Intel® Software Development Products

Getting Started Tutorial: Analyzing Threading Errors

Intel® Inspector XE 2011 for Windows* OS

Fortran Sample Application Code

Document Number: 326599-001

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

Legal Information

Contents

Legal Information

Overview

Discover how to find and fix threading errors using the Intel® Inspector XE and the `nqueens_fortran` Fortran 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 threading errors • To editing application code to fix the threading 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 threading 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 threading 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® 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/.

Navigation Quick Start

1

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

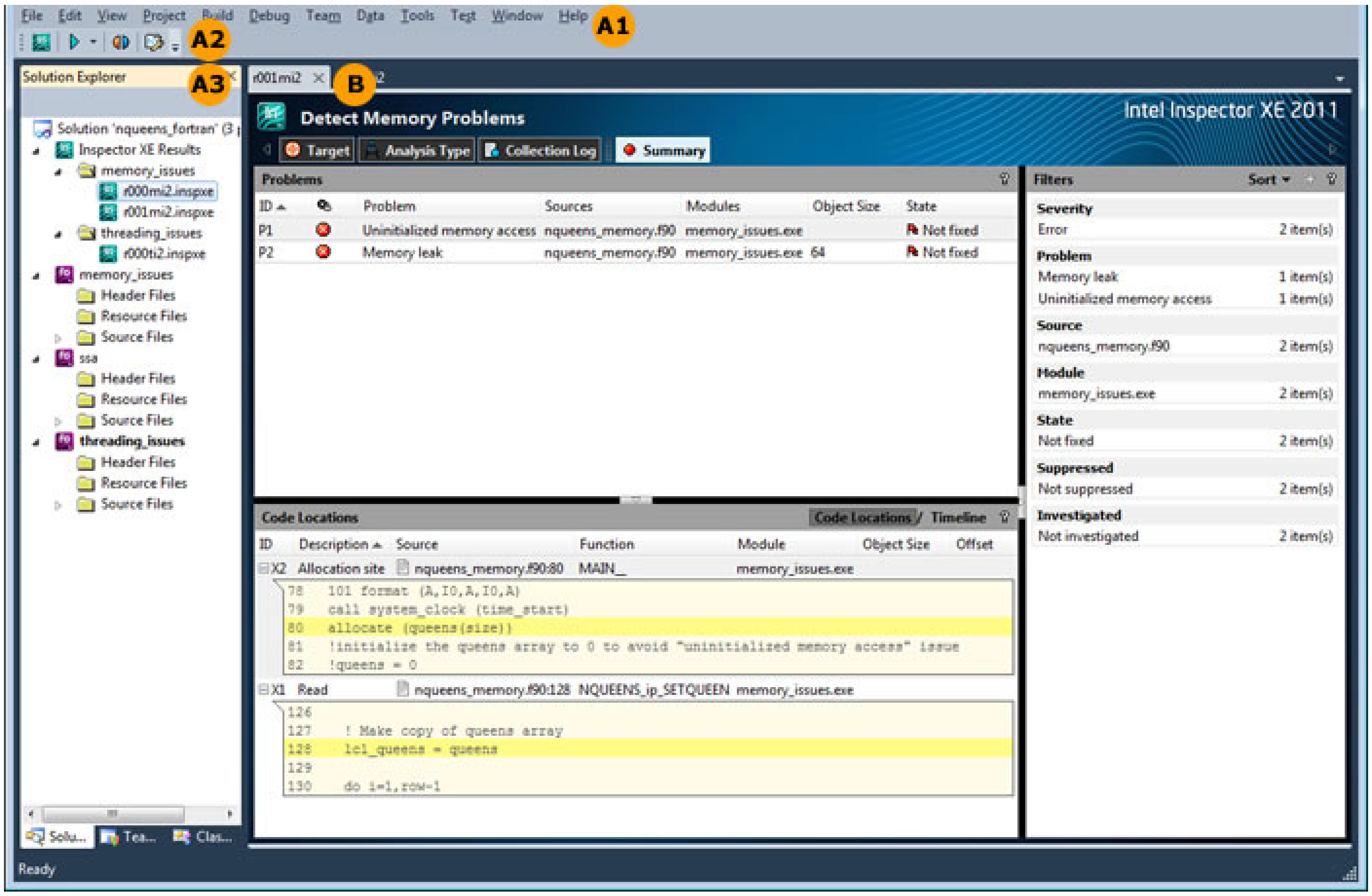
Intel Inspector XE Access

To access the Intel Inspector XE in the Visual Studio* IDE: From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2011** > **Parallel Studio XE 2011 with [VS2005 | VS2008 | VS2010]** .

To access the Standalone Intel Inspector XE GUI, do one of the following:

- From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2011** > **Intel Inspector XE 2011**.
- From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2011** > **Command Prompt** > **Parallel Studio XE with Intel Compiler** > **IA-32 Visual Studio [2005 | 2008 | 2010] mode** to set up your environment, then type inspxe-gui.

Intel Inspector XE/Visual Studio* IDE Integration



ID		Problem	Sources	Modules	Object Size	State
P1	●	Uninitialized memory access	nqueens_memory.f90	memory_issues.exe		Not fixed
P2	●	Memory leak	nqueens_memory.f90	memory_issues.exe	64	Not fixed

ID	Description	Source	Function	Module	Object Size	Offset
X2	Allocation site	nqueens_memory.f90:80	MAIN__	memory_issues.exe		

```
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 memory access" issue
82    !queens = 0
```

ID	Description	Source	Function	Module	Object Size	Offset
X1	Read	nqueens_memory.f90:128	NQUEENS_ip_SETQUEEN	memory_issues.exe		

```
126
127     ! Make copy of queens array
128     lcl_queens = queens
129
130     do i=1,row-1
```

Severity	
Error	2 item(s)
Problem	
Memory leak	1 item(s)
Uninitialized memory access	1 item(s)
Source	
nqueens_memory.f90	2 item(s)
Module	
memory_issues.exe	2 item(s)
State	
Not fixed	2 item(s)
Suppressed	
Not suppressed	2 item(s)
Investigated	
Not investigated	2 item(s)

A The menu, toolbar, and **Solution Explorer** offer different ways to perform many of the same functions.

A1 Use the **Tools > Intel Inspector XE 2011** menu to create dynamic analysis results, compare results, and import dynamic analysis results.

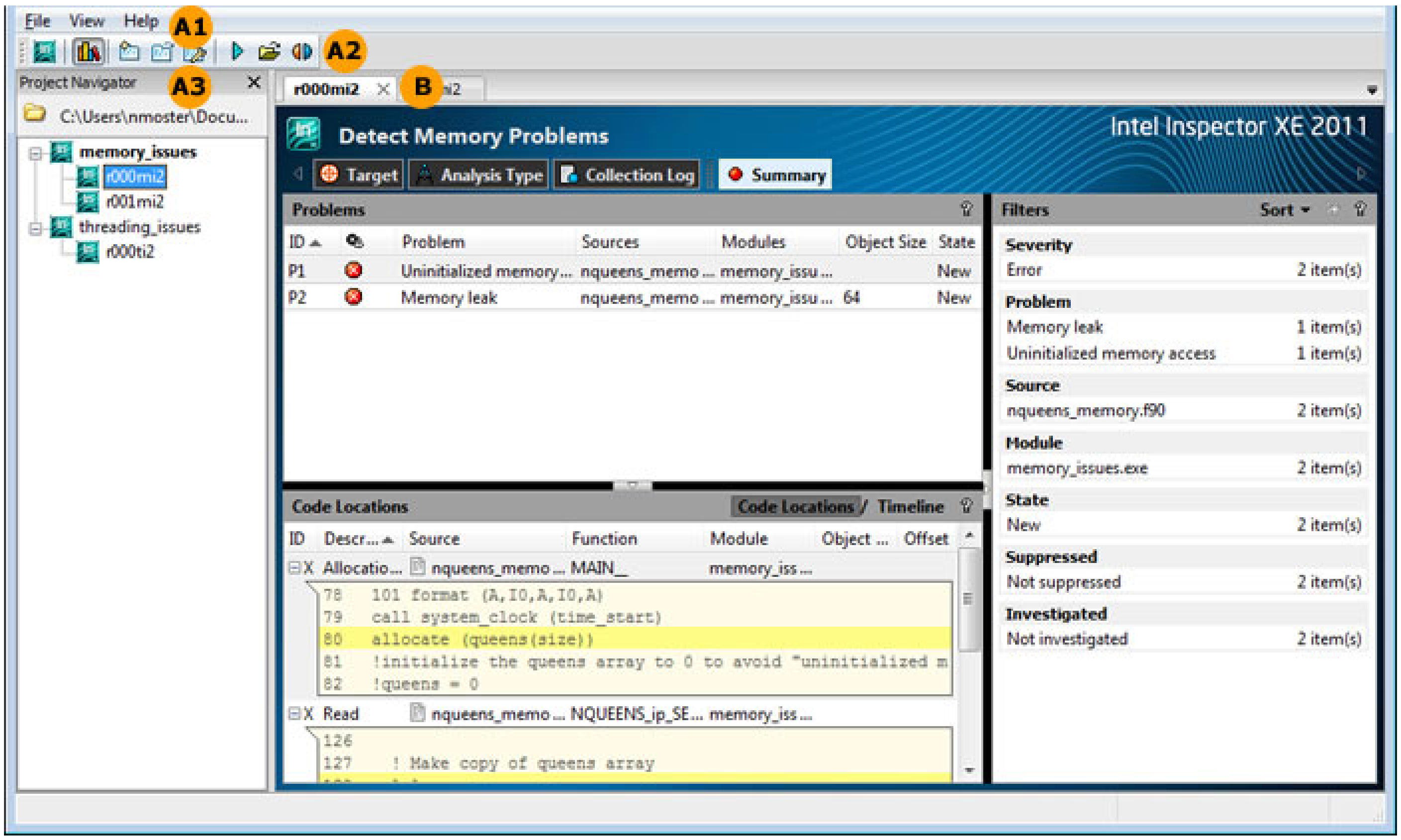
A2 Use the **Intel Inspector XE** toolbar to open the Intel Inspector XE *Getting Started Tutorials*, create dynamic analysis results, compare results, and configure projects.

A3 **Solution Explorer** context menus:

- Use the **Intel Inspector XE 2011** menu on the **Solution Explorer** project context menu to create dynamic analysis results and configure projects.
- Use the context menu on a result in the **Inspector XE Results** folder to open results, create dynamic analysis results, and manage results.

B Use the Intel Inspector XE result tabs to manage result data.

Standalone Intel Inspector XE GUI



ID		Problem	Sources	Modules	Object Size	State
P1	●	Uninitialized memory...	nqueens_memo...	memory_issu...		New
P2	●	Memory leak	nqueens_memo...	memory_issu...	64	New

Severity	
Error	2 item(s)
Problem	
Memory leak	1 item(s)
Uninitialized memory access	1 item(s)
Source	
nqueens_memory.f90	2 item(s)
Module	
memory_issues.exe	2 item(s)
State	
New	2 item(s)
Suppressed	
Not suppressed	2 item(s)
Investigated	
Not investigated	2 item(s)

ID	Descr...	Source	Function	Module	Object ...	Offset
X	Allocatio...	nqueens_memo...	MAIN__	memory_iss...		

```
101 format (A,I0,A,I0,A)
call system_clock (time_start)
allocate (queens(size))
!initialize the queens array to 0 to avoid "uninitialized m
!queens = 0
```

ID	Descr...	Source	Function	Module	Object ...	Offset
X	Read	nqueens_memo...	NQUEENS_ip_SE...	memory_iss...		

```
! Make copy of queens array
```

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

A1 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*.

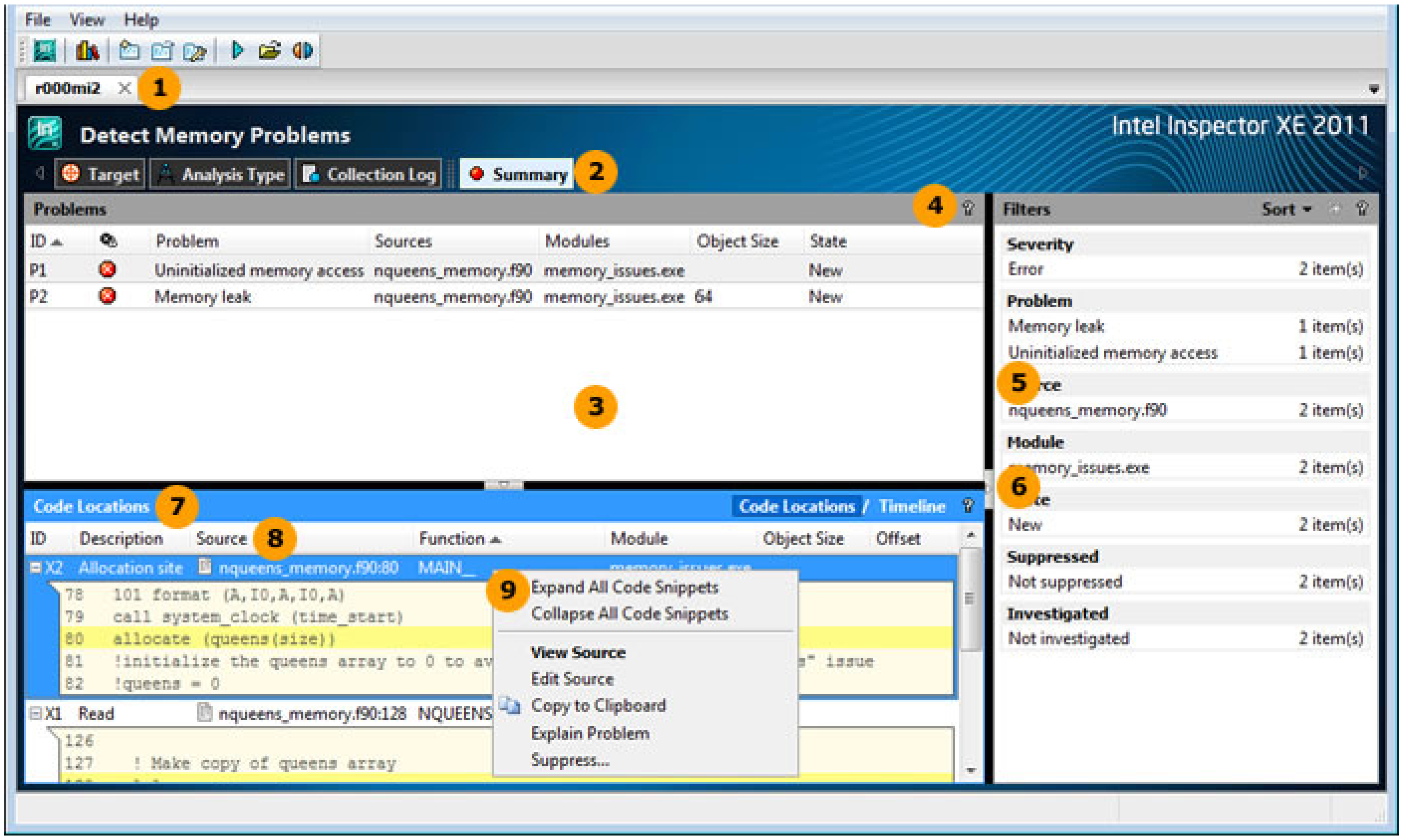
A2 Use the toolbar to open the Intel Inspector XE *Getting Started Tutorials*; create, configure, and open projects; create, open, and compare results; and open the **Project Navigator**.

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 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



ID		Problem	Sources	Modules	Object Size	State
P1	⊗	Uninitialized memory access	nqueens_memory.f90	memory_issues.exe		New
P2	⊗	Memory leak	nqueens_memory.f90	memory_issues.exe	64	New

Severity	
Error	2 item(s)
Problem	
Memory leak	1 item(s)
Uninitialized memory access	1 item(s)
...rce	
nqueens_memory.f90	2 item(s)
Module	
...mory_issues.exe	2 item(s)
...te	
New	2 item(s)
Suppressed	
Not suppressed	2 item(s)
Investigated	
Not investigated	2 item(s)

ID	Description	Source	Function	Module	Object Size	Offset
X2	Allocation site	nqueens_memory.f90:80	MAIN__	memory_issues.exe		
X1	Read	nqueens_memory.f90:128	NQUEENS			

```
101 format (A,I0,A,I0,A)
call system_clock (time_start)
allocate (queens(size))
!initialize the queens array to 0 to av...
!queens = 0
```

```

! Make copy of queens array
```

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 , , , and controls to show/hide window panes.

7 Use title bars to identify window panes.

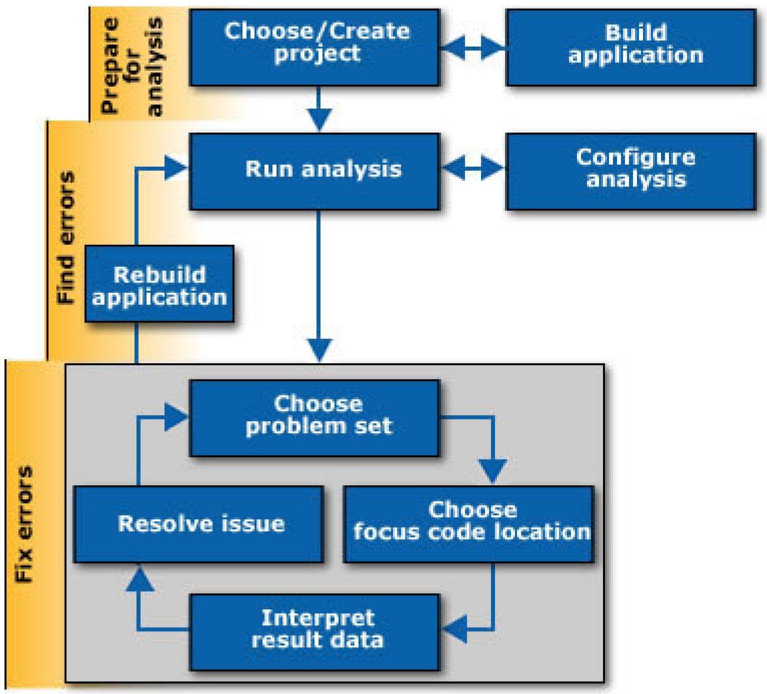
8 Data column headers - Drag to reposition the data column; drag the left or right border to resize the data column; click to sort results in ascending or descending order by column data.

9 Right-click data in window panes to display context menus that provide access to key capabilities.

Analyzing Threading Errors

2

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 threading errors in parallel programs, is one way to help maximize your productivity as quickly as possible.



Step 1: Prepare for analysis	Do one of the following: • In the Visual Studio* IDE: Choose a project, verify settings, and build an application to inspect for threading errors. • In the Standalone Intel Inspector XE GUI: Build an application to inspect for threading errors and create a new project.
Step 2: Find errors	• Configure a threading error analysis. • Run the threading error analysis on the application.
Step 3: Fix errors	• Choose a problem set and focus code location 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 threading error analysis.

Visual Studio* IDE: Choose Project and Build Application

To create an application the Intel Inspector XE can inspect for threading errors:

- Get software tools.
- Open a Visual Studio* solution.
- Set a startup project.
- Verify optimal compiler/linker options.
- Verify the application is set to build in debug mode.
- Verify optimal data set size.
- Build and test the application.

Get Software Tools

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

- Intel Inspector XE, including sample applications
- `.zip` file extraction utility
- Supported compiler (see *Release Notes* for more information)

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/.

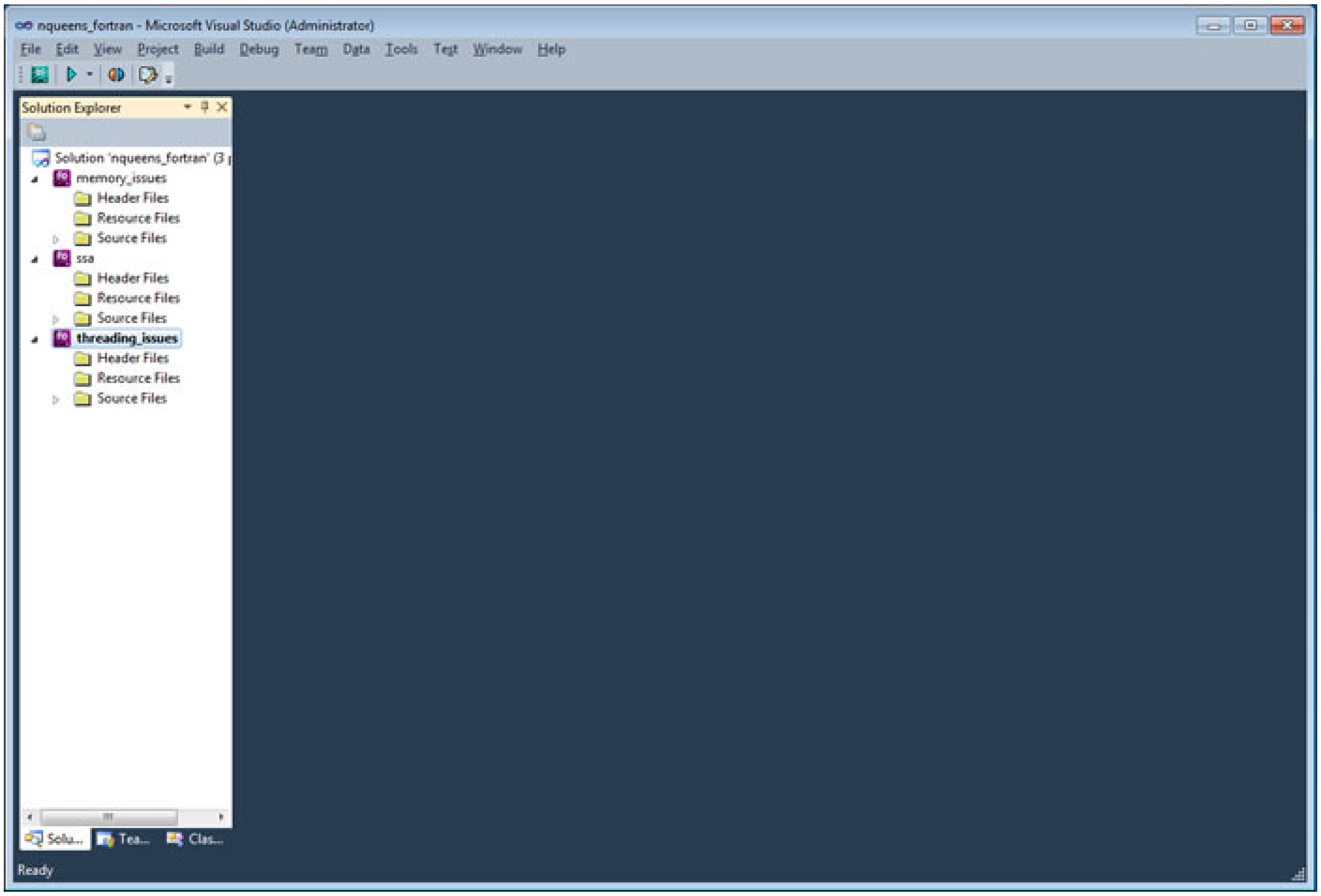
Install and Set Up Intel Inspector XE Sample Applications

1. Copy the `nqueens_fortran.zip` file from the `<install-dir>\samples\<locale>\Fortran` directory to a writable directory or share on your system. The default installation path is `C:\Program Files\Intel\Inspector XE 2011\` (on certain systems, instead of `Program Files`, the directory name is `Program Files (x86)`).
2. Extract the sample from the `.zip` file.

- 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.

Open a Visual Studio* Solution

1. Choose **File** > **Open** > **Project/Solution**.
2. In the **Open Project** dialog box, open the `nqueens_fortran\nqueens_fortran.sln` file to display the **nqueens_fortran** solution in the **Solution Explorer**:



Choose a Startup Project

1. Right-click the **threading_issues** project.
2. Choose **Set as StartUp Project**.

Verify Optimal Compiler/Linker Options

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.

Compiler/Linker Options	**Correct C/C++ Setting**	**Correct Fortran Setting**	**Impact If Not Set Correctly**
Debug information	Enabled (`/Zi` or `/ZI`)	Enabled (`/debug:full`)	Missing file/line information
Optimization	Disabled (`/Od`)	Disabled (`/Od`)	Incorrect file/line information

Compiler/Linker Options	Correct C/C++ Setting	Correct Fortran Setting	Impact If Not Set Correctly
Dynamic runtime library	Selected (`/MD` or `/MDd`)	Selected (`/libs:dll`)	False positives or missing code locations
Basic runtime error checks	Disabled (do not use `/RTC`; **Default** option in Visual Studio* IDE)	Disabled (`/check:[no]bounds`)	False positives

1. Right-click the **threading_issues** project in the **Solution Explorer**.
2. Choose **Properties** to display the **Property Pages** dialog box.
3. Verify the **Configuration** drop-down list is set to **Debug** or **Active(Debug)**.
4. In the left pane, choose **Configuration Properties** > **Fortran** > **Debugging**.
5. Verify the **Debug Information Format** is set to **Full (/debug:full)**.
6. In the left pane, choose **Configuration Properties** > **Fortran** > **Optimization**.
7. Verify the **Optimization** field is set to **Disable (/Od)**.
8. In the left pane, choose **Configuration Properties** > **Fortran** > **Libraries**.
9. Verify the **Runtime Library** field is set to **Multithread DLL (/libs:dll)**.
10. In the left pane, choose **Configuration Properties** > **Fortran** > **Run-time**.
11. Verify the **Check Array and String Bounds** field is set to **No**.
12. In the left pane, choose **Configuration Properties** > **Linker** > **Debugging**.
13. Verify the **Generate Debug Info** field is set to **Yes (/DEBUG)**.

Verify the Application is Set to Build in Debug Mode

1. Click the **Configuration Manager** button.
2. Verify the **Active solution configuration** drop-down list is set to **Debug**.
3. Click the **Close** button to close the **Configuration Manager** dialog box.
4. Click the **OK** button to close the **Property Pages** dialog box.

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 that fully create threads with minimal to moderate work per thread. *Minimal to moderate* means just enough work to demonstrate all the different behaviors a thread can perform. Data sets with runs in the seconds time range are ideal. Create additional data sets to ensure all your code is inspected.

Build and Test the Application

1. Choose **Build** > **Project Only** > **Build Only threading_issues**.
2. Choose **Debug** > **Start Without Debugging**.
3. If the Visual Studio* IDE responds any projects are out of date, click **No**.
4. Check for output similar to the following:

```
Usage: threading_issues.exe boardSize
Using default size of 10
Starting nqueens solver for size 10 with 2 thread(s)
```

```
Number of solutions: 1344
Incorrect result!
Calculations took 31 ms.
Press any key to continue...
```

Key Terms

False positive

Standalone GUI: Build Application and Create New Project

To create an application the Intel Inspector XE can inspect for threading 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 Standalone 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
- `.zip`
- Supported compiler (see *Release Notes* for more information)

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 `nqueens_fortran.zip` file from the `<install-dir>\samples\<locale>\Fortran\` directory to a writable directory or share on your system. The default installation path is `C:\Program Files \Intel\Inspector XE 2011\` (on certain systems, instead of `Program Files`, the directory name is `Program Files (x86)`).
2. Extract the sample from the `.zip` file to create the `nqueens_fortran` directory.

- 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.

Compiler/Linker Options	**Correct C/C++ Setting**	**Correct Fortran Setting**	**Impact If Not Set Correctly**
Debug information	Enabled (`/Zi` or `/ZI`)	Enabled (`/debug:full`)	Missing file/line information
Optimization	Disabled (`/Od`)	Disabled (`/Od`)	Incorrect file/line information
Dynamic runtime library	Selected (`/MD` or `/MDd`)	Selected (`/libs:dll`)	False positives or missing code locations
Basic runtime error checks	Disabled (do not use `/RTC`; **Default** option in Visual Studio* IDE)	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 that fully create threads with minimal to moderate work per thread. *Minimal to moderate* means just enough work to demonstrate all the different behaviors a thread can perform. 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. From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2011** > **Command Prompt** > **Parallel Studio XE with Intel Compiler** > **IA-32 Visual Studio [2005 | 2008 | 2010] mode** to set up your environment.
2. Change directory to the `nqueens_fortran` directory in its unzipped location.
3. If you choose **IA-32 Visual Studio 2008** or **IA-32 Visual Studio 2010 mode**, type devenv nqueens_fortran.sln to convert the `nqueens_fortran.sln` solution. When conversion is complete, close the Visual Studio* IDE.
4. Type devenv nqueens_fortran.sln /Build to build all projects in the solution.

Verify the Application Runs Outside the Intel Inspector XE

1. Change directory to `threading_issues\Debug\`.
2. Type threading_issues.exe to execute the application.
3. Check for output similar to the following:

```
Usage: threading_issues.exe boardSize
Using default size of 10
Starting nqueens solver for size 10 with 2 thread(s)
Number of solutions: 1333
Incorrect result!
Calculations took 31 ms.
```

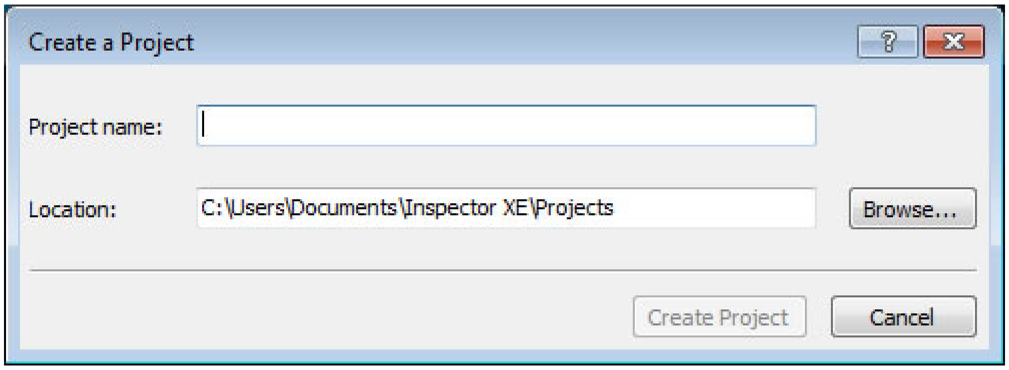
Open the Standalone Intel Inspector XE GUI

From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2011** > **Intel Inspector XE 2011**.

TIP Keep the command prompt window open.

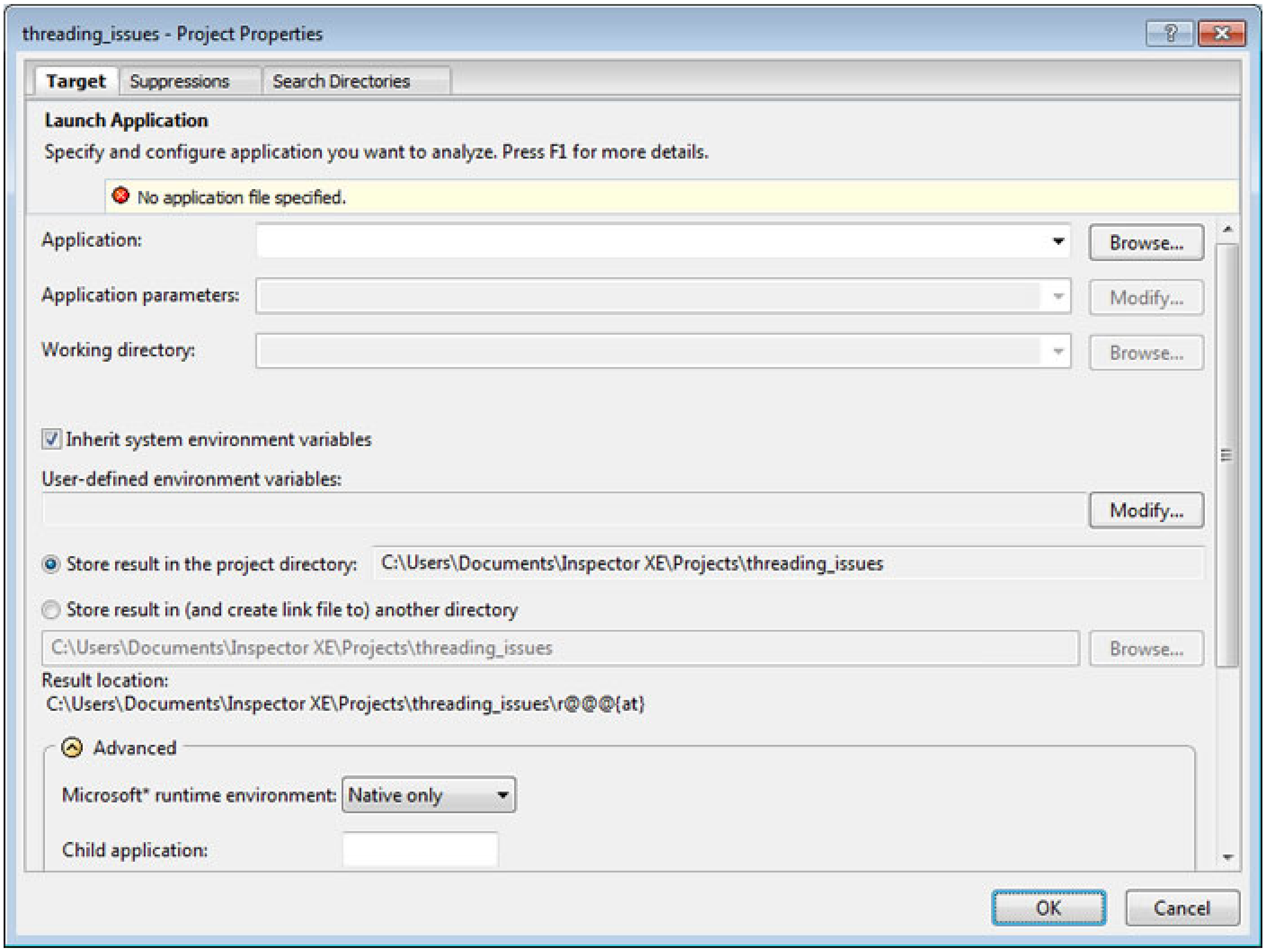
Create a New Project

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



Location: C:\Users\Documents\Inspector XE\Projects

2. In the **Project name** field, type threading_issues. Then click the **Create project** button to create a `config.inspxeproj` file in the `\Inspector XE\Projects\threading_issues\` directory (default location) and display a dialog box similar to the following:



Specify and configure application you want to analyze. Press F1 for more details.

No application file specified.

[✓] Inherit system environment variables

(●) Store result in the project directory: C:\Users\Documents\Inspector XE\Projects\threading_issues

( ) Store result in (and create link file to) another directory: C:\Users\Documents\Inspector XE\Projects\threading_issues

Result location:
C:\Users\Documents\Inspector XE\Projects\threading_issues\r@@@{at}

Microsoft* runtime environment: Native only

3. Click the **Browse** button next to the **Application** field and select the `nqueens_fortran\threading_issues\Debug\threading_issues.exe` application. Notice the Intel Inspector XE autofills the project **Working directory** field for you. Then click the **OK** button to display a **threading_issues project is open** window.

Key Terms

False positive

Configure Analysis

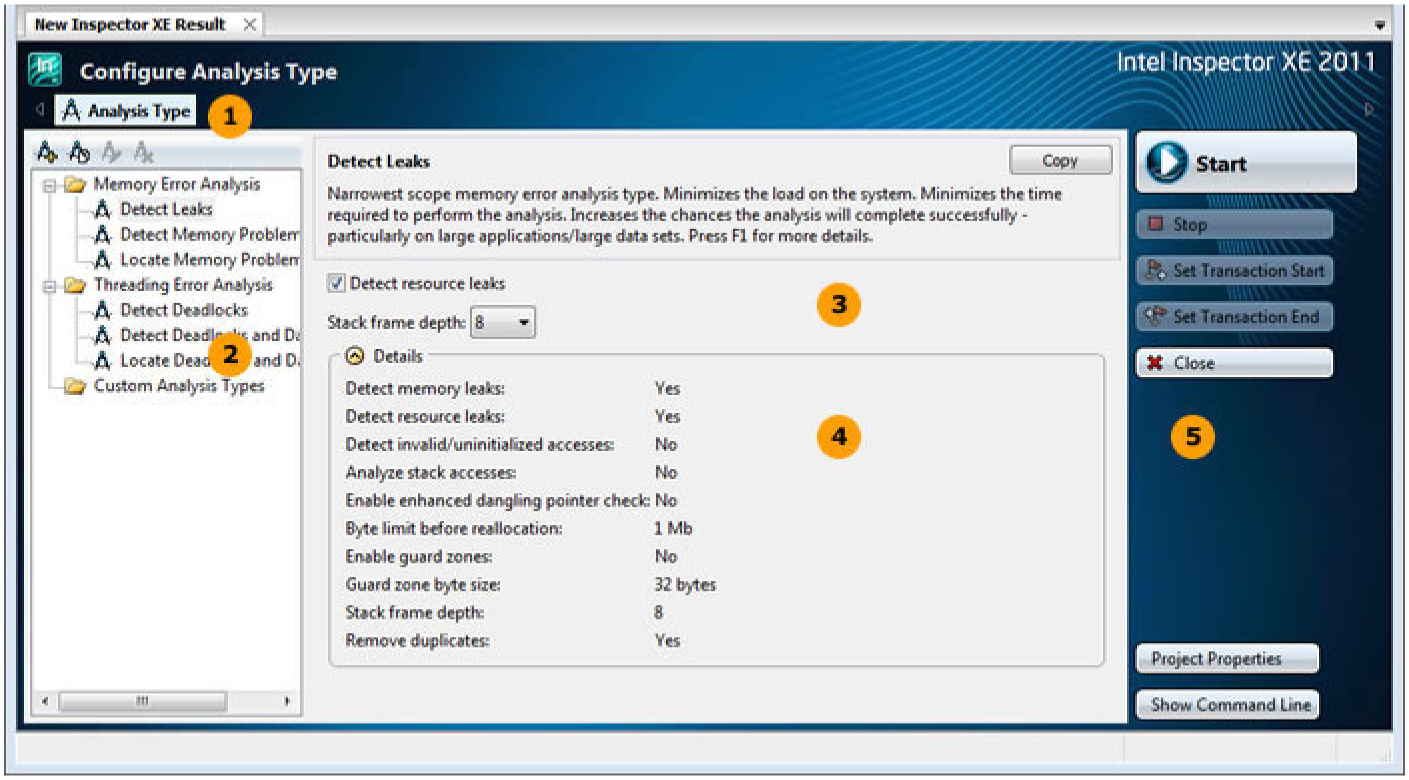
The Intel Inspector XE offers a range of preset threading analysis types to help you control analysis scope and cost. The analysis type with the narrowest scope minimizes the load on the system and the time and resources required to perform the analysis; however, it detects the narrowest set of errors 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 threading error analysis, choose a threading analysis type.

Choose Threading Error Analysis Type

1. To display an **Analysis Type** window similar to the following:

- From the Visual Studio* menu, choose **Tools** > **Intel Inspector XE 2011** > **New Analysis...**.
- From the Standalone Intel Inspector XE GUI menu, choose **File** > **New** > **Analysis...**.



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

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

2. After you finish experimenting, choose the **Detect Deadlocks and Data Races** analysis type.

Key Terms

Analysis

Run Analysis

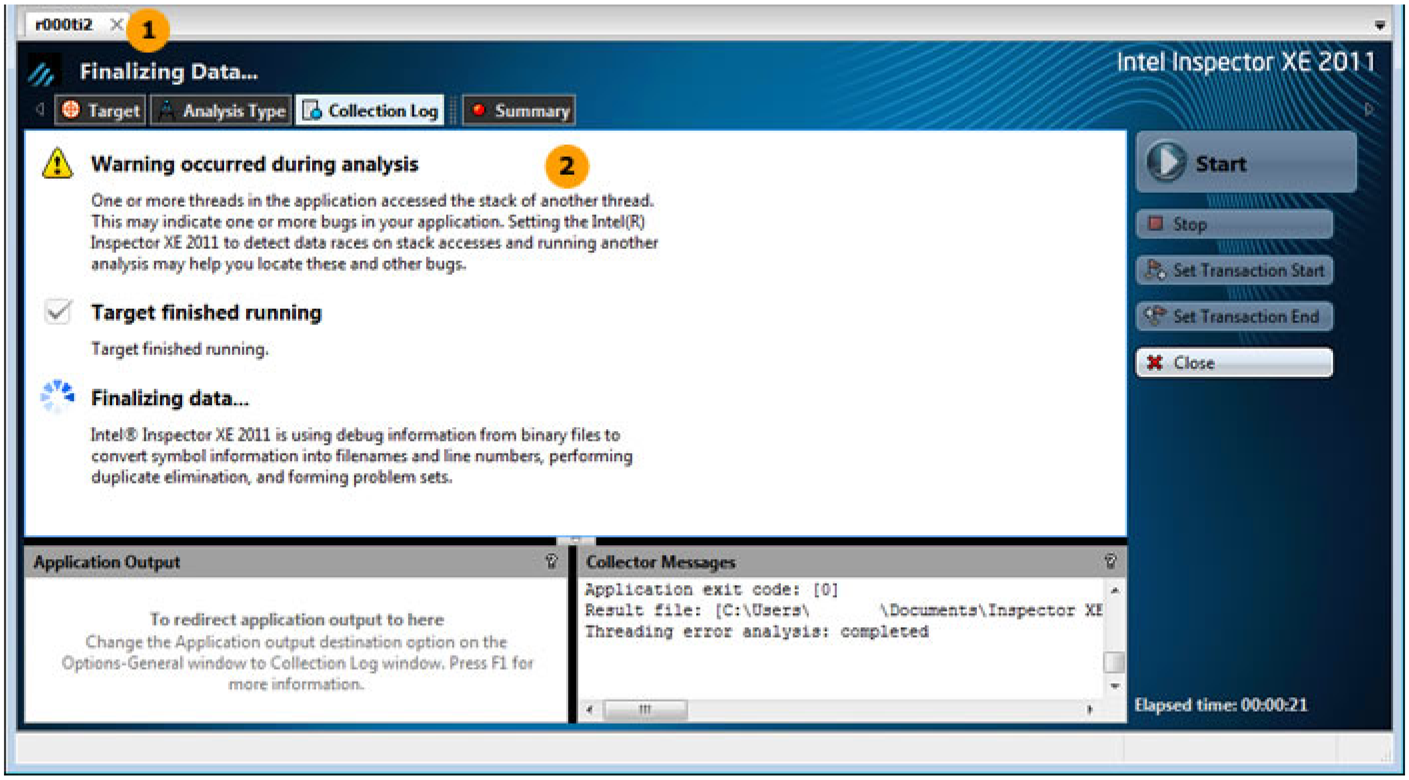
To find threading errors that may need fixing, run a threading error analysis.

Run Threading Error Analysis

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

- Executes the `threading_issues.exe` application.
- Identifies threading errors that may need handling.
- Collects the result in a directory in the `nqueens_fortran\threading_issues\My Inspector XE Results - threading_issues\` directory.
- Finalizes the result (converts 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:



Warning occurred during analysis

One or more threads in the application accessed the stack of another thread. This may indicate one or more bugs in your application. Setting the Intel(R) Inspector XE 2011 to detect data races on stack accesses and running another analysis may help you locate these and other bugs.

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.

To redirect application output to here
Change the Application output destination option on the Options-General window to Collection Log window. Press F1 for more information.

```
Application exit code: [0]
Result file: [C:\Users\        \Documents\Inspector XE
Threading error analysis: completed
```

1 The result name appears in the tab. Here, the name of the result (and the name of the result directory in the `nqueens_fortran\threading_issues\My Inspector XE Results - threading_issues\` directory) is `r000ti2`, where

- `r` = constant
- `000` = next available number
- `ti` = threading error analysis type
- `2` = preset analysis type of medium scope

NOTE Intel Inspector XE also offers a pointer to the result in the **Solution Explorer** (Visual Studio* IDE) and **Project Navigator** (standalone GUI).

2 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

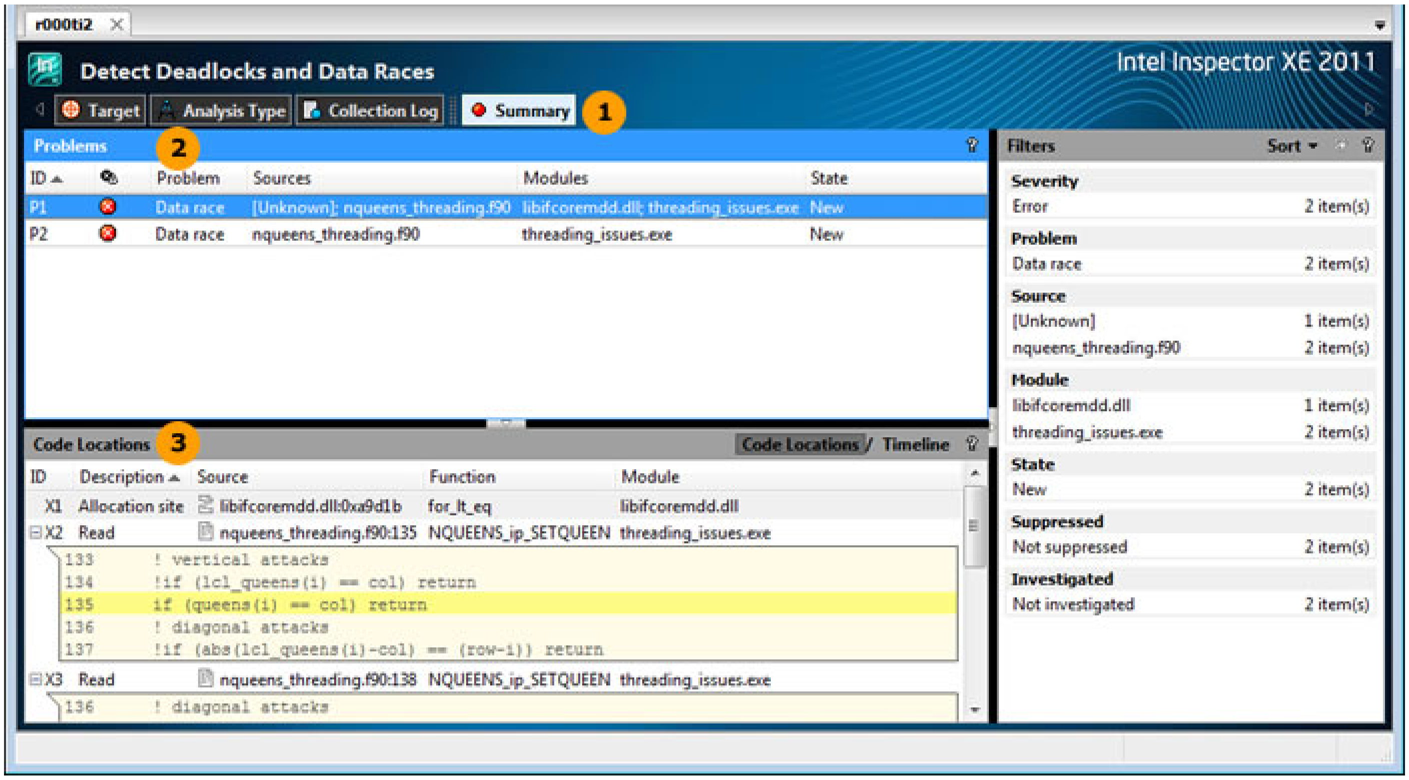
- Analysis
- Collection
- Finalization

Choose Problem Set and Focus Code Location

To start exploring a detected threading error:

- Understand window panes.
- Choose a problem set.
- Choose a focus code location.

Understand Summary Window Panes



Problems

ID		Problem	Sources	Modules	State
P1	●	Data race	[Unknown]; nqueens_threading.f90	libifcoremdd.dll; threading_issues.exe	New
P2	●	Data race	nqueens_threading.f90	threading_issues.exe	New

Filters

Sort	
Severity	
Error	2 item(s)
Problem	
Data race	2 item(s)
Source	
[Unknown]	1 item(s)
nqueens_threading.f90	2 item(s)
Module	
libifcoremdd.dll	1 item(s)
threading_issues.exe	2 item(s)
State	
New	2 item(s)
Suppressed	
Not suppressed	2 item(s)
Investigated	
Not investigated	2 item(s)

Code Locations

ID	Description	Source	Function	Module
X1	Allocation site	libifcoremdd.dll:0xa9d1b	for_lt_eq	libifcoremdd.dll
X2	Read	nqueens_threading.f90:135	NQUEENS_ip_SETQUEEN	threading_issues.exe
X3	Read	nqueens_threading.f90:138	NQUEENS_ip_SETQUEEN	threading_issues.exe

```
133        ! vertical attacks
134        !if (lcl_queens(i) == col) return
135        if (queens(i) == col) return
136        ! diagonal attacks
137        !if (abs(lcl_queens(i)-col) == (row-i)) return
```

```
136        ! diagonal attacks
```

1	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.
2	Think of the **Problems** pane as a *to-do* list. Start at the top and work your way down.
3	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

If necessary, click the data row for the P1 **Data Race** problem set.

Choose a Focus Code Location

Double-click the data row for the X2 **Read** code location set to display the **Sources** window, which provides more visibility into the cause of the error.

Key Terms

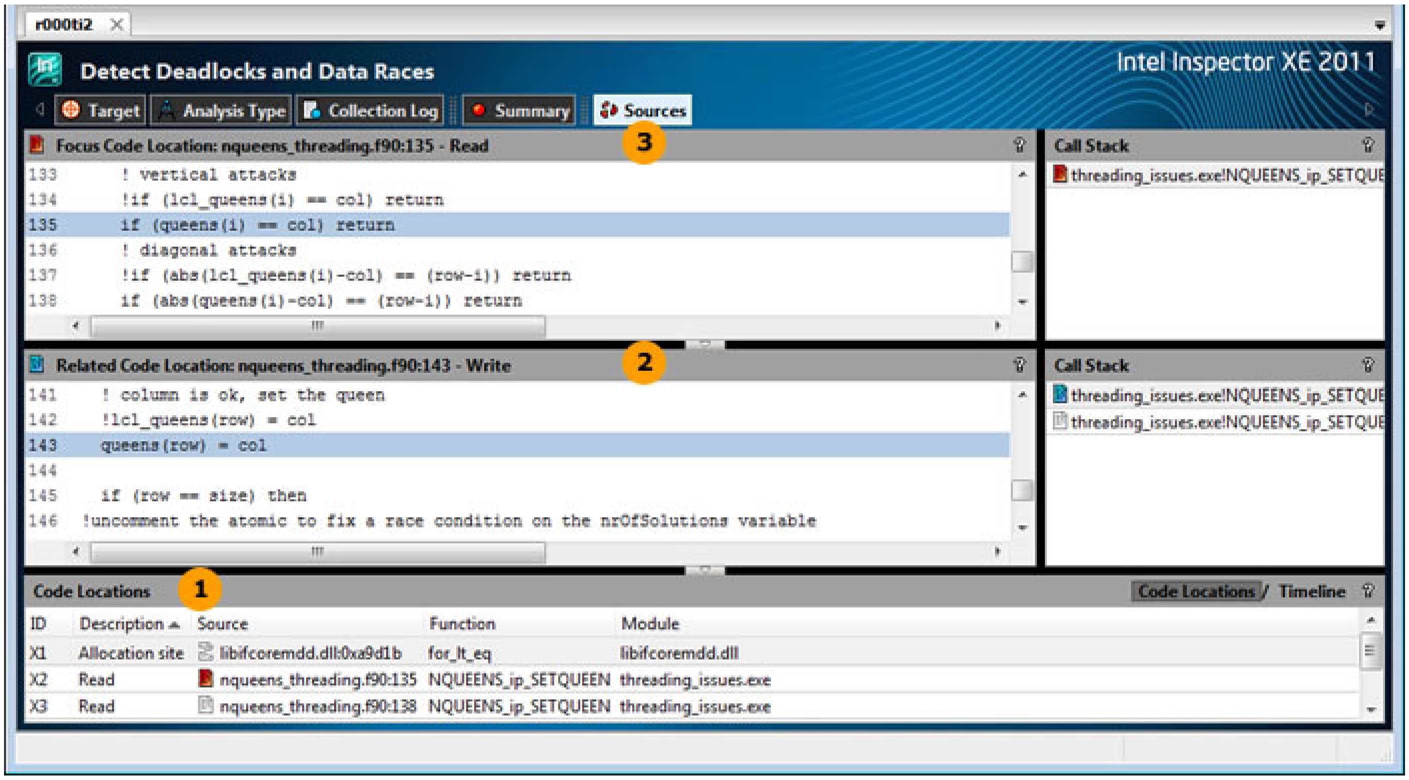
- Code location
- Problem
- Problem set
- Result

Interpret Result Data

To determine the cause of the detected threading error:

- Interpret window panes and icons.
- View source code for another code location.
- Access more information on interpreting and resolving problems.

Interpret Sources Window Panes and Icons



Focus Code Location: nqueens_threading.f90:135 - Read

```
133       ! vertical attacks
134       !if (lcl_queens(i) == col) return
135       if (queens(i) == col) return
136       ! diagonal attacks
137       !if (abs(lcl_queens(i)-col) == (row-i)) return
138       if (abs(queens(i)-col) == (row-i)) return
```

Call Stack
threading_issues.exe!NQUEENS_ip_SETQUE

Related Code Location: nqueens_threading.f90:143 - Write

```
141     ! column is ok, set the queen
142     !lcl_queens(row) = col
143     queens(row) = col
144
145     if (row == size) then
146   !uncomment the atomic to fix a race condition on the nrOfSolutions variable
```

Call Stack
threading_issues.exe!NQUEENS_ip_SETQUE
threading_issues.exe!NQUEENS_ip_SETQUE

Code Locations — Code Locations / Timeline

ID	Description	Source	Function	Module
X1	Allocation site	libifcoremdd.dll:0xa9d1b	for_lt_eq	libifcoremdd.dll
X2	Read	nqueens_threading.f90:135	NQUEENS_ip_SETQUEEN	threading_issues.exe
X3	Read	nqueens_threading.f90:138	NQUEENS_ip_SETQUEEN	threading_issues.exe

1 Like the pane on the **Summary** window, the **Code Locations** pane shows all the code locations in one Write -> Write **Data race** problem and two Write -> Read **Data race** problems in the P1 **Data race** problem set.

The Write -> Write **Data race** problem contains three code locations:

- The X4 **Write** code location represents the instruction and associated call stack of the thread responsible for a memory write.

- The X7 **Write** code location represents the instruction and associated call stack of the thread responsible for a concurrent memory write.
- The X1 **Allocation site** code location represents the location and associated call stack from which the memory block was allocated.

Each Write -> Read **Data race** problem also contains three code locations:

- The X4 **Write** code location represents the instruction and associated call stack of the thread responsible for a memory write.
- The X2 and X3 **Read** code locations represent the instructions and associated call stacks of the threads responsible for a concurrent memory read.
- The X1 **Allocation site** code location represents the location and associated call stack from which the memory block was allocated.

Notice the X4 **Write** and X1 **Allocation site** code locations are in all problems.

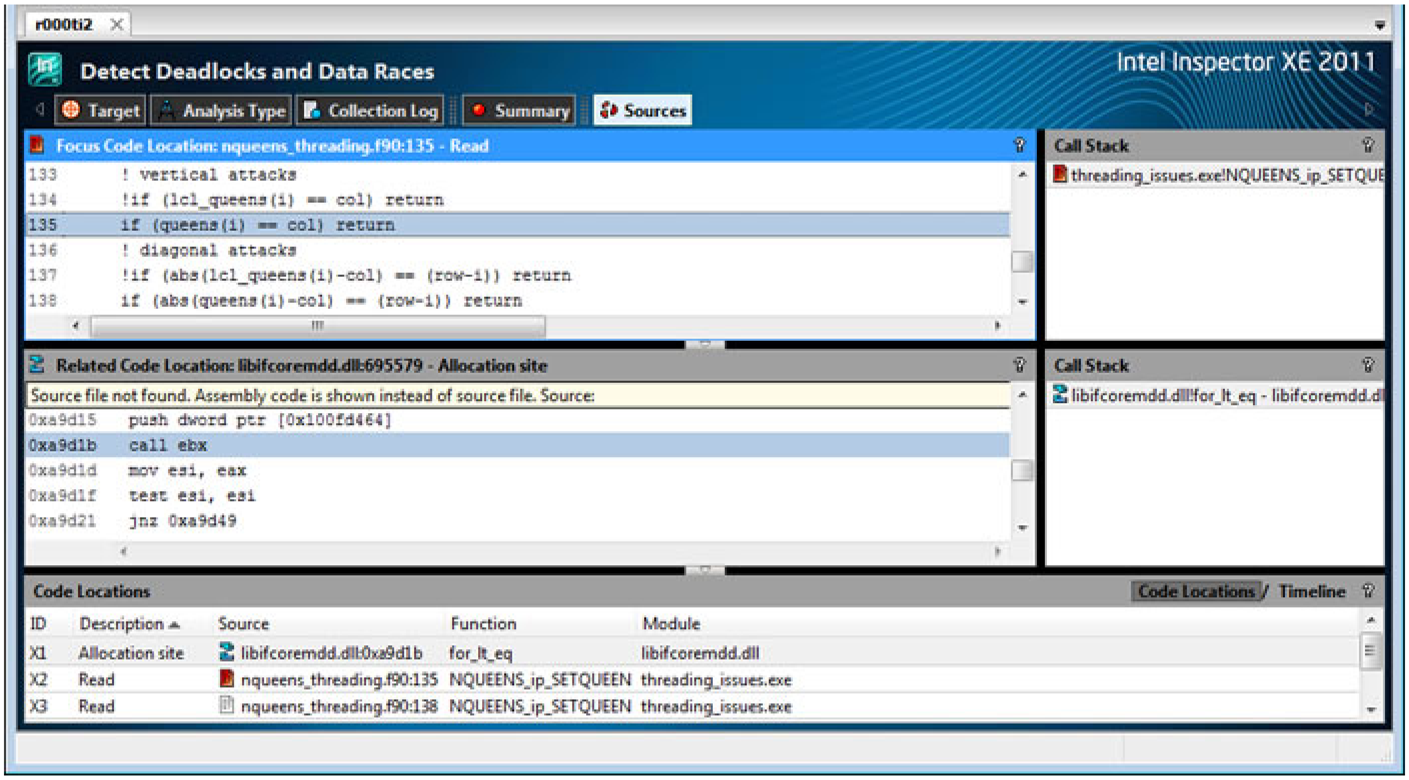
2 The **Related Code Location** pane shows the source code in the `nqueens_threading.f90` source file surrounding the **Write** code location. Also notice the 🟦 icon in the pane title matches the 🟦 icon on the **Write** code location data row in the **Code Locations** pane. The source code corresponding to the **Write** code location is highlighted.

3 The **Focus Code Location** pane shows the source code in the `nqueens_threading.f90` source file surrounding the **Read** code location. Notice the 🟥 icon in the pane title matches the 🟥 icon on the **Read** code location data row in the **Code Locations** pane. The source code corresponding to the **Read** code location is highlighted.

Icon	Meaning	
🟥	This code location is the focus code location. You chose it when you double-clicked the **Read** code location on the **Summary** window. Its source code is currently displayed in the **Focus Code Location** pane.	Code location source code is available for viewing in the Intel Inspector XE and editing in an editor.
🟦	This code location is related to the focus code location. Its source code is currently displayed in the **Related Code Location** pane.	
▯	This is another code location in the problem or problem set. Its source code is not currently displayed on screen.	
▨	This is another code location in the problem or problem set for which the Intel Inspector XE did not find the source file.	Code location source code is not available for viewing in the Intel Inspector XE and editing in an editor.

View Source Code for Another Code Location

Double-click the data row for the **Allocation site** code location in the **Code Locations** pane to display a window similar to the following:



Focus Code Location: nqueens_threading.f90:135 - Read

```
133        ! vertical attacks
134        !if (lcl_queens(i) == col) return
135        if (queens(i) == col) return
136        ! diagonal attacks
137        !if (abs(lcl_queens(i)-col) == (row-i)) return
138        if (abs(queens(i)-col) == (row-i)) return
```

threading_issues.exe!NQUEENS_ip_SETQUE

Related Code Location: libifcoremdd.dll:695579 - Allocation site

Source file not found. Assembly code is shown instead of source file. Source:

```
0xa9d15    push dword ptr [0x100fd464]
0xa9d1b    call ebx
0xa9d1d    mov esi, eax
0xa9d1f    test esi, esi
0xa9d21    jnz 0xa9d49
```

libifcoremdd.dll!for_lt_eq - libifcoremdd.dl

ID	Description	Source	Function	Module
X1	Allocation site	libifcoremdd.dll:0xa9d1b	for_lt_eq	libifcoremdd.dll
X2	Read	nqueens_threading.f90:135	NQUEENS_ip_SETQUEEN	threading_issues.exe
X3	Read	nqueens_threading.f90:138	NQUEENS_ip_SETQUEEN	threading_issues.exe

Notice the window changes:

- The **Related Code Location** pane now shows the source code for the **Allocation site** code location and the icon for the **Allocation site** code location is now instead of throughout the **Sources** window.
- The icon for the X4 **Read** code location is now instead of .

Double-click the data row for X4 **Read** code location.

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 **Data race** problem type.

Key Terms

- Code location
- Problem
- Problem set
- Related code location

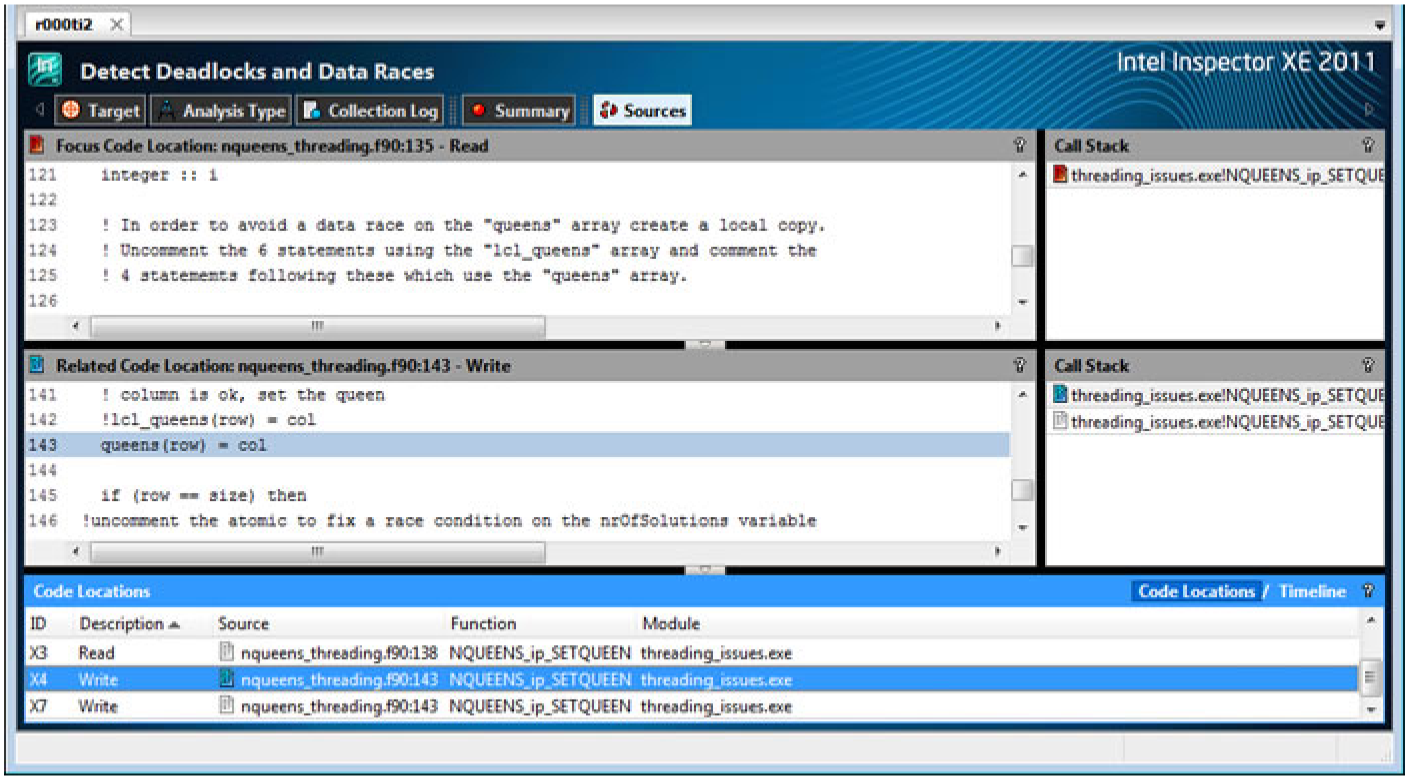
Resolve Issue

To fix the detected threading error:

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

Investigate the Issue

Scroll to near line 123 in the **Focus Code Location** pane to display a window similar to the following:



Focus Code Location: nqueens_threading.f90:135 - Read

```
    integer :: i

    ! In order to avoid a data race on the "queens" array create a local copy.
    ! Uncomment the 6 statements using the "lcl_queens" array and comment the
    ! 4 statements following these which use the "queens" array.

```

threading_issues.exe!NQUEENS_ip_SETQUE

Related Code Location: nqueens_threading.f90:143 - Write

```
    ! column is ok, set the queen
    !lcl_queens(row) = col
    queens(row) = col

    if (row == size) then
  !uncomment the atomic to fix a race condition on the nrOfSolutions variable
```

threading_issues.exe!NQUEENS_ip_SETQUE
threading_issues.exe!NQUEENS_ip_SETQUE

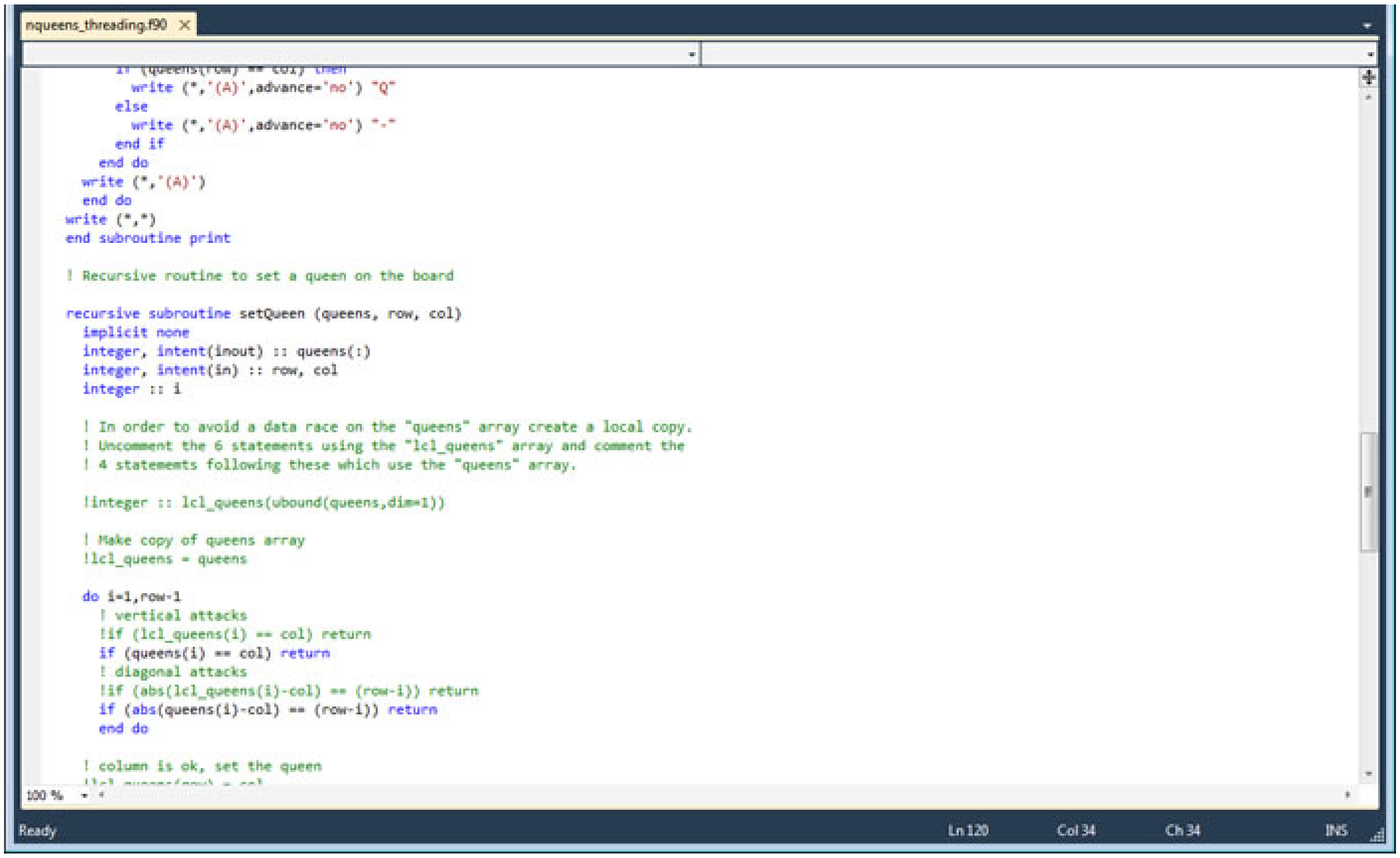
Code Locations

ID	Description	Source	Function	Module
X3	Read	nqueens_threading.f90:138	NQUEENS_ip_SETQUEEN	threading_issues.exe
X4	Write	nqueens_threading.f90:143	NQUEENS_ip_SETQUEEN	threading_issues.exe
X7	Write	nqueens_threading.f90:143	NQUEENS_ip_SETQUEEN	threading_issues.exe

The commenting in the **Focus Code Location** window identifies the cause of the **Data race** problems: Multiple threads are concurrently accessing the global `queens` array. One possible correction strategy: Change the global array to a local array.

Access Editor

Double-click anywhere in the **Focus Code Location** pane to open the `nqueens_threading.f90` source file in an editor:



```
        write (*,'(A)',advance='no') "Q"
      else
        write (*,'(A)',advance='no') "-"
      end if
    end do
  write (*,'(A)')
  end do
write (*,*)
end subroutine print

! Recursive routine to set a queen on the board

recursive subroutine setQueen (queens, row, col)
  implicit none
  integer, intent(inout) :: queens(:)
  integer, intent(in) :: row, col
  integer :: i

  ! In order to avoid a data race on the "queens" array create a local copy.
  ! Uncomment the 6 statements using the "lcl_queens" array and comment the
  ! 4 statememts following these which use the "queens" array.

  !integer :: lcl_queens(ubound(queens,dim=1))

  ! Make copy of queens array
  !lcl_queens = queens

  do i=1,row-1
    ! vertical attacks
    !if (lcl_queens(i) == col) return
    if (queens(i) == col) return
    ! diagonal attacks
    !if (abs(lcl_queens(i)-col) == (row-i)) return
    if (abs(queens(i)-col) == (row-i)) return
    end do

  ! column is ok, set the queen
```

Change the Source Code

1. Search the file and uncomment six statements using the `lcl_queens` array. Beneath four of those six statements, comment out the statements using the `queens` array.
2. Save your edits (automatic if you are using the Visual Studio* editor in the Visual Studio* IDE) 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

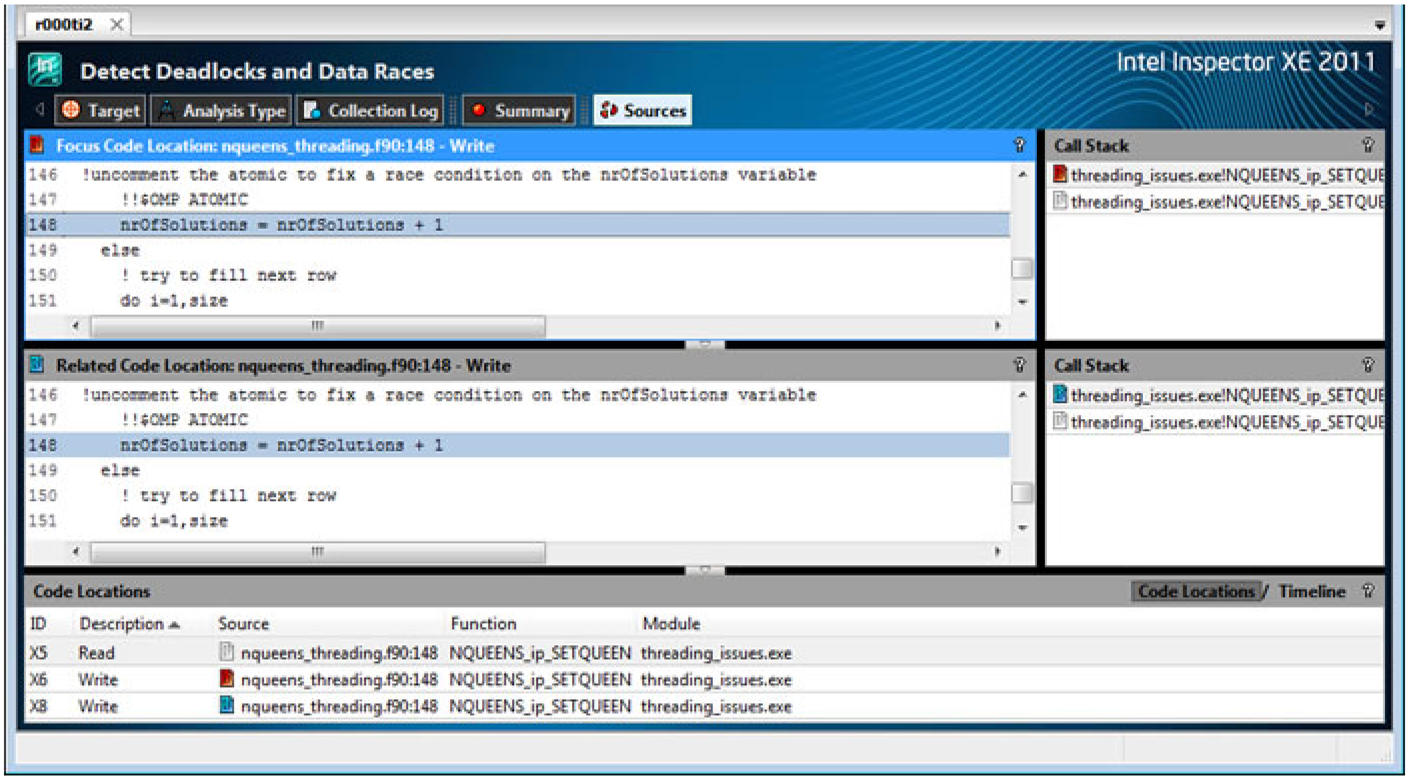
Resolve Next Issue

To fix another detected threading error:

- Choose another problem set.
- Fix the threading error.

Choose Another Problem Set

In the **Problems** pane on the **Summary** window, double-click the data row for the P2 **Data race** problem set to display the **Sources** window:



Focus Code Location: nqueens_threading.f90:148 - Write

```
146 !uncomment the atomic to fix a race condition on the nrOfSolutions variable
147     !!$OMP ATOMIC
148     nrOfSolutions = nrOfSolutions + 1
149   else
150     ! try to fill next row
151     do i=1,size
```

Call Stack

threading_issues.exe!NQUEENS_ip_SETQUE
threading_issues.exe!NQUEENS_ip_SETQUE

Related Code Location: nqueens_threading.f90:148 - Write

```
146 !uncomment the atomic to fix a race condition on the nrOfSolutions variable
147     !!$OMP ATOMIC
148     nrOfSolutions = nrOfSolutions + 1
149   else
150     ! try to fill next row
151     do i=1,size
```

Call Stack

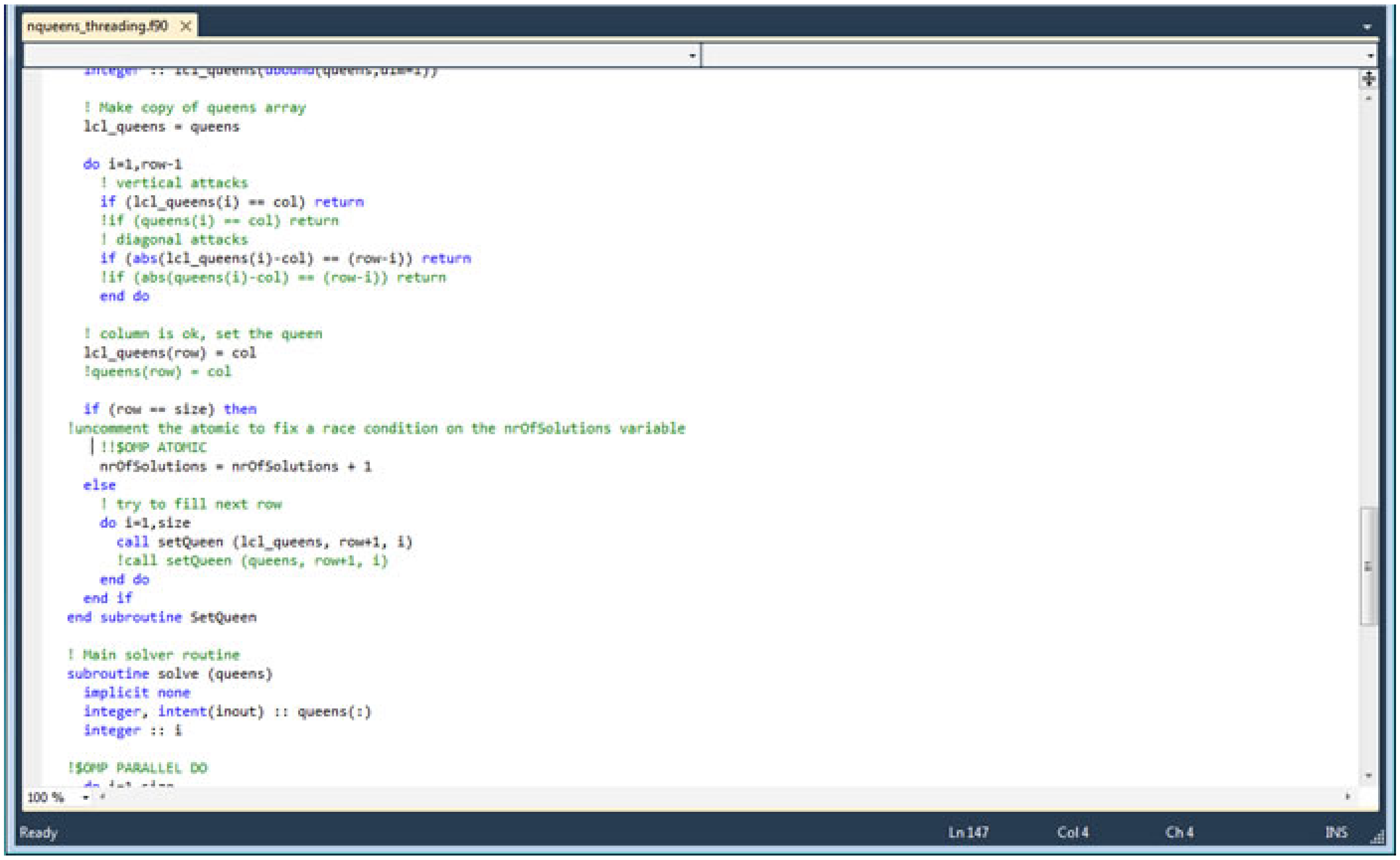
threading_issues.exe!NQUEENS_ip_SETQUE
threading_issues.exe!NQUEENS_ip_SETQUE

Code Locations — Code Locations / Timeline

ID	Description	Source	Function	Module
X5	Read	nqueens_threading.f90:148	NQUEENS_ip_SETQUEEN	threading_issues.exe
X6	Write	nqueens_threading.f90:148	NQUEENS_ip_SETQUEEN	threading_issues.exe
X8	Write	nqueens_threading.f90:148	NQUEENS_ip_SETQUEEN	threading_issues.exe

Fix the Threading Error

1. Double-click line 147 in either the **Focus Code Location** or **Related Code Location** pane to open the `nqueens_threading.90` source file in your editor:



```
  ! Make copy of queens array
  lcl_queens = queens

  do i=1,row-1
    ! vertical attacks
    if (lcl_queens(i) == col) return
    !if (queens(i) == col) return
    ! diagonal attacks
    if (abs(lcl_queens(i)-col) == (row-i)) return
    !if (abs(queens(i)-col) == (row-i)) return
    end do

  ! column is ok, set the queen
  lcl_queens(row) = col
  !queens(row) = col

  if (row == size) then
!uncomment the atomic to fix a race condition on the nrOfSolutions variable
   !!$OMP ATOMIC
    nrOfSolutions = nrOfSolutions + 1
  else
    ! try to fill next row
    do i=1,size
      call setQueen (lcl_queens, row+1, i)
      !call setQueen (queens, row+1, i)
    end do
  end if
end subroutine SetQueen

! Main solver routine
subroutine solve (queens)
  implicit none
  integer, intent(inout) :: queens(:)
  integer :: i

!$OMP PARALLEL DO
```

2. Uncomment `!!$OMP ATOMIC`.
3. Save your edits (automatic if you are using the Visual Studio* editor in the Visual Studio* IDE) and return to the **Sources** window.

Key Terms

- Code location
- Problem
- Problem set

Rebuild and Rerun Analysis

To check if your edits resolved the threading errors:

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

Rebuild the Application

If you are using the Visual Studio* IDE:

1. Choose **Build** > **Clean Solution**.
2. Choose **Build** > **Rebuild Solution**.

If you are using the Standalone Intel Inspector XE GUI:

1. In a command prompt window, change directory to the `nqueens_fortran` directory.

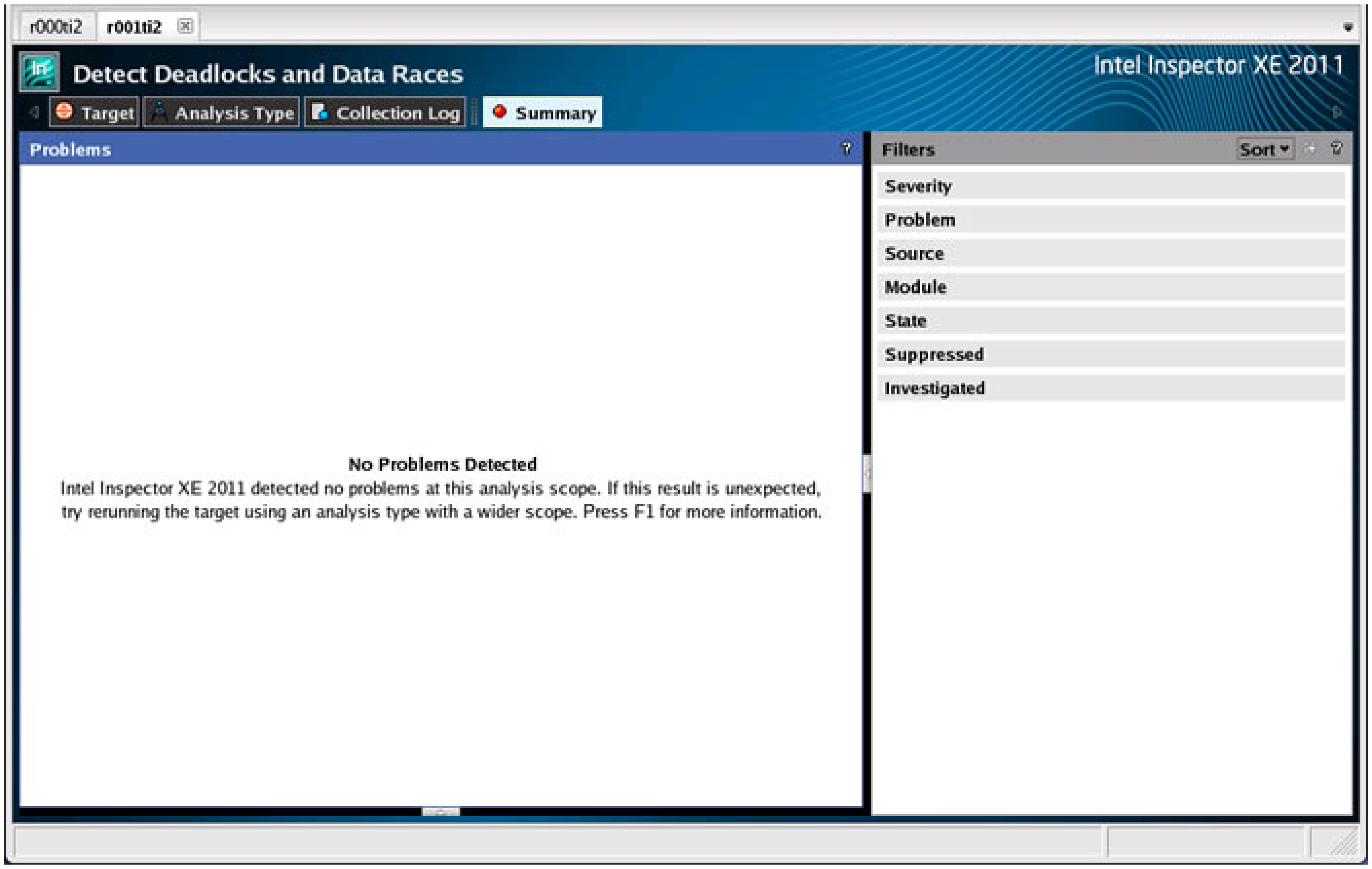
2. Type devenv nqueens_fortran.sln /Clean.
3. Type devenv nqueens_fortran.sln /Build.

Rerun the Analysis

To run another analysis of the same analysis type:

- From the Visual Studio* menu, choose **Tools** > **Intel Inspector XE 2011** > **Threading Error Analysis / Detect Deadlocks and Data Races**.
- From the Standalone Intel Inspector XE GUI menu, choose **File** > **Threading Error Analysis / Detect Deadlocks and Data Races**.

The **Summary** window automatically displays after analysis (both collection and finalization) completes successfully:



No Problems Detected
Intel Inspector XE 2011 detected no problems at this analysis scope. If this result is unexpected, try rerunning the target using an analysis type with a wider scope. Press F1 for more information.

Notice the Intel Inspector XE:

- Created a new result tab.
- No longer detects any threading problems.

Key Terms

Analysis

Summary

3

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

Step	Tutorial Recap	Key Tutorial Take-aways
1. Prepare for analysis	If you used the Visual Studio* IDE: You chose a project; verified the project is set to produce the most accurate, complete results; built and ensured the application runs on your system outside the Intel Inspector XE. If you used the standalone GUI: 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: `/debug:full`, `/Od`, `/libs:dll`, and `/check:[no] bounds`. • 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 **Solution Explorer** (Visual Studio* IDE) or **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 **Tools** menu (Visual Studio* IDE), **File** menu (Standalone Intel Inspector XE GUI), 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.

Step	**Tutorial Recap**	**Key Tutorial Take-aways**
		• Double-click a code location on the **Sources** window to open an editor.
4. Check your work	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

4

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.