



Frequently Asked Questions for the Intel® Trace Analyzer and Collector

Copyright © 2007–2010 Intel Corporation

All Rights Reserved

Document Number: 318118-006

Revision: 8.0 Update 3

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

Disclaimer and 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. UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

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/

MPEG-1, MPEG-2, MPEG-4, H.261, H.263, H.264, MP3, DV, VC-1, MJPEG, AC3, AAC, G.711, G.722, G.722.1, G.722.2, AMRWB, Extended AMRWB (AMRWB+), G.167, G.168, G.169, G.723.1, G.726, G.728, G.729, G.729.1, GSM AMR, GSM FR are international standards promoted by ISO, IEC, ITU, ETSI, 3GPP and other organizations. Implementations of these standards, or the standard enabled platforms may require licenses from various entities, including Intel Corporation.

BlueMoon, BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, E-GOLD, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, 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 XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Moblin, Pentium, Pentium Inside, Puma, skool, the skool logo, SMARTi, Sound Mark, 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 other countries.

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

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

Java is a registered trademark of Oracle and/or its affiliates.

Copyright (C) [2003]–[2011], Intel Corporation. All rights reserved.

This product includes software developed by the University of California, Berkley and its contributors, and software derived from the Xerox Secure Hash Function. It also includes libraries developed and © by SGI and Michael Riepe. They are licensed under the GNU Lesser General Public License (LGPL) and their source code can be found in the "third party" directory.

Contents

1	General Questions about Intel® Trace Analyzer	4
1.1	What is Intel® Trace Analyzer?	4
1.2	I have created Structured Trace Format (STF) files with Intel® Trace Collector 5.0. Will Intel® Trace Analyzer version 6.0 or greater still be able to process the STF file?	4
1.3	Does Intel® Trace Analyzer and Collector run on Microsoft Windows*?	4
1.4	What are Charts in Intel® Trace Analyzer?.....	4
1.5	What are Views in Intel® Trace Analyzer?.....	4
1.6	I have a very large application and my Charts are a bit crowded. Is there any way to search or highlight certain function events, messages, and collective operations in Intel® Trace Analyzer?	5
1.7	What is Aggregation and how many types of it are available in Intel® Trace Analyzer?.....	5
1.8	Can trace files be on the order of gigabytes of storage?	5
1.9	I cannot open a timeline chart. How can I get the respective menu-entries enabled?.....	5
1.10	My message profile shows ridiculously great numbers. What went wrong?	6
1.11	Why do user-defined activities not work?	6
1.12	Why are some messages not shown?	6
2	General Questions about the Intel® Trace Collector	7
2.1	How can I limit the trace file size?	7
2.2	How can I limit the memory consumption of Intel® Trace Collector?.....	7
2.3	How can I manage the Intel® Trace Collector API calls?.....	8
2.4	What happens when a program fails?	8
2.5	I cannot find the trace file, where is it?.....	9
2.6	What is this “Bad Clock Resolution” all about?	9
2.7	Does Intel® Trace Collector support non-MPI applications?	10
2.8	I get “Error: libVT.a: file not recognized”, what’s wrong?.....	10
3	Platform Specific Questions.....	11
3.1	Linux* OS: Cannot find libelf.....	11
3.2	Error: Unsupported Architecture	11

1 General Questions about Intel® Trace Analyzer

1.1 What is Intel® Trace Analyzer?

Intel® Trace Analyzer is a graphical tool that displays and analyzes event trace data generated by Intel® Trace Collector. It helps understanding the behavior of the application, detecting performance problems or programming errors.

1.2 I have created Structured Trace Format (STF) files with Intel® Trace Collector 5.0. Will Intel® Trace Analyzer version 6.0 or greater still be able to process the STF file?

Yes, STF files (or trace files) created with Intel® Trace Collector 5.0 are readable by Intel® Trace Analyzer version 6.0 or greater.

1.3 Does Intel® Trace Analyzer and Collector run on Microsoft Windows*?

Yes, Intel® Trace Analyzer and Collector runs on Microsoft Windows CCS*, supporting Intel® MPI Library, MPICH2*, Microsoft Visual Studio 2005* and later and Intel® Professional Edition Compilers. Intel® Trace Analyzer is also supported on Microsoft* Windows* Compute Cluster Server 2003, Microsoft* Windows* HPC Server 2008, Microsoft* Windows* XP and Microsoft* Windows Vista*.

1.4 What are Charts in Intel® Trace Analyzer?

Charts in Intel® Trace Analyzer are graphical or alphanumeric diagrams that are parameterized with a time interval, a process grouping, a function grouping and an optional filter. Together they define the structure in which data is presented and the amount of data to be displayed.

The Charts supported by Intel Trace Analyzer are divided into:

1. Timelines: the **Event Timeline**, the **Qualitative Timeline**, the **Quantitative Timeline** and the **Counter Timeline**.
2. Profiles: the **Function Profile**, the **Message Profile**, and the **Collective Operations Profile**.

While the Timelines show trace data in graphical form over a horizontal axis representing runtime, the Profiles show statistical data. All these Charts are found under the **Charts Menu** item. Opening a file in the Intel Trace Analyzer, the default display is a View containing the **Function Profile Chart** for the opened file.

1.5 What are Views in Intel® Trace Analyzer?

A View holds a collection of Charts in a single window. Those Charts, inherent in the same View, use the same perspective on the data. This perspective is made up of the following attributes: the time interval, process aggregation, function aggregation and filters. This helps to flexibly analyze a trace file by looking at multiple partitions of the data from various points of view.

Whenever an attribute in the current perspective is changed for one of the Charts, all other Charts follow. Opening several Views offers a very flexible and variable mechanism for exploring, analyzing, and comparing trace data.

1.6 I have a very large application and my Charts are a bit crowded. Is there any way to search or highlight certain function events, messages, and collective operations in Intel® Trace Analyzer?

The filtering and tagging functions of Intel® Trace Analyzer are the tools to use when looking for something specific.

The filtering dialog box is accessed through **Main Menu > Advanced > Filtering**. This dialog box allows specifying filter expressions that describe which function events, messages, and collective operations are to be analyzed and shown. The two fundamental modes of input are generating the filter expression through a graphical interface or typing it in manually. If the current expression can not be converted into a proper filter definition, then the dialog shows a red warning that indicates the reason. After applying a filtering expression, the Charts will display only events that satisfy the condition of the expression. All other events will be suppressed as if they had never been written to the trace file.

Tagging works the same way as filtering and uses the same grammar to create the tagging expression. The only difference is that tagging highlights events that satisfy the specific user-defined conditions. That emphasis varies depending on the Chart to which the tagging expression is applied. For specific details, please refer to the Intel Trace Analyzer Reference Guide.

1.7 What is Aggregation and how many types of it are available in Intel® Trace Analyzer?

Aggregation reduces the amount of data displayed by accumulating the events into thread groups and into function groups.

Process Aggregation focuses on the processes that are important and aggregates the results into Process Groups. It accesses the Process Group Editor (**Advanced > Process Aggregation**) where the appropriate process groups can be constructed and/or chosen.

Function Aggregation focuses on a subset of functions and aggregates them into Function Groups, without causing a distraction for other functions that are currently not significant. Use the Function Group Editor (**Advanced > Function Aggregation**) to do this.

1.8 Can trace files be on the order of gigabytes of storage?

Intel® Trace Analyzer was designed with data structures to efficiently process trace files that have large magnitude. Therefore, handling trace files that are on the order of gigabytes is generally not a problem. Each timeline calculates a resolution for the analysis that describes a time span that can reasonably be painted and selected with the mouse. Thus, the performance optimization is achieved by merging events (function events, messages and collective operations) if required by the current Resolution. In addition, when zooming in on a certain section of a timeline, the extra trace information that is not currently displayed is hidden. Due to all of these enhancements, Intel Trace Analyzer can theoretically load trace files of any size.

1.9 I cannot open a timeline chart. How can I get the respective menu-entries enabled?

You are loading a large trace file. Intel® Trace Analyzer is still loading it and creating its internal data structures. While it is doing this, it shows you precomputed statistical data in the profile charts which

you can use almost in a normal fashion. Once the entire trace has been processed, Intel Trace Analyzer will allow you to open timeline charts.

1.10 My message profile shows ridiculously great numbers. What went wrong?

Most probably the message was recorded in reverse order meaning that the message was received before it was sent. This can happen on systems with coarse clock resolution and/or very fast communication hardware. If the clock resolution cannot be increased, there is currently no solution for it.

1.11 Why do user-defined activities not work?

In order to minimize the instrumentation overhead, Intel® Trace Collector does not check for global consistency of the activity codes specified by calls to `VT_symdef ()` or `VTSYMDEF ()`. When using user-defined activities you should ensure that:

- The same code is used for the same activity in all processes.
- Two different symbols never share the same code.

If these rules are violated, Intel® Trace Analyzer might complain about duplicate activities, or activities may be mislabeled in the Intel Trace Analyzer displays.

1.12 Why are some messages not shown?

In order for messages to be indicated in the Intel® Trace Analyzer displays, both the calls to the sending and the receiving MPI routines must be traced. For non-blocking receives, the call to the MPI wait or test routine that did complete the receive request must be logged.

If tracing has been disabled during runtime it can happen that for some messages, either the sending or the receiving call has not been traced. As a consequence, these messages are not shown by Intel Trace Analyzer, and other messages can appear to be sent to or received at the wrong place. Similarly, filtering out some of the above mentioned MPI routines has the same effect.

2 General Questions about the Intel® Trace Collector

2.1 How can I limit the trace file size?

Although Intel® Trace Collector uses a compact binary format to store the trace data, trace files for real-world applications can get extremely large. The best approach is to limit the number of events to be logged by scaling down the application, like, for example, iteration count, number of processes, problem size and so on. This also shortens the time required to run a test. Quite often, this is not acceptable because reduced input datasets are not available or performance analysis for reduced problems is simply not interesting. In that case there are basically four other options:

- Enable trace data collection for a subset of the application's runtime only: by inserting calls to `VT_traceoff()` and `VT_traceon()`, an application programmer can easily limit the profiling to interesting parts of an application or a subset of iterations. This will require recompilation of (a subset of) the application though, which may not be possible, or at least inconvenient.
- If the application has a complex call graph, for instance due to automatic function tracing, then folding of functions can prune the call tree a lot at run-time and thus cut down the trace file size. This feature is not supported by all Intel Trace Collector versions.
- Use the activity/symbol filtering mechanism to limit the set of logged events. For this, the application does not have to be changed in any way. However, it is necessary that the user has an idea of which events are interesting enough to be traced, and which events can be discarded. As every MPI routine call generates roughly the same amount of trace data the possible reduction in data volume is quite high: concentrate on the calls actually communicating data, and do not trace the administrative MPI routines.
- Use the process or node or time interval filters to limit data collection to a subset of processes.

2.2 How can I limit the memory consumption of Intel® Trace Collector?

During the application run, Intel® Trace Collector first stores trace data in memory buffers. There are two options that control the allocation of these buffers: `MEM-BLOCKSIZE` specifies the size of each memory block in bytes, and `MEMMAXBLOCKS` determines the maximum number of memory blocks. Intel Trace Collector will not exceed the memory limits set by `MEMMAXBLOCKS` multiplied by `MEM-BLOCKSIZE`. When this trace data memory is exhausted, one of three actions is taken:

- If the `AUTOFLUSH` option is enabled (default), the collected trace data is flushed to disk, and the trace collection continues. The spill files are automatically merged when the application finalizes, so that all records will appear in the trace file.
- If `AUTOFLUSH` is disabled and `MEM-OVERWRITE` is enabled, the trace buffers will be overwritten from the beginning, in effect recording the last n records.
- Otherwise, the trace collection stops, in effect collecting the first n records.

Placing trace data in main memory can slow down the application if it needs the memory itself. Setting `MEM-MAXBLOCKS` puts a hard limit on the amount of memory used by Intel Trace Collector, but can disrupt the application when a process waits for flushing of trace data. To avoid this, Intel Trace Collector can be told to start flushing earlier in the background with the `MEM-FLUSHBLOCKS` option. This option is only available in more recent thread-safe versions of Intel Trace Collector.

To understand how much memory is currently in use, Intel Trace Collector can add counter data to the trace. If enabled, each process will store its own values for these counters in the trace each time they change. This makes it possible to take the effect of buffer handling into account when doing the

analysis of the trace. These counters are not enabled by default. It is necessary to add the following lines to a configuration file (see usage of `VT_CONFIG`) to enable each counter:

```
COUNTER data_in_ram ON COUNTER data_in_file ON COUNTER flush_active ON
```

At runtime, Intel Trace Collector also provides feedback on the amount of data collected: with the default setting of 500MB for the `MEM-INFO` configuration option a message is printed each time more than this amount of new data is recorded by a process. The value is chosen so that the message serves as a warning when the amount of trace data exceeds the amount that can usually be handled without problems. In order to use it as a kind of progress report a much lower value would be more appropriate.

2.3 How can I manage the Intel® Trace Collector API calls?

The API routines greatly extend the functionality of Intel® Trace Collector. Unfortunately, manual instrumenting the application source code with the Intel Trace Collector API makes code maintenance harder. An application that contains calls to the Intel Trace Collector API requires the Intel Trace Collector library to link and incurs a certain profiling overhead. The dummy API library `libVTnull.a` helps in this situation: all the API calls map to empty subroutines, and no trace data is ever gathered if an application is linked to it. Still, the extraneous function calls remain and may cause a slight overhead.

It is recommended that the C pre-processor (or an equivalent tool for Fortran) is used to guard all the calls to the Intel Trace Collector API by `#ifdef` directives. This will allow easy generation of a plain vanilla version and an instrumented version of a program.

2.4 What happens when a program fails?

The Intel® Trace Collector library stores trace data first in buffers in the application memory, and then in flush files (one per MPI process) when the buffers have been filled. In normal operation, the library will merge the trace data from each process during execution of the `MPI_Finalize()` routine, and write the trace data into a single trace file suitable for input to Intel® Trace Analyzer. If a program fails, `MPI_Finalize()` is never executed, and the traditional Intel Trace Collector does not write a trace file. To get a trace file until the program crash, you can link against the fail-safe version `libVTfs`.

Troubleshooting:

The Intel Trace Collector library can report four basic error classes:

1. Setup errors
2. Invalid configuration file format
3. Erroneous use of the API routines
4. Insufficient memory

The first category includes invalid settings of the `VT_` environment variables, failure to open the specified trace file and so on. A warning message is printed; the library ignores the erroneous setup and tries to continue with default settings.

For the second class, a warning message is printed, the faulty configuration file line is ignored, and the parser continues with the next line.

When an Intel Trace Collector API routine is called with invalid parameters, a negative value is returned (as a function result in C, in the error parameter in Fortran), and operation continues. Invoking any API routines before `MPI_Init()` or after `MPI_Finalize()` is considered erroneous, and the call is silently ignored.

An insufficient memory error can occur during execution of an API routine or within any MPI routine if tracing is enabled. In the first case, an error code (`VT_ENOMEM` or `VTENOMEM`) is returned to the calling process; in any case, Intel Trace Collector prints an error message and attempts to continue by

disabling the collection of trace data. Within `MPI_Finalize()`, the library will try to generate a trace file from the data gathered before the insufficient memory error.

Although Intel Trace Collector tries to handle out-of-memory situations gracefully, library calls in the application might not be as tolerant, or the operating system does not handle such a situation well enough. To avoid a memory error in the first place, try to limit the amount of trace data as explained in the section [Limiting Memory Consumption](#). The memory requirements of Intel Trace Collector can be reduced with the `MEM-BLOCKSIZE` and `MEM-MAXBLOCKS` configuration options. The `AUTOFLUSH` option needs to remain enabled if you want to see a trace of the whole application run.

2.5 I cannot find the trace file, where is it?

Unless told otherwise in the configuration file, Intel® Trace Collector will write the trace data to the file `argv[0].stf`, with `argv[0]` being the application name in the command line (same as `getarg(0)` in Fortran). Note that your MPI library or the MPI execution script may interfere with `argv[0]`, and that only the process actually writing the trace file (usually the one with rank 0 in `MPI_COMM_WORLD`) will look at it. A relative pathname will be interpreted relative to that process current working directory.

You can however change the trace file name with the `LOGFILE-NAME` directive in a configuration file.

If it turns out that Intel Trace Collector cannot create the specified trace file, it will attempt to write to the file `/tmp/VT-<PID>.stf`, with `<PID>` being the Unix process id of the trace file-writing MPI process.

In any case, an information message with the actual trace file name will be printed by Intel Trace Collector within `MPI_Finalize()`.

On systems where not all processes see the same files, be sure to look for the trace file in the correct process file system. You can influence which process will write the file by setting an environment variable or by a directive in the configuration file.

2.6 What is this “Bad Clock Resolution” all about?

If the clock resolution is very low (the timer function returns the same value for a long period of time), then many events will be recorded on the same time stamp and analysis of such a trace becomes very hard. In particular the Global Timeline becomes useless.

If Intel® Trace Collector 4.0.2 detects this, it will issue a warning like:

```
minimum clock increment 1e-3s is very high, please fix system setup to
obtain better traces
```

The minimum clock increment is always stored in the trace file info, because the timer base also listed there may be lower than the real value.

This problem was observed on certain Red Hat Enterprise Linux* 3.0 kernel versions. The following releases (see `/etc/r*release*` on Red Hat derived systems) are definitely affected:

- Rocks release 3.1.0* (Matterhorn*)

These are not:

- Rocks release 3.2.0*
- Red Hat Enterprise Linux AS release 3* (Taroon Update 1*), kernel 2.4.21-9.EL

Something that might help with Red Hat Enterprise Linux* 3.0 Taroon systems running on Intel® 64 architecture is to reboot the kernel with the `tsc` option. For further information, try http://kbase.redhat.com/faq/FAQ_79_3728.shtm.

2.7 Does Intel® Trace Collector support non-MPI applications?

The tracing part is not restricted to programs using MPI; however it is more complicated to use the tracing library in this case. You can use `itcpin` to insert instrumentation probes. The `itcpin` utility program can manipulate a binary executable so that an Intel® Trace Analyzer and Collector library is inserted as if the file has been linked against it. It can also insert code into the executable so that function entry and exit events are recorded for more detailed analysis of the user's code. For more information on the usage of `itcpin`, see the Intel® Trace Collector User's Guide.

If you have access to the source codes of the application then you could instrument the code using API calls to the tracing functions of the library. The executable would have to be recompiled and relinked with the tracing library. The resulting binary then writes a trace file that contains the calls to the API as events and can be analyzed using the Intel® Trace Analyzer.

2.8 I get "Error: libVT.a: file not recognized", what's wrong?

Make sure that the libraries are installed using the `./install` scripts provided with the tool. Otherwise, the libraries will be locked and will be in a format not recognized as valid, thus showing the error above.

3 Platform Specific Questions

3.1 Linux* OS: Cannot find libelf

If you compile your MPI program on Linux* OS, you may run into the following linker problem:

```
/usr/bin/ld: cannot find -lelf
```

This means that the linker cannot find the library file `libelf.a`. Some distributions do not install this library by default. In some older Intel® Trace Collector distributions, it was not included either, so you had to install this package from your Linux installation media. But now this error should no longer occur because now a version of `libelf` is included in the same directory as `libVT` itself.

3.2 Error: Unsupported Architecture

We do not test or validate Intel® Trace Collector on systems using non-Intel processors. Because of potential architectural differences, we cannot ensure that crucial performance results are correct. Therefore, rather than allow test or validations that could lead to potentially incorrect results, we prevent our tool from running on systems using non-Intel processors.