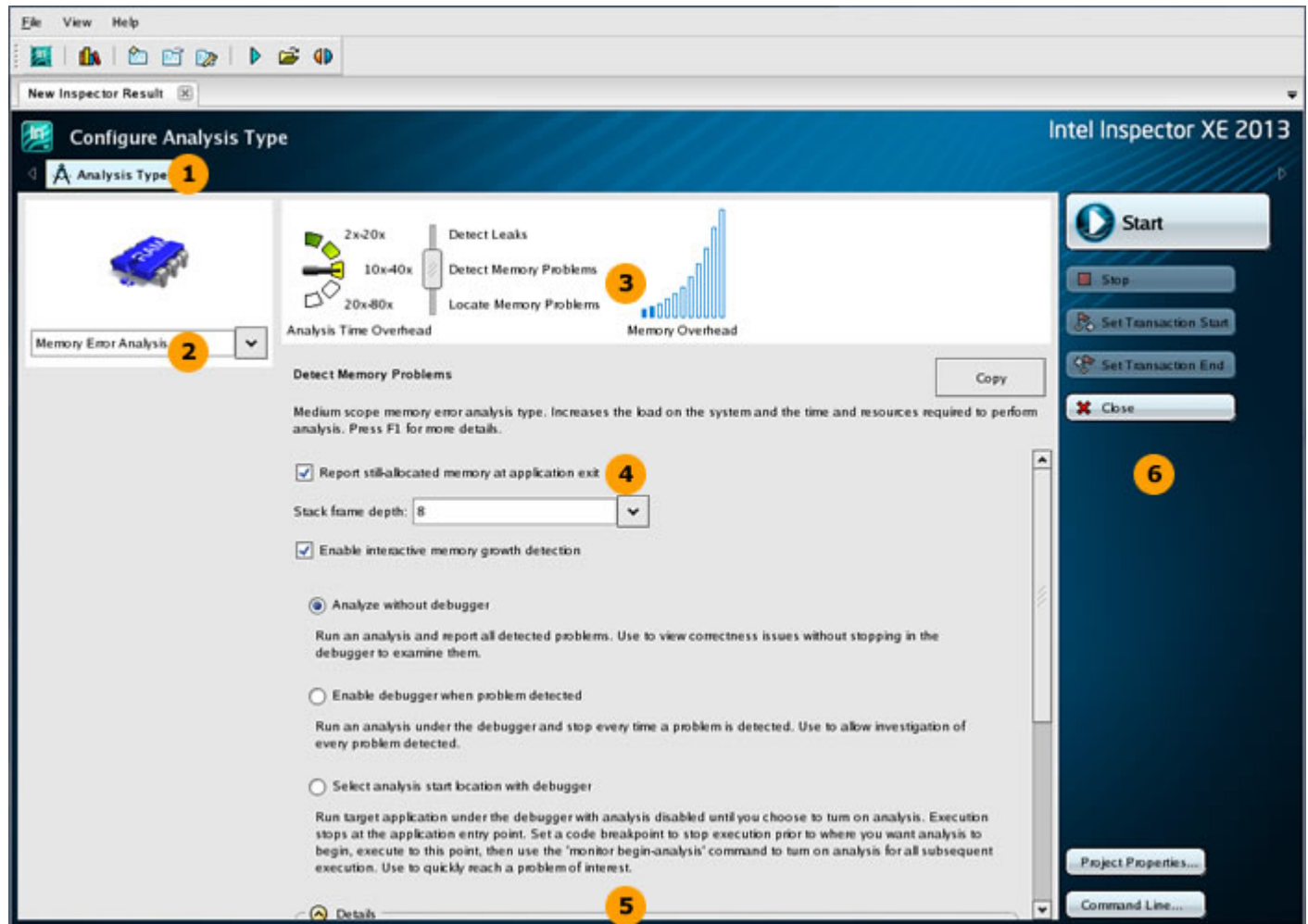


minimal details. The analysis type with the widest scope maximizes the load on the system and the time and resources required to perform the analysis; however, it detects the widest set of errors and provides context and the maximum amount of detail for those errors.

To configure a memory error analysis, choose a memory analysis type.

## Choose a Memory Error Analysis Type

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



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

Use the Analysis Type drop-down list to choose an analysis type category: **Memory Error Analysis**, **Threading Error Analysis**, or **Custom Analysis Types**.

Choose the **Memory Error Analysis** type category.



**NOTE** This tutorial covers memory error analysis types, which you can use to search for resource leak, incorrect `memcpy` call, invalid deallocation, invalid memory access, invalid partial memory access, memory growth, memory leak, memory not deallocated, mismatched allocation/deallocation, missing allocation, uninitialized memory access, and uninitialized partial memory access errors. Use threading error analysis types to search for data race, deadlock, lock hierarchy violation, and cross-thread stack access errors.

---

3

Use the configuration slider to choose a preset analysis type and the corresponding gauges to assess the *cost* of that choice. The preset analysis type at the top of the slider has the narrowest scope; the preset analysis type at the bottom has the widest.

Choose the **Detect Memory Problems** preset analysis type.

The **Analysis Time Overhead** gauge helps you quickly estimate the time it may take to collect a result using this preset analysis type. Time is expressed in relation to normal application execution time. For example, 2x - 20x is 2 to 20 times longer than normal application execution time. If normal application execution time is 5 seconds, estimated collection time is 10 to 100 seconds.

The **Memory Overhead** gauge helps you quickly estimate the memory the Intel Inspector XE may consume to detect errors using this preset analysis type. Memory is expressed in blue-filled bars.



**NOTE** The gauge does not show memory used by the running application during analysis.

---

4

Use the checkbox(es), radio button(s), and drop-down list(s) to fine-tune some, but not all, preset analysis type settings.



**NOTE** If you need to fine-tune more analysis type settings, you can choose another analysis type or create a custom analysis type.

---

5

Use the **Details** region to view all current settings for this analysis type.

6

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.

If you experimented with various settings, make sure you choose the **Detect Memory Problems** preset analysis type (and ensure your settings match the image above) before proceeding.

### Key Terms

Analysis

## Run Analysis

---



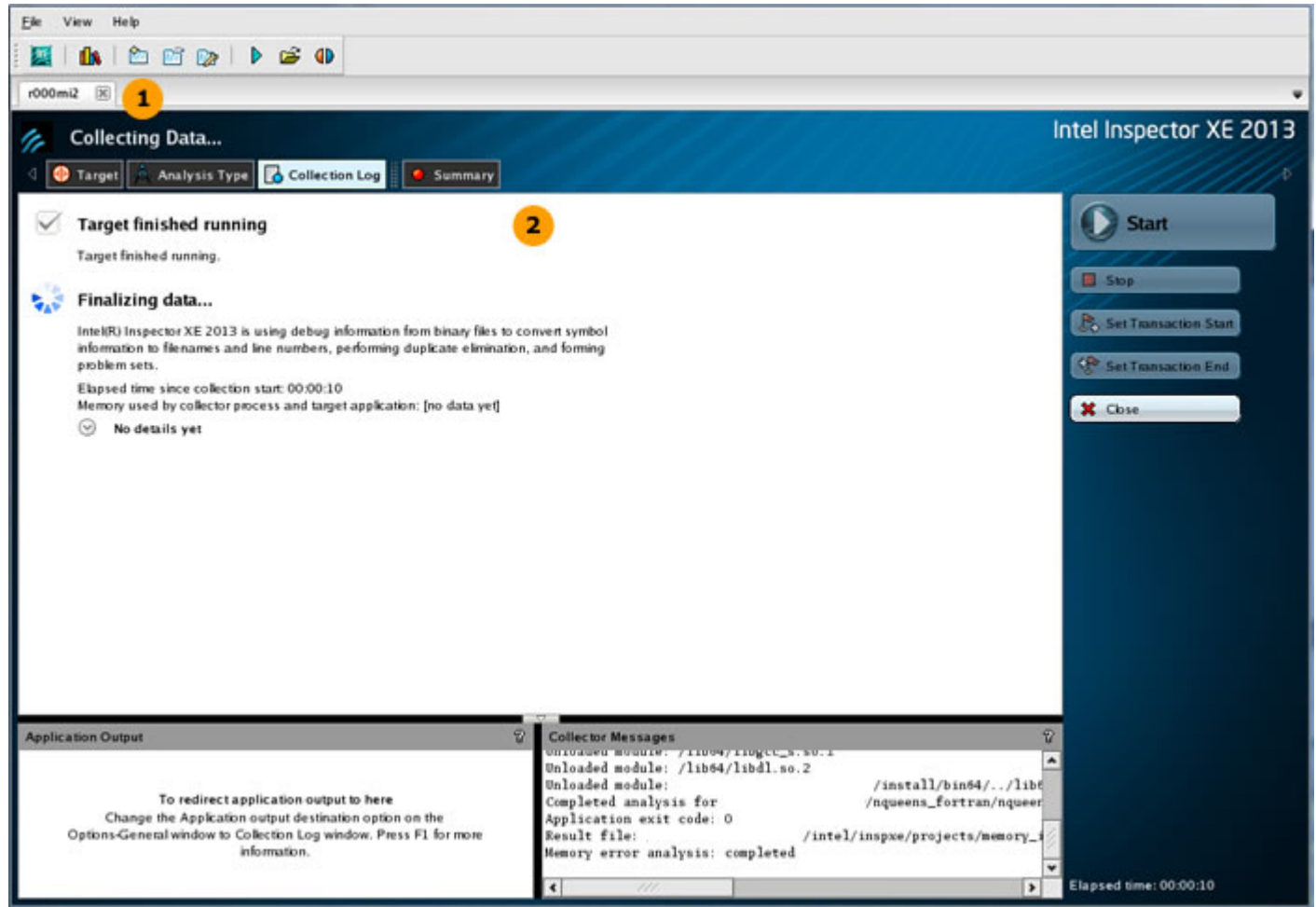
To find memory errors that may need fixing, run a memory error analysis.

## Run Memory Error Analysis

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

- Executes the `nqueens_memory_debug` application.
- Identifies memory errors that may need handling.
- Collects the result in a directory in the `intel/inspxe/projects/memory_issues/` directory.
- Finalizes the result.

During analysis, the Intel Inspector XE displays a **Collection Log** window similar to the following:



1

The result name appears in the tab. Here, the name of the result 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**.

2

The **Collection Log** pane shows analysis progress and 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 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



To start exploring a detected memory error:

- Understand Summary window panes.
- Choose a problem.


### Understand Summary Window Panes

The screenshot shows the Intel Inspector XE 2013 Summary window. The interface includes a menu bar (File, View, Help), a toolbar, and a target selection bar (r000mi2). The main area is divided into several panes:

- Summary Tab:** The active tab, showing a list of detected problems.
 

ID	Problem	Sources	Modules	Object Size	State
P1	Uninitialized memory access	nqueens_memory.890	nqueens_memory_debug		New
P2	Memory not deallocated	nqueens_memory.890	nqueens_memory_debug		New
P3	Memory leak	nqueens_memory.890	nqueens_memory_debug	64	New
- Filters:** A sidebar on the right showing filters for Severity (Error, Warning), Problem (Memory leak, Memory not deallocated, Uninitialized memory access), Source (nqueens\_memory.890), Module (nqueens\_memory\_debug), State (New), and Sort (3 items).
- Code Locations:** A pane at the bottom showing the source code for the selected problem (Uninitialized memory access). It displays a list of code locations with their descriptions, sources, functions, modules, object sizes, and offsets.
 

Description	Source	Function	Module	Object Size	Offset
Read	nqueens_memory.890:128	setqueen	nqueens_memory_debug		
126					
127	! Make copy of queens array				
128	!cl_queens = queens				
129					
130	do i=1,row-1				
Allocation site	nqueens_memory.890:80	nqueens	nqueens_memory_debug		
78	101 format (A,I0,A,I0,A)				
79	call system_clock (time_start)				
80	allocate (queens(size))				
81	!initialize the queens array to 0 to avoid "uninitialized"				
82	!queens = 0				
- Timeline:** A pane on the right showing a timeline of events, including the start of the analysis (start Q2070) and the execution of the OMP Worker Thread #1 (Q2095).

- 1 The **Summary** window is the starting point for managing result data. It groups problems into problem sets and then prioritizes the problem sets by severity and size.
- 2 Think of the **Problems** pane as a *to-do* list. Start at the top and work your way down. Try viewing the problems in various problem sets by clicking the corresponding  icon.
- 3 The **Code Locations** pane shows the code location summary, surrounding source code snippet, call stack, and thread and timeline information for all code locations in one or all occurrences of the problem(s) highlighted in the **Problems** pane.

### Choose a Problem

When you are finished experimenting, double-click the data row for the **Memory leak** 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

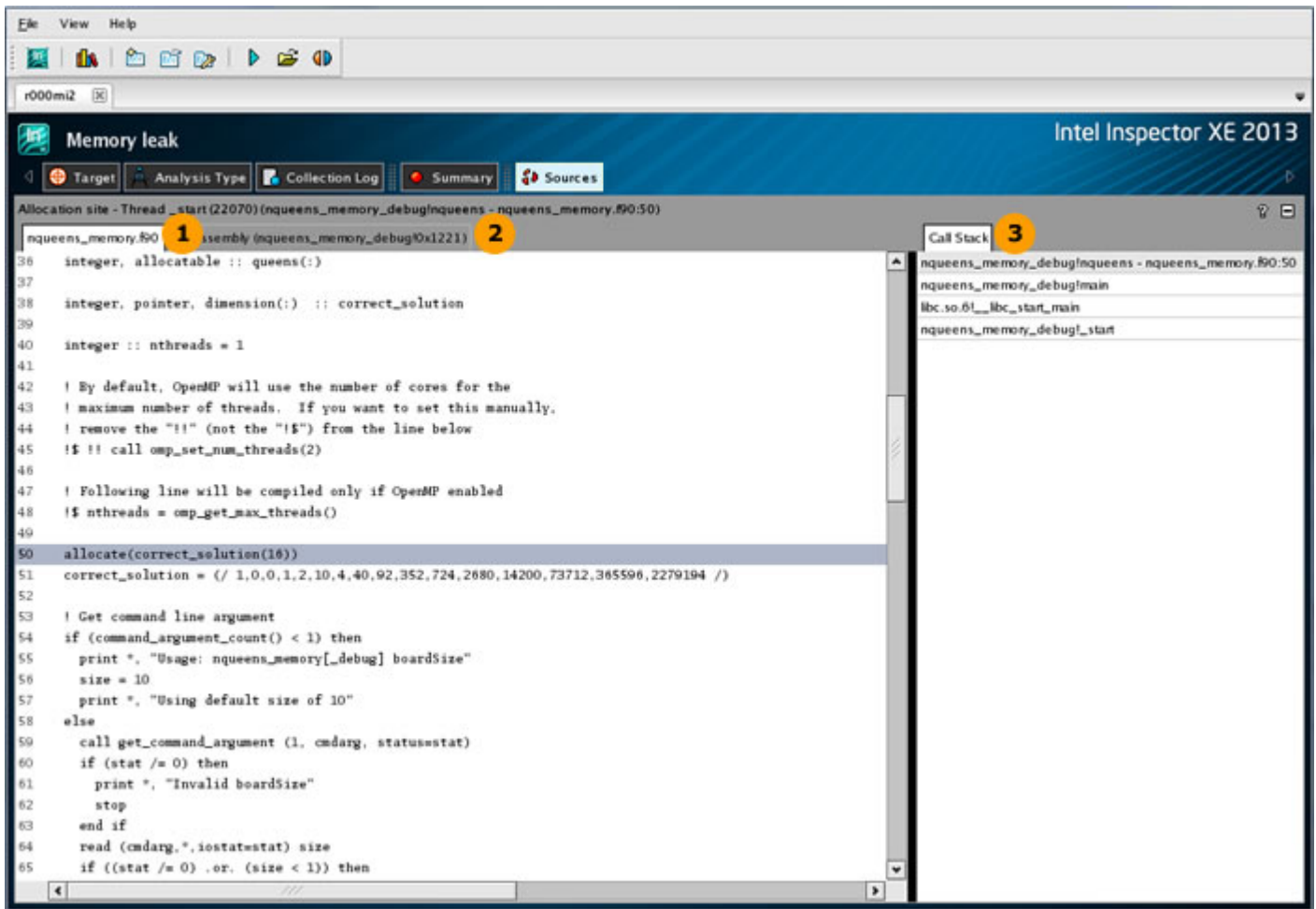
---



To determine the cause of the detected memory error:

- [Interpret Sources window pane tabs.](#)
- [Access more information on interpreting and resolving problems.](#)

### Interpret Sources Window Pane Tabs



1

The **Source** tab shows the complete source surrounding one code location in the **Memory leak** problem: **Allocation site**.



The **Allocation site** code location represents the location from which the memory block was allocated. Notice the source code corresponding to the **Allocation site** code location is highlighted.

2

The **Disassembly** tab shows low-level operations for the **Allocation site** code location in the **Memory leak** problem.

3

The **Call Stack** tab shows the complete call stack for the **Allocation site** code location in the **Memory leak** problem.

Use the / icons to expand/collapse source, disassembly, and call stack information for each code location region in a problem.

### Access More Information on Interpreting and Resolving Problems

1. Right-click anywhere in the **Source** or **Disassembly** tab.
2. Choose **Explain Problem** to display the Intel Inspector XE Help information for the **Memory leak** problem type.

### Key Terms

- Code location
- Problem



## Resolve Issue



To fix the detected memory error:

- Investigate the issue.
- Access an editor directly from the Intel Inspector XE.
- Change the source code.

### Investigate the Issue

Scroll to near line 94. The embedded commenting in the `nqueens_memory.f90` sample file reveals the cause of the **Memory leak** error: The `deallocate` call near line 95 that releases the memory is commented out.

### Access an Editor

Double-click near line 94 in the **Source** tab to open the `nqueens_memory.f90` source file in an editor:

```
call system_clock (time_end, count_rate)
print 101, "Number of solutions: ", nrOfSolutions
if (nrOfSolutions == correct_solution(size+1)) then
  print 101, "Correct Result!"
else
  print 101, "Incorrect Result!"
end if

print 101, "Calculations took ", (time_end-time_start) / (count_rate/1000), "ms."

deallocate (queens)
!deallocate the pointer to avoid a memory leak
!deallocate (correct_solution)
contains

! Routine to print the board
subroutine print (queens)
  implicit none
  integer, intent(in) :: queens(:)
  integer :: row, col

  do row=1,size
```

### Change the Source Code

1. Uncomment `!deallocate (correct_solution).`
2. Save your edits.
3. Click the result tab to 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.

4. Click the **Summary** button to display the **Summary** window.

### Key Terms

Code location

## Investigate Problem Using Interactive Debugging



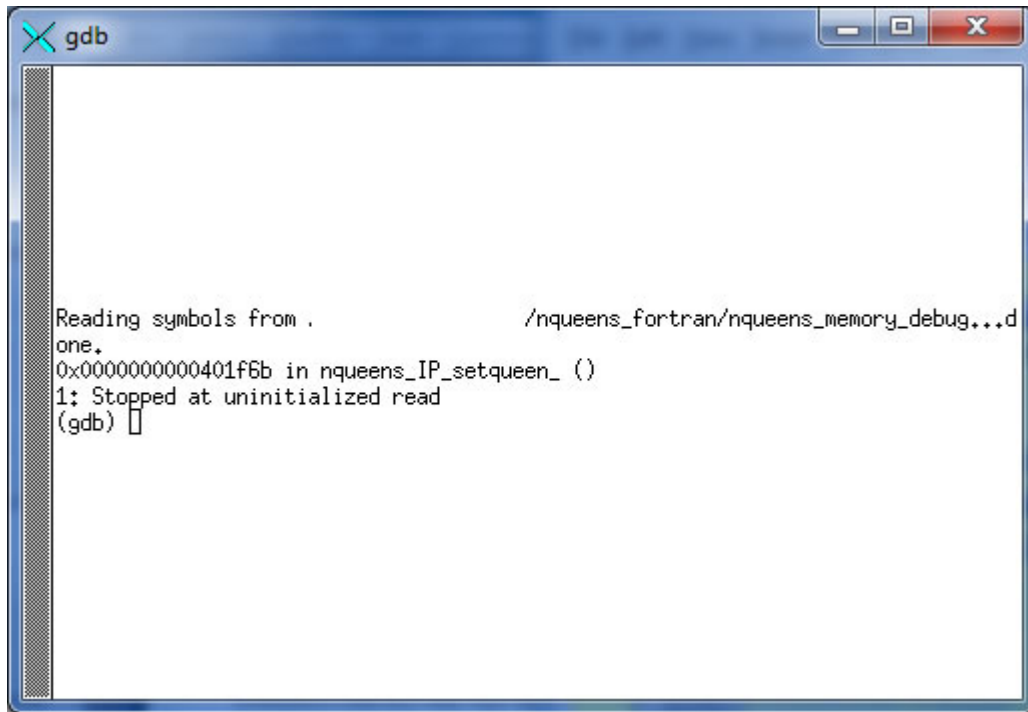
Investigate a problem more deeply with an interactive debugging session during analysis.

- Launch an interactive debugging session.
- Display extended debugging commands.
- Stop the analysis and interactive debugging session.

### Launch an Interactive Debugging Session

In the **Problems** pane on the **Summary** window, right-click the data row for the **Uninitialized memory access** problem set to display a context menu, then choose **Debug this problem** to:

- Launch a new analysis, **r001mi2**, with the same configuration settings used to create the **r000mi2** result, except the analysis is optimized to detect only the **Uninitialized memory access** problem.
- Halt application execution and open a debugging session when the Intel Inspector XE detects the first occurrence of the **Uninitialized memory access** problem.



The debugger displays problem breakpoint information in a *prob-brk-id: prob\_brk\_desc* format, where:

- *prob-brk-id* is a debugger-assigned integer value that identifies the problem breakpoint.
- *prob-brk-desc* is a short problem description.



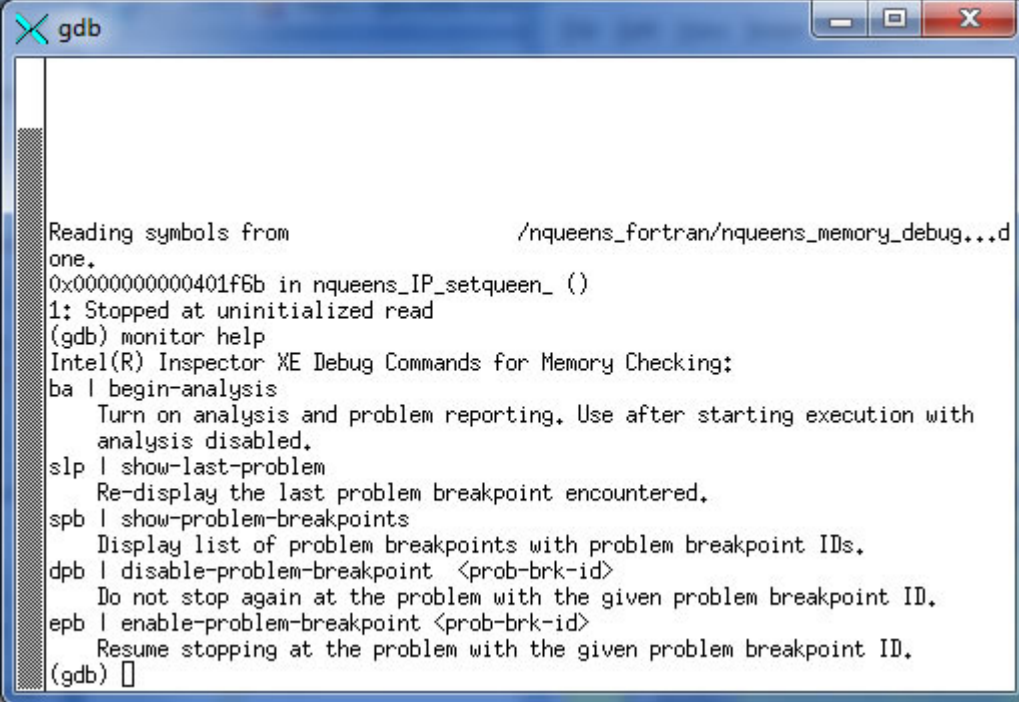
**NOTE** Do not confuse the *prob-brk-id* values assigned by the debugger with problem IDs prefaced with the letter *P* in the Intel Inspector XE result data.



## Display Extended Debugging Commands

During the interactive debugging session, use the normal debugger actions to examine memory and other state information, set code breakpoints, and continue execution. Only the use of data breakpoints is not supported.

Intel Inspector XE also offers a set of command extensions to use within an interactive debugging session during analysis. Type `monitor help` to view a list applicable to memory error analyses:



```

gdb
Reading symbols from /nqueens_fortran/nqueens_memory_debug...d
one.
0x0000000000401f6b in nqueens_IP_setqueen_ ()
1: Stopped at uninitialized read
(gdb) monitor help
Intel(R) Inspector XE Debug Commands for Memory Checking:
ba | begin-analysis
    Turn on analysis and problem reporting. Use after starting execution with
    analysis disabled.
slp | show-last-problem
    Re-display the last problem breakpoint encountered.
spb | show-problem-breakpoints
    Display list of problem breakpoints with problem breakpoint IDs.
dpb | disable-problem-breakpoint <prob-brk-id>
    Do not stop again at the problem with the given problem breakpoint ID.
epb | enable-problem-breakpoint <prob-brk-id>
    Resume stopping at the problem with the given problem breakpoint ID.
(gdb) 

```

## Stop the Analysis and Interactive Debugging Session

When you are finished experimenting in the debugger workspace, stop both the interactive debugging session and the analysis:

1. Return to the Intel Inspector XE window. Notice the new, still-running, **r001mi2** result.
2. Click the **Stop** button on the Intel Inspector XE **Command** toolbar.

When finalization is complete, the Intel Inspector XE displays a **Summary** window for the new **r001mi2** result that contains only the **Uninitialized memory access** problem.



**TIP** Intel Inspector XE automatically adjusts the debugging session analysis to return to the **Uninitialized memory access** problem more quickly; however the analysis adjustments do not correspond to individual problem types. Consequently, the Intel Inspector XE may detect and report additional problems, but it will break only for the **Uninitialized memory access** problem.

## Key Terms

- Problem
- Problem breakpoint
- Problem set

## Rebuild and Rerun Analysis



To check if your edits resolved the **Memory leak** problem:

- Rebuild the application with your edited source code.
- Rerun the analysis.

### Rebuild the Application

In another terminal session, type `make nqueens_memory_debug` in the `nqueens_fortran/` 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:

The screenshot displays the Intel Inspector XE 2013 interface. The main window is titled "Detect Memory Problems" and shows a "Summary" tab. The "Problems" table lists two issues:

ID	Problem	Sources	Modules	Object Size	State
P1	Uninitialized memory access	nqueens_memory.f90	nqueens_memory_debug		Not fixed
P2	Memory not deallocated	nqueens_memory.f90	nqueens_memory_debug		New

The "Filters" panel on the right shows the following counts:

Severity	Count
Error	1 item(s)
Warning	1 item(s)

The "Problem" filter shows 1 item(s) for "Memory not deallocated" and 1 item(s) for "Uninitialized memory access". The "Source" filter shows 2 item(s) for "nqueens\_memory.f90". The "Module" filter shows 2 item(s) for "nqueens\_memory\_debug". The "State" filter shows 1 item(s) for "New" and 1 item(s) for "Not fixed".

The bottom pane shows the "Code Locations: Uninitialized memory access" table:

Description	Source	Function	Module	Object Size	Offset
Read	nqueens_memory.f90:128	setqueen	nqueens_memory_debug		
126					
127	! Make copy of queens array				
128	!c1_queens = queens				
129					
130	do i=1,row-1				
Allocation site	nqueens_memory.f90:80	nqueens	nqueens_memory_debug		
78	101 format (A,IO,A,IO,A)				
79	call system_clock (time_start)				
80	allocate (queens(size))				
81	! initialize the queens array to 0 to avoid "uninitialized"				
82	!queens = 0				

Notice the Intel Inspector XE:

- Created a new result tab: **r002mi2**.
- No longer detects the **Memory leak** problem.

### Key Terms

Analysis





# Summary



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

Step	Tutorial Recap	Key Tutorial Take-aways
<b>1. Set up</b>	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 and linked in debug mode using the following options produce the most accurate and complete analysis results: -debug, -O0, -shared-intel, and -check:none.
<b>2. Collect result</b>	<p>You chose an analysis type and ran an analysis. During analysis, the Intel Inspector XE:</p> <ul style="list-style-type: none"> <li>• Ran the application, identified errors that may need handling, collected a result, and displayed the result in a result tab.</li> <li>• Added a pointer to the result in the <b>Project Navigator</b>.</li> </ul>	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Run error analyses from the <b>File</b> menu, toolbar, or command line using the <code>inspxe-cl</code> command.</li> </ul>
<b>3. Investigate result</b>	You explored detected problems, interpreted the result data, accessed an editor directly from the Intel Inspector XE, and changed source code. You also investigated a problem using interactive debugging.	<ul style="list-style-type: none"> <li>• Key terms: A <i>code location</i> is a fact the Intel Inspector XE observes at a source code location. A <i>problem</i> is one or more occurrences of a detected issue. A <i>problem set</i> is a group of problems with a common problem type and a shared code location that might share a common solution.</li> <li>• Think of the <b>Problems</b> pane on the <b>Summary</b> window as a <i>to-do</i> list: Start at the top and work your way down.</li> <li>• Double-click a code location or problem on the <b>Summary</b> window to navigate to the <b>Sources</b> window. Click the <b>Summary</b> button on the <b>Sources</b> window to return to the <b>Summary</b> window.</li> <li>• Right-click various places on the <b>Summary</b> or <b>Sources</b> window to display a context menu, then choose <b>Explain Problem</b> to access more information on interpreting and resolving the problem.</li> <li>• Double-click a code location on the <b>Sources</b> window to open an editor.</li> <li>• Right-click a problem, then choose <b>Debug This Problem</b> to launch an interactive debugging session.</li> </ul>
<b>4. Check your work</b>	You recompiled, relinked, and reinspected the application.	

**Next step:** Prepare your own application(s) for analysis. Then use the Intel Inspector XE to find and fix errors.

# Key Terms

---



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. Previously called an *observation*.

**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 (if requested), and forms problem sets.

**problem:** One or more occurrences of a detected issue, such as an uninitialized memory access. Multiple occurrences have the same call stack but a different thread or timestamp. You can view information for a problem as well as for each occurrence.

**problem breakpoint:** A breakpoint that halts execution when a memory or threading analysis detects a problem. In a Linux\* debugger, a problem breakpoint is indicated by display of source line information, a unique problem breakpoint ID, and a short problem description.

**problem set:** A group of problems with a common problem type and a shared code location that might 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; a collection of configurable attributes, including suppression rules and search directories; and a container for analysis results.

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

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

