Getting Started Tutorial: Analyzing Threading Errors

Intel® Inspector XE 2013 for Windows* OS

Fortran Sample Application Code

Document Number: 327652-002

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

Contents

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/

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, skoool, the skoool 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.

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

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: **1.** Build an application to produce an optimal inspection result. **2.** Inspect an application to find threading errors. **3.** Edit application code to fix the threading errors. **4.** Investigate threading errors using interactive debugging. **5.** Rebuild and reinspect the application.
Estimated Duration	10-15 minutes.
Learning Objectives	After you complete this tutorial, you should be able to: • List the steps to find and fix threading errors using the Intel Inspector XE. • Define key Intel Inspector XE terms. • Identify compiler/linker options that produce the most accurate and complete analysis results. • Run threading error analyses. • Influence analysis scope and running time. • 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. • Launch an interactive debugging session to investigate problems more deeply.
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 other Intel® 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 analysis results created by Intel® compilers in various suite products.

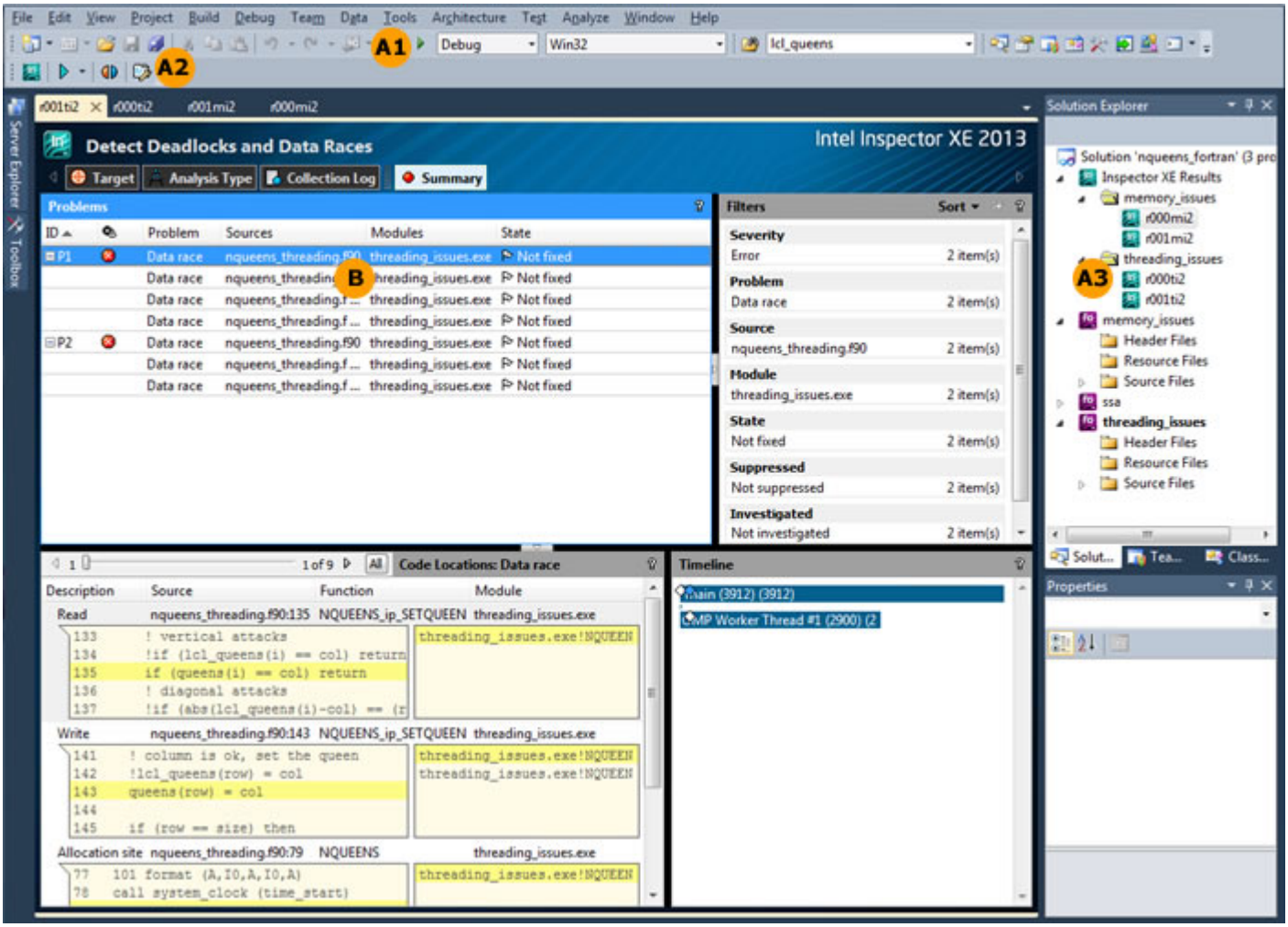
Intel Inspector XE Access

To access the Intel Inspector XE in the Visual Studio* IDE: From the Windows* **Start** menu, choose **All Programs** > **Intel Parallel Studio XE 2013** > **Parallel Studio XE 2013 with [VS2008 | VS2010]**.

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

- From the Windows* **Start** menu, choose **All Programs** > **Intel Parallel Studio XE 2013** > **Intel Inspector XE 2013**.
- From the Windows* **Start** menu, choose **All Programs** > **Intel Parallel Studio XE 2013** > **Command Prompt** > **Parallel Studio XE with Intel Compiler** > **IA-32 Visual Studio [2008 | 2010] mode** to set up your environment, then type `inspxe-gui` in the **Command Prompt** window.

Intel Inspector XE/Visual Studio* IDE Integration



A1 - A3 The menu, toolbar, and **Solution Explorer** offer different ways to perform many of the same functions.

A1 Use the **Tools > Intel Inspector XE 2013** menu to create dynamic analysis results, compare results, and import result archive files and results from other Intel® error-detection products.

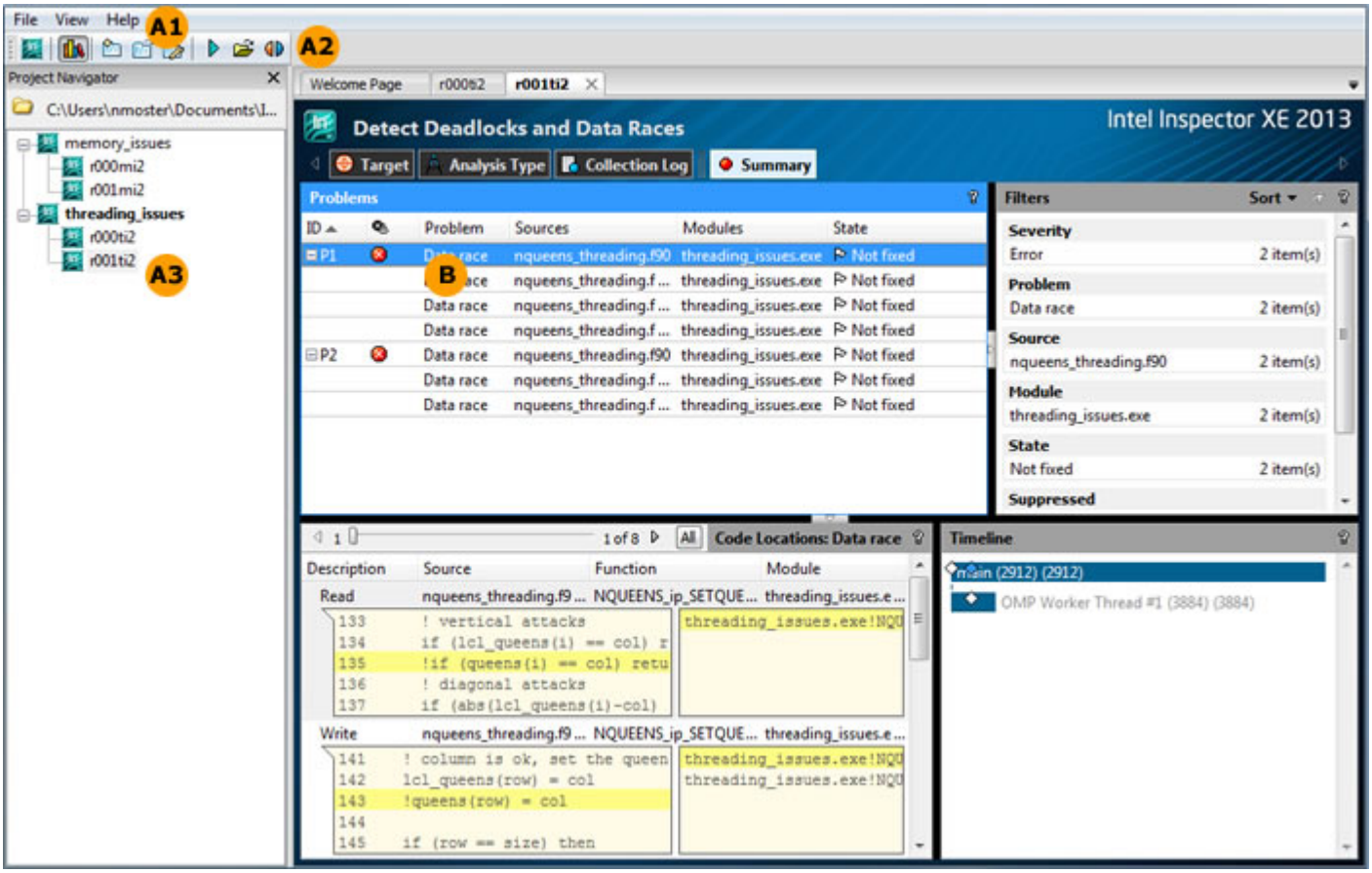
A2 Use the **Intel Inspector XE** toolbar to open the *Getting Started* page, create dynamic analysis results, compare results, and configure projects.

A3 **Solution Explorer** context menus (right-click to open):

- Use the **Intel Inspector XE 2013** 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, export result archive files, create dynamic analysis results, and manage results.

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

Standalone Intel Inspector XE GUI



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*.

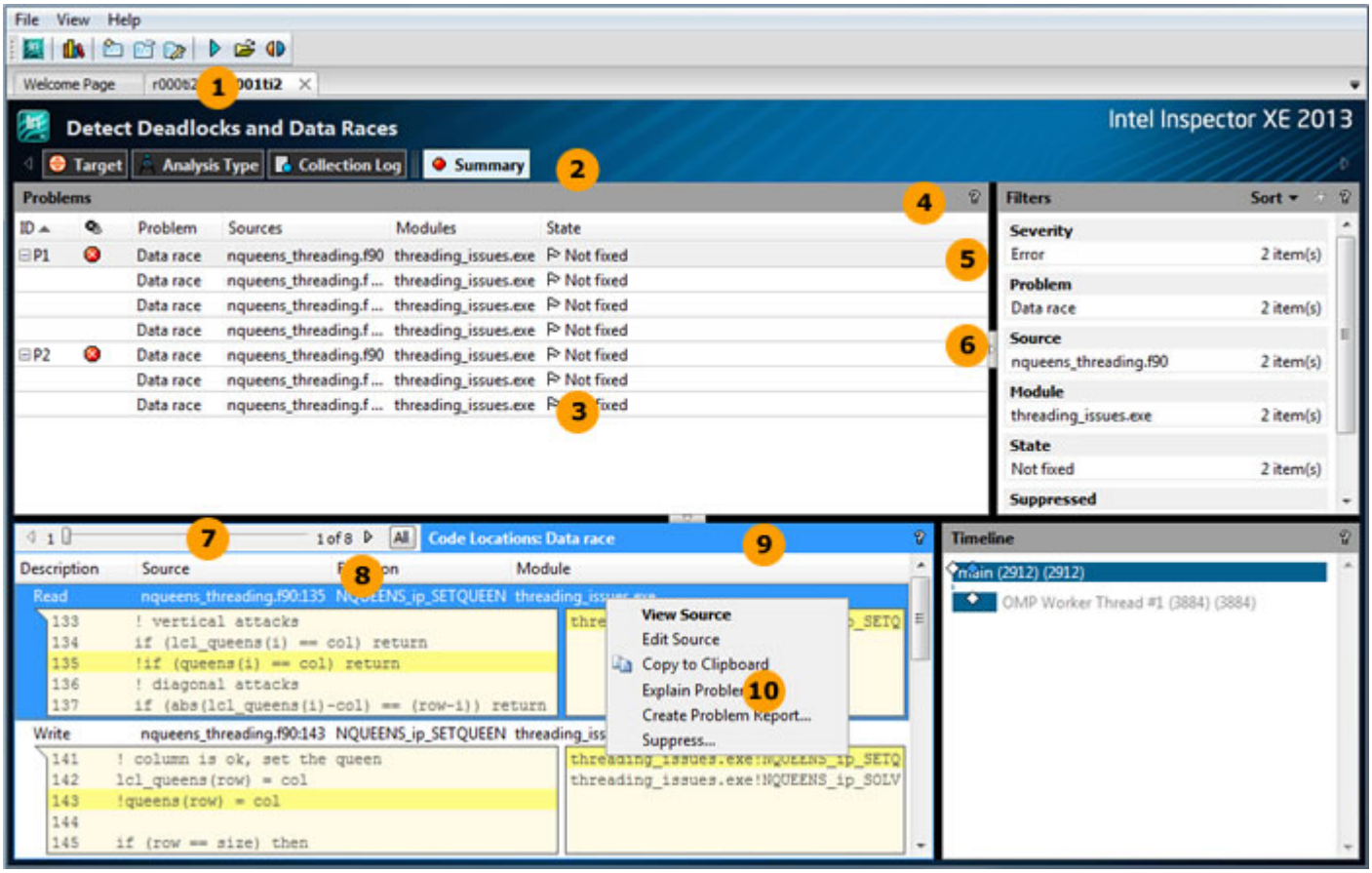
A2 Use the toolbar to open the *Getting Started* page; create, configure, and open projects; create dynamic analysis results; and open and compare results.

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  ,  ,  , and  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.

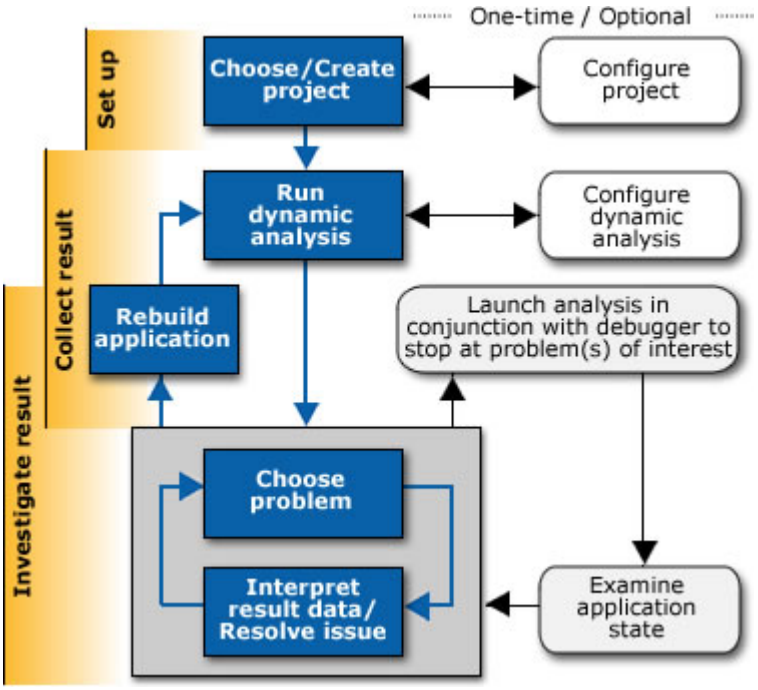
9 Use title bars to identify window panes.

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

Analyzing 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: Unpack and set up sample 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: Collect result	• Configure a threading error analysis. • Run the threading error analysis on the application.
Step 3: Investigate result	• Choose a problem in the analysis result. • Interpret the result data. • Resolve the issue. • Investigate another problem using interactive debugging.
Step 4: Check your work	Rebuild the application and rerun the threading error analysis.

Visual Studio* IDE: Choose Project and Build Application

 Follow these steps only if you are using the Intel Inspector XE plug-in to the Visual Studio* IDE to complete this tutorial.

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

- Get software tools.
- Open a Visual Studio* solution.
- Choose a startup project.
- Verify optimal compiler/linker settings.
- Verify the application is set to build in debug mode.
- 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 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

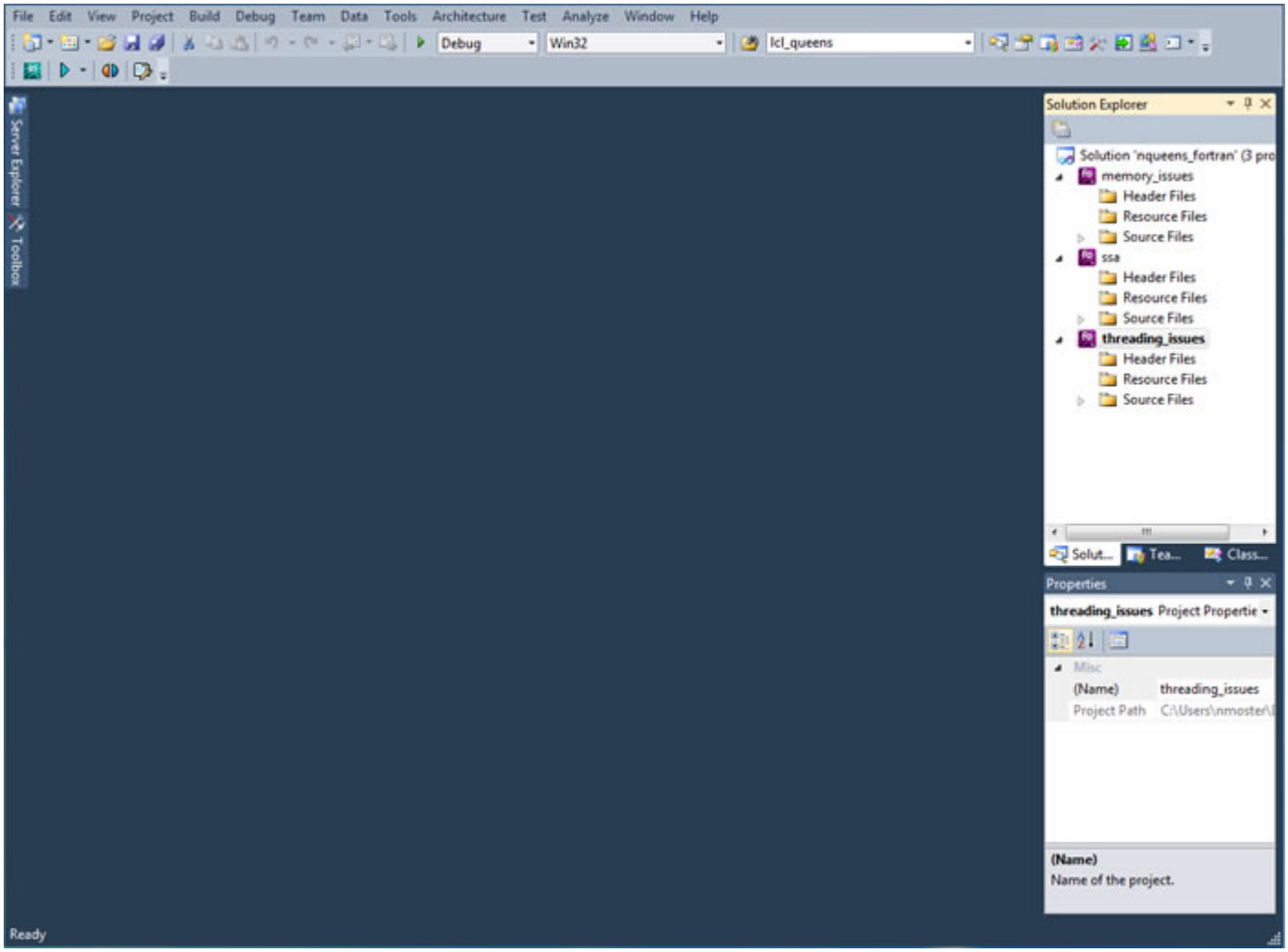
1. Copy the `nqueens_fortran.zip` file from the `<install-dir>\samples\<locale>\Fortran` directory to a writable directory or share on your system. The default `<install-dir>` is `C:\Program Files (x86)\Intel\Inspector XE 2013\` (on certain systems, instead of `Program Files (x86)`, the directory name is `Program Files`).
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**.

NOTE The `nqueens_fortran.sln` solution was created using the Visual Studio* 2008 IDE. If the Visual Studio* conversion wizard appears, follow the steps to convert the solution to run on your installed version of the Visual Studio* IDE.



Choose a Startup Project

If the **threading_issues** project is not the startup project (project typeface in the **Solution Explorer** is not bold),

1. Right-click the **threading_issues** project in the **Solution Explorer**.
2. Choose **Set as StartUp Project**, which changes the project typeface to bold.

Verify Optimal Compiler/Linker Settings

You can use the Intel® Inspector XE to analyze:

- Memory errors in debug and release modes of:
  - C++ binaries - The Intel Inspector XE can analyze native code in native and mixed native/managed binaries.
  - Fortran binaries - The Intel Inspector XE can analyze native code in native binaries.
- Threading errors in debug and release modes of:
  - C++ binaries - The Intel Inspector XE can analyze native and managed code in native, managed, and mixed native/managed binaries.
  - Fortran binaries - The Intel Inspector XE can analyze native code in native binaries.

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 (`/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/threads` or `libs:dll/threads/dbglibs`)	False positives or missing code locations
Basic runtime error checks	Disabled (do not use `/RTC`; **Default** option in Visual Studio* IDE)	None (`/check:none`)	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**.
   - Choose **Debugging** and verify the **Debug Information Format** field is set to **Full (/debug:full)**.
   - Choose **Optimization** and verify the **Optimization** field is set to **Disable (/Od)**.
   - Choose **Libraries** and verify the **Runtime Library** field is set to **Multithread DLL (/libs:dll/threads)** or **Debug Multithread DLL (libs:dll/threads/dbglibs)**.
   - Choose **Run-time** and verify the **Runtime Error Checking** field is set to **None (/check:none)**.
5. In the left pane, choose **Configuration Properties** > **Linker** > **Debugging** and 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.

Build and Test the Application

1. Choose **Build** > **Project Only** > **Build Only threading_issues** to build a single project in the solution.
2. Check the messages in the **Output** window to confirm the build succeeded.
3. Choose **Debug** > **Start Without Debugging** to test the application.
4. If the Visual Studio* IDE responds any projects are out of date, click **No**.
5. Check for non-deterministic application output (that also varies by number of cores) 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
Correct result!
Calculations took 31 ms.
Press any key to continue...
```

Key Terms

False positive

Standalone GUI: Build Application and Create New Project


Follow these steps only if you are using the Standalone Intel Inspector XE GUI to complete this tutorial.

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

- Get software tools.
- Verify optimal compiler/linker settings.
- 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 , including sample applications
- `.zip` file extraction utility
- Supported compiler (see *Release Notes* for more information)

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.zip` file from the `<install-dir>\samples\<locale>\Fortran\` directory to a writable directory or share on your system. The default `<install-dir>` is `C:\Program Files\Intel (x86)\Inspector 2013\` (on certain systems, instead of `Program Files (x86)`, the directory name is `Program Files`).
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:
  - C++ binaries - The Intel Inspector XE can analyze native code in native and mixed native/managed binaries.
  - Fortran binaries - The Intel Inspector XE can analyze native code in native binaries.
- Threading errors in debug and release modes of:
  - C++ binaries - The Intel Inspector XE can analyze native and managed code in native, managed, and mixed native/managed binaries.
  - Fortran binaries - The Intel Inspector XE can analyze native code in native binaries.

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 (`/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/threads` or `libs:dll/threads/dbglibs`)	False positives or missing code locations
Basic runtime error checks	Disabled (do not use `/RTC`; **Default** option in Visual Studio* IDE)	None (`/check:none`)	False positives

Build the Application

1. From the Windows* **Start** menu, choose **All Programs** > **Intel Parallel Studio XE 2013** > **Command Prompt** > **Parallel Studio XE with Intel Compiler** > **IA-32 Visual Studio [2008 | 2010] mode** to set up your environment.
2. Change directory to the `nqueens_fortran` directory in its unzipped location.
3. If you chose **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.

NOTE Do not be concerned if you see a `Build: 2 succeeded, 1 failed` message because of static analysis build errors.

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 non-deterministic application output (that also varies by number of cores) similar to the following:

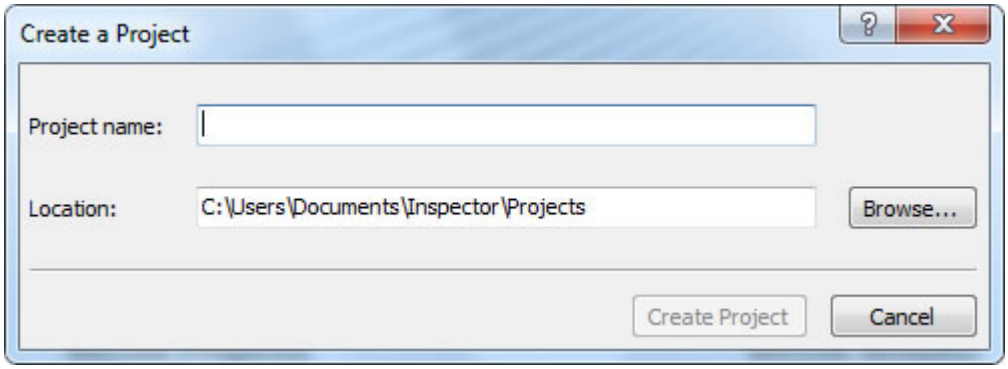
```
Usage: threading_issues.exe boardSize
Using default size of 10
Starting nqueens solver for size 10 with 2 thread(s)
Number of solutions: 1333
Incorrect result!
Calculations took 31 ms.
```

Open the Standalone Intel Inspector XE GUI

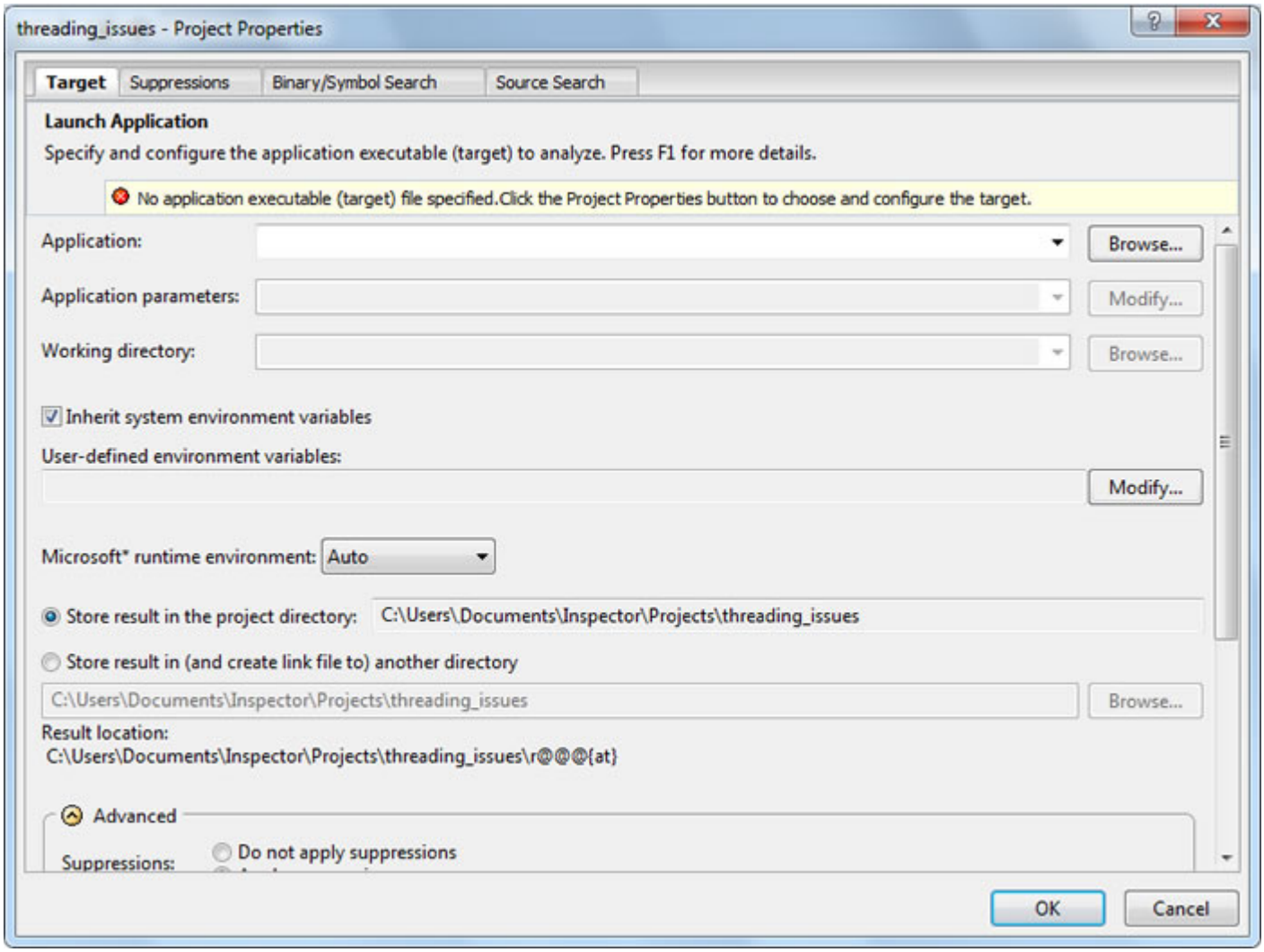
From the Windows* **Start** menu, choose **All Programs** > **Intel Parallel Studio XE 2013** > **Intel Inspector XE 2013**.

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 `threading_issues`. Then click the **Create project** button to create a `config.inspxeproj` file in the `\Inspector\Projects\threading_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 \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 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

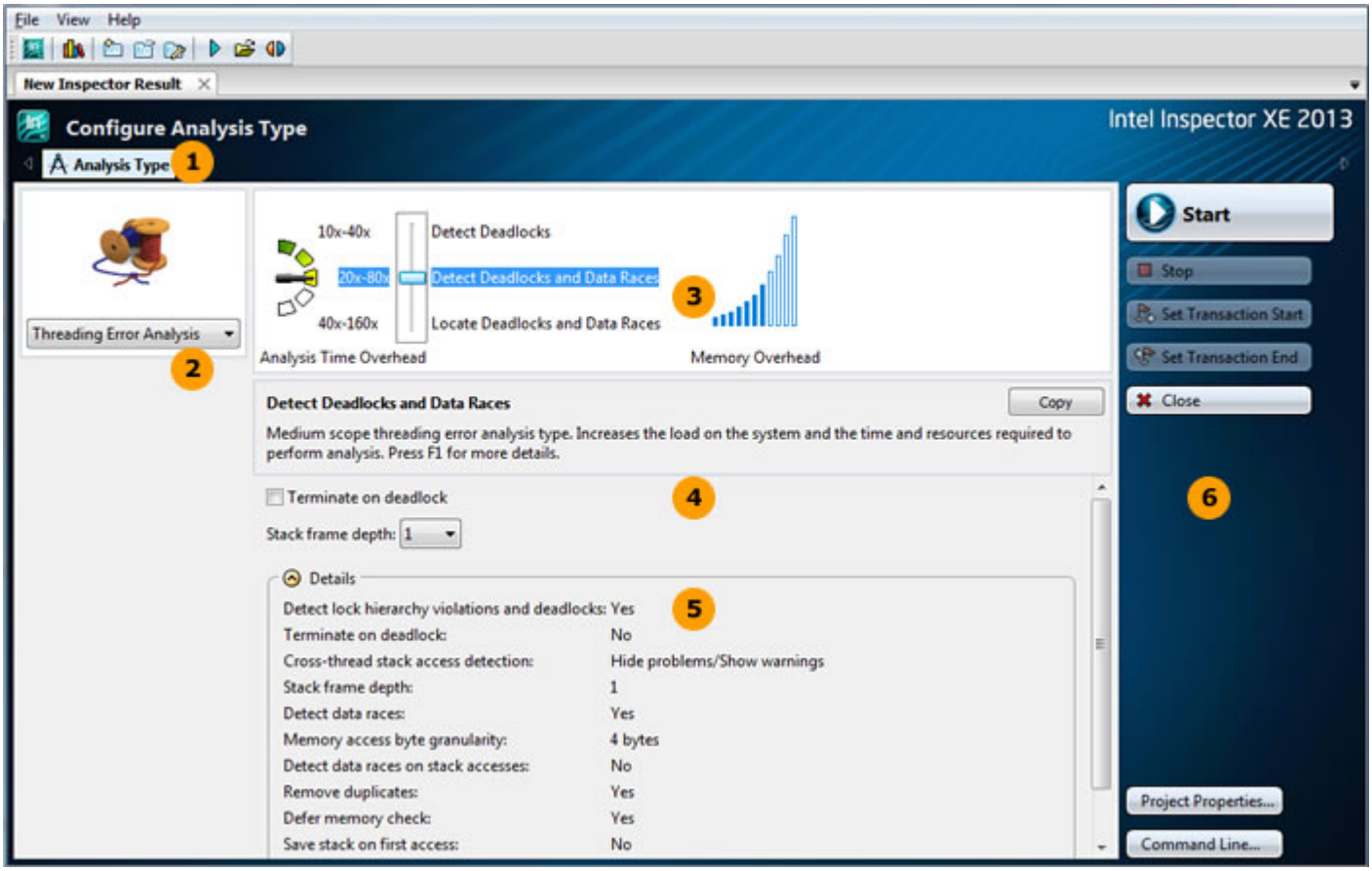
Intel Inspector XE offers a range of preset threading analysis types to help you control analysis scope and cost. The analysis type with the narrowest scope minimizes the load on the system and the time and resources required to perform the analysis; however, it detects the narrowest set of errors and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

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

- From the Visual Studio* menu, choose **Tools** > **Intel Inspector XE 2013** > **New Analysis...**.
- From the Standalone Intel Inspector XE GUI menu, choose **File** > **New** > **Analysis...**.



NOTE Your screen includes interactive debugging options if you are working in the Visual Studio* IDE.

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 **Threading Error Analysis** type category.

   NOTE This tutorial covers threading error analysis types, which you can use to search for data race, deadlock, lock hierarchy violation, and cross-thread stack access errors. Use memory error analysis types 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.

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 Deadlocks and Data Races** 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, 10x - 40x is 10 to 40 times longer than normal application execution time. If normal application execution time is 5 seconds, estimated collection time is 50 to 200 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) 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 Deadlocks and Data Races** preset analysis type (and ensure your settings match the image above) before proceeding.

Key Terms

Analysis

Run Analysis

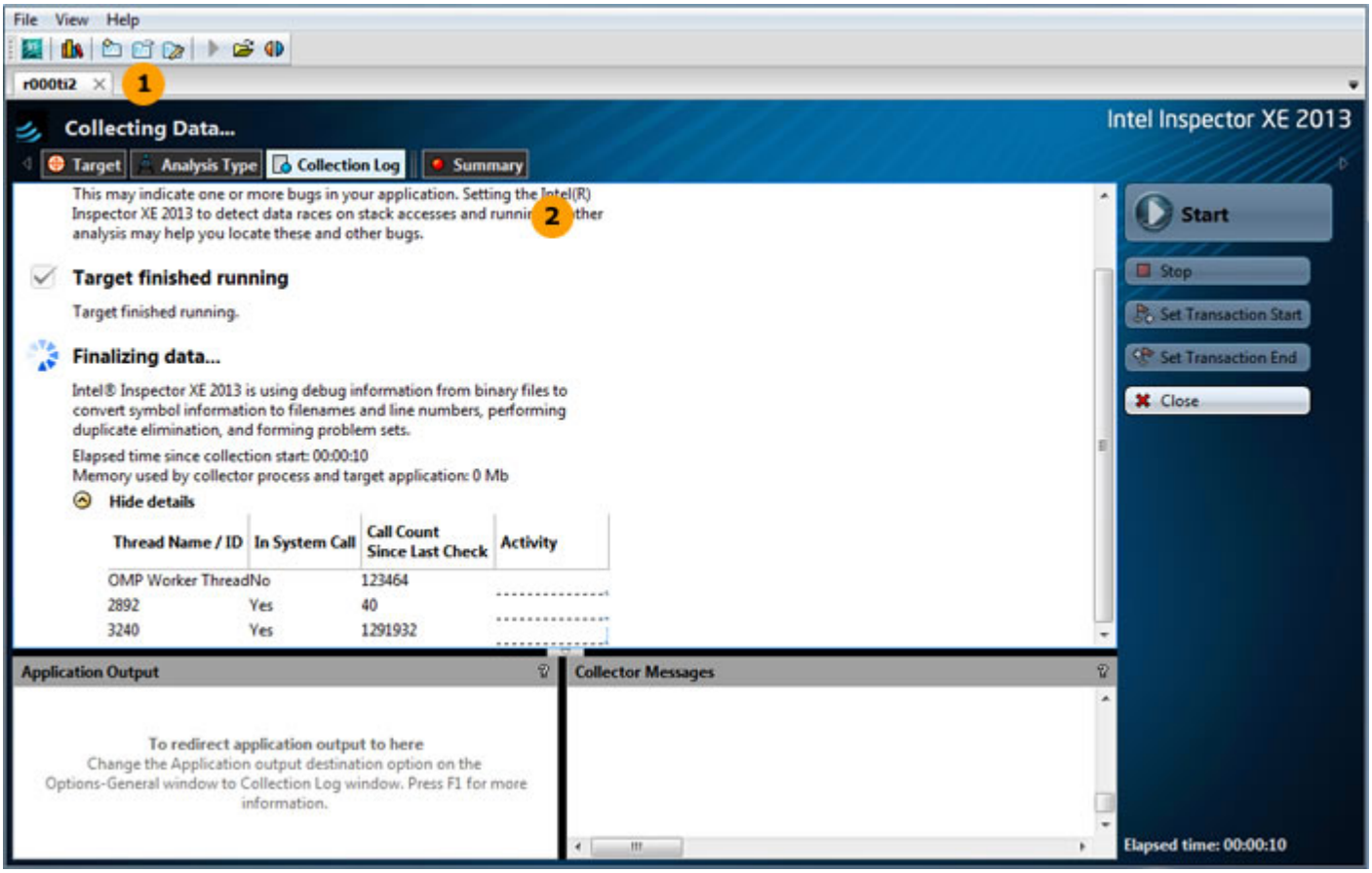
To find threading errors that may need fixing, run a threading error analysis.

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 (Visual Studio* IDE)
  - `Inspector\Projects\threading_issues\` directory (Standalone Intel Inspector XE GUI)
- 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 **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).

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

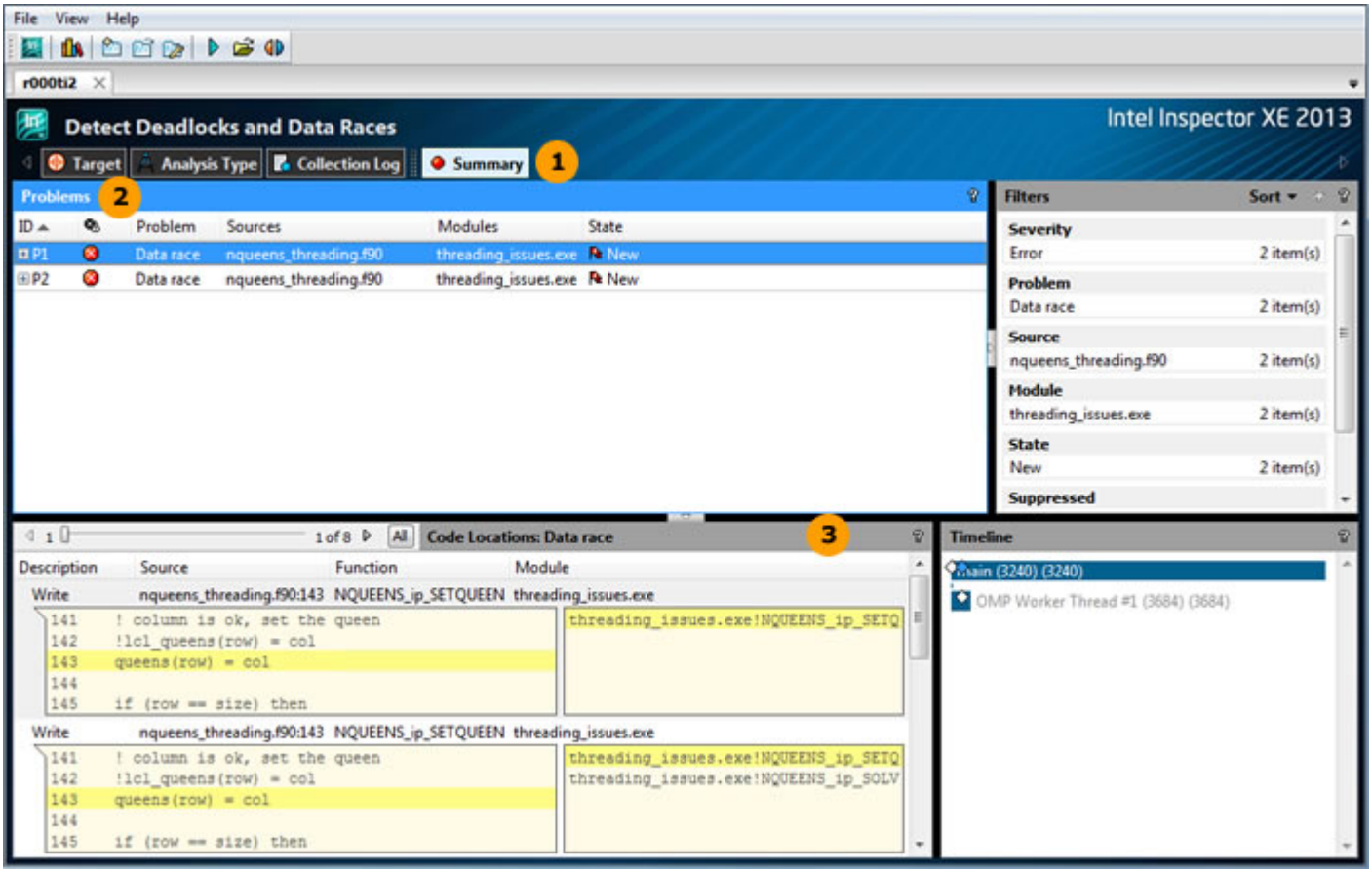
- Analysis
- Collection
- Finalization

Choose Problem

To start exploring a detected threading error:

- Understand Summary window panes.
- Choose a problem.

Understand Summary Window Panes



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.

Try viewing information for different occurrences of the highlighted problem by clicking the ▷ or ◁ controls, or clicking the **All** button.

Choose a Problem

When you are finished experimenting:

1. Click the ⊞ icon for both **Data race** problem sets.
2. Double-click the data row for one of the **Data race** problems at line 148 in the `nqueens_threading.f90` file to display the **Sources** window, which provides more visibility into the cause of the error.

Key Terms

- Code location
- Problem

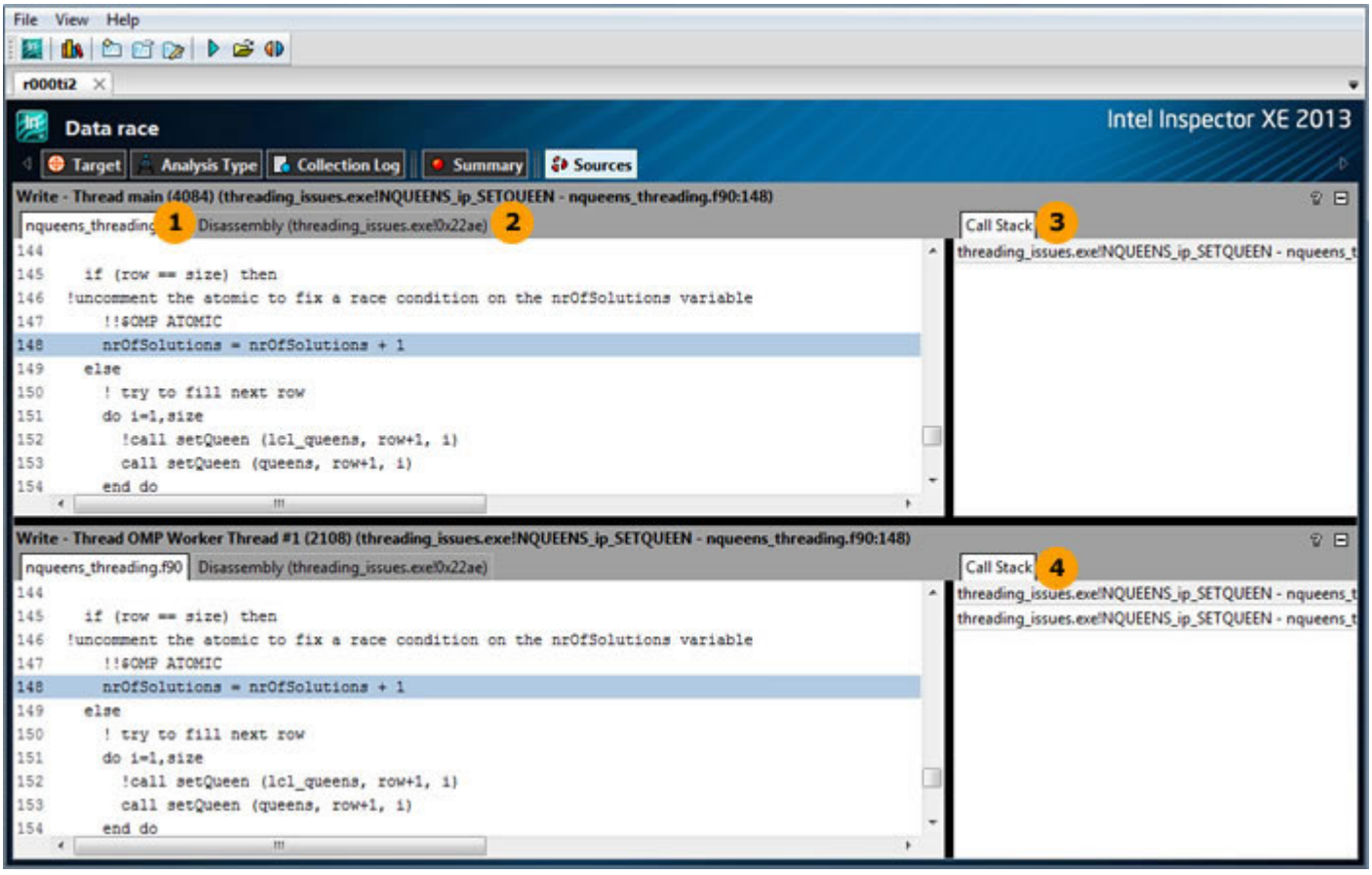
- Problem set
- Result

Interpret Result Data

To determine the cause of the detected threading error:

- Interpret 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 **Data race** problem: A memory **Write**. Notice the source code corresponding to the **Write** code location is highlighted.

2. The **Disassembly** tab shows low-level operations for the **Write** code location in the **Data race** problem.

3. The **Call Stack** tab shows the complete call stack for the **Write** code location in the **Data race** problem.

4 This region shows source, disassembly, and call stack information for another code location in the **Data race** problem: A concurrent memory **Write**.

Use the ⊞/⊟ icons to expand/collapse source, disassembly, and call stack information for each code location region in the **Data race** 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 **Data race** problem type.

Key Terms

- Code location
- Problem

Resolve Issue


To fix the detected threading error:

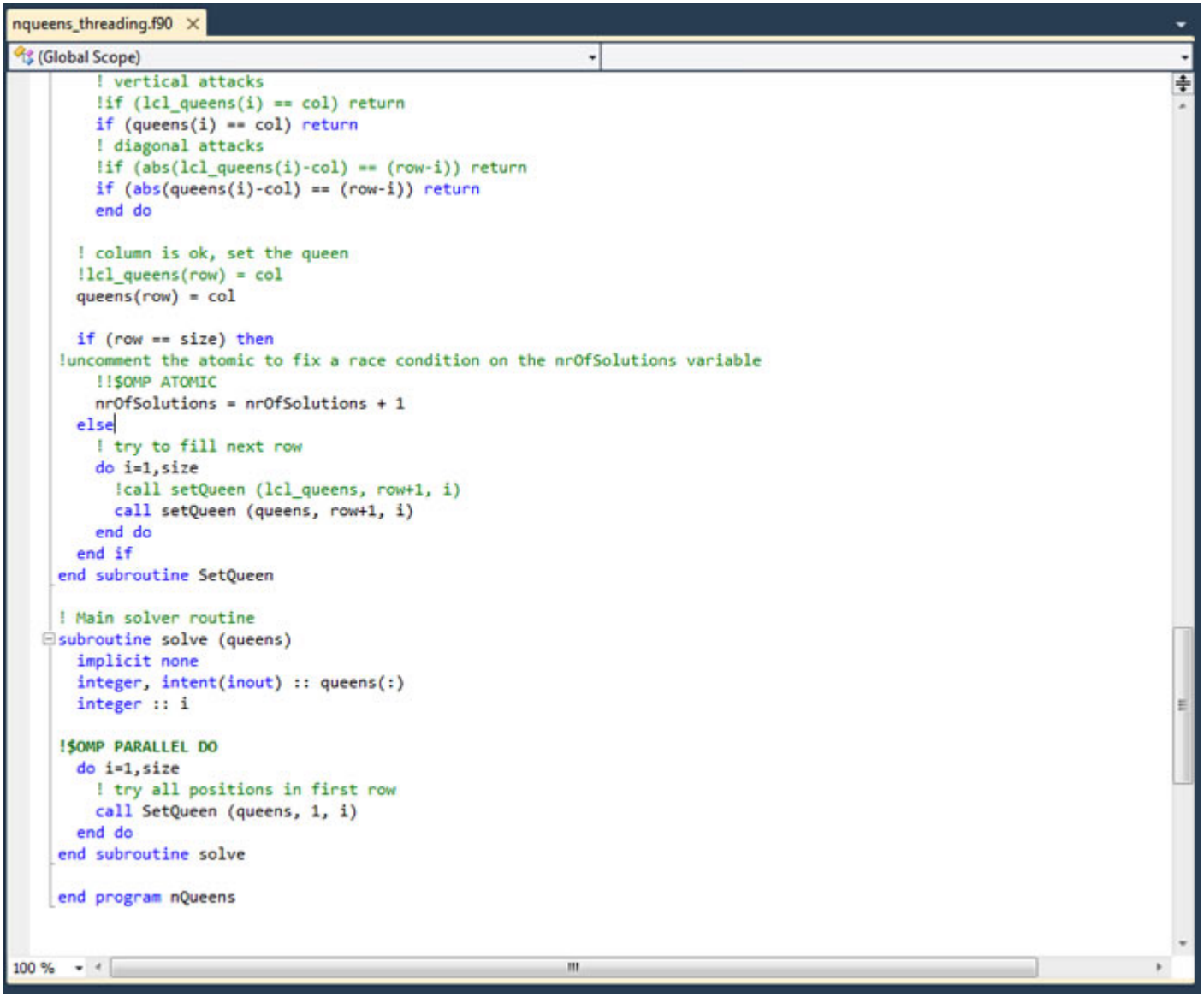
- Investigate the issue.
- Access an editor directly from the Intel Inspector XE.
- Change the source code.

Investigate the Issue

Scroll to near line 146. The embedded commenting in the `nqueens_threading.f90` sample file identifies how to fix the **Data race** problem: Uncomment the atomic on the `nrOfSolutions` variable.

Access an Editor

Double-click near line 146 to open the `nqueens_threading.f90` source file in an editor:



Change the Source Code

1. Uncomment `!!$OMP ATOMIC`.
2. Save your edits (automatic if you are using the Visual Studio* IDE editor).
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


If you are using the Visual Studio* IDE, you can investigate a problem more deeply with an interactive debugging session during analysis.

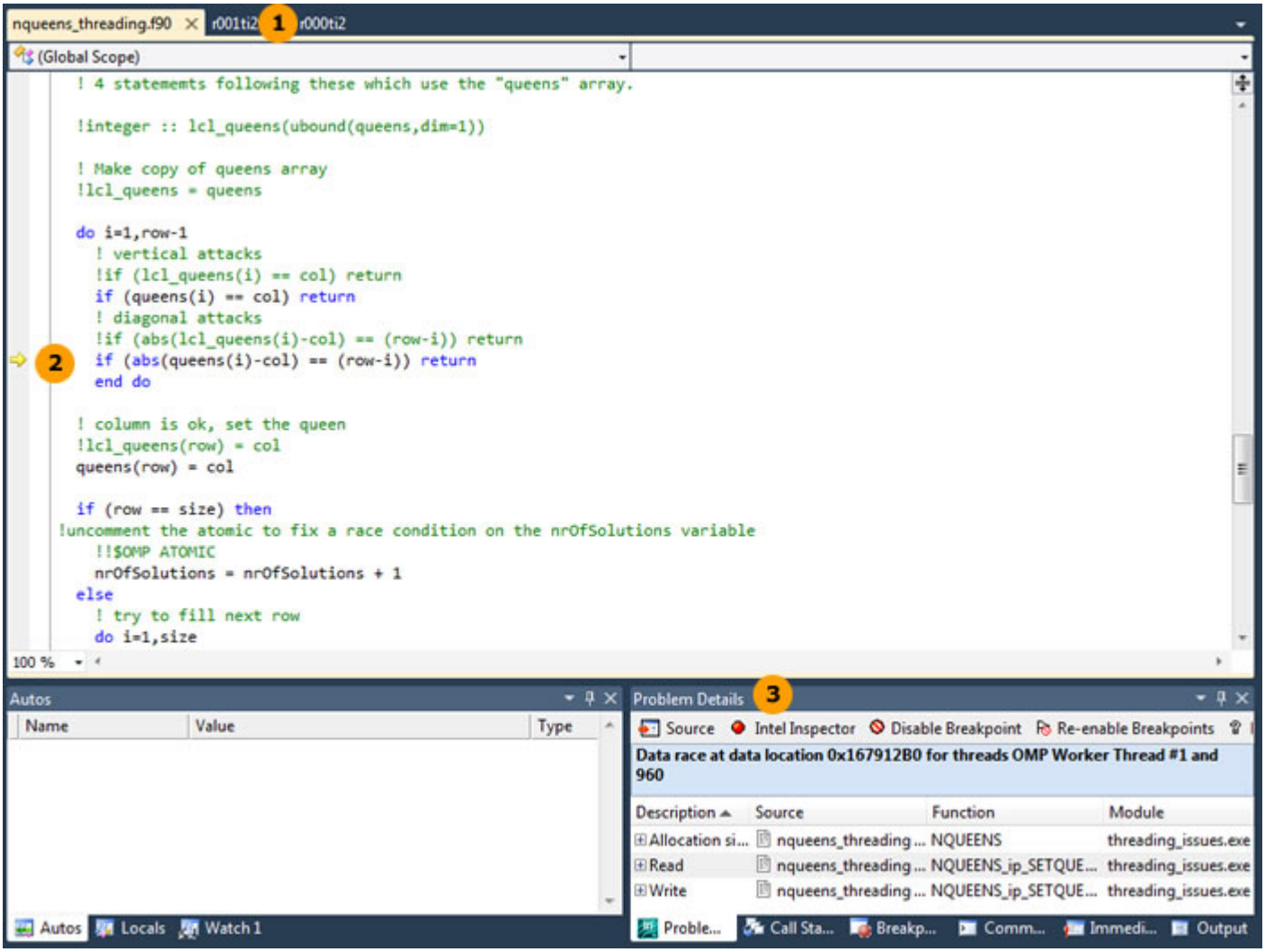
- Launch an interactive debugging session.
- Experiment using standard debugging commands.
- Stop the analysis and interactive debugging session.

Launch an Interactive Debugging Session

In the **Problems** pane on the **Summary** window, right-click the data row for the **Data race** problem set with three **Data race** problems (at lines 135, 138, and 143) to display a context menu, then choose **Debug this problem** to:

- Launch a new analysis, of the same type, optimized to detect only the **Data race** problems in this problem set.
- Halt application execution and open a debugging session when the Intel Inspector XE detects the first occurrence of the **Data race** problems.


NOTE If the Intel Inspector XE tells you to rebuild the project, click the **OK** button to continue using the original build.



1. Intel Inspector XE launches a new analysis, **r001ti2**, with the same configuration settings used to create the **r000ti2** result, except the analysis is optimized to detect only the **Data race** problems in the selected problem set.

2. Intel Inspector XE halts execution and opens a debugging session when it detects the first occurrence of the **Data race** problems. In the Visual Studio* IDE, this problem breakpoint is indicated by a yellow arrow at the appropriate source line.

3. The **Problem Details** pane appears when the problem breakpoint occurs. Use this pane to view the same problem details found in the Intel Inspector XE result, duplicated within the debugger workspace for convenient referencing while you examine the application state for debugging purposes.

Experiment Using Standard Debugging Commands

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

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. Click the **Intel Inspector** button on the **Problem Details** pane to display the still-running **r001ti2** result.
2. Click the **Stop** button on the Intel Inspector XE **Command** toolbar.
3. Click the **Collection Log** button for the new **r001ti2** result to wait for finalization to complete.

When finalization is complete, the Intel Inspector XE displays a **Summary** window for the new **r001ti2** result that contains only the selected **Data race** problems in the selected problem set.

TIP Intel Inspector XE automatically adjusts the debugging session analysis to return to the **Data race** 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 **Data race** problem.

Key Terms

- Problem
- Problem breakpoint
- Problem set

Rebuild and Rerun Analysis

To check if your edits resolved the problems in the **Data race** problem set:

- 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`.

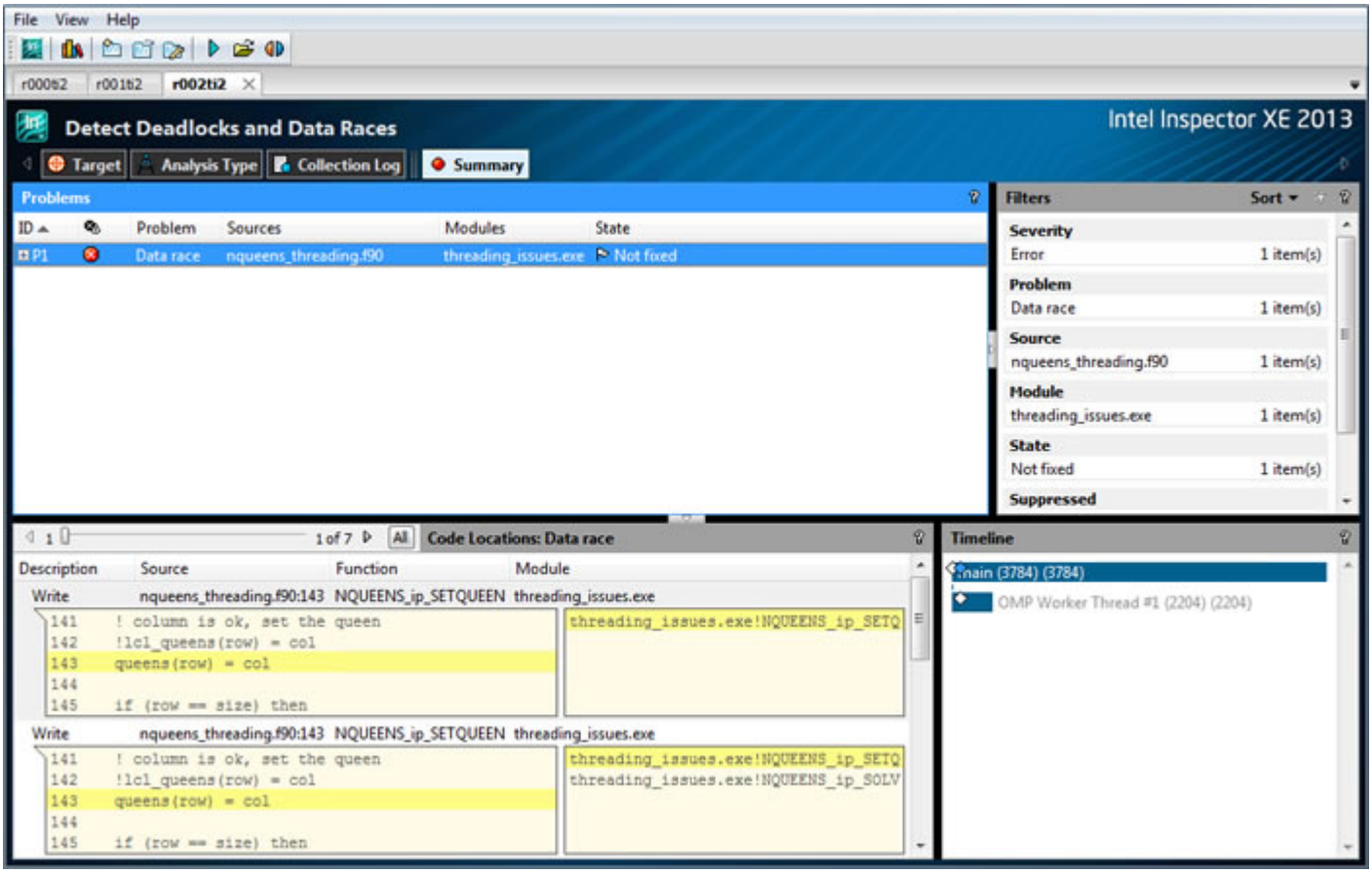
NOTE Do not be concerned if you see a `Build: 2 succeeded, 1 failed` message because of static analysis build errors.

Rerun the Analysis

To run another analysis of the same analysis type:

- From the Visual Studio* menu, choose **Tools** > **Intel Inspector XE 2013** > **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:



Notice the Intel Inspector XE:

- Created a new result tab: **r002ti2**.
- Detects only one threading problem set.

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. Set up	If you used the Visual Studio* IDE: You chose a project; verified the project is set to produce the most accurate and complete analysis 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 and linked in debug mode using the following options produce the most accurate and complete analysis results: `/debug:full`, `/Od`, `/libs:dll/threads` or `libs:dll/threads/dbglibs`, and `/check:none`.
2. Collect result	You chose an analysis type and ran an analysis. During analysis, the Intel Inspector XE: • Ran the application, identified errors that may need handling, collected a result, and displayed the result in a result tab. • 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. Investigate result	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 if you used the Visual Studio* IDE.	• Key terms: A *code location* is a fact the Intel Inspector XE observes at a source code location. A *problem* is one or more occurrences of a detected issue. A *problem set* is a group of problems with a common problem type and a shared code location that might 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 on the **Summary** window to navigate to the **Sources** window. Click the **Summary** button on the **Sources** window to return to the **Summary** window.

Step	Tutorial Recap	Key Tutorial Take-aways
		• Right-click various places on the **Summary** or **Sources** window to display a context menu, then choose **Explain Problem** to access more information on interpreting and resolving the problem. • Double-click a code location on the **Sources** window to open an editor. • Right-click a problem, then choose **Debug This Problem** to launch an interactive debugging session (Visual Studio* IDE).
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.

4 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 the Visual Studio* debugger, a problem breakpoint is indicated by a yellow arrow at the source line where execution halts.

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.