

ReplayEngine User Guide

Version 2024.3
September, 2024

PERFORCE

www.perforce.com



© 2024 Perforce Software, Inc. All rights reserved.
© 2007-2024 by Rogue Wave Software, Inc., a Perforce company ("Rogue Wave"). All rights reserved.
© 1998–2007 by Etnus LLC. All rights reserved.
© 1996–1998 by Dolphin Interconnect Solutions, Inc.
© 1993–1996 by BBN Systems and Technologies, a division of BBN Corporation.

Perforce and other identified trademarks are the property of Perforce Software, Inc., or one of its affiliates. Such trademarks are claimed and/or registered in the U.S. and other countries and regions. All third-party trademarks are the property of their respective holders. References to third-party trademarks do not imply endorsement or sponsorship of any products or services by the trademark holder. Contact Perforce Software, Inc., for further details.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise without the prior written permission of Rogue Wave.

Perforce has prepared this manual for the exclusive use of its customers, personnel, and licensees. The information in this manual is subject to change without notice, and should not be construed as a commitment by Perforce. Perforce assumes no responsibility for any errors that appear in this document.

TotalView and TotalView Technologies are registered trademarks of Rogue Wave. TVD is a trademark of Rogue Wave.

Perforce uses a modified version of the Microline widget library. Under the terms of its license, you are entitled to use these modifications. The source code is available at <https://rwkbp.makekb.com/>.

All other brand names are the trademarks of their respective holders.

ACKNOWLEDGMENTS

Use of the Documentation and implementation of any of its processes or techniques are the sole responsibility of the client, and Perforce Software, Inc., assumes no responsibility and will not be liable for any errors, omissions, damage, or loss that might result from any use or misuse of the Documentation

PERFORCE SOFTWARE, INC. MAKES NO REPRESENTATION ABOUT THE SUITABILITY OF THE DOCUMENTATION. THE DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. PERFORCE SOFTWARE, INC. HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE DOCUMENTATION, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT SHALL PERFORCE SOFTWARE, INC. BE LIABLE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT, PUNITIVE, OR EXEMPLARY DAMAGES IN CONNECTION WITH THE USE OF THE DOCUMENTATION.

The Documentation is subject to change at any time without notice.

TotalView by Perforce
<http://totalview.io>

Contents

About ReplayEngine

How ReplayEngine Works	2
Play It Backwards	2
The Process of Recording and Playback	2
System Resource Issues	4
Replaying Your Program	5
Threads and Processes	7
Attaching to Running Programs	8
Saving and Loading the Execution History	9

Using ReplayEngine

Enabling and Disabling ReplayEngine	11
Enabling ReplayEngine at Program Load	11
Enabling and Disabling ReplayEngine for a Loaded Program	12
Enabling Replay	12
Disabling Replay	12
Examining Program State and History	14
Replay Bookmarks	15
Creating bookmarks	15
Activating bookmarks	16
Setting Preferences for ReplayEngine	17
CLI Support	18

Known Issues and Limitations

Performance Issues	23
--------------------------	----

Index	24
--------------------	-----------

About ReplayEngine

- [How ReplayEngine Works](#) on page 2
- [System Resource Issues](#) on page 4
- [Replaying Your Program](#) on page 5
- [Threads and Processes](#) on page 7
- [Attaching to Running Programs](#) on page 8
- [Saving and Loading the Execution History](#) on page 9

ReplayEngine is embedded functionality on the Linux x86-64 platform that allows you to go backwards in a debugging session.

NOTE: If your platform does not support ReplayEngine, the ReplayEngine toolbar and Replay-Engine-related menu items do not display.

For information on using ReplayEngine in a debugging session, see [Using ReplayEngine](#) on page 10.

ReplayEngine complements TotalView, so this discussion assumes a working knowledge of how the TotalView product works.

How ReplayEngine Works

Play It Backwards

The hardest step in locating software bugs involves working backward from a failure to identify the error that caused it. Conventional debugging techniques don't make it easy to find the cause of an error, as they allow you to control program execution only in a forward direction.

Instead of going back to the beginning to try to recreate the conditions of a problem, ReplayEngine starts from the point of failure and works backward in time to find the cause. Recreating the conditions of a crash, sometimes the hardest problem in conventional forward debugging, is no longer necessary. You can now move to locate errors that occurred long before the failure they caused.

The Process of Recording and Playback

In order to move backward in your program, ReplayEngine saves state information as your program executes. This information includes the order in which your program executes and any changes to its data. When ReplayEngine is saving state information, it's in *record mode*.

The saved state information is the program's execution *history*. You can save the execution history at any time and then reload the recording when debugging the executable in a subsequent session.

Using a ReplayEngine command, either by clicking a toolbar button or by entering a command into the CLI, shifts ReplayEngine into *replay mode*. In this mode, you can move to any previously executed statement, at which point ReplayEngine displays its saved state information. The information displayed in replay mode is identical to the information displayed in record mode.

Most debugging commands work the same in replay mode as in record mode. Commands such as viewing a variable or setting a breakpoint work as you would expect. Debugging commands that do not work as expected are those that change or alter a recorded state. Typically, these are commands that:

- Change a variable's value
- Call functions that alter memory
- Run threads asynchronously

If your program unlinks a file in record mode, the file remains always unlinked in replay mode. That is, in replay mode, the file will be unlinked even if you move back to before the unlink statement.

When executing in record mode, your program runs more slowly than without ReplayEngine turned on. Usually, you will not notice the extra execution time. However, when you are in replay mode, the computational overhead required to recreate the program's state may be noticeable. When it needs extra time, ReplayEngine displays a dialog box to cancel the operation.

System Resource Issues

ReplayEngine writes internal information in **/tmp**. Normally, this uses very little space, but in some situations **/tmp** can grow large, and if your system has a small **/tmp** area, ReplayEngine may fill it up. If this occurs, you can:

- Increase the amount of storage allocated to **/tmp**.
- Use the **TMPDIR** environment variable to point to another disk location.
- Define a special TotalView variable, **TVD_REPLAY_TMPDIR**, for ReplayEngine to use as the base directory for writing its temporary information. For example:

```
setenv TVD_REPLAY_TMPDIR /home/user/smith/replayTempDir
```

ReplayEngine also changes the amount of memory your program uses because it keeps history and state information in memory.

While in replay mode, ReplayEngine creates extra processes (usually around 10) but depending on the complexity of the application you are debugging, you may see more. Ignore these processes as they are used only by ReplayEngine.

RELATED TOPICS

Controlling the history and state information storage [Examining Program State and History](#) on page 14

Saving and reloading execution history [Saving and Loading the Execution History](#) on page 9










Replaying Your Program

Before you can replay your program's statements, you must stop your program's execution. Either halt your program, or TotalView can stop execution when your program encounters a breakpoint.

Figure 1, ReplayEngine Toolbar in Active State



The ReplayEngine commands include:

- **Record** (), a toggle that enables and disables ReplayEngine. See [Enabling and Disabling ReplayEngine](#) on page 11.
- **Go Back** () displays the state that existed at the last action point. If no action point is encountered, ReplayEngine displays the state that existed at the start of its recorded history.
- **Prev** () displays the state that existed when the previous statement executed. If that line had a function call, **Prev** skips over the call.
- **Unstep** () displays the state that existed when the previous statement executed. If that line had a function call, ReplayEngine moves to the last statement in that function.
- **Caller** () displays the state that existed before the current routine was called.
- **Back To** () displays the program's state for the line you select. This line must have executed prior to the currently displayed line. If you wish to move forward within replay mode, select a line and select the **Run To** button from the main debugging toolbar.
- **Live** () shifts from replay mode to record mode. It also displays the statement that would have executed had you not moved into ReplayMode.
- **Bookmark** () creates a ReplayEngine bookmark at a selected location.
- **Save** () saves the current replay recording session to a file.

The above commands are also available on the Process menu, with indications of the keyboard shortcuts.

NOTE: The ReplayEngine toolbar commands appear only if you are using TotalView on a Linux-x86-64 machine. On this platform, these buttons are permanently disabled if you do not have a ReplayEngine license.

To move forward within the program's history, use the **Step**, **Next**, **Run To**, and **Out** buttons. These commands do the same thing in replay or record modes.

You can also set breakpoints in previously executed statements. After setting a breakpoint, pressing the **Go** button moves you to that statement. You can transform a breakpoint to an evalpoint if the evalpoint uses simple expressions such as `"if (x=y+z) $stop"`. You cannot, however, create barrierpoints.

If you reach the line that would have been executed if you hadn't gone into replay mode, you are automatically switched back to record mode and you can then resume program execution. You can also switch back to record mode by clicking the **Live** button.

Threads and Processes

When recording, ReplayEngine runs and records one thread at a time. In a multi-threaded or multi-process program, normally ReplayEngine decides the order in which threads are run and recorded. ReplayEngine runs all threads in succession and saves state information for each thread as it executes.

If you need to control the way threads execute, use the TotalView asynchronous threading commands while in record mode. With these commands you can:

- Single-step a process or lockstep group.
- Hold threads so they do not run.

In replay mode, all actions must occur in the same order as recorded. This implies that you cannot influence the order in which threads execute, and you cannot hold a thread.


Attaching to Running Programs

If you attach to a program, ReplayEngine begins recording that program's execution at the time you attached to it. This implies that you cannot go back further than when you attached to the process.

Saving and Loading the Execution History

TotalView can save the current ReplayEngine execution history to a recording file at any time. The saved recording file can then be loaded into TotalView where all the replay options are available to go back and forth within the time boundaries of the saved recording.

To save a recording, either:

- Select the **Save** button on the ReplayEngine toolbar:  , or
- Choose the **Save Recording File ...** menu item in the **File** menu.

A **Save** dialog launches. Edit the default filename if you wish, then select a location to save the file. By default, the name of the recording file is `replay_<executable-name>_<date>_<time>.recording`.

Alternatively, use the CLI **dhistory** command:

```
dhistory -save filename
```

The *filename* can be either a path or a simple file name, in which case it is saved into the current working directory. If no *filename* is specified, the recording is saved in the current working directory as `replay_pid-hostname.recording`.

The saved recording can be loaded into TotalView as follows:

- At startup, using the same syntax as when opening a core file:
totalview executable recording-file
TotalView recognizes the recording file for what it is and acts appropriately.
- After TotalView is running, using the **dattach** command with option **-c**:
dattach executable -c recording-file
- On the Start Page view by selecting **Load Core File or Replay Recording File**. (See [Debug a Core or Replay Recording File](#) in the *TotalView User Guide*.)

Performing any of the above displays the dialog for selecting the record file and application used during the recording session when the recording session was saved.

Again, TotalView recognizes it is dealing with a recording file.

Using ReplayEngine

- [Enabling and Disabling ReplayEngine](#) on page 11
- [Examining Program State and History](#) on page 14
- [Replay Bookmarks](#) on page 15
- [Setting Preferences for ReplayEngine](#) on page 17
- [CLI Support](#) on page 18

There is very little difference between running TotalView and running ReplayEngine. With ReplayEngine enabled on a running program, use the special buttons **Go Back**, **Prev**, **Unstep**, **Caller**, or **Back To** to go back in your program's history to the statement you wish to examine.

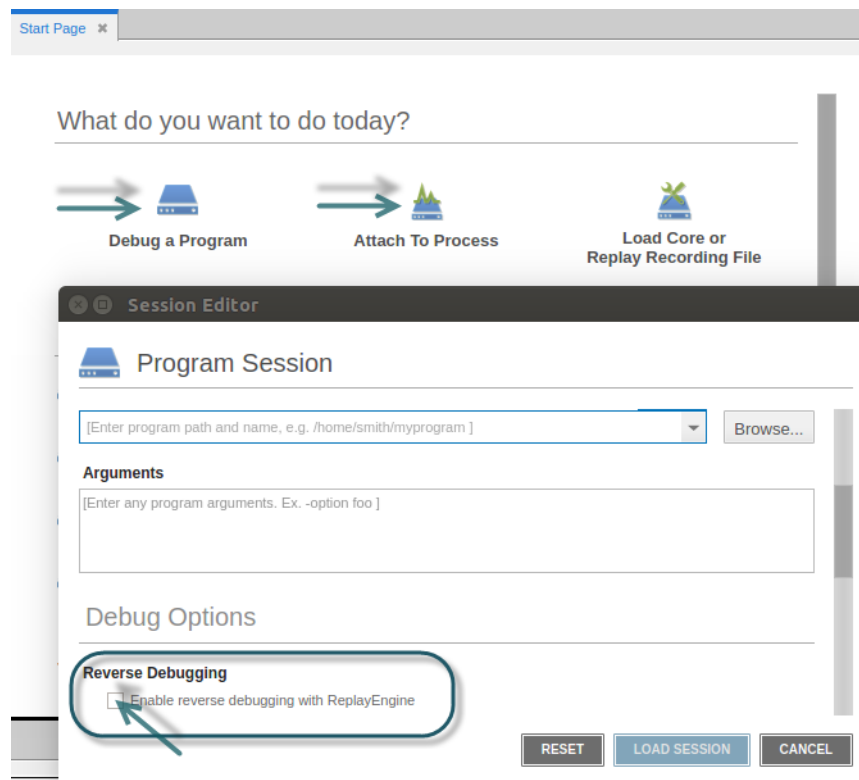
Enabling and Disabling ReplayEngine

You can prepare a program for replay when you first load it into TotalView. Once the program is loaded, there are a number of ways to enable replay.

Enabling ReplayEngine at Program Load

To enable ReplayEngine when loading a program into TotalView, select the checkbox **Enable reverse debugging with Replay Engine** in either the **Debug a Program** or **Attach to Process** dialogs available on the Start Page.

Figure 2, Enabling ReplayEngine from the Start Page



For a program already under TotalView control, you can use the Command Line view to enter the **dattach** command with the **-replay** option.

```
dattach -replay program-path
```

For a new program, ReplayEngine begins recording instructions as soon as you begin executing the program. For a running process to which you have attached, ReplayEngine starts recording the next time you restart the process.

Enabling and Disabling ReplayEngine for a Loaded Program

Once a program is loaded into TotalView, enable and disable replay via several options.

Enabling Replay

Replay behavior differs depending on whether or not program execution has begun.

The program is not yet executing

If the program is loaded but has not started executing, enable ReplayEngine in any of the following ways:

- Click the **Record** toolbar button



- Select the **Debug > Enable ReplayEngine** menu item
- Execute the CLI command **dhistory -enable**

ReplayEngine begins recording when the process starts executing. If you restart the process, ReplayEngine begins recording from the beginning of process execution.

To stop recording, exit the program and explicitly disable ReplayEngine. You cannot turn replay off while a process is executing.

The program is executing but halted

If a process is already executing and stopped, you can immediately enable replay with any of the methods used when your program is not yet executing — but replay will then be enabled *only* while the program executes that single time. At process exit and restart, ReplayEngine will no longer be enabled unless you explicitly re-enable it.

Enabling ReplayEngine during program execution also means that you cannot step backward beyond the point at which ReplayEngine was enabled.

Disabling Replay

You cannot disable ReplayEngine for a process that is executing. You must:

1. Kill the executing process.

2. Disable ReplayEngine either by
 - Clicking the **Record** button or de-selecting the **Debug > Enable ReplayEngine** menu item, both of which are toggles.
 - Entering **dhistory -disable** in a CLI prompt focused on the process.

If you now restart the process, ReplayEngine will be disabled for the executing process.

After killing the process, you can also return to the Start Page, click the **Edit** option for your most recent session (the pencil icon), and then de-select the **Enable reverse debugging with Replay Engine** option in the resulting dialog.

Examining Program State and History

After enabling ReplayEngine, you can begin controlling your program's execution using the same execution commands used when ReplayEngine is not enabled. For example, you might set a breakpoint and press the **Go** button, or select a line and press the **Run To** button.

When you wish to review the program's state at some previous point, halt your program and use the **Go Back**, **Prev**, **Unstep**, **Caller**, or **Back To** buttons to go to the statement you wish to examine. These four buttons are similar to the **Next**, **Step**, **Out**, and **Run To** toolbar buttons, differing only in that the Replay buttons go backwards in the program's history. The **Process** pull-down menu contains the menu bar equivalents to these commands.

While you are in replay mode, note that the **Next**, **Step**, **Out**, and **Run To** toolbar buttons are still active to move forward in the history.

When you're in replay mode, TotalView changes the highlight line from yellow to orange within the Source Pane.


Figure 3, Source View with ReplayEngine

The screenshot shows a source code editor with the following code:

```

1152 my_pid = getpid();
1153
1154
1155 if (argc > arg_count && (arg = argv[arg_count]) && isdigit(*arg))
1156 {
1157     fork_count = atoi (arg);
1158     arg_count++;
1159 } /* if */
1160
1161 if (args_ok && argc > arg_count && (arg = argv[arg_count]) && isdigit(*arg))
1162 {
1163     threads_per_copy = atoi (arg);
1164     if (threads_per_copy > FORK_LOOP_MAX_THREADS)
  
```

Line 1155 is highlighted in orange. A play button icon (a red triangle pointing right) is located to the left of line 1155. A blue box highlights the line number 1155 and the code on that line.

The Source View always shows the last line executed – the “Live” location – within record mode using the  symbol. When you are in replay mode, this symbol is where ReplayEngine shifts from replay mode back to record mode.

NOTE: ReplayEngine supports process width only; therefore, the scoping commands at the far left side of the main toolbar have no effect in replay mode.

Replay Bookmarks

Replay bookmarks mark a point in the execution of a program, allowing you to quickly jump back to that point in time.


NOTE: Bookmarks are set at a specific point during the program's execution history. If you restart your program, it is not guaranteed that the execution history will be the same. Even if you did not recompile your application, data inputs and other factors may alter the execution paths taken and, as a result, a bookmark might not map to the same point in execution history as when you initially created it.

Creating bookmarks

Create bookmarks at any point while stepping through the code of your program by:

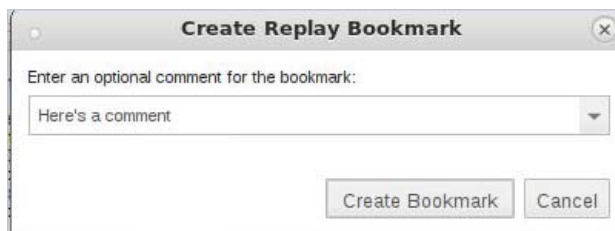
- Selecting the **Debug > Create Replay Bookmark...** menu item.



- Using the **Ctrl+Shift+D** keyboard shortcut.
- Clicking the **Bookmark** icon () on the ReplayEngine toolbar.

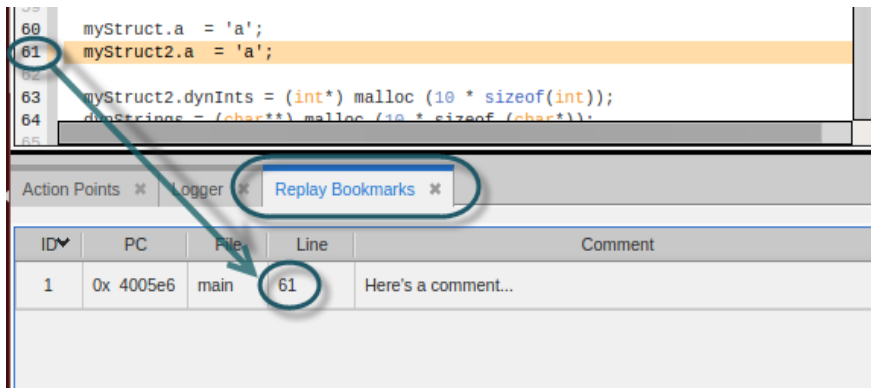
You can add an optional comment from the **Create Replay Bookmark** dialog.

Figure 4, Create Replay Bookmark dialog



Once the bookmark is created, it displays in the **Replay Bookmarks view**.

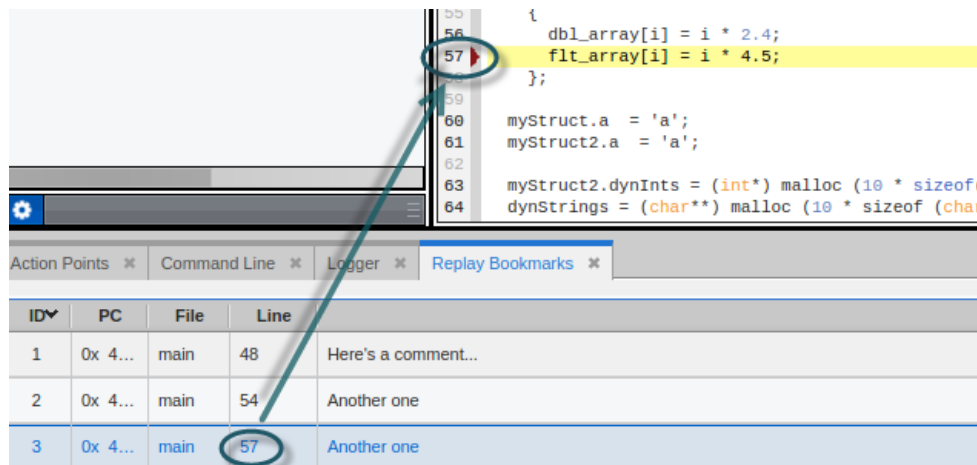
Figure 5, Replay Bookmarks view




Activating bookmarks

Activate a bookmark by double-clicking on the bookmark in the **Replay Bookmarks view**. ReplayEngine takes you to that point in your program's execution history where you have the full power of TotalView and Replay-Engine to examine the state of your program, run forward or backward, set breakpoints, and so forth.

Figure 6, Activated Replay bookmark



Return to the live point in your program by clicking the **Live** icon  on the ReplayEngine toolbar, selecting **Live** from the **Process** menu, or using the **Alt+Shift+L** keyboard shortcut.

Setting Preferences for ReplayEngine

You can set these preferences for ReplayEngine:

- The maximum amount of memory to allocate to ReplayEngine
- The preferred behavior when the memory maximum is reached

Set memory size preference in the CLI, like so:

```
dset TV::replay_history_size value
```

The value can be a number, or a number followed by 'K' or 'M' for kilobytes or megabytes. The default value '0' specifies to limit the maximum size by available memory only.

For example:

```
dset TV::replay_history_size 1024M
```

Sets the maximum history size to 1024 megabytes.

```
dset TV::replay_history_size 1000000
```

Sets the maximum history size to 1000000 bytes.

The behavior preference defines ReplayEngine behavior when the maximum memory size is reached. By default, the oldest history is discarded so that recording can continue. Alternatively, you can specify that the recording process simply stops when the allocated memory is used up.

For example:

```
dset TV::replay_history_mode 1
```

Discard oldest history and continue recording (the default).

```
dset TV::replay_history_size 1000000
```

Stop the process being recorded and stop recording.

CLI Support

CLI support is accessed through the Command Line view on the interface. If that view is not currently displayed, you can show it either by right-clicking in the menu and toolbar area and selecting the Command Line view in the context menu, or through the Window > Views menu.

- The **dload** and **dattach** CLI commands have the **-replay** option for enabling and disabling ReplayEngine. For example:

```
dload -replay myProgram
```

- The **dgo**, **dnnext**, **dnnexti**, **dout**, **dstep**, **dstepi**, and **duntil** commands let you step or run backwards by using the **-back** option. For example:

```
dnnext -back  
duntil -back 22
```

The **dhistory** command has the following options:

- **-info**
Displays the current time. The output of this command shows an integer value followed by an address. The first integer value is a virtual timestamp. This virtual timestamp does not refer to the exact point in time; it has a granularity that is typically a few lines of code. The address value is a PC value that corresponds to a precise point within that block of code.
- **-enable**
If the program has not been started, ReplayEngine is enabled when the program is started. If the program is already running, ReplayEngine is enabled immediately. Recording begins at the point that ReplayEngine was enabled and moving back beyond that point is not possible.
- **-disable**
Disables ReplayEngine for the next restart for the process.
- **-create_bookmark** *[comment]*
Creates a Replay bookmark at the current execution location so you can return to it later. You can specify an optional comment to this command and it will be stored with the bookmark for display when you use the **show_bookmarks** command. A bookmark is created with a unique numeric ID, which is the return value
- **-goto_bookmark** *ID*
Goes to the bookmark with the specified ID. This returns the focus process to the execution location where the bookmark was first created.
- **-go_live**
Resets the process back to record mode.

- **-show_bookmarks**

Displays all Replay bookmarks. This command shows the bookmark ID along with information about what line number, PC and function the bookmark is on. If you added a comment to help you remember the significance of the bookmark, it displays this as well.

- **-delete_bookmark *ID***

Deletes the bookmark with the given ID.

- **-clear_bookmarks**

Deletes all Replay bookmarks.

- **-get_time** — Deprecated. Use the bookmark options.

Displays the current time. The output of this command shows an integer value followed by an address. The first integer value is a virtual timestamp. This virtual timestamp does not refer to the exact point in time; it has a granularity that is typically a few lines of code. The address value is a PC value that corresponds to a precise point within that block of code.

- **-go_time *time*** — Deprecated. Use the bookmark options.

Moves the process to an execution point represented by the *time* argument. The *time* argument is a virtual timestamp as reported by **dhistory -get_time**. You cannot use this command to move to a specific instruction but you can use it to get to within a small block of code (usually within a few lines of your intended point in execution history). This command is typically used either for roughly bookmarking a point in code or for searching execution history. It may need to be combined with stepping and **duntil** commands to return to an exact position.

These CLI commands are explained in detail in the *TotalView Reference Guide*.

Known Issues and Limitations

- [Limitations](#) on page 20
- [Performance Issues](#) on page 23

Limitations

- **Obscure instructions:** Use of AMD 3DNow! and other extended AMD instructions is not supported (though Intel SSE, SSE2, SSE3 and SSE4 instructions are supported). Instructions that modify CS, DS, ES or SS registers are also not supported.
- **AsyncIO:** ReplayEngine does not support asynchronous IO operations. **io_cancel**, **io_destroy**, **io_getevents**, **ioperm**, **iopl**, **io_setup**, and **io_submit** system calls are all unsupported.
- **Exec:** ReplayEngine does not support the **execve** syscall, as used by libc's **execl()**, **execlp()**, **execle()**, **execv()**, **execvp()**, and **execve()** functions. If the target program attempts to issue this system call, forward execution will not be possible beyond this point (though reverse execution is still possible).
- **Obscure system calls:** Certain rarely used system calls are not supported. If the target program attempts to issue an unsupported system call, forward execution will not be possible beyond this point (though reverse execution is still possible). The following system calls are either esoteric or obsolete, and only maintained in the kernel for backward compatibility with binaries written for early 2.x series kernels: **ssetmask**, **modify_ldt**, **pivot_root**, **vm86**, and **unshare**.
- **Use of setrlimit():** If the target program uses **setrlimit** to reduce the amount of memory, processes, or other resources consumed, ReplayEngine may not be able to operate properly due to lack of resources.

- **Use of x86 inter-segment (aka 'far') jumps/calls:** ReplayEngine does not support the use of far jumps/calls in the target program. Any such attempt will result in forward execution not being able to continue from the point at which the far jump/call instruction is issued.
- **Non-executable memory:** ReplayEngine ignores the executable status of memory when running code, so code that would usually fail because it is in non-executable memory will run successfully.
- **Disk usage:** Depending on the target program, ReplayEngine can create large temporary files within `/tmp`. See [System Resource Issues](#) for information on how to use alternative temporary directories.
- **Self-modifying code:** ReplayEngine mostly works with self-modifying code, but in some situations the effects of writing into the currently executing “basic block” may be delayed (that is, writing instructions just ahead of the current program counter such that the processor executes the newly written code by virtue of “running in to” rather than “jumping to” it).
- **Shared memory accesses straddling valid and invalid pages:** Accessing shared memory where the instruction's operand straddles a page boundary such that the first part of the operand is in accessible shared memory, but the second part is in mapped shared memory which is not backed by a valid shared object (e.g. because the file which is mapped has been truncated) should receive signal **SIGBUS**. Under ReplayEngine, a target program making such an access will not receive **SIGBUS** but will read zeros for the part of the operand that straddles into unbacked memory. Note that normal attempted access to shared memory not backed by a shared object will generate a **SIGBUS** as normal; the issue applies only when a single instruction's access that lies half in valid memory and half in invalid memory that should generate a **SIGBUS**.
- **Breakpoints:** All breakpoints used with ReplayEngine work like hardware breakpoints. In particular, if the code where the breakpoint resides is not modified, writing to that code will not remove the breakpoint, and setting a breakpoint that is not at the first byte of an instruction will have no effect.
- **System call output buffers:** Any system calls that write to memory must be passed a buffer entirely within writable memory. For example, if `read()` is passed an 8k buffer of which only the first 4k is in user-writable memory, if that `read()` would normally return 4k or fewer characters then natively it may succeed, but on ReplayEngine it will fail with **EFAULT**. If a system call that writes to memory is passed a buffer which is not in writable memory at all, but fails for some other reason before the kernel tries to write to the buffer, then natively it may fail with some error other than **EFAULT**, but on ReplayEngine it may fail with **EFAULT**. If two buffers which overlap are passed to a system call which writes to both of them or reads from one and writes to the other, the behavior in ReplayEngine may differ from the native behavior (although behavior in such cases is liable to vary between kernel versions, too.)

- **Adjust Flag:** According to the Intel manuals, the state of the Adjust Flag (AF) after some instructions is “undefined.” On some processor models, different executions of the same code can produce different states of AF. If the behavior of a program depends on the state of AF when it is supposed to be undefined, the program may not run correctly with ReplayEngine.
- **SIGCHLD while attaching:** If a **SIGCHLD** arrives for a process while ReplayEngine is in the middle of attaching to the process, the **SIGCHLD** may be silently lost. Once the process has been attached to, **SIGCHLD** is handled normally.
- **Loading a previous recording session:** The successful reloading and debugging of a previously saved replay recording session requires that both the environment that saved the session and the environment replaying the recording session be exactly the same.

Performance Issues

High TLB rates with certain multi-threaded target programs

When reverse debugging an application in which many threads make frequent system calls on a multi-processor platform, binding the application process to a single processor can improve performance. This is because such applications put stress on ReplayEngine's heap management, which in turn stresses the processor's TLB (translation lookaside buffer). If the application is bound to a single processor, it is less likely to suffer TLB misses caused by process migration. Since user threads are automatically serialized during reverse debugging, there is no loss of concurrency due to binding.

If the application is to be launched under TotalView, one way to accomplish binding is to preface the totalview command with a **taskset(1)** command specifying a single processor. For example:

```
taskset --cpu-list 3 totalview -replay myapp
```

To accomplish binding when TotalView is to be attached to a running application, find the PID (process identifier) of the application process, and use **taskset** to bind that process to a single processor before attaching to it with TotalView. For example:

```
taskset --pid --cpu-list 3 <PID of myapp>
```

We have noticed the need for such binding when debugging MySQL applications with ReplayEngine.

Index

Symbols

/tmp

use with ReplayEngine 4

B

BackTo toolbar button

(ReplayEngine) 5

barriers with ReplayEngine 6

breakpoints

and ReplayEngine 6

C

Caller toolbar button

(ReplayEngine) 5

CLI

commands supporting
ReplayEngine 18

CLI commands

-back option for replay
debugging 18

dattach -replay 18

dhistory 18

dhistory -disable 13

dhistory -replay 12

dload -replay 18

dnext and dnexti -back
commands 18

supporting ReplayEngine 18

code highlighting

during ReplayEngine replay 14

code marker for end of replay

(ReplayEngine) 14

D

dattach command

-replay option for
ReplayEngine 18

Debug > Enable ReplayEngine

menu item

disabling replay 12

enabling replay 12

Debug menu 12

Debug New Program dialog

enabling ReplayEngine 11

debugging

CLI commands supporting re-
play debugging 18

switching between replay and
live debugging 6

debugging behavior in Replay-

Engine replay mode 6, 14

debugging commands in

ReplayEngine 2

dhistory command 18

disabling ReplayEngine 12

Debug > Enable ReplayEngine
menu item 12

with CLI dhistory -disable
command 13

with Process > Startup Param-
eters dialog 13

with Record button 12

dload command

-replay option for
ReplayEngine 18

dload -replay and -noreplay

options 18

dnext and dnexti -back command-
line options 18

E

enabling ReplayEngine

with CLI dhistory command 12

with Debug > Enable Replay-
Engine menu item 12

eval breakpoints with

ReplayEngine 6

G

GoBack toolbar button

(ReplayEngine) 5

H

highlighting

during ReplayEngine replay 14

L

Live toolbar button

(ReplayEngine) 5, 6

M

menus

Debug 12

Process 5, 14

save ReplayEngine recording
file 9

P

performance with ReplayEngine 3

Prev toolbar button

(ReplayEngine) 5

process creation by

ReplayEngine 4

Process menu 5

ReplayEngine commands 14

program state in ReplayEngine 2

R

Record button (ReplayEngine)

disabling replay 12

record mode 2

Record toolbar button

(ReplayEngine) 5

recording program history 2

replay mode 2

ReplayEngine

breakpoints, evalpoints, and
barrierpoints 6

canceling operations in replay
mode 3

CLI command support 18

CLI commands for debugging
during replay 18

debugging commands 2

debugging commands during
replay 6, 14

debugging through the CLI 18

dhistory CLI command 18

disabling 12

- enabling 11
- end of replay marker 14
- highlighting in code during
replay 14
- performance when using 3
- process creation 4
- program state 2
- record mode 2
- replay mode 2
- saving a recording file 9
- scoping disabled during
replay 14
- supports only process
width 14
- switching between replay and
live debugging 6
- thread execution in replay
mode 7
- ReplayEngine commands 5
- replayengine_about 1

S

- scoping
 - disabled during replay
(ReplayEngine) 14
- space requirements with
ReplayEngine 4
- state of program in ReplayEngine 2

T

- temp space requirements with
ReplayEngine 4
- temp space with Cray XT 4
- thread execution during Replay-
Engine replay 7
- TMPDIR environment variable and
ReplayEngine 4
- toolbars
 - ReplayEngine 14
- TVD_REPLAY_TMPDIR variable 4

U

- Unstep toolbar button
(ReplayEngine) 5
- using ReplayEngine 10