Diagnostics View

The Diagnostics view displays all the diagnostics that the Intel® Thread Checker Analysis Engine generated for your program. Diagnostics are categorized by severity and indicate issues or events that may be non-deterministic. These are issues that can lead to indeterminate or incorrect results. Usually they are due to a conflict like a data race, dead lock, or stalled thread.

To help you focus your analysis, you can group, filter, and sort diagnostics according to different criteria.

To..	Do this..
Group diagnostics	Drag and drop column headers to the grey area at the top of the list.
Filter diagnostics out of view	Right-click on a diagnostic and select **Filter Diagnostic**. You can then **Show Filters** you applied.
View applied filters	Right-click in the Diagnostics view tab and select **Show Filters**.
Sort diagnostics	Click a column header to sort diagnostics by that column within each group.
Add a column	Right-click and select **Show Column**.
Remove a column	Right-click and select **Hide Column**.
View source code location	Double-click on a diagnostic.
Understand diagnostic	Right-click on a diagnostic and select **Diagnostic Help**.
Understand column	Right-click on a column and select **Column Help**.

Tips for reviewing diagnostics

There are many ways to review Thread Checker's diagnostics. Each developer will have his or her own methods. Here are a few ideas to get you started.

One approach for reviewing diagnostics is to group by **2nd Access [Best]** category. You can consider diagnostics that occur on the same 2nd Access line number as the same issue. Resolving these first may yield the greatest benefit.

You can also sort by **Short Description** to find Write->Write data-races first and try to fix these. Many times these can be resolved by making the variable private to each thread.

Once you feel comfortable with the changes you have made to your code, you can filter out these diagnostics so they are hidden from view. You can also re-run the Intel® Thread Checker Activity to see if the errors have been resolved.

Accessing Source Code

Double-click a diagnostic in the Diagnostics view to examine detail about an error. If source code is available, it is displayed in any of the four source views. These consist of the Context, Definition, 1st Access, and 2nd Access views. You can configure the source views as individual tabs, or in a split or tiled format via Configure > Options > Intel® Thread Checker. Use the source view toolbar to manipulate the view and navigate your source code.

Access views - Typically in a diagnostic, two threads are in conflict over a shared object. This object may be a system object or a region of memory. The **1st Access** view shows the thread which accessed the object first. The **2nd Access** view shows the thread which later referenced the object.

Definition view - The **Definition** view shows the object that is the dynamic allocation point of the object if known.

Context view - If the two threads are part of an OpenMP* team then the **Context** view shows the parallel region containing these threads.

Source view toolbar buttons

Item	Name	Purpose
☰	**Source only**	Display high-level programming language. Viewing this mode requires symbol information.
☰▾	**Mixed**	Display high-level programming language together with its related assembly instructions. Click on the arrow to choose from the different mixed code formats.
☰	**Disassembly only**	Display the disassembly code of the module.
	Toggle bookmark	Add a new bookmark in this line or delete an existing bookmark.
	Clear bookmarks	Delete all bookmarks.
	Go to previous bookmark	Go to the previous bookmark.
	Go to next bookmark	Go to the next bookmark.
▽	**Go to next portal**	Go to the next portal; enabled only when line number information is not available.
△	**Go to previous portal**	Go to the previous portal; enabled only when line number information is not available.
	Go to start of function	Go to the current function declaration.
	Go to next function	Go to the next function declaration.

OpenMP*, POSIX*, and Windows* API Support

Thread Checker supports analysis of threaded programs that use OpenMP*, POSIX*, and Windows* API. Thread Checker supports analysis of POSIX* and OpenMP* threaded programs on IA32 and Itanium® Processor Family based systems running Linux* via the Thread Checker remote data collector.

OpenMP*

For OpenMP* threaded programs in either a Windows* or Linux environment, Thread Checker performs additional checking to determine if the program's execution is inconsistent or could possibly be inconsistent with the execution of the corresponding sequential program.

If your program uses OpenMP* directives like pragma's, you must use the appropriate /Qopenmp or -openmp switch when building your debug application for analysis using Intel® Thread Checker. Otherwise, the Intel® Compiler ignores OpenMP* directives in your code.

See the OpenMP* web site for simple, portable, scalable SMP programming: http://www.openmp.org

POSIX*

Thread Checker supports the POSIX* multi-threading API otherwise know as pthreads*. Thread Checker supports pthreads* on Linux via Thread Checker's remote data collector. For more information about the POSIX* multi-threading API, perform a web search on Posix* or pthreads*. There are numerous resources on the internet that discuss POSIX* threading API implementation and pthreads*.

Windows* API

For information about Windows* multi-threading APIs, consult the MSDN* website at http://msdn.microsoft.com/library. There you will find complete guides to the following types of APIs:

- Process and Thread Functions
- Synchronization Functions

For product and purchase information visit the Intel® Software Development Products site at: www.intel.com/software/products

Document coding: xxxx/xxx/xxx/xx/xxxx xxxxxx-xxx

Print Date: 02.01.2006

Intel® Software Development Products

Intel® Thread Checker Quick Reference Guide

Use Intel® Thread Checker to check your multi-threaded code for potential threading problems.

A step by step approach to application threading diagnostics using Intel® Thread Checker

1. **Select a data set** – Consider using a small representative data set rather than a large repetitive data set to save time. It may be faster to run several small datasets rather than one large dataset.
2. **Create and run an Intel® Thread Checker Activity** - Create an Activity using the Intel® Thread Checker Configuration Wizard. The Intel® Thread Checker data collector gathers information about threading problems found during the specific run of your application.
3. **Investigate / Review code design** – Review Thread Checker diagnostics. Drill down to the code, if available, to see where diagnostics occurred.
4. **Implement code changes** – Modify your code to resolve race conditions and other thread errors that Thread Checker diagnosed. Use the filter feature to filter out benign race conditions.
5. **Re-run the Activity** - Re-run the Activity until all threading errors have been resolved.

Thread Faster with **Intel® Threading Tools**

Build and Linker Switches

You can perform binary instrumentation on an executable which has been compiled without the recommended build switches. However, Intel® Thread Checker will have less information to work with resulting in less specific diagnostics; for example, Thread Checker may report incorrect line numbers for optimized code. To optimize the level of diagnostics available, use the recommended compiler and linker switches when building your application.

Recommended compiler and linker switches

Windows*	Linux*	Description
/Od	-O0	Disable optimizations. This will enable Thread Checker source code information (especially line numbers) to be accurate.
/MD, /MDd [/MT, /MTd]	n/a	Dynamically linked or statically linked thread safe runtime libraries. Dynamically linked RTL or /MD is recommended.
/Zi,/ZI,/Z7	-g	Produce symbolic debug information in object.
/fixed:no	n/a	Linker switch which prevents compiler from stripping base relocation information from results.

Microsoft* Windows* Compiler example:

```
cl /MD /Od /Zi <source> -o <target> /link /fixed:no
```

Intel® Compiler specific switches:

Windows*	Linux*	Description
/Qtcheck	-tcheck	Generate Thread Checker instrumentation to detect multi-threading bugs.
/Qopenmp	-openmp	Enable the compiler to generate multi-threaded code based on the OpenMP directives embedded in your code.

Intel® C/C++ Compiler Windows* example:

```
icl /MD /Od /Zi /Qtcheck <source> -o <target> /link /fixed:no
```

Intel® C/C++ Compiler Linux* example:

```
icc -g -O0 -tcheck <source> -o <target>
```

Intel® Fortran* Compiler Windows* example:

```
ifort /Fe<target> /MD /Zi /Od /Qtcheck <source> /link /fixed:no
```

Intel® Fortran* Compiler Linux* example:

```
ifort -o <target> -g -O0 -tcheck <source>
```

Instrumentation

Intel® Thread Checker can collect data directly from your executable binary file by making changes to its image. This enables the binary to connect to the Intel® Thread Checker Analysis Engine without source code. Alternatively, Thread Checker can connect your program to the Analysis Engine when you compile your program using the Intel® Compiler.

Binary instrumentation method

Intel® Thread Checker automatically performs binary instrumentation when you run an Intel® Thread Checker Activity within the VTune™ environment.

- Binary instrumentation only works if you can execute your program from within the VTune™ environment.
- If you have debug information in your source code, Thread Checker can display source code, if available, even when using binary instrumentation.
- If source code is not available, Thread Checker disassembles the binary and provides disassembly information.

Source instrumentation method

Intel® Thread Checker can connect your program to the Analysis Engine at compile time using the Intel® Compiler.

- To perform source instrumentation, use the /Qtcheck or -tcheck switches.
- Source instrumentation can provide additional symbolic information specifying the symbolic expressions performing the memory references in the reported diagnostics.
- Use source instrumentation when you want to run the instrumented program outside of the VTune™ environment. For example, to instrument a server application.

Data Collection

You can run an analysis on a binary or source instrumented application from within Thread Checker by creating a Thread Checker Activity. You can also compile your source using source instrumentation then execute the resulting instrumented binary. This will create a Thread Checker results file for your binary. The results file can be manually opened in Thread Checker for analysis.

To create an Intel® Thread Checker results file:

1. Compile your application using the source instrumentation method. Use the appropriate Thread Checker compiler switch for your environment. Use /Qtcheck in Windows or -tcheck in Linux environments.
2. Execute your application natively, outside the VTune™ environment. The Analysis Engine records data about diagnostics found. The data is stored in a file with a .thr extension in the current directory.
3. Once your program's run is complete, start the VTune™ environment and import the .thr results file. To import a .thr file, go to **File > Open File**. Select **Thread Checker [*.thr]** from the drop-down list. Browse to your .thr results file. Click **OK**.

Linux* Remote Data Collection

The Intel® Threading Tools Remote Data Collection Server, **ittserver**, enables you to collect data on a Linux* system. You can then analyze results gathered from the remote system on a Windows* host system running Intel® Thread Checker.

To use the Remote Data Collector to analyze an application on a Linux* system, do the following:

1. Install the Thread Checker Remote Agent on your Linux* system. Instructions for installing the Remote Agent, which includes the Thread Checker Collector and ittserver, are included in the product release notes.
2. If needed, set Thread Checker environment variables by sourcing tcvars.sh or tcvars.csh. Set environment variables for Intel® Compiler by sourcing iccvars.sh or iccvars.csh.
3. Start ittserver. By default, ittserver is installed in /opt/intel/itt/bin/32/. Launch ittserver from your home directory or a directory where ittserver can write its remote files. Once ittserver is launched, information related to ittserver appears:
   - If "Intel® Thread Checker Collector" is displayed, then this indicates a good install of the Thread Checker Collector library component on your Linux* system.
   - If "Server is ready..." is displayed, then this indicates that the ittserver is active and ready for signals from the Windows* host system.
   - To stop ittserver press Ctrl-C.
4. In Windows*, create and run a Thread Checker Activity using the Intel® Thread Checker Configuration Wizard. Click the **Remote...** button to select and set options specific to the remote Linux* system. Enter the path to the executable as it is seen on the system running ittserver. For example, enter "/home/myhome/a.out." Click **Next** then click **Finish**. When the remote collector finishes, it notifies ittserver, which in turn notifies Thread Checker on Windows*. The analysis results are sent back to Thread Checker and displayed in the GUI.
5. Analyze your Activity results on the Windows* host system. You can analyze the diagnostics using all of the features available in Intel® Thread Checker.

Mapping source files for remote analysis

When viewing Activity results originating from a Linux system, you may be prompted for the location of source files that cannot be automatically determined.

If your Linux machine has the Samba service running, then when prompted by Thread Checker for files that cannot be located, you can click the ellipse button to browse the file system on your remote machine.

Selective Instrumentation

You can set different levels of instrumentation that Intel® Thread Checker uses to collect data for specific modules or specified functions. To access an Activity's instrumentation settings, right-click on a Thread Checker Activity and select **Modify Collectors**. The **About to Modify Activity with Results** dialog box may appear. Click **OK**. The **Configure Intel® Thread Checker** dialog box appears. Click on the **Instrumentation** tab to view settings.

The following table shows instrumentation settings, from the highest (most expensive and most detailed analysis) to the lowest level (least expensive and least detailed analysis):

Level	Description
Full Image	Enables each instruction to be checked. This level instruments modules with or without symbolic information.
Custom Image	Same as "Full Image" except that you can select specific functions within a module to instrument.
All Functions (default)	Enables instrumentation for modules that were compiled with symbolic debugging information.
Custom Functions	Same as "All Functions" except that you can select specific functions within a module to instrument.
API Imports	Only system API functions that are needed to be instrumented by the tool are instrumented. No user code is instrumented. The specific set of API functions is tool dependent.
Module Imports	Disables instrumentation. Enables Thread Checker to continue performing diagnostic detection in modules that are loaded from a parent module.

Selective instrumentation tips:

- If you want to disable diagnostic detection for a specific module, use the "Module Imports" level.
- System images and images without base relocations can only be instrumented with the "Module Imports" option. To produce base relocations for an image, use /fixed:no linker option while building the image.
- You may wish to adjust the instrumentation level based on response time, system resources, or to control the amount of information gathered.