Intel® Inspector Tutorial: Analyzing Threading Errors in a C++ Sample Application for Windows* OS

Contents

Intel® Inspector Tutorial: Analyzing Threading Errors in a C++ Sample Application for Windows* OS

1

NOTE
Sanitizers in Intel® oneAPI DPC++/C++ Compiler and Intel® Fortran Compiler help effectively pinpoint memory, address, threading, and undefined behavior related issues early in the development process. Sanitizers offer faster time to results, fewer false positives, and improved compiler integration compared to Intel Inspector. As a result, **Intel Inspector will no longer be included in the Intel® HPC Toolkit**. It continues to be downloadable as a standalone package and it will be discontinued in 2025 or later. Customers who have purchased Intel® Priority Support will continue to receive support. Please see Intel Inspector deprecation article for more information.

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows* and Linux* operating systems.

This tutorial - and C++ sample application you can use to follow along - show how to use the Intel inspector on a Windows* platform to analyze threading errors.

Intel® Inspector is available as a standalone product and as part of the following products:

- Intel® HPC Toolkit
- Intel® IoT Toolkit

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.** Rebuild and reinspect the application. It was last updated for the 2017 product release.
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. • Define key Intel Inspector 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 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 Intel® Software Documentation Library.

These sites also offer tutorials for other Intel products.

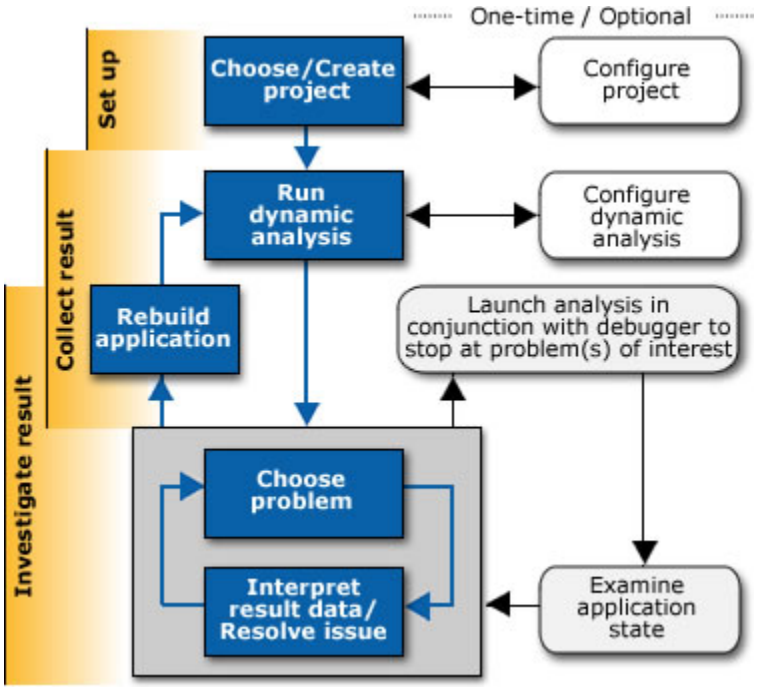
In addition, you can find more resources in:

- Getting Started with Intel® Inspector
- Intel Inspector

Threading Error Analysis Use Case

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

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



This tutorial implements this workflow in the following manner:

Step 1: Unpack and set up sample for analysis	Do one of the followin • In the Visual Studi ject, then build and test an application to inspect for threadi • In the Intel Inspec uild an application to inspect for threading errors, set up the Intel Inspector environment, and create a new Intel Inspector 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.
Step 4: Check your work	Rebuild the application and rerun the threading error analysis.

Navigation Quick Start

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

- Set up the Intel Inspector environment.
- Open the Intel Inspector standalone GUI.
- Navigate the Intel Inspector standalone GUI.
- Navigate the Intel Inspector in the Microsoft Visual Studio* IDE.
- Navigate the Intel Inspector result tabs.

Set up the Intel Inspector Environment

NOTE
Setting up the Intel Inspector environment is necessary only if you plan to use the `inspxe-gui` command to launch the Intel Inspector standalone GUI or the `inspxe-cl` command to run the command line interface.

For the standaloneIntel Inspector, run the `<inspector-install-dir>\inspxe-vars.bat` command.

The default installation path, `<inspector-install-dir>`, is inside `C:\Program Files (x86)\IntelSWTools\` (on certain systems, instead of `Program Files (x86)`, the directory name is `Program Files`).

For the application as part of an Intel® HPC Toolkit or Intel® IoT Toolkit installation, run the `<oneapi-install-dir>\env\vars.bat` command. The default installation path, `<oneapi-install-dir>`, is inside `C:\Program Files (x86)\Intel\oneAPI` .

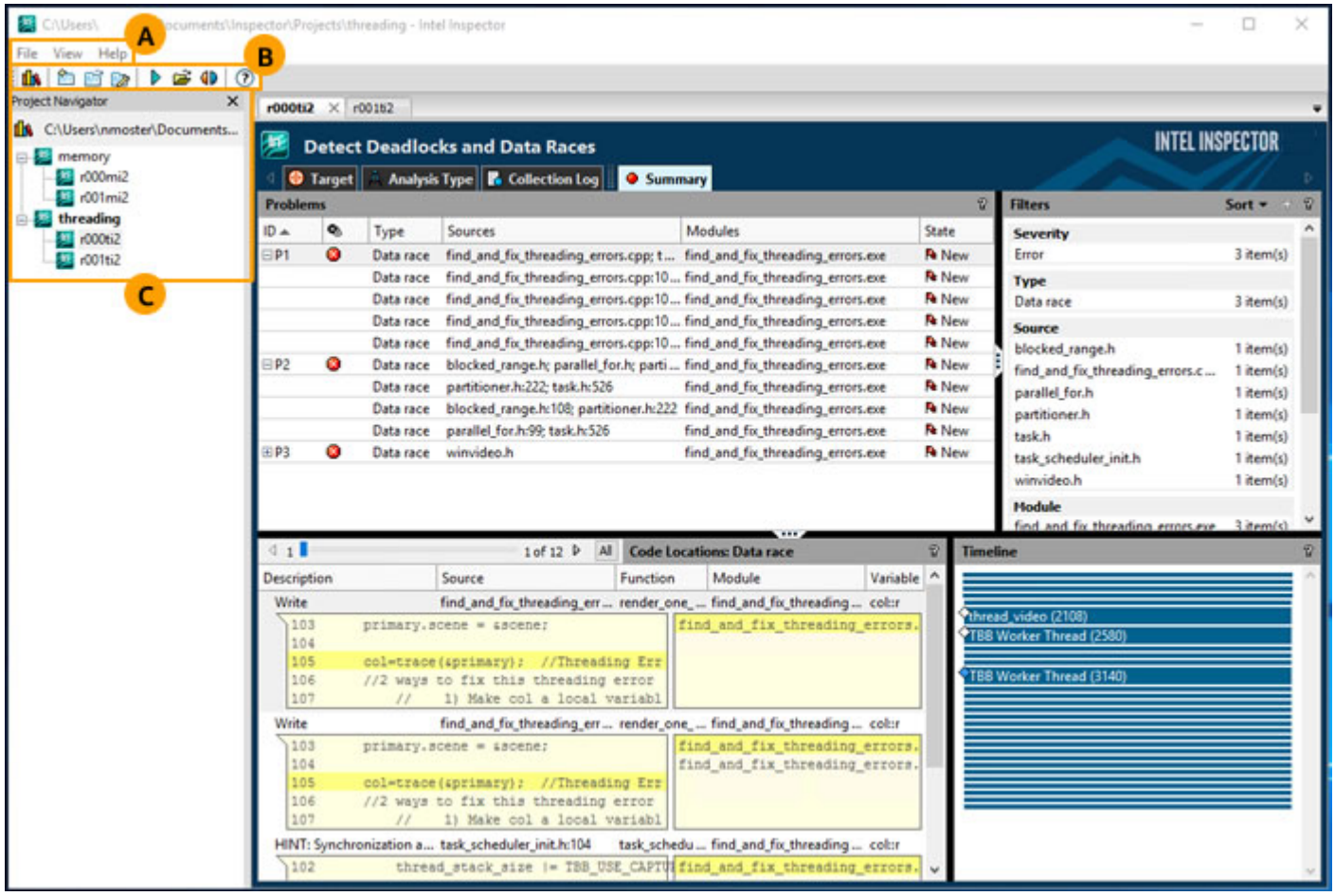
Open the Intel Inspector Standalone GUI

For the Intel Inspector standalone GUI, do one of the following:

- Run the `inspxe-gui` command.

- From the Microsoft Windows* 7 **Start** menu, select **Intel Inspector [version]**.
- From the Microsoft Windows* 8/8.1/10 **All Apps** screen, select **Intel Inspector [version]**.

Navigate the Intel Inspector Standalone GUI



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

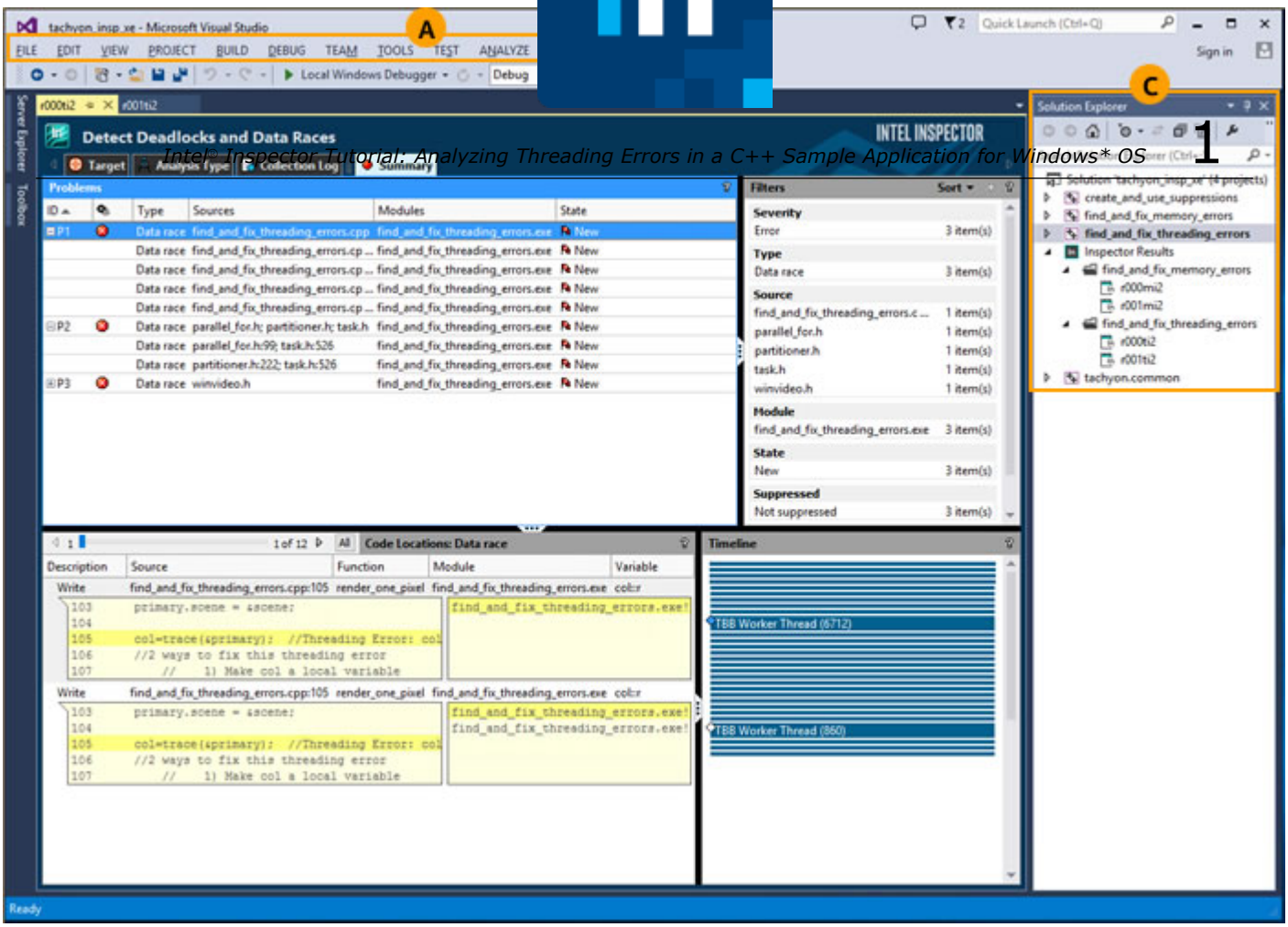
A Use the menu to create projects and dynamic analysis results, import result archive files and results from other Intel error-detection products, open projects and results, compare results, configure projects, set various options, and access the *Getting Started* page and *Help*.

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

C 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.

Navigate the Intel Inspector in the Micro IDE



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

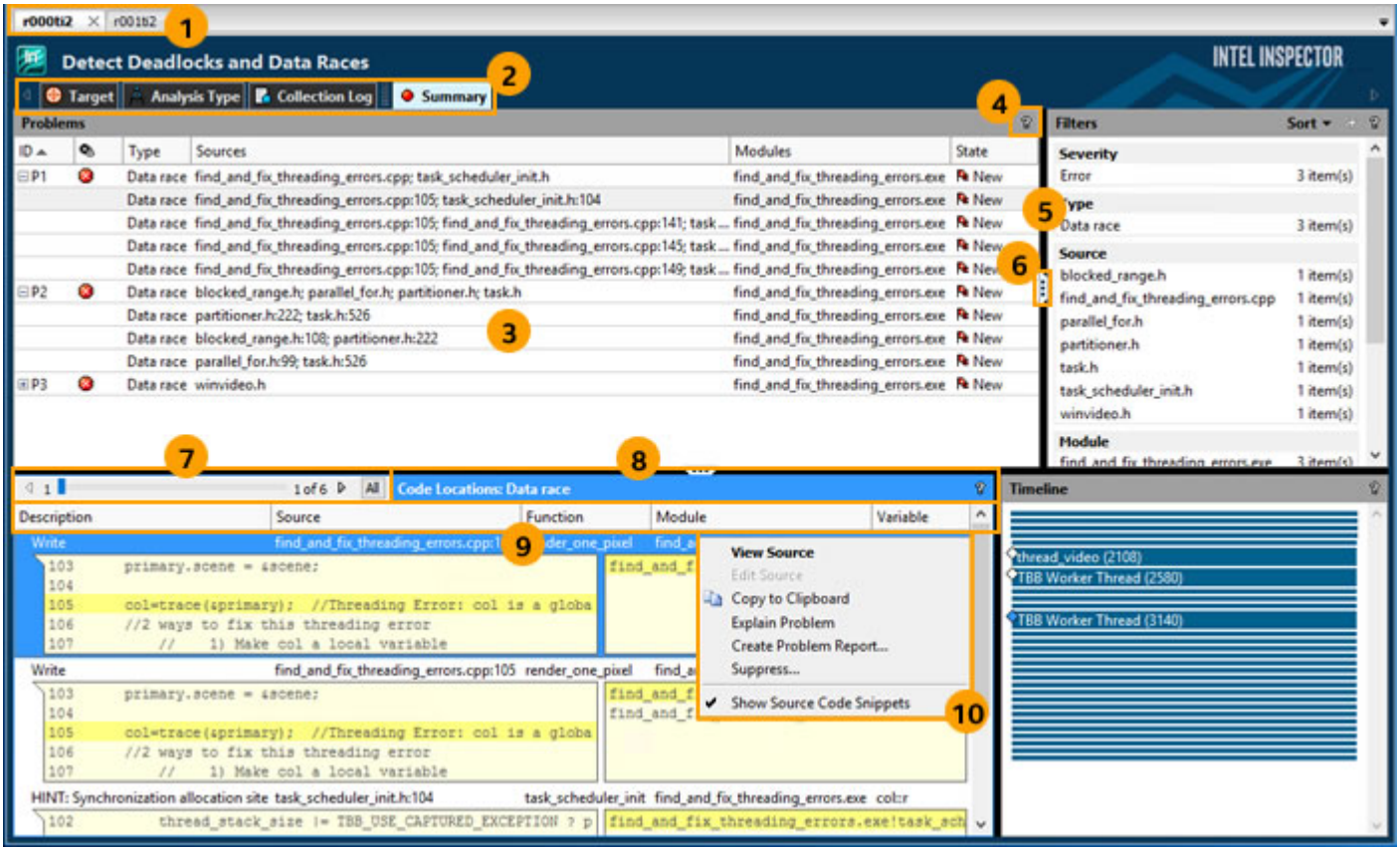
A Use the **Tools > Intel Inspector [version]** menu to create analysis results; compare results; and import result archive files, results not associated with a project, and results from other Intel error-detection products into the current project.

B Use the **Intel Inspector** toolbar to create analysis results, compare results, configure projects, and open documentation resources.

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

- Use the **Intel Inspector [version]** menu on the **Solution Explorer** project context menu to create analysis results and configure projects.
- Use the context menu on a result in the project result folder to open results, create analysis results, refinalize (re-resolve) results, export result archive files, and manage results.

Navigate the Intel Inspector 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 adj the pane (and possibly in adjacent panes).

8 Use title bars to identify window panes

9 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.

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

Visual Studio* IDE: Choose Project and Build Application

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

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

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

- Get software tools.
- Open a Visual Studio* solution.
- Choose a startup project.
- Understand optimal/linker settings.
- Build and test the application.

Get Software Tools

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

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

Acquire the Intel Inspector

If you do not already have access to the Intel Inspector, you can download it for free as follows:

- standalone version
- part of the Intel® oneAPI HPC Toolkit

Install and Set Up the Intel Inspector Sample Applications

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

NOTE

- 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 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 `tachyon_insp_xe\vc10\tachyon_insp_xe.sln` file to display the **tachyon_insp_xe** solution in the **Solution Explorer**.

NOTE
The `tachyon_insp_xe.sln` solution was created using the Visual Studio* 2010 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 **find_and_fix_threading_errors** project is not the startup project (project typeface in the **Solution Explorer** is not bold),

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

Understand Optimal Compiler/Linker Settings

You can use the Intel® Inspector to analyze memory and threading errors in both debug and release modes of C++ and Fortran binaries; however, applications compiled/linked in debug mode using the following settings produce the most accurate and complete analysis results.

Compiler/Linker Property	Correct C/C++ Setting	Impact If Not Set Correctly
Debug information	Enabled (`/Zi` or `/ZI`)	Missing file/line information
Optimization	Disabled (`/Od`)	Incorrect file/line information
Dynamic runtime library	Selected (`/MD` or `/MDd`)	False positives or missing code locations
Basic runtime error checks	Disabled (do not use `/RTC`; **Default** option in Visual Studio* IDE)	False positives

Compiler/Linker Property	Correct Fortran Setting	Impact If Not Set Correctly
Debug information	Enabled (`/debug:full`)	Missing file/line information
Optimization	Disabled (`/Od`)	Incorrect file/line information
Dynamic runtime library	Selected (`/libs:dll/threads` or `libs:dll/threads/dbglibs`)	False positives or missing code locations
Basic runtime error checks	None (`/check:none`)	False positives

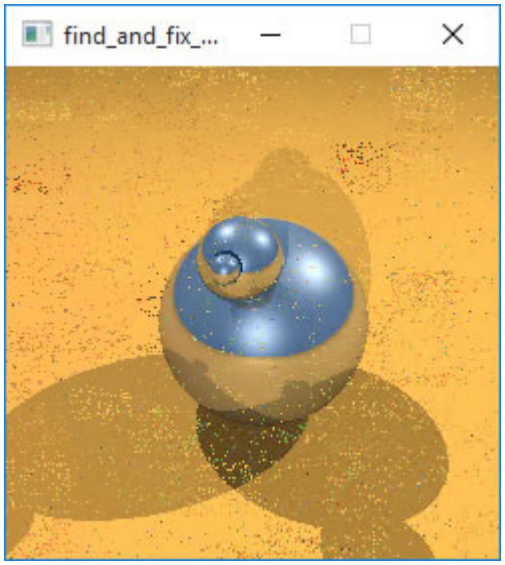
The sample code is already set. Follow these steps if you need to change settings for your application.

1. If the **Solutions Configuration** drop-down on the Visual Studio* **Standard** toolbar is set to **Release**, change it to **Debug**.
2. Right-click the **find_and_fix_threading_errors** project in the **Solution Explorer**, then choose **Properties** to display the **Property Pages** dialog box.

3. In the left pane, choose **Configuration** **+**.
   - Choose **General** and verify the **Debu** **mat** field is set to **Program Database (/Zi)** or **Program Database for Ed** .
   - Choose **Optimization** and verify the s set to **Disabled (/Od)**.
   - Choose **Code Generation**. Verify the **Runtime Library** field is set to **Multi-threaded DLL (/MD)** or **Multi-threaded Debug DLL (/MDd)** and the **Basic Runtime Checks** field is set to **Default**.
4. In the left pane, choose **Configuration Properties** > **Linker** > **Debugging** and verify the **Generate Debug Info** field is set to **Yes (/DEBUG)**.
5. Click the **Apply** button, then click the **OK** button to close the **Property Pages** dialog box.

Build and Test the Application

1. Choose **Build** > **Solution** to build all projects 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. Check for non-deterministic application output similar to the following:



Notice the off-color dots in the image. The cause: Threading errors.

Standalone GUI: Build Application and Create New Project

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

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

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

- Get software tools.
- Understand optimal compiler/linker settings.
- Build the application.
- Verify the application runs outside the Intel Inspector.
- Set up the Intel Inspector environment.
- Open the Intel Inspector standalone GUI.

- Create a new project.

Get Software Tools

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

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

Acquire the Intel Inspector

If you do not already have access to the Intel Inspector, you can download it for free as follows:

- standalone version
- part of the Intel® oneAPI HPC Toolkit

Install and Set Up the Intel Inspector Sample Applications

1. Copy the `tachyon_inxp_xe.zip` file from the `<install-dir>\samples\<locale>\C++\` directory to a writable directory or share on your system. The default `<install-dir>` is below `C:\Program Files (x86)\Intel\` (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 `tachyon_insp_xe` directory.

NOTE

- 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 features; they do not represent best practices for creating code.

Understand Optimal Compiler/Linker Settings

You can use the Intel® Inspector to analyze memory and threading errors in both debug and release modes of C++ and Fortran binaries; however, applications compiled/linked in debug mode using the following settings produce the most accurate and complete analysis results.

Compiler/Linker Property	Correct C/C++ Setting	Impact If Not Set Correctly
Debug information	Enabled (`/Zi` or `/ZI`)	Missing file/line information
Optimization	Disabled (`/Od`)	Incorrect file/line information
Dynamic runtime library	Selected (`/MD` or `/MDd`)	False positives or missing code locations
Basic runtime error checks	Disabled (do not use `/RTC`; **Default** option in Visual Studio* IDE)	False positives

Compiler/Linker Property	Correct Fortran Setting	Impact If Not Set Correctly
Debug information	Enabled (`/debug:full`)	Missing file/line information
Optimization	Disabled (`/Od`)	Incorrect file/line information

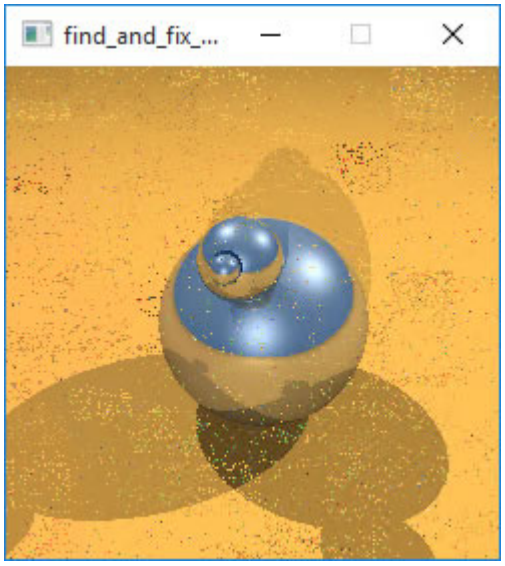
Compiler/Linker Property	Correc	Impact If Not Set Correctly
Dynamic runtime library	Selecte ds or libs dbglibs)	False positives or missing code locations
Basic runtime error checks	None (/RTC1 /RTCs)	False positives

Build the Application

1. Find **Visual Studio Tools** for your Visual Studio* and OS version, and select one of the command prompt shortcuts. For example, from the Microsoft Windows* 10 **All Apps** screen, select **Visual Studio 2013** > **Visual Studio Tools** > **VS2013 x64 Native Tools Command Prompt**.
2. In the command prompt window, change directory to the `tachyon_inspxe\` directory in its unzipped location.
3. If you are using Microsoft Visual Studio* version 2012 or later, type `devenv vc10\tachyon_insp_xe.sln /Upgrade` to convert the `tachyon_insp_xe.sln` solution.
4. Type `devenv vc10\tachyon_insp_xe.sln /Build` to build all projects in the solution.

Verify the Application Runs Outside the Intel Inspector

1. Change directory to `vc10\find_and_fix_threading_errors_Win32\Debug\`.
2. Type `find_and_fix_threading_errors.exe ..\..\..\dat\simpleballs.dat` to execute the application.
3. Check for non-deterministic application output similar to the following:



Notice the off-color dots in the image. The cause: Threading errors.

Tip
Keep the command prompt window open.

Set up the Intel Inspector Environment

NOTE
Setting up the Intel Inspector environment is necessary only if you plan to use the `inspxe-gui` command to launch the Intel Inspector standalone GUI or the `inspxe-cl` command to run the command line interface.

For the standalone Intel Inspector, run the `<inspector-install-dir>\inspxe-vars.bat` command.

The default installation path, `<inspector-install-dir>`, is below `C:\Program Files (x86)\Intel\oneAPI\inspector` (on certain systems, instead of `Program Files (x86)`, the directory name is `Program Files`).

For the application as part of an Intel® HPC Toolkit or Intel® IoT Toolkit installation, run the `<oneapi-install-dir>\env\vars.bat` command. The default installation path, `<oneapi-install-dir>`, is inside `C:\Program Files (x86)\Intel\oneAPI` .

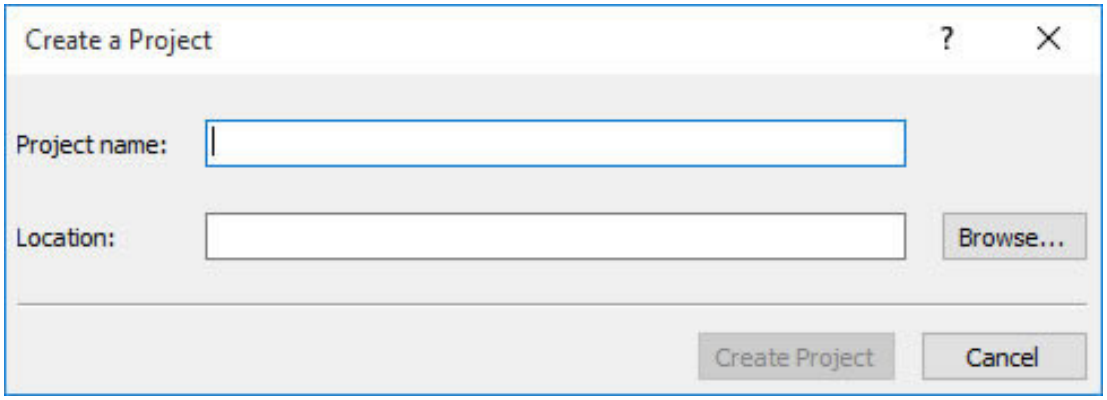
Open the Intel Inspector Standalone GUI

For the Intel Inspector standalone GUI, do one of the following:

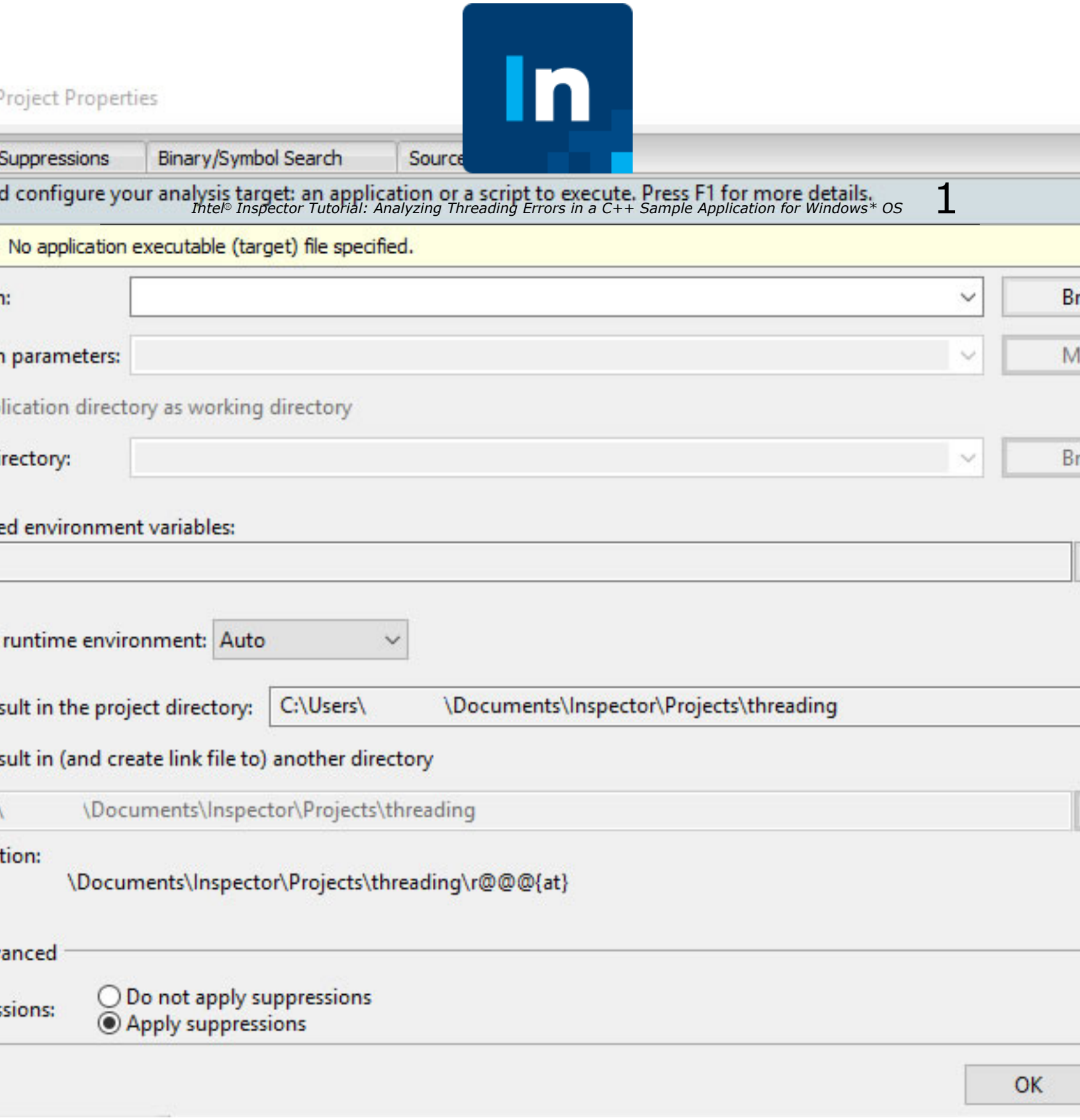
- Run the `inspxe-gui` command.
- From the Microsoft Windows* **Start** menu, search for **Intel Inspector**.

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`. Then click the **Create project** button to create a `config.inspxeproj` file in the `\Inspector\Projects\threading\` directory (default location) and display a dialog box similar to the following:



3. Click the **Browse...** button next to the **Application** field and select the `tachyon_insp_xe \vc10\find_and_fix_threading_errors_Win32\Debug\find_and_find_threading_errors.exe` application. Notice the Intel Inspector autofills the project **Working directory** field for you.

4. Click the **Modify** button next to the **Application Parameters** field. In the **Application Parameters** window, click the **Browse to insert file path** button, change the **Select the file to launch** window to show all files, select the `tachyon_insp_xe\dat\simpleballs.dat` file, click the **OK** button to close the **Application Parameters** window, 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**.)

Configure Analysis

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

Intel Inspector 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:

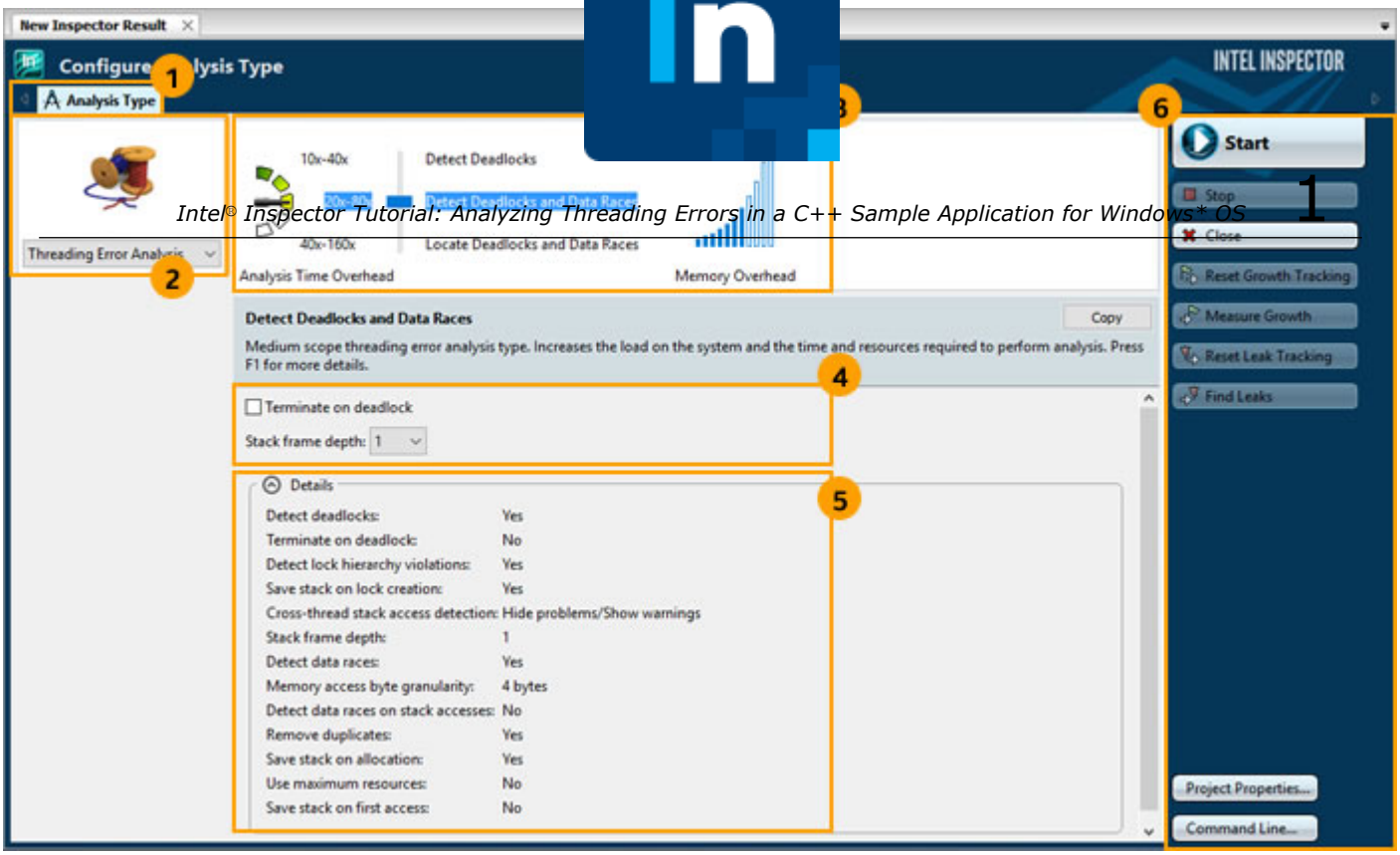
- Understand the Analysis Type window.
- Choose a threading error analysis type.

Understand the Analysis Type Window

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

1. Do one of the following:
   - From the Visual Studio* menu, choose **Tools** > **Intel Inspector [version]** > **New Analysis...**.
   - From the Intel Inspector standalone GUI menu, choose **File** > **New** > **Analysis...**.
2. In the Analysis Type drop-down list, choose **Threading Error Analysis**.
3. Use the configuration slider to choose the **Detect Deadlocks and Data Races** preset analysis type.

NOTE
Your window may also show three interactive debugger radio buttons.



1 Use the **Navigation** toolbar to navigate among the Intel Inspector 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**.

> **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.

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 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. Use the **Command Line** button to open the **Corresponding inspxe-cl Command Options** dialog box, where you can review - and, if desired, copy to the clipboard - the command to perform the same analysis using the Intel Inspector command line interface (`inspxe-cl` command).

Choose a Threading Error Analysis Type

Make sure your settings match those in the image above before proceeding. If any interactive debugger radio buttons are displayed, make sure the **Analyze without Debugger** radio button is selected.

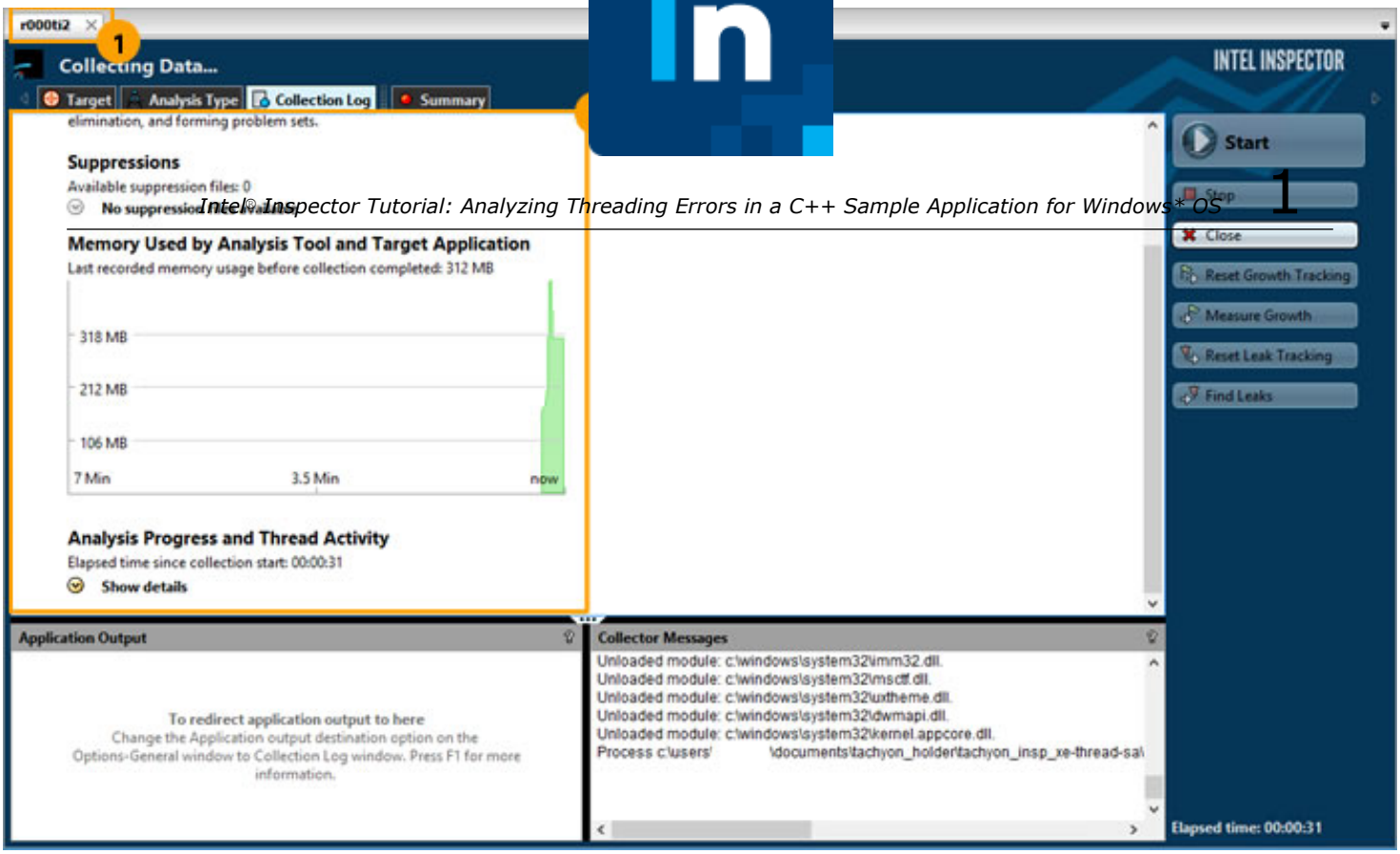
Run Analysis

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

To find threading errors that may need fixing, click the **Start** button on the **Analysis Type** window. The Intel Inspector:

- Executes the `find_and_fix_threading_errors.exe` application.
- Identifies threading errors that may need handling.
- Collects the result in a directory in the:
  - `tachyon_insp_xe\vc10\My Inspector Results - find_and_fix_threading_errors\` directory (Visual Studio* IDE)
  - `Inspector\Projects\threading\` directory (Intel Inspector standalone GUI)
- Finalizes the result.

During analysis, the Intel Inspector 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 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 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 standalone GUI. You can also use the Intel Inspector command line interface (`inspxe-cl` command) to run an analysis.

The **Summary** window automatically displays after analysis completes successfully.

Choose Problem

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

To start exploring a detected threading error:

- Understand key terms.
- Understand Summary window panes.
- Choose a problem.

Understand Key Terms

code location: A fact the Intel Inspector observes at a source code location, such as a *write* code location. Previously called an *observation*.

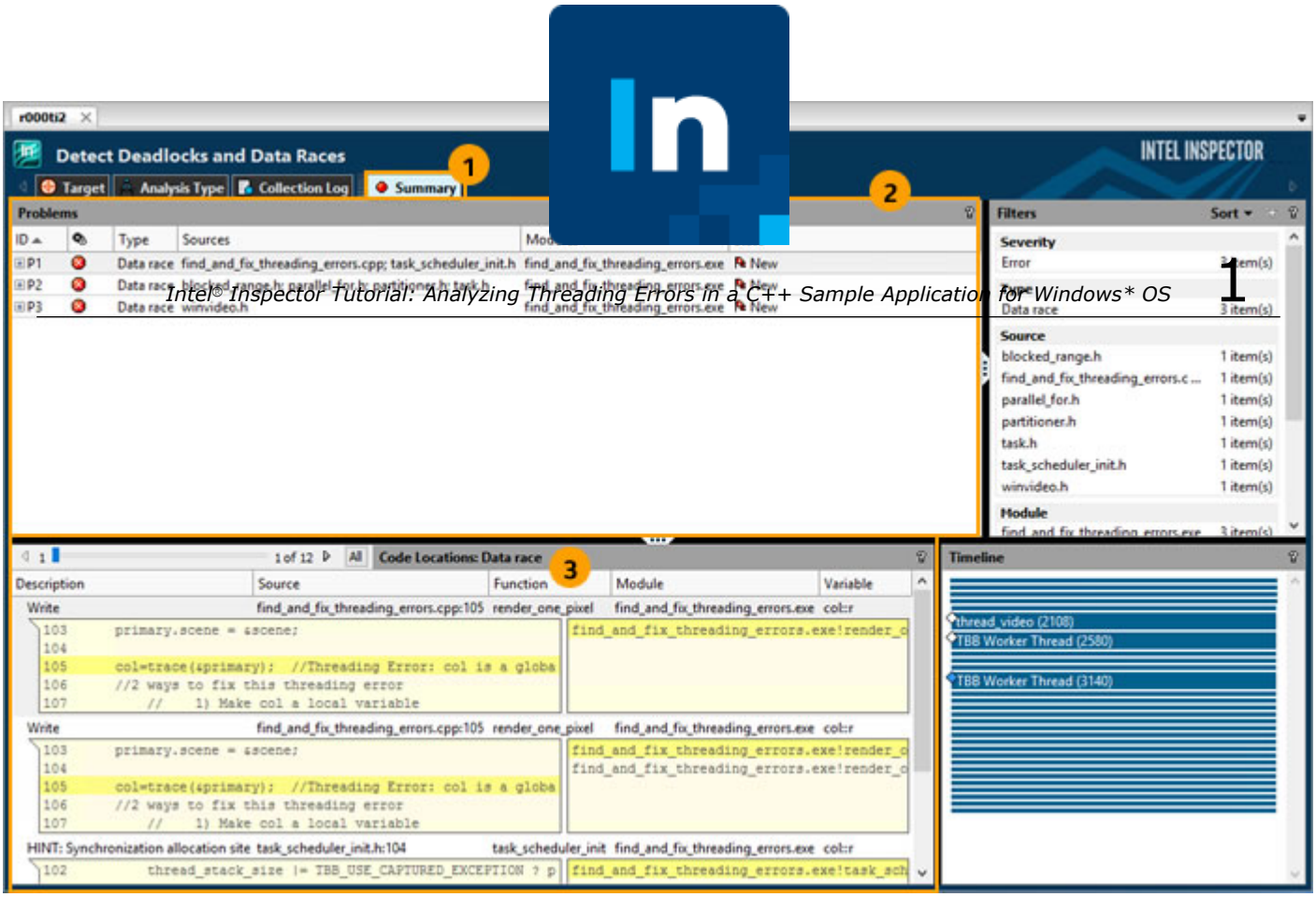
problem: One or more occurrences of a detected issue, such as an uninitialized memory access. Multiple occurrences have the same call stack but a different thread or timestamp. You can view information for a problem as well as for each occurrence.

problem 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 application execution. You can view problem sets only after analysis is complete.

See the Intel® Inspector Glossary for more key terminology.

Understand Summary Window Panes

A **Summary** window similar to the following automatically displays after analysis completes successfully.



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, and call stack information for all code locations in one or all occurrences of the problem(s) highlighted in the **Problems** pane.

Try using the various controls on the slider to view information for different occurrences of the highlighted problem.

Choose a Problem

When you are finished experimenting:

1. Click the

icon for the **Data race** problem set in the `find_and_fix_threading_errors.cpp` source file to view all the problems in the problem set.
2. Double-click the data row for the first **Data race** problem in the problem set to display the **Sources** window, which provides more visibility into the cause of the error.

Interpret Result Data

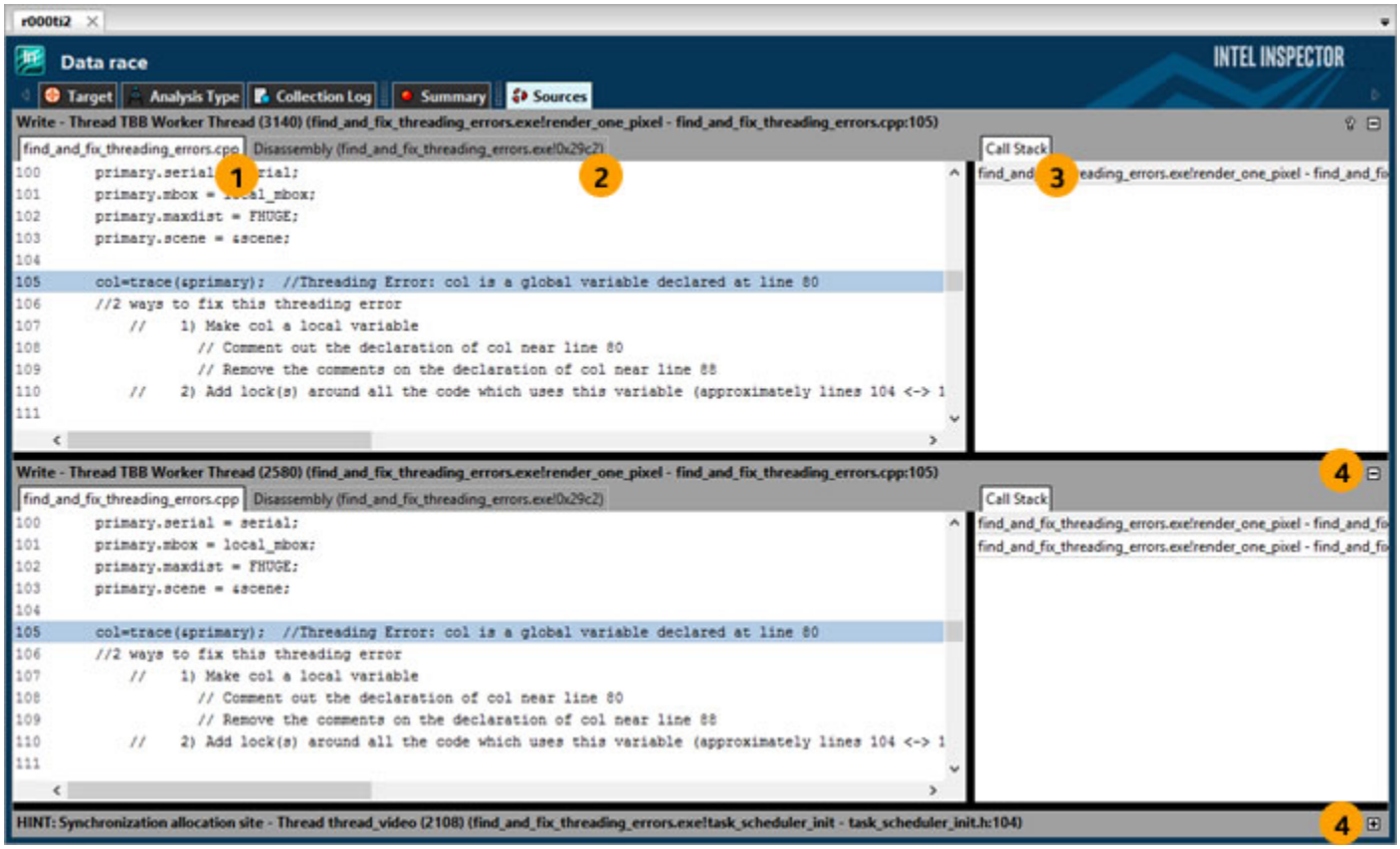
Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

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

A **Sources** window similar to the following displays after you double-click the data row for the first **Data race** problem in the problem set. It provides more visibility into the cause of the error.



1 The **Source** tab shows the complete source surrounding one code location in the **Data race** problem. (Samples are non-deterministic; you may see a memory **Write** or a memory **Read** code location.)

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

3 The **Call Stack** tab shows the complete call stack for the 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. (Samples are non-deterministic; you may see a memory **Write** or a memory **Read** code location.)

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 Help information for the **Data race** problem type.

Resolve Issue

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

To fix the detected threading error:

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

Investigate the Issue

The commenting embedded throughout the `find_and_fix_threading_errors.cpp` sample file reveals the cause of the data race problems: Multiple threads are concurrently accessing the global variable `col`. One possible correction strategy: Change the global variable to a local variable.

Access an Editor

Scroll to near line 80 in either code location region and double-click the `color col;` line in the **Source** tab to open the `find_and_fix_threading_errors.cpp` source file in an editor.

Change the Source Code

1. Comment `color col;` and uncomment `//color col;` near line 88 to localize the variable `col` to the function `render_one_pixel`.
2. Save your edits.
3. Click the result tab to return to the **Sources** window.

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

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

Rebuild and Rerun Analysis

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

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

- 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 Intel Inspector standalone GUI:

1. In the previously opened command prompt window, change directory to the `tachyon_insp_xe\` directory.
2. Type `devenv vc10\tachyon_insp_xe.sln /Clean`.
3. Type `devenv vc10\tachyon_insp_xe.sln /Build`.

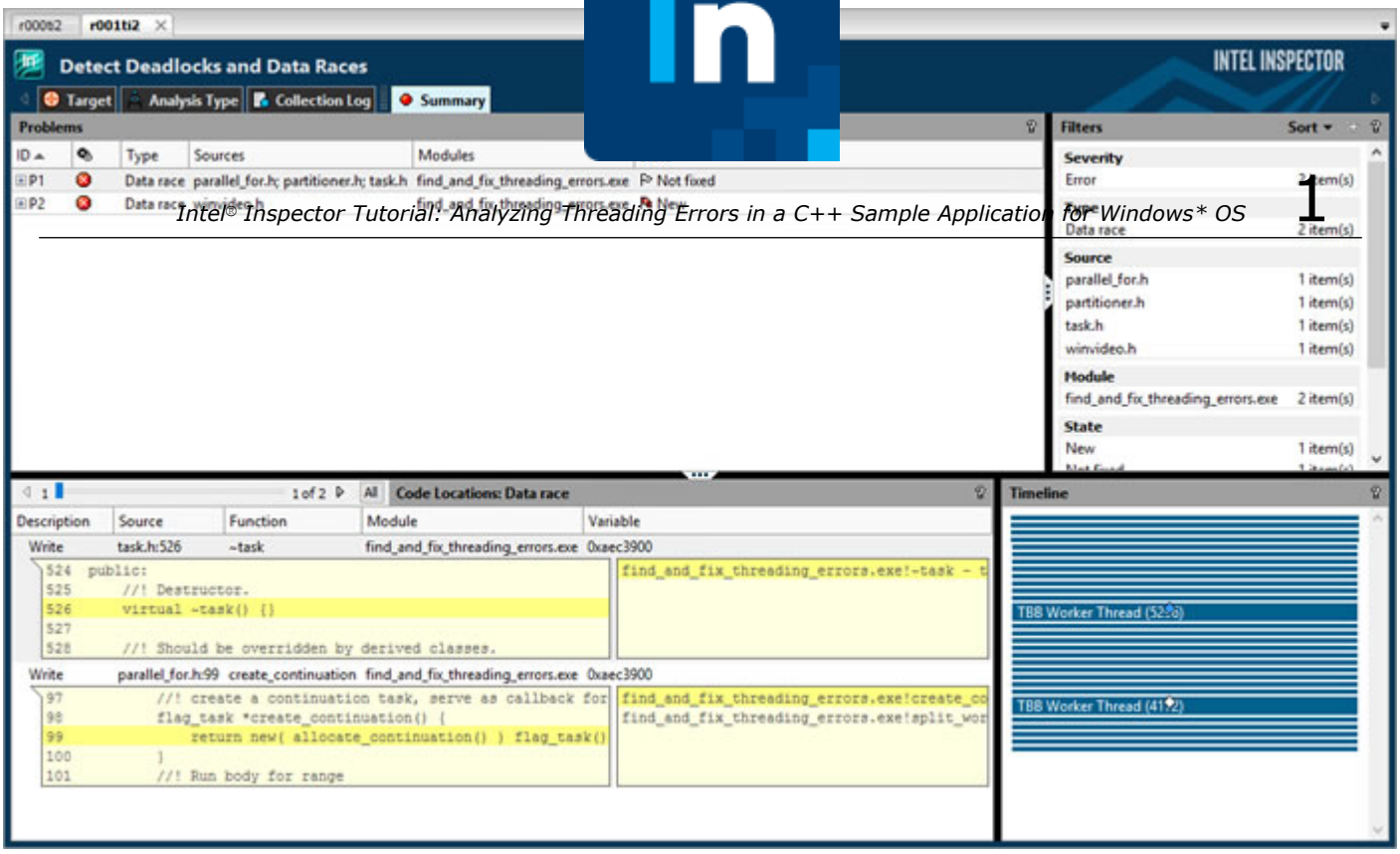
Rerun the Analysis

To run another analysis of the same analysis type:

- From the Visual Studio* menu, choose **Tools** > **Intel Inspector [version]** > **Threading Error Analysis / Detect Deadlocks and Data Races**.
- From the Intel Inspector standalone GUI menu, choose **File** > **New** > **Threading Error Analysis / Detect Deadlocks and Data Races**.

Notice the application output now displays properly.

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



Notice the Intel Inspector:

- Created a new result tab: **r002ti2**.
- No longer detects **Data race** problems in the `find_and_find_threading_errors.cpp` source file.

Summary

Intel® Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications on Windows and Linux* operating systems. This topic is part of a **tutorial** that shows how to find and fix threading errors using the Intel Inspector and a C++ sample application.*

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, and built and ensured the application runs on your system outside the Intel Inspector.	Applications compiled and linked in debug mode using the following options produce the most accurate and complete analysis results: `/Zi` or `/ZI`, `/Od`, `/MD` or `/MDd`, and no `/RTC`.

Step	Tutorial Recap	Key Tutorial Take-aways
	If you used the standalone GUI: You built the application using optimal compiler/linker settings, ensured the application runs on your system outside the Intel Inspector, set up the Intel Inspector environment, and created a project to hold analysis results.	
2. Collect result	You chose an analysis type and ran an analysis. During analysis, the Intel Inspector: • 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 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 (Intel Inspector standalone 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, and changed source code.	• Key terms: A *code location* is a fact the Intel Inspector 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. • 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.
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 to find and fix errors.

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.