TotalView 2017.1 Release Notes

Updated: April 2017

These release notes contain a summary of new features and enhancements, late-breaking product issues, migration from earlier releases, and bug fixes.

PLEASE NOTE: The version of this document in the product distribution is a snapshot at the time the product distribution was created. Additional information may be added after that time because of issues found during distribution testing or after the product is released. To be sure you have the most up-to-date information, see the version of this document on the Rogue Wave web site:
http://www.roguewave.com/support/product-documentation/totalview.aspx

Additions and Updates

Early Access to the NextGen TotalView User Interface

The TotalView team continues to add more features to the next generation TotalView user interface. The interface is gaining in maturity and we are excited to provide an early version. To try out the new user interface start TotalView with the -newUI switch:

```
totalview -newUI
```

New features added to the NextGen user interface in this release include:

- **Replay Bookmarks**
  For ReplayEngine supported platforms, Replay Bookmarks allow you to easily mark a point during your programs execution history and easily jump back to that point time! Read more about how to utilize Replay Bookmarks in the Chapter 8 of the User Guide.
- **Early Access to Mixed Language Debugging with Python and C/C++**
  This release introduces mixed debugging of Python and C/C++ applications, enabling you to easily see a fully integrated call stack across the language barriers and to examine data passed between the layers. Read more about the Python and C/C++ debugging capabilities in Chapter 7 of the NextGen TotalView for HPC User Guide.
- **Support added to modify program arguments after debug session is started**
  Support has been added to modify program arguments after a debug session has been launched. The new arguments and settings will be used the next time the debug session is restarted.

Currently supported platforms:

- Linux x86 64-bit, Linux PowerLE, Linux ARM64 and Apple's macOS/Mac OS X

Release over release we will be adding new functionality to the new user interface based on a priority list that you can help influence. Please send email to tv-beta@roguewave.com with your feedback and feature priorities! We welcome all feedback and feature requests for the new user interface!

New License Model for ARM64 and PowerLE Platforms

The ARM64 and PowerLE platforms have a new file based FlexNet Embedded license model. Customers who have purchased a license for these platforms will receive a new license file. Instructions for installing the license are in the Installation Guide.

GNU DebugFission Split DWARF on Linux

Split DWARF is a new approach that dramatically shrinks the space required to store DWARF debug information, speeds-up application link times, reduces system memory and IO requirements, and speeds-up debugger startup. Split DWARF places the majority of the debug information for an "object file" (.o) in a separate "DWARF object file" (.dwo) that is *not* processed by the linker.

TotalView now supports the GNU DebugFission variant of Split DWARF, which is described in more detail at https://gcc.gnu.org/wiki/DebugFission. The DWARF 5 variant of Split DWARF is not yet supported. For more information, check out the Saving Time and Space with Split DWARF white paper.

Building your application for GNU DebugFission Split DWARF:

- **Compiling:**
  - Use the "**-gsplit-dwarf**" compiler option to generate ".dwo" (DWARF object) files containing the full DWARF debug information, and a ".o" (object) file containing the code, data, and skeleton DWARF debug information. Note: the DWARF debug information in the ".o" file points to the ".dwo" file, therefore the ".dwo" file must not be deleted to debug that module.
- **Linking:**
  - Use the "**-fuse-ld=gold**" compiler option to use the gold linker (**ld.gold**).
  - Use the "**-Wl,--gdb-index**" compiler option to pass the "**--gdb-index**" option to the gold linker. Note: The resulting executable or shared library image file will contain a "**.gdb_index**" section that the debugger can use for faster startup.

Distribution Tar Bundle Name Change for 32-bit Linux

To more clearly identify 32-bit versus 64-bit tar distribution bundles for x86 based Linux architectures, the 32-bit tar bundle now has “-32” appended to the end of it.

Distribution Tar Bundle Name Change for macOS

The distributed tar bundles for macOS now has “-64” appended to their names to more clearly identify the architecture.

Bug fixes and Improvements

There have been a significant number of bug fixes and improvements added to the 2017.1 release

Bug Fixes for 2017.1

TVT-19670	Process wide breakpoint is disabled on restart.
TVT-22428	Saved conditional breakpoints get error when program has minor change and rebuild.
TVT-22668	On macOS Sierra, a race condition fetching the dyld base address raises an error.
TVT-22735	Hitting ‘next’ in a C++ program with inlining, causes the program to run to completion.
TVT-23137	Unable to load MemoryScape .mdbg files that contain spaces in the name.
TVT-23138	MemoryScape Comparison Report is missing leaked data.
TVT-23213	Excessively long time reading symbol table.
TVT-23244	Internal error when trying to display the contents of a std::shared_ptr object.
TVT-23254	Fatal error when using Intel MPI and MPMD.
TVT-23278	Fatal error:Soid… has been lost.
TVT-23349	A long module list in Intel Fortran can cause TotalView to abort.

Deprecation Notices

None

Known Issues

Licensing

TotalView releases built with FlexNet Publisher 11.13.1 must have licenses served by a license server at 11.13.1 or higher

FlexNet Publisher client library version 11.13.1 is built into our recent releases. This means, according to FlexNet Publisher's component version compatibility rules (near the end of FNP's License Administration Guide PDF), the license server *must be* at v11.13.1 or higher. Although these rules have long been in place with no problems, we've recently been receiving reports of license checkout failures when using the license server v11.12.1 from previous TotalView releases. In this case the vendor daemon's debug log file shows "(toolworks) Request denied: Client (11.13) newer than Vendor Daemon (11.12). (Version of vendor daemon is too old. (-83,21049))". As noted in FNP's License Administration Guide PDF, this issue can be avoided by making sure our latest license server components are in place.

TotalView sometimes cannot acquire license due to FlexNet bug

If you are using Linux Power, Linux Itanium or AIX then you are still affected by the bug in the FlexNet Publisher software that results in TotalView's inability to acquire a license when your license file contains multiple licenses with different maintenance expiration dates (i.e. the 4th field on the INCREMENT line). The licensing software skips some of the licenses in this case. If you know that your license file is being read and is correct, and you think you might be running into this bug, we recommend that you add the "sort" keyword and value (such as sort=1, sort=2, sort=3) to each INCREMENT line in the license file in any order. This bug has been reported to Flexera and is identified as SIOC-000145042.

Here is an example of adding the "sort" keyword:

```
SERVER linux-power 0050569b402c
VENDOR toolworks
INCREMENT TotalView_Enterprise toolworks 2014.1231 permanent 1 \
        sort=1 VENDOR_STRING="processors=16 platform=linux-power" \
        SIGN=3350A26C395A
INCREMENT TotalView_Enterprise toolworks 2015.1231 permanent 1 \
        sort=2 VENDOR_STRING="processors=16 platform=linux-ia64" \
        SIGN=C7D11FB667C8
INCREMENT TotalView_Enterprise toolworks 2016.1231 permanent 1 \
        sort=3 VENDOR_STRING="processors=16 platform=linux-x86_64" \
        SIGN=BEC7534A248A
```

macOS

With multiple displays attached to a macOS machine, some TotalView windows may not be noticeable

When a user attaches an external monitor to a running Macbook Pro, or adds multiple displays to a Mac, window rearrangement may move some windows off-screen. This may result in a TotalView modal window not being found until you use the Mac command to display all the windows (Mission Control). This appears to be an interaction between XQuartz and Darwin. It has been seen in Mavericks, but it's possible it will show up in other releases. There may be a workaround in System Preferences->Mission Control by disabling “Displays have separate Spaces.”

Physical console access needed when starting TotalView

Starting in Mountain Lion, OS X security policies require that users meet a password challenge in order to use TotalView, and the challenge can be issued only to the console (the OS X desktop). After the password challenge is met once, you can run TotalView repeatedly from the same login session without further challenges.

It is possible to work around this need for physical access with the following steps. The first few of these are likely already set in order to allow TotalView to run.

- Install XQuartz and TotalView
- Ensure every user needing debugging is in the _developer group
- Allow X11 forwarding in the sshd_config file (disabled by default)

- In a terminal window enter the following two commands:
  - DevToolsSecurity -enable (this step is optional if this was already enabled)

- sudo security authorizationdb write system.privilege.taskport allow

Visualizer fails under macOS Sierra and Xquartz

Attempts to use the visualizer tool fails with a message 'Error: attempt to add non-widget child "dsm" to parent "vismain"' which supports only widgets.

Linux

Split-DWARF and .gdb_index support, and related options

State variable **TV::dwarf_global_index** is a boolean flag that controls whether or not TotalView consider using the DWARF global index sections (.debug_pubname, .debug_pubtypes, .debug_typenames, etc.) in executable and shared library image files. It defaults to **true**. It may be useful to set this flag to **false** if you have an image file that has incomplete global index sections, and you want to force TotalView to skim the DWARF instead, which may cause TotalView to slow down when indexing symbol tables. Command option **-dwarf_global_index** sets the flag to **true**,and **-no_dwarf_global_index** set the flag to **false**.

State variable **TV::gdb_index** -is a boolean flag that controls whether or not TotalView consider using the .gdb_index section in executable and shared library image files. It defaults to **true**. It may be useful to set this to **false** if you have an image file that has an incomplete .gdb_index section and you want to force TotalView to skim the DWARF instead. Command option **-gdb_index** sets the flag to **true**,and **-no_gdb_index** set the flag to **false**.

ReplayEngine On-Demand Records Can Show Invalid Stack Trace

In some circumstances in which a ReplayEngine debugging session is driven to the beginning of recorded history, the debugger will display an invalid stack trace and stack frame. We have observed this when debugging a ReplayEngine recording file that was created during a live debugging session in which ReplayEngine was enabled on-demand.

To recover a valid stack, simply step or continue the session - that is, move forward in history. If the beginning of history is specifically of interest, it can be reached directly by opening a CLI window and issuing the command "dhistory -go_time 1".

We expect this issue to be fixed in the next release.

OpenMPI 1.8.4 with ReplayEngine enabled on older Linux releases.

We have observed a problem with the combination of Replay Engine, Open MPI 1.8.4, and older Linux releases such as RHEL5 (Red Hat Enterprise Linux). When the MPI runtime system closes shared libraries during its startup, a munmap(2) system call may attempt to unmap memory which is in use by Replay Engine. The error message "Unsupported memory access with syscall (11). Conflict with replay private internal memory." is displayed. This error is unrecoverable, but it may be possible to use Replay Engine on the same application by enabling it after the application has completed its MPI_Init call. We have not seen this problem with newer Linux releases such as RHEL6.

TotalView Message Queue and Intel MPI 5.0

By default, the TotalView Message Queue will not work with Intel MPI 5.0 without setting correctly LD_LIBRARY_PATH to the Intel MPI debug libraries. This can be done by sourcing one of the "mpivars.sh/csh" scripts provided by Intel with an added "debug" argument. For example, issue the command "source PATH/impi/5.0.3.048/bin64/mpivars.sh debug", making sure to replace PATH with the path to your Intel MPI compiler installation. TotalView will then properly pick up the MPI message queue information and display it in its Message Queue window.

Memory Debugging and Intel MPI 5.0+

If a user wants to do memory debugging and they statically link their MPI program with the Intel MPI 5.0+ libraries, MemoryScape will detect a Double Allocation error. This is because, starting with Intel MPI 5.0, the MPI libraries redefine free() and MemoryScape depends on the system free() to see the deallocations. To work around this problem, the user will need to link dynamically or fall back to the Intel MPI 4.0+ libraries.

Debugging IBM Platform MPI Jobs in TotalView

Users have seen some issues when trying to use TotalView on an IBM Platform MPI job. If you try to launch the job from the Session Manager, or the Parallel Tab of the Startup Parameters window, TotalView may show an error that the target program has crashed while trying to load shared libraries. When run under mpirun, this error does not show since mpirun sets up the environment correctly. One can avoid the problem by setting the environment variable LD_LIBRARY_PATH to add the path to the library directory containing the missing libraries. The libraries should be in the 'lib' directory that is the same level as the 'bin' directory containing mpirun.

While testing the above issue it was noted that the classic launch method of

```
totalview  mpirun –a –np 4 ./foo
```

did not appear to work correctly. TotalView would attach to all the processes, but only rank 0 was held at the point where the job went parallel. The other processes would run to a point where they were waiting on rank 0. To work around this, launch through the GUI as described above, or use the –tv switch for mpirun

```
mpirun –tv –np 4 ./foo
```

Newer Linux kernels that prohibit non-root access to /proc/self/pagemap and ReplayEngine

If non-root access to /proc/self/pagemap is prohibited, the ReplayEngine will emit an ignored assertion warning when the program being debugged enters record mode for the first time. Furthermore, any unknown syscalls will subsequently be handled a little more slowly.
The change affects Ubuntu 15.04 and likely other new distribution releases.

While using ReplayEngine, attaching to 32-bit application from 64-bit hosts sometimes fails

On some 64-bit hosts, attaching to a 32-bit target fails and results in a crash. The underlying technology behind ReplayEngine assumes that in a 64-bit environment, the target is also a 64-bit application and was not explicitly designed to support a mixed environment.

Benign Warning Messages Displayed when ReplayEngine is run on SuSE Linux Enterprise Server 11 with Service Pack 1. (SLES 11 SP1)

As of the 2016.06 release, ReplayEngine no longer fails when attempting to do replay mode operations (move the target backward into history) on Linux x86-64 platforms running SuSE Linux Enterprise Server 11 with Service Pack 1 but some warning messages such as the following are displayed:

```
127083  client/set_current_child.c:456:set_current_child_fn
[78347:78347]: Failed to restore process name for pid 78357: -5
127084  client/set_current_child.c:179:set_current_child_fn
[78347:78347]: Failed to set process name for pid 78357: -5
```

These messages are benign and your reverse debugging session will work normally.
We have only observed this problem on SLES 11 SP1. It is possible however, that other platforms running the same Linux kernel version (2.6.32.12-0.7) will also run into this problem. The only available workarounds are to update the OS to a more recent release (for example, installing Service Pack 2).

std::string shows as opaque value compiled with Clang 3.5

When trying to debug a program compiled with Clang 3.5 that uses a std::string variable, the variable is listed as type std::string:64 with a value of

Opaque std::basic_string<char,std::char_traits<char>,std::allocator<char>>

When the same program is compiled with the Clang 3.3 compiler, the std string is seen as a simple STL container, and the string value is seen as 'abcd'.

Clang supports a number of optimizations to reduce the size of debug information in the binary. These optimizations work based on the assumption that the debug type information can be spread over multiple compilation units. For instance, Clang does not emit type definitions for types that are not needed by a module and could be replaced with a forward declaration. Further, Clang only emits type info for a dynamic C++ class in the module that contains the vtable for the class.

It has been found that using the **-fstandalone-debug** option which turns off these optimizations works around the problem with the opaque value above.

The **–fstandalone-debug** option is useful when working with 3rd-party libraries that don't come with debug information.
Note that Clang never emits type information for types that are not referenced at all by the program.
-fstandalone-debug is the default on macOS.
-fno-standalone-debug is the default on Linux-x86-64. To work around the opaque value problem above, use the **–fstandalone-debug** option.

Linux - Ubuntu

Memory debugging by linking against the TotalView libraries may not work

There are a number of cases in which it is recommended to link the Heap Interposition Agent (HIA) into the target program to allow memory debugging without having to enable it in the GUI each time. Starting in Ubuntu 12, the linker does not link in libraries that are not directly used by the program. This means that the link line for the agent, with TVLIB pointing to the TotalView library,

```
gcc –g –o memprog memprog.c –L$TVLIB –ltvheap –Wl,-rpath,$TVLIB
```

may not pick up the HIA. Instead, add the option "-Wl,--no-as-needed" before the inclusion of the tvheap library. The new compile/link line will look like

```
gcc –g –o memprog memprog.c –Wl,--no-as-needed –L$TVLIB –ltvheap –Wl,-rpath,$TVLIB
```

CUDA

CUDA 8.0 Debugger API Internal Error

Certain NVIDIA driver versions associated with CUDA 8.0 may provoke "CUDA Debugger API internal error" messages from TotalView or CUDA-GDB. Known driver versions that have this problem are r361, r367, and r375, however the problem may exist in other driver versions. The

internal error typically involves debugging a multi-thread application, when multiple host threads in the process are launching CUDA kernels. NVIDIA has confirmed the driver error, and plans to release a fixed driver version. A release date is not yet available. If the problem occurs, TotalView may print a message and the program may appear to hang.

NVIDIA Pascal Unified Memory Debugger Internal Error

CUDA applications running on Pascal under the debugger may cause a debugger internal error (for example, a SEGV) when the application process exits. Known driver versions that have this problem are r361 and r375, however the problem may exist in other driver versions. The debugger internal error typically involves debugging a CUDA application that exits after using unified memory. NVIDIA has confirmed the driver error, and plans to release a fixed driver version. A release date is not yet available. If the problem occurs, TotalView will exit with an internal error.

Dynamic parallelism not fully supported

With CUDA 5.5-7.0, we have limited support for dynamic parallelism. You should be able to use TotalView in the CUDA 5.5/CUDA 6.5 runtime with applications that display dynamic parallelism; however, we plan improvements to our functionality for displaying the relationships between dynamically launched kernels and navigating the various running kernels.

Layered textures not supported

TotalView does not yet support CUDA layered textures. If you try to examine a layered texture in the TotalView Data Pane, a "Bad address" message will be displayed and you will see "ERROR: Reading Texture memory not currently supported" displayed on the console. If you require layered textures support, please contact TotalView support at support@roguewave.com and let us know how you are using textures so we can develop the best solution to support you.

Solaris

Oracle Studio 12u4 – TotalView unable to evaluate virtual function calls

When debugging applications compiled with the latest version of the Oracle Studio 12u4 compiler, TotalView is unable to call virtual functions through its expression system. This appears to be a shortcoming in debug information from the compiler and should be addressed in cooperation with the Oracle compiler team.

SGI

Memory debugging MPI programs on SGI systems needs special linking

The TotalView and MemoryScape memory debugging Heap Interposition Agent (HIA) technology conflicts with the SGI memory manager when used in MPI programs. The easiest way to get around this problem is to disable the SGI memory manager by unsetting the MPI_MEM_ALIGN environment variable. Without this variable set, the SGI memory manager will not be loaded and the HIA will work correctly, enabling memory debugging to take place.

Cray

Debugging your program within supercomputer environments can often be challenging. Reference the sections below to learn pointers on how to successfully enable memory debugging, and perform reverse debugging on your program within a Cray environment.

Memory debugging on Cray systems

Use the pointers below to help achieve a successful memory debugging session within the Cray environment:

- **Install TotalView on a shared file system visible to the Cray compute nodes.**
  In order for the required memory debugging shared libraries to be located when your program is running on a compute node it is best to install TotalView on a shared file system.
- **Cray TotalView Support Module is not required for TotalView 8.15.0 and later.**
  As of TotalView 8.15.0 and its use of the MRNet tree framework TotalView no longer requires the Cray TotalView Support Library to be installed in order to run. If the MRNet tree framework is turned off then the Cray TotalView Support Module will be required.
- **Statically linking your program against the tvheap_cnl library.**
  One of the most foolproof ways of enabling memory debugging is to statically link against the tvheap_cnl library that is shipped with TotalView. The static library fully supports multithreaded applications. Statically linking your application with the tvheap_cnl library will automatically enable memory debugging in your program when debugged under TotalView and MemoryScape. See the "Linking Your Application with the Agent" discussion in the User Guide for more information on how to statically link your applications with the library.
- **Dynamically linking your program against the tvheap_cnl library.**
  It is possible to dynamically link your application against the dynamic version of the tvheap_cnl library. In this scenario the tvheap_cnl library must be visible to the Cray

Compute Node systems, either through a shared file system or by the Cray environment automatically staging the applications shared library dependencies on the compute node.

- **Do not enable memory debugging on the aprun starter process.**
  Turning on memory debugging for the aprun starter process will cause it to fail and prevent the job from starting. The proper way to enable memory debugging is to use the static or dynamic linking options described above.

Reverse debugging on Cray systems

Use the pointers below to help achieve a successful reverse debugging session within the Cray environment:

- **Do not enable reverse debugging on the aprun starter process.**
  Turning on reverse debugging for the aprun starter process will cause it to fail and prevent the job from starting. The proper way to turn on reverse debugging is to launch your parallel job and reach a breakpoint in the code and then dynamically turn on the Replay Engine reverse debugging option. It will begin recording the execution of the program from that point forward.