Contents

About this Guide

Overview ................................................................. 1
TotalView Family Differences .................................... 2
TotalView for HPC Documentation ............................... 3
Conventions ............................................................ 4
Contacting Us ......................................................... 5

Part 1: CLI Commands

Chapter 1  CLI Command Summary  .................. 7

Chapter 2  CLI Commands  ............................ 19

Command Overview .................................................... 19
  General CLI Commands ........................................... 19
  CLI Initialization and Termination Commands ................. 19
  Program Information Commands ............................. 20
  Execution Control Commands .................................. 21
  Action Points ....................................................... 21
  Platform-Specific CLI Commands ............................ 22
  Other Commands .................................................. 22

alias ................................................................. 23
capture ............................................................. 25
dactions .............................................................. 27
dassign ............................................................... 32
dattach ............................................................... 34
dbarrier ............................................................... 37
dbreak ................................................................. 42
dcache ............................................................... 46
dcalltree ............................................................. 47
dcheckpoint ......................................................... 51
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcont</td>
<td>53</td>
</tr>
<tr>
<td>dcuda</td>
<td>55</td>
</tr>
<tr>
<td>ddelete</td>
<td>60</td>
</tr>
<tr>
<td>ddetach</td>
<td>62</td>
</tr>
<tr>
<td>ddisable</td>
<td>64</td>
</tr>
<tr>
<td>ddlopen</td>
<td>66</td>
</tr>
<tr>
<td>ddown</td>
<td>69</td>
</tr>
<tr>
<td>denable</td>
<td>71</td>
</tr>
<tr>
<td>dexamine</td>
<td>73</td>
</tr>
<tr>
<td>dfocus</td>
<td>76</td>
</tr>
<tr>
<td>dfocus</td>
<td>79</td>
</tr>
<tr>
<td>dgo</td>
<td>82</td>
</tr>
<tr>
<td>dga</td>
<td>84</td>
</tr>
<tr>
<td>dgroups</td>
<td>86</td>
</tr>
<tr>
<td>dhalt</td>
<td>91</td>
</tr>
<tr>
<td>dheap</td>
<td>92</td>
</tr>
<tr>
<td>dhistory</td>
<td>94</td>
</tr>
<tr>
<td>dhold</td>
<td>96</td>
</tr>
<tr>
<td>dkill</td>
<td>98</td>
</tr>
<tr>
<td>dlappend</td>
<td>99</td>
</tr>
<tr>
<td>dlist</td>
<td>100</td>
</tr>
<tr>
<td>dload</td>
<td>103</td>
</tr>
<tr>
<td>dmstat</td>
<td>106</td>
</tr>
<tr>
<td>dnext</td>
<td>109</td>
</tr>
<tr>
<td>dnexti</td>
<td>112</td>
</tr>
<tr>
<td>dout</td>
<td>115</td>
</tr>
<tr>
<td>dprint</td>
<td>118</td>
</tr>
<tr>
<td>dptsets</td>
<td>124</td>
</tr>
<tr>
<td>drerun</td>
<td>127</td>
</tr>
<tr>
<td>drestart</td>
<td>130</td>
</tr>
<tr>
<td>drun</td>
<td>132</td>
</tr>
<tr>
<td>dsession</td>
<td>135</td>
</tr>
<tr>
<td>dset</td>
<td>136</td>
</tr>
<tr>
<td>dstatus</td>
<td>139</td>
</tr>
<tr>
<td>dstep</td>
<td>143</td>
</tr>
<tr>
<td>dstepe</td>
<td>147</td>
</tr>
</tbody>
</table>
Chapter 3

**CLI Namespace Commands**

Command Overview ........................................ 179
  Accessor Functions .................................. 179
  Helper Functions .................................. 180
actionpoint ............................................. 181
dec2hex .................................................. 184
dll ......................................................... 185
errorCodes ............................................. 187
expr ....................................................... 189
focus_groups .......................................... 191
focus_processes .................................... 192
focus_threads ....................................... 193
group ................................................... 194
hex2dec ................................................ 196
process ............................................... 197
read_symbols ....................................... 201
respond ............................................. 202
scope .................................................. 203
source_process_startup ......................... 205
symbol ............................................... 206

dunhold ................................................ 150
dunset ............................................... 152
duntil ............................................... 153
dup ...................................................... 156
dwait .................................................. 158
dwatch ............................................... 159
dwhat ................................................. 162
dwhere .............................................. 166
dworker ............................................... 169
exit .................................................... 170
help ..................................................... 171
quit ..................................................... 172
spurs .................................................. 173
stty .................................................... 178
unalias ............................................... 179
Chapter 4  
Batch Debugging Using tvscript  
Overview ................................................................. 235

tvscript Command Syntax ........................................... 236
  tvscript Options .................................................... 240
tvscript External Script Files ................................. 246
  Logging Functions API ........................................... 246
  Process Functions API .......................................... 246
  Thread Functions API ........................................... 246
  Action Point API .................................................. 246
  Event API .......................................................... 247
  Example tvscript Script File .................................. 248

Chapter 5  
TotalView Variables  
Overview ................................................................. 252

Top-Level (::) Namespace ........................................... 253
TV:: Namespace ....................................................... 261
TV::MEMDEBUG:: Namespace ..................................... 297
TV::GUI:: Namespace .............................................. 299

Chapter 6  
Creating Type Transformations  
Overview ................................................................. 307

Why Type Transformations ........................................ 308
Creating Structure and Class Transformations ............... 310
  Transforming Structures ....................................... 310
  build_struct_transform Function .......................... 312
  Type Transformation Expressions ............................ 312
  Using Type Transformations .................................. 316

C++View ............................................................... 318
  Writing a Data Display Function ............................. 319
    TV_ttf_type_ascii_string .................................. 319
    TV_ttf_type_int ............................................ 319
  Templates ......................................................... 321
  Precedence - Searching for TV_ttf_display_type .......... 322
  TV_ttf_add_row ................................................... 322
  TV_ttf_ec_ok .................................................... 323

thread ........................................................................ 219
type ........................................................................ 222
type_transformation ................................................. 225
Part 2: Running TotalView

Chapter 7 TotalView Command Syntax 338
Overview ........................................... 338
Command-Line Syntax ................................ 339
Command-Line Options ................................ 340

Chapter 8 TotalView Debugger Server Command Syntax 352
Overview ........................................... 352
The tvdsvr Command and Its Options ................. 353
Description ......................................... 353
Options ................................................ 353

Contents vii
Part 3: Platforms and Operating Systems

Chapter 9  Platforms and Compilers

Overview ................................................................. 360
Compiling with Debugging Symbols ......................... 361
  Apple Running Mac OS X ................................. 361
  IBM AIX on RS/6000 Systems ......................... 361
  IBM Blue Gene ............................................... 363
  IBM Power Linux ............................................. 363
  Linux Running on an x86 Platform ..................... 363
  Linux Running on an x86-64 Platform ................. 364
  Linux Running on an Itanium Platform ............... 365
  Sun Solaris ..................................................... 365
Using gnu_debuglink Files ........................................ 366
  Total View Command-Line Options and CLI State Variables 366
  Searching for the gnu_debug_link File ............... 367
Linking with the dbfork Library ............................... 368
  dbfork on IBM AIX on RS/6000 Systems ............... 368
    Linking C++ Programs with dbfork .................. 368
  Linux or Mac OS X ......................................... 369
  SunOS 5 SPARC .............................................. 369

Chapter 10  Operating Systems

Operating Systems ................................................ 371
Supported Operating Systems .................................. 372
Troubleshooting Mac OS X Installations ..................... 373
  Problem Description ....................................... 373
    For Mac OS X Versions 10.8 (Mountain Lion) or Later 373
    For Mac OS X Versions 10.11 (Capitan) or Later ...... 373
    Remotely Debugging without Console Access .......... 374
Mounting the /proc File System ............................... 375
  Mounting /proc with SunOS 5 .......................... 375
Swap Space ....................................................... 376
  Swap Space on IBM AIX .................................. 376
  Swap Space on Linux ..................................... 376
  Swap Space on SunOS 5 .................................... 377
Shared Libraries .................................................. 378
  Changing Linkage Table Entries and LD_BIND_NOW .... 378
# Debugging Your Program's Dynamically Loaded Libraries

- dlopen Options for Scalability .......................... 382
- Filtering dlopen Events ................................. 382
- Handling dlopen Events in Parallel ..................... 384
- Known Limitations ....................................... 385

# Chapter 11 Architectures 388

- Remapping Keys ........................................... 386
- Expression System ......................................... 387
- Expression System on IBM AIX-Power and Blue Gene/Q ... 387

Overview .................................................. 388

- AMD and Intel x86-64 .................................... 389
  - x86-64 General Registers ............................... 389
  - x86-64 Floating-Point Registers ....................... 390
  - x86-64 FPCR Register .................................. 391
    - Using the x86-64 FPCR Register ..................... 392
  - x86-64 FPSR Register .................................. 392
  - x86-64 MXCSR Register ................................ 393

- Power Architectures ..................................... 394
  - Power General Registers ............................... 394
  - Blue Gene Power Registers ............................. 396
  - Blue Gene/Q QPX Floating-Point Registers .......... 397
  - Power MSR Register ................................... 397
  - Power Floating-Point Registers ....................... 398
  - Power FPSCR Register .................................. 398
    - Using the Power FPSCR Register ..................... 400

- Intel IA-64 ................................................. 401
  - Intel IA-64 General Registers ......................... 401
  - IA-64 Processor Status Register Fields (PSR) ........ 402
  - Current Frame Marker Register Fields (CFM) .......... 403
  - Register Stack Configuration Register Fields (RSC) .. 404
  - Previous Function State Register Fields (PFS) ........ 404
  - Floating Point Registers .............................. 405
  - Floating Point Status Register Fields ................. 405

- Intel x86 .................................................. 407
  - Intel x86 General Registers ........................... 407
  - Intel x86 Floating-Point Registers .................... 408
  - Intel x86 FPCR Register ................................ 409
    - Using the Intel x86 FPCR Register ................. 410
  - Intel x86 FPSR Register ................................ 410
Contents

Part 4: Appendices

Appendix A  MPI Startup ................................................. 417
  Overview ......................................................... 417
  Customizing Your Parallel Configuration ...................... 418
  TotalView ....................................................... 418
  Standalone MemoryScape ....................................... 419
  Example Parallel Configuration Definitions .................. 420

Index ................................................................. 425
About this Guide

Overview

The information in this guide is organized in parts:

- **Part I, “CLI Commands,”** on page 6 contains descriptions of all the CLI commands, the variables that you can set using the CLI, and other CLI-related information.

- **Part II, “Running TotalView,”** on page 337 documents all possible command-line options as well as those that customize the behavior of the `tvdsvr`.

- **Part III, “Platforms and Operating Systems,”** on page 359 provides general information on compilers, runtime environments, operating systems, and supported architectures.

- **Part IV, “Appendices,”** on page 416 includes Appendix A which describes how to create startup profiles for environments that TotalView does not define.
TotalView Family Differences

This manual describes the TotalView for HPC, TotalView Team, and TotalView Individual debuggers. Each of these supports the use of the CLI debugger as well. In all cases, TotalView for HPC and TotalView Team have the same features, differing only in the way they are licensed. TotalView Individual differs in its feature set.

**NOTE >>** The most fundamental differences between TotalView Team and TotalView for HPC are the way resources are shared and used. When you purchase TotalView Team, you are purchasing “tokens.” These tokens represent debugging capabilities. For example, if you have 64 tokens available, 64 programmers could be active, each debugging a one-process job; or two programmers, each debugging a 32-process job. In contrast, a TotalView for HPC license is based on the number of users and the number of licensed processors. You’ll find more precise information on our web site.

The basic differences are:

<table>
<thead>
<tr>
<th>TotalView Team &amp; HPC</th>
<th>TotalView Individual</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute on any licensed computer of the same architecture</td>
<td>Node locked.</td>
<td>You can execute TotalView Individual only on the computer you install it on.</td>
</tr>
<tr>
<td>Number of users is determined by license</td>
<td>Only one user</td>
<td>The TotalView for HPC license limits the number of users. TotalView Team does not.</td>
</tr>
<tr>
<td>Number of processes limited by license. No limit on threads</td>
<td>No more than 16 processes and threads.</td>
<td></td>
</tr>
<tr>
<td>Your license determines the number of processors upon which your program can run.</td>
<td>A program can execute on no more than two cores.</td>
<td>TotalView for HPC licenses the full capabilities of all machines upon which it runs. TotalView Team can acquire part of your machine.</td>
</tr>
<tr>
<td>Processes can execute on any computers in the same network.</td>
<td>Remote processes are not allowed.</td>
<td>Processes must execute on the installed computer.</td>
</tr>
<tr>
<td>Remote X Server connections allowed.</td>
<td>No remote X Server connections are allowed.</td>
<td>Programmers cannot remotely log into a computer and then execute TotalView Individual.</td>
</tr>
<tr>
<td>Memory debugging is bundled.</td>
<td>No memory debugging.</td>
<td></td>
</tr>
</tbody>
</table>
## TotalView for HPC Documentation

The following table describes all available TotalView for HPC documentation.

<table>
<thead>
<tr>
<th>Product</th>
<th>Title</th>
<th>Description</th>
<th>HTML</th>
<th>PDF</th>
<th>Print</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General TotalView for HPC Documentation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Getting Started with TotalView for HPC Products</td>
<td>Introduces the basic features of TotalView for HPC, MemoryScape, and         X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ReplayEngine, with links for more detailed information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TotalView for HPC Platforms and System Requirements</td>
<td>Defines platform and system requirements for TotalView for HPC, MemoryScape,</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>and ReplayEngine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TotalView for HPC Evaluation Guide</td>
<td>Brochure that introduces basic TotalView for HPC features</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>User Guides</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TotalView for HPC User Guide</td>
<td>Primary resource for information on using the TotalView for HPC GUI and the</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Debugging Memory Problems with MemoryScape</td>
<td>How to debug memory issues, relevant to both TotalView for HPC and the      X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MemoryScape standalone product</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reverse Debugging with Replay Engine</td>
<td>How to perform reverse debugging using the embedded add-on ReplayEngine     X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Reference Guides</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TotalView for HPC Reference Guide</td>
<td>A reference of CLI commands, how to run TotalView for HPC, and platform-specific</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>detail</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Installation Guides</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TotalView for HPC Install Guide</td>
<td>Installing TotalView for HPC and the FLEXlm license manager</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MemoryScape Install Guide</td>
<td>Installing MemoryScape as a standalone product</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>In-Product Help</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Help screens launched from within the product's GUI</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Conventions

The following table describes the conventions used in this book:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Brackets are used when describing optional parts of a command.</td>
</tr>
<tr>
<td>arguments</td>
<td>In a command description, text in italics represents information you enter. Elsewhere, italics is used for emphasis.</td>
</tr>
<tr>
<td><strong>Bold text</strong></td>
<td>In a command description, <strong>bold text</strong> represents keywords or options that must be entered exactly as displayed. Elsewhere, it represents words that are used in a programmatic way rather than their normal way.</td>
</tr>
<tr>
<td><strong>Example text</strong></td>
<td>In program listings, this represents a program or something you’d enter in response to a shell or CLI prompt. Bold text here indicates exactly what you should type. If you’re viewing this information online, example text is in color.</td>
</tr>
</tbody>
</table>
Contacting Us

Please contact us if you have problems installing this product, questions that are not answered in the product documentation or on our Web site, or suggestions for new features or improvements.

By Email

- For technical support: support@roguewave.com
- For documentation issues: rwonlinedocs@roguewave.com

By Phone

800-856-3766 in the United States

(+1) 508-652-7700 worldwide

If you are reporting a problem, please include the following information:

- The version of the product and the platform on which you are running the product.
- An example that illustrates the problem.
- A record of the sequence of events that led to the problem.
This part of the reference guide describes the TotalView Command Line Interface (CLI).

Chapter 1, “CLI Command Summary,” on page 7  
Summarizes all CLI commands.

Chapter 2, “CLI Commands,” on page 19  
Describes all commands in the CLI’s unqualified (top-level) namespace. These are the commands that you use day-in and day-out, and those that are most often used interactively.

Chapter 3, “CLI Namespace Commands,” on page 179  
Describes commands found in the TV:: namespace. These commands are seldom used interactively, as they are most often used in scripts.

Chapter 4, “Batch Debugging Using tvscript,” on page 235  
Discusses how to create batch scripts that run TotalView unattended.

Chapter 5, “TotalView Variables,” on page 252  
Describes all TotalView variables, including those uses to set GUI behaviors. These variables reside in three namespaces: unqualified (top-level), TV:: and TV::GUI. For the most part, you set these variables to alter TotalView behaviors.

Chapter 6, “Creating Type Transformations,” on page 307  
Discusses how to customize data display using CLI routines. This is useful if you do not wish to see all the members of a class or structure or would like to alter the way TotalView displays these elements.
This chapter contains a summary of all TotalView debugger CLI commands. The commands are described in detail in Chapter 2, “CLI Commands,” on page 19 and Chapter 3, “CLI Namespace Commands,” on page 179.

**actionpoint**

Gets and sets action point properties

```
TV::actionpoint action [ object-id ] [ other-args ]
```

**alias**

Creates a new user-defined pseudonym for a command

```
alias alias-name defin-body
```

Views previously defined aliases

```
alias [ alias-name ]
```

**capture**

Returns a command's output as a string

```
capture [-out | -err | -both] [-f filename] command
```

**dactions**

Displays information about action points

```
dactions [ ap-id-list ] [ -at source-loc ]
  [ -enabled | -disabled ]
  [ -enabled_blocks | -disabled_blocks ]
  [ -block_images ]
  [ -block_lines ]
```

Saves action points to a file
dactions -save [filename]
Loads previously saved action points

dactions -load [filename]

dassign
Changes the value of a scalar variable
    dassign target value

dattach
Brings currently executing processes under TotalView control
    dattach [-g gid] [-r hname]

        [-ask_attach_parallel | -no_attach_parallel]
        [-replay | -no_replay]
        [-go | -halt] [-rank num]
        [-c { core-file | recording-file }]
        [-e] executable [pid-list]

        [-parallel_attach_subnet subset_specification]

dbarrier
Creates a barrier breakpoint at a source location
    dbarrier breakpoint-expr [-stop_when_hit { group | process | none }]

        [-stop_when_done { group | process | none }][ -pending]

Creates a barrier breakpoint at an address
    dbarrier -address addr [-stop_when_hit { group | process | none }]

        [-stop_when_done { group | process | none }][ -pending]

dbreak
Creates a breakpoint at a source location
    dbreak breakpoint-expr [-p | -g | -t] [[ -l lang] -e expr][ -pending]

Creates a breakpoint at an address
    dbreak -address addr [-p | -g | -t] [[ -l lang] -e expr][ -pending]

dcache
Clears the remote library cache

    dcache -flush

dcalltree
Displays parallel backtrace data
    [-data pbv_data_array] [-show_details] [-sort columns] [-hide_backtrace]

    [-save_as_csv filename] [-save_as_dot filename]
dcheckpoint
- Creates a checkpoint on IBM AIX
  dcheckpoint [-delete | -halt ]

dcont
- Continues execution and waits for execution to stop
  dcont

dcuda
- Manages NVIDIA® CUDA™ GPU threads, providing the ability to inspect them, change the focus, and display their status.
  dcuda

ddelete
- Deletes some action points
  ddelete action-point-list
- Deletes all action points
  ddelete -a

ddetach
- Detaches from the processes
  dddetach

ddisable
- Disables some action points
  ddisable action-point-list [-block number-list ]
- Disables all action points
  ddisable -a

ddlopen
- Loads a shared object library
  ddlopen [-now | -lazy ] [-local | -global ] [-mode int ] filespec
- Displays information about shared object libraries
  ddlopen [-list dll-ids...] 

ddown
- Moves down the call stack
  ddown [ num-levels ]
**dec2hex**

Converts a decimal number into hexadecimal

```
TV::dec2hex number
```

**denable**

Enables some action points

```
denable action-point-list
```

Enables all disabled action points in the current focus

```
denable -a
```

**dexamine**

Display memory contents

```
```

**dflush**

Removes the top-most suspended expression evaluation

```
dflush
```

Removes all suspended `dprint` computations

```
dflush -all
```

Removes `dprint` computations preceding and including a suspended evaluation ID

```
dflush susp-eval-id
```

**dfocus**

Changes the target of future CLI commands to this P/T set

```
dfocus p/t-set
```

Executes a command in this P/T set

```
dfocus [p/t-set command]
```

**dga**

Displays global array variables

```
dga [-lang lang_type] [handle_or_name] [slice]
```

**dgo**

Resumes execution of target processes

```
dgo
```

**dgroups**

Adds members to thread and process groups


dgroups -add [-g gid] [id-list]

Deletes groups

dgroups -delete [-g gid]

Intersects a group with a list of processes and threads

dgroups -intersect [-g gid] [id-list]

Prints process and thread group information

dgroups [-list] [pattern-list]

Creates a new thread or process group

dgroups -new [thread_or_process] [-g gid] [id-list]

Removes members from thread or process groups

dgroups -remove [-g gid] [id-list]

dhalt

Suspends execution of processes

dhalt

dheap

Shows Memory Debugger state

dheap [-status]

Applies a saved configuration file

dheap -apply_config {default | filename}

Shows information about a backtrace

dheap -backtrace [subcommands]

Compares memory states

dheap -compare subcommands [optional_subcommands]
  [process | filename [process | filename]]

Enables or disables the Memory Debugger

dheap {enable | disable}

Enables or disables event notification

dheap -event_filter subcommands

Writes memory information

dheap -export subcommands

Specifies which filters the Memory Debugger uses

dheap -filter subcommands

Writes guard blocks (memory before and after an allocation)

dheap -guard [subcommands]
Enables and disables the retaining (hoarding) of freed memory blocks

```dheap -hoard [ subcommands ]```

Displays Memory Debugger information

```dheap -info [ -backtrace ] [ start_address [ end_address ] ]```

Indicates whether an address is within a deallocated block

```dheap -is_dangling address```

Locates memory leaks

```dheap -leaks [ -check_interior ]```

Enables or disables Memory Debugger event notification

```dheap -[no]notify```

Paints memory with a distinct pattern

```dheap -paint [ subcommands ]```

Enables and disables the ability to catch bounds errors and use-after-free errors retaining freed memory blocks

```dheap -red_zones [ subcommands ]```

Enables and disables allocation and reallocation notification

```dheap -tag_alloc subcommand start_address [ end_address ]```

Displays the Memory Debugger's version number

```dheap -version```

**dhistory**

Displays information about the state of the program as it is being replayed. If you have received a timestamp, you can go back to the line that was executing at that time.

```dhistory [ -info ] [ -get_time ] [ -go_time time ] [ -go_live ] [ -enable ] [ -disable ]```

**dhold**

Holds processes

```dhold -process```

Holds threads

```dhold -thread```

**dkill**

Terminates execution of target processes

```dkill [ -remove ]```

**dlappend**

Appends list elements to a TotalView variable
**dlappend** *variable-name value* [ ... ]

**dlist**
Displays code relative to the current list location

```shell
dlist [-n num-lines]
```
Displays code relative to a named location

```shell
dlist breakpoint-expr [-n num-lines]
```
Displays code relative to the current execution location

```shell
dlist -e [-n num-lines]
```

**dll**
Manages shared libraries

```shell
TV::dll action [ dll-id-list ] [-all]
```

**dload**
Loads debugging information

```shell
dload [-g gid] [-mpi starter_value] [-r hname]
[-replay] [-noreplay]
[-env variable=value] ... [-e] executable
[-parallel_attach_subset subset_specification]
```

**dmstat**
Displays memory use information

```shell
dmstat
```

**dnext**
Steps source lines, stepping over subroutines

```shell
dnext [-back] [num-steps]
```

**dnexti**
Steps machine instructions, stepping over subroutines

```shell
dnexti [-back] [num-steps]
```

**dout**
Executes until just after the place that called the current routine

```shell
dout [-back] [frame-count]
```

**dprint**
Prints the value of a variable or expression

```shell
dprint [-nowait] [-slice slice_expr] [-stats [-data]] variable_or_expression
```
dptsets
Shows the status of processes and threads in an array of P/T expressions

\texttt{dptsets} [\textit{ptset\_array}] ...

drerun
Restarts processes

\texttt{drerun} [\textit{cmd\_arguments}] [ < \textit{infile}]
[ > [ > ]] [ & ] \textit{outfile}
[ 2> [ > ]] \textit{errfile}

drestart
Restarts a checkpoint on AIX

\texttt{drestart} [-halt] [-g \textit{gid}] [-r \textit{host}] [-no_same_hosts]

Restarts a checkpoint on SGI

\texttt{drestart} [process-state] [-no_unpark] [-g \textit{gid}] [-r \textit{host}]
[-ask_attach_parallel] [-no_attach_parallel]
[-no_preserve_ids] \textit{checkpoint\_name}

drun
Starts or restarts processes

\texttt{drun} [\textit{cmd\_arguments}] [ < \textit{infile}]
[ > [ > ]] [ & ] \textit{outfile}
[ 2> [ > ]] \textit{errfile}

dsession
Loads a session

\texttt{dsession} [-load \textit{session\_name}]

dset
Creates or changes a CLI state variable

\texttt{dset} \textit{debugger-var} \textit{value}

Views current CLI state variables

\texttt{dset} [\textit{debugger-var}]

Sets the default for a CLI state variable

\texttt{dset} -set\_as\_default \textit{debugger-var} \textit{value}

dstatus
Shows current status of processes and threads

\texttt{dstatus}
**dstep**
Steps lines, stepping into subfunctions
```
dstep [-back] [num-steps]
```

**dstepl**
Steps machine instructions, stepping into subfunctions
```
dstepl [-back] [num-steps]
```

**dunhold**
Releases a process
```
dunhold -process
```
Releases a thread
```
dunhold -thread
```

**dunset**
Restores a CLI variable to its default value
```
dunset debugger-var
```
Restores all CLI variables to their default values
```
dunset -all
```

**duntil**
Runs to a line
```
duntil [-back] line-number
```
Runs to an address
```
duntil [-back] -address addr
```
Runs into a function
```
duntil proc-name
```

**dup**
Moves up the call stack
```
dup [num-levels]
```

**dwait**
Blocks command input until the target processes stop
```
dwait
```

**dwatch**
Defines a watchpoint for a variable
```
dwatch variable [-length byte-count] [-p | -g | -t] [[-l lang] -e expr] [-t type]
```
Defines a watchpoint for an address

\texttt{dwatch -address addr-length byte-count [ -p | -g | -t ]}
\texttt{[ [ -l lang ] -e expr ] [ -t type ]}

\texttt{dwhat}

Determines what a name refers to

\texttt{dwhat symbol-name}

\texttt{dwhere}

Displays locations in the call stack

\texttt{dwhere [-level level-num] [ num-levels ] [ -args ] [ -locals ] [ -registers ]}
\texttt{[ -noshow_pc ] [ -noshow_fp ] [ -show_image ]}

Displays all locations in the call stack

\texttt{dwhere -all [ -args ] [ -locals ] [ -registers ]}
\texttt{[ -noshow_pc ] [ -noshow_fp ] [ -show_image ]}

\texttt{dworker}

Adds or removes a thread from a workers group

\texttt{dworker { number | boolean }}

\texttt{errorCodes}

Returns a list of all error code tags

\texttt{TV::errorCodes}

Returns or raises error information

\texttt{TV::errorCodes number_or_tag [ -raise [ message ] ]}

\texttt{exit}

Terminates the debugging session

\texttt{exit [ -force ]}

\texttt{expr}

Manipulates values created by \texttt{dprint -nowait}

\texttt{TV::expr action [ susp-eval-id ] [ other-args ]}

\texttt{focus_groups}

Returns a list of groups in the current focus

\texttt{TV::focus_groups}

\texttt{focus_processes}

Returns a list of processes in the current focus

\texttt{TV::focus_processes [ -all | -group | -process | -thread ]}
**focus_threads**
Returns a list of threads in the current focus

TV::focus_threads [ -all | -group | -process | -thread ]

**group**
Gets and sets group properties

TV::group action [ object-id ] [ other-args ]

**help**
Displays help information

help [ topic ]

**hex2dec**
Converts to decimal

TV::hex2dec number

**process**
Gets and sets process properties

TV::process action [ object-id ] [ other-args ]

**quit**
Terminates the debugging session

quit [ -force ]

**read_symbols**
Reads symbols from libraries

TV::read_symbols -lib lib-name-list

Reads symbols from libraries associated with a stack frame

TV::read_symbols -frame [ number ]

Reads symbols for all frames in the backtrace

TV::read_symbols -stack

**respond**
Provides responses to commands

TV::respond response command

**scope**
Gets and sets internal scope properties

TV::scope action [ object-id ] [ other-args ]
source_process_startup

“Sources” a .tvd file when a process is loaded

TV::source_process_startup process_id

spurs

Manages threads using commands modeled after the GDB SPU Runtime System (SPU) library.

spurs add [ directory directory-list ... ]

stty

Sets terminal properties

stty [ stty-args ]

symbol

Returns or sets internal TotalView symbol information

TV::symbol action [ object-id ] [ other-args ]

thread

Gets and sets thread properties

TV::thread action [ object-id ] [ other-args ]

type

Gets and sets type properties

TV::type action [ object-id ] [ other-args ]

type_transformation

Creates type transformations and examines properties

TV::type_transformation action [ object-id ] [ other-args ]

unalias

Removes an alias

unalias alias-name

Removes all aliases

unalias -all
Command Overview

This chapter lists all of CLI commands with a brief description.

General CLI Commands

These commands provide information on the general CLI operating environment:

- **alias**: Creates or views pseudonyms for commands and arguments.
- **capture**: Sends output to a variable for commands that print information.
- **dlappend**: Appends list elements to a TotalView variable.
- **dset**: Changes or views values of TotalView variables.
- **dunset**: Restores default settings of TotalView variables.
- **help**: Displays help information.
- **stty**: Sets terminal properties.
- **unalias**: Removes a previously defined alias.

CLI Initialization and Termination Commands

These commands initialize and terminate the CLI session, and add processes to CLI control:
• **dattach**: Brings one or more processes currently executing in the normal runtime environment (that is, outside TotalView) under TotalView control.

• **ddetach**: Detaches TotalView from a process.

• **ddlopen**: Dynamically loads shared object libraries.

• **dgroups**: Manipulates and manages groups.

• **dkill**: Kills existing user processes, leaving debugging information in place.

• **dload**: Loads debugging information about the program into TotalView and prepares it for execution.

• **drerun**: Restarts a process.

• **drun**: Starts or restarts the execution of user processes under control of the CLI.

• **dsession**: Loads a session into TotalView.

• **exit, quit**: Exits from TotalView, ending the debugging session.

**Program Information Commands**

The following commands provide information about a program's current execution location, and support browsing the program's source files:

• **dcalltree**: Displays parallel backtrace data.

• **ddown**: Navigates through the call stack by manipulating the current frame.

• **dexamine**: Displays memory contents.

• **dflush**: Unwinds the stack from computations.

• **dga**: Displays global array variables.

• **dlist**: Browses source code relative to a particular file, procedure, or line.

• **dmstat**: Displays memory usage information.

• **dprint**: Evaluates an expression or program variable and displays the resulting value.

• **dptsets**: Shows the status of processes and threads in a P/T set.

• **dstatus**: Shows the status of processes and threads.

• **dup**: Navigates through the call stack by manipulating the current frame.

• **dwhat**: Determines what a name refers to.

• **dwhere**: Prints information about the thread's stack.
Execution Control Commands

The following commands control execution:

- **dcont**: Continues execution of processes and waits for them.
- **dfocus**: Changes the set of processes, threads, or groups upon which a CLI command acts.
- **dgo**: Resumes execution of processes (without blocking).
- **dhalt**: Suspends execution of processes.
- **dhistory**: Provides information for ReplayEngine and supports working with timestamps.
- **dhold**: Holds threads or processes.
- **dnext**: Executes statements, stepping over subfunctions.
- **dnexti**: Executes machine instructions, stepping over subfunctions.
- **dout**: Runs out of current procedure.
- **dstep**: Executes statements, moving into subfunctions if required.
- **dsteipi**: Executes machine instructions, moving into subfunctions if required.
- **dunhold**: Releases held threads.
- **duntil**: Executes statements until a statement is reached.
- **dwait**: Blocks command input until processes stop.
- **dworker**: Adds or removes threads from a workers group.

Action Points

The following action point commands define and manipulate the points at which the flow of program execution should stop so that you can examine debugger or program state:

- **dactions**: Views information on action point definitions and their current status; this command also saves and restores action points.
- **dbarrier**: Defines a process barrier breakpoint.
- **dbreak**: Defines a breakpoint.
- **ddelete**: Deletes an action point.
- **ddisable**: Temporarily disables an action point.
- **denable**: Re-enables an action point that has been disabled.
• **dwatch**: Defines a watchpoint.

### Platform-Specific CLI Commands

- **dcuda**: Manages NVIDIA® CUDA™ GPU threads, providing the ability to inspect them, change the focus, and display their status.
- **spurs**: Manages threads using commands modeled after the GDB SPU Runtime System (SPU) library.

### Other Commands

The commands in this category do not fit into any of the other categories:

- **dassign**: Changes the value of a scalar variable.
- **dcache**: Clears the remote library cache.
- **dcheckpoint**: Creates a file that can later be used to restart a program.
- **dheap**: Displays information about the heap.
- **drestart**: Restarts a checkpoint.
alias

Creates or views pseudonyms for commands

**Format**
Creates a new user-defined pseudonym for a command

```
alias alias-name defn-body
```

Views previously defined aliases

```
alias [ alias-name ]
```

**Arguments**

- `alias-name`
  - The name of the command pseudonym being defined.

- `defn-body`
  - The text that Tcl substitutes when it encounters `alias-name`.

**Description**

The `alias` command associates a specified name with some defined text. This text can contain one or more commands. You can use an alias in the same way as a native TotalView or Tcl command. In addition, you can include an alias as part of the definition of another alias.

If you do not enter an `alias-name` argument, the CLI displays the names and definitions of all aliases. If you specify only an `alias-name` argument, the CLI displays the definition of the alias.

Because the `alias` command can contain Tcl commands, `defn-body` must comply with all Tcl expansion, substitution, and quoting rules.

The TotalView global startup file, `tvdinit.tvd`, defines a set of default one or two-letter aliases for all common commands. To see a list of these commands, type `alias` with no argument in the CLI -window.

You cannot use an alias to redefine the name of a CLI-defined command. You can, however, redefine a built-in CLI command by creating your own Tcl procedure. For example, the following procedure disables the built-in `dwatch` command. When a user types `dwatch`, the CLI executes this code instead of the built-in CLI code.

```
proc dwatch {} {
    puts "The dwatch command is disabled"
}
```

**NOTE >>** Be aware that you can potentially create aliases that are nonsensical or incorrect because the CLI does not parse `defn-body` (the command's definition) until it is used. The CLI detects errors only when it tries to execute your alias.

When you obtain help for any command, the help text includes any TotalView predefined aliases.

To delete an alias, use the `unalias` command.
Examples

alias nt dnext
  Defines a command called nt that executes the dnext -command.
alias nt
  Displays the definition of the nt alias.
alias
  Displays the definitions of all aliases.
alias m {dlist main}
  Defines an alias called m that lists the source code of function main().
alias step2 {dstep; dstep}
  Defines an alias called step2 that does two dstep commands. This new command applies to the focus that exists when this alias is used.
alias step2 {s ; s}
  Creates an alias that performs the same operations as that in the previous example, differing in that it uses the alias for dstep. You could also create the following alias which does the same thing: alias step2 {s 2}.
alias step1 {f p1. dstep}
  Defines an alias called step1 that steps the first user thread in process 1. All other threads in the process run freely while TotalView steps the current line in your program.

RELATED TOPICS

- Initializing TotalView in the TotalView for HPC User Guide
- unalias Command
capture

Returns a command’s output as a string

Format

```
capture [-out | -err | -both] [-f filename] command
```

Arguments

- **-out**
  Captures only output sent to `stdout`.

- **-err**
  Captures only output sent to `stderr`.

- **-both**
  Captures output sent to both `stdout` and `stderr`. This is the default.

- **-f filename**
  Sends the captured output to `filename`. The file must be a writable Tcl file descriptor.

- **command**
  The CLI command (or commands) whose output is being captured. If you specify more than one command, you must enclose them within braces `{}`.

Description

The `capture` command executes `command`, capturing in a string all output that would normally go to the console. After `command` completes, it returns the string. This command is analogous to the UNIX shell’s back-tick feature (`command`). The `capture` command obtains the printed output of any CLI command so that you can assign it to a variable or otherwise manipulate it.

Examples

```
set save_stat [ capture st ]
```

Saves the current process status to a Tcl variable.

```
set arg [ capture p argc]
```

Saves the printed value of argc into a Tcl variable.

```
set vbl [ capture {foreach i {1 2 3 4} \ {p int2_array($i )}} ]
```

Saves the printed output of four array elements into a Tcl variable. Here is sample output:

```
int2_array(1) = -8 (0xffff8)
int2_array(2) = -6 (0xfffa)
int2_array(3) = -4 (0xfffc)
int2_array(4) = -2 (0xfffe)
```

Because the `capture` command records all information sent to it by the commands in the `foreach` loop, you do not have to use a `dlist` command.
exec cat << [ capture help commands ] > cli_help.txt

  Writes the help text for all CLI commands to the cli_help.txt file.
set ofile [open cli_help.txt w]
capture -f $ofile help commands
close $ofile

  Also writes the help text for all CLI commands to the cli_help.txt file. This set of commands is more efficient than the previous command because the captured data is not -buffered.

RELATED TOPICS

  drun Command
drerun Command
dactions

Displays information, and saves and reloads action points

Format

Displays information about action points.

```
dactions [ ap-id-list ] [ -at source-loc ] [ -enabled | -disabled ] [ -enabled_blocks | -disabled_blocks ]
[ -block_images | -block_lines ]
```

Saves action points to a file.

```
dactions -save [ filename ]
```

Loads previously saved action points.

```
dactions -load [ filename ]
```

Suppresses or unsuppresses action points.

```
dactions [ -suppress | -unsuppress ]
```

Arguments

`ap-id-list`

A list of action point identifiers. If you specify individual action points, the information that appears is limited to these points.

Do not enclose this list within quotes or braces. See the examples at the end of this section for more information.

Without this argument, the CLI displays summary information about all action points in the processes in the focus set. If you enter one ID, the CLI displays full information for it. If you enter more than one ID, the CLI displays just summary information for each.

`-at source-loc`

Displays the action points at `source-loc`. See `dbreak` for the details on the form of `source-loc`.

`-enabled`

Shows only enabled action points.

`-disabled`

Shows only disabled action points.

`-supress`

Effectively disables all existing action points. If the code is run, threads will not stop at any action points. Although you can create new action points (and delete existing ones), the new action points too will be effectively disabled.

`-unsuppress`

Restores all action points to the state they were in when suppressed. Any new action points added are set as enabled.
-enabled_blocks
  When displaying the full information for an action point, only shows the enabled address blocks. (See example below.)

-disabled_blocks
  When displaying the full information for an action point, only shows the disabled address blocks. (See example below.)

-block_images
  When displaying the full information for an action point, shows the image name of each address block.

-block_lines
  When displaying the full information for an action point, shows the source line of each address block.

-full
  Forces the display of full information.

-save
  Writes information about action points to a file.

-load
  Restores action point information previously saved in a file.

filename
  The name of the file into which TotalView reads and writes action point information. If you omit this file name, TotalView writes action point information to a file named program_name:TVD.v3breakpoints, where program_name is the name of your program.

Description
The dactions command displays information about action points in the processes in the current focus. If you do not indicate a focus, the default focus is at the process level. The information is printed; it is not returned.

Using the Action Point Identifier
To get the action point identifier, just enter dactions with no arguments. You need this identifier to delete, enable, and disable action points.

The identifier is returned when TotalView creates the action point. The CLI prints this ID when the thread stops at an action point.

You can include action point identifiers as arguments to the command when more detailed information is needed. The -enabled and -disabled options restrict output to action points in one of these states.

You cannot use the dactions command when you are debugging a core file or before TotalView loads executables.
**Saving and Loading Action Points**

The `-save` option writes action point information to a file so that either you or TotalView can restore your action points later. The `-load` option immediately reads the saved file. Using the `filename` argument with either option writes to or reads from this file. If you do not use this argument, TotalView names the file `program_name.TVD.v3breakpoints` (where `program_name` is the name of your program), and writes it to the directory in which your program resides.

The information saved includes expressions associated with the action point and whether the action point is enabled or disabled. For example, if your program's name is `foo`, TotalView writes this information to `foo.TVD.v3breakpoints`.

---

**NOTE >>** TotalView does not save information about watchpoints.

If a file with the default name exists, TotalView can read this information when it starts your program. When TotalView exits, it can create the default. For more information, see the File > Preference Action Points Page information in the TotalView for HPC online Help.

**Suppressing and Unsuppressing Action Points**

Suppress effectively disables all existing action points. If the code is run, threads will not stop at any action points. Although you can create new action points (and delete existing ones), the new action points too will be effectively disabled. Unsuppress restores all action points to the state they were in when suppressed. Any new action points added are set as enabled.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac</td>
<td>dactions</td>
<td>Displays all action points</td>
</tr>
</tbody>
</table>

**Examples**

`ac -at 81`

Displays information about the action points on line 81. (This example uses the alias instead of the full command name.) Here is the output from this command:

```
ac -at 81
1 shared action point for group 3:
  1 addr=0x10001544 [arrays.F#81] Enabled
    Share in group: true
    Stop when hit: group
```

`dactions 1 3`

Displays information about action points 1 and 3, as follows:

```
2 shared action points for process 1:
```
If you have saved a list of action points as a string or as a Tcl list, you can use the eval command to process the list's elements.

For example:

d1.> dactions
2 shared action points for group 3:
   3 [global_pointer_ref.cxx#52] Enabled
   4 [global_pointer_ref.cxx#53] Enabled

d1.> set group1 "3 4"
3 4

d1.> eval ddisable $group1
d1.> ac
2 shared action points for group 3:
   3 [global_pointer_ref.cxx#52] Disabled
   4 [global_pointer_ref.cxx#53] Disabled

dfocus p1 dactions
Displays information about all action points defined in process 1.
dfocus p1 dactions -enabled
Displays information about all enabled action points in process 1
dactions n [-enabled_blocks|-disabled_blocks]
This extended example demonstrates the use of these two options.

Set a break point:

d1.> b {bar<std::vector<int, std::allocator<int>> >::bar(int)}
Incorporating 10079 bytes of DWARF '.debug_info' information for tx_test2.cxx (linenumber)...done
1

Entering dactions reports on only the top-level action point associated with this action point number:

d1.> dactions
1 shared action point for group 3:
   1 [bar<std::vector<int, std::allocator<int>> >::bar(int)] Enabled

Entering dactions n reports on all action point instances (the address block) associated with this action point number:

d1.> dactions 1
1 shared action point for group 3:
   1 [bar<std::vector<int, std::allocator<int>> >::bar(int)] Enabled
       Address 0: [Enabled] bar<std::vector<int, std::allocator<int>> >::bar+0x12 (0x004013d2)
       Address 1: [Enabled] bar<std::vector<int, std::allocator<int>> >::bar+0x84 (0x00401444)
Address 2: [Disabled] bar<std::vector<double,std::allocator<double> > ::bar+0x12 (0x00401496)
Address 3: [Disabled] bar<std::vector<double,std::allocator<double> > ::bar+0x86 (0x0040150a)
  Share in group: true
  Stop when hit: process

Using `-enabled_blocks` reports on only enabled action point instances (the address block) associated with this action point number:

d1.<> dactions 1 -enabled_blocks
1 shared action point for group 3:
  1 [bar<std::vector<int,std::allocator<int> > ::bar(int) Enabled
     Address 0: [Enabled] bar<std::vector<int,std::allocator<int> > ::bar+0x12
     (0x004013d2)
     Address 1: [Enabled] bar<std::vector<int,std::allocator<int> > ::bar+0x84
     (0x00401444)
       Share in group: true
       Stop when hit: process

Using `-disabled_blocks` reports on only disabled action point instances (the address block) associated with this action point number:

d1.<> dactions 1 -disabled_blocks
1 shared action point for group 3:
  1 [bar<std::vector<int,std::allocator<int> > ::bar(int) Enabled
     Address 2: [Disabled] bar<std::vector<double,std::allocator<double> > ::bar+0x12
     (0x00401496)
     Address 3: [Disabled] bar<std::vector<double,std::allocator<double> > ::bar+0x86
     (0x0040150a)
       Share in group: true
       Stop when hit: process

d1.<>

You could use this information, for example, to enable the currently disabled action point addresses:

d1.<> denable -block 2 3

**RELATED TOPICS**

- **Setting Action Points** in the *TotalView for HPC User Guide*
- **Saving Actions Points to a File** in the *TotalView for HPC User Guide*
- **Action Point > Enable** in the TotalView online Help
- **Action Point > Disable** in the TotalView online Help
- **Action Point > Load All** in the TotalView online Help
- **Action Point > Save All** in the TotalView online Help
- **Action Point > Save As** in the TotalView online Help
- **TV::auto_save_breakpoints Variable**
dassign

Changes the value of a scalar variable

**Format**

dassign target value

**Arguments**

target

The name of a scalar variable in your program.

value

A source-language expression that evaluates to a scalar value. This expression can use the name of another variable.

**Description**

The dassign command evaluates an expression and replaces the value of a variable with the evaluated result. The location can be a scalar variable, a dereferenced pointer variable, or an element in an array or structure.

The default focus for the dassign command is thread. If you do not change the focus, this command acts upon the thread of interest (TOI). If the current focus specifies a width that is wider than t (thread) and is not d (default), dassign iterates over the threads in the focus set and performs the assignment in each. In addition, if you use a list with the dfocus command, the dassign command iterates over each list member.

The CLI interprets each symbol name in the expression according to the current context. Because the value of a source variable might not have the same value across threads and processes, the value assigned can differ in your threads and processes. If the data type of the resulting value is incompatible with that of the target location, you must cast the value into the target's type. (Casting is described in the TotalView for HPC User Guide.)

**Assigning Characters and Strings**

- If you are assigning a character to a target, place the character value within single-quotation marks; for example, 'c'.

- You can use the standard C language escape character sequences; for example, \n and \t. These escape sequences can also be in a character or string assignment.

- If you are assigning a string to a target, place the string within quotation marks. However, you must escape the quotation marks so they are not interpreted by Tcl; for example, "The quick brown fox".

If value contains an expression, the TotalView expression system evaluates the expression. See “Using the Evaluate Window” in the TotalView for HPC User Guide for more information.
Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>dassign</td>
<td>Changes a scalar variable's value</td>
</tr>
</tbody>
</table>

Examples

dassign scalar_y 102

Stores the value 102 in each occurrence of variable `scalar_y` for all processes and threads in the current set.

dassign i 10*10

Stores the value 100 in variable `i`.

dassign i i*i

Does not work and the CLI displays an error message. If `i` is a simple scalar variable, you can use the following statements:

```
set x [lindex [capture dprint i] 2]
dassign i [expr $x * $x]
```

f {p1 p2 p3} as scalar_y 102

Stores the value 102 in each occurrence of variable `scalar_y` contained in processes 1, 2, and 3.

RELATED TOPICS

- Changing the Value of Variables in the TotalView for HPC User Guide
- Changing a Variable's Data Type in the TotalView for HPC User Guide
**dattach**

Brings currently executing processes under TotalView control

**Format**

```
dattach [-g gid] [-r hname]
          [-ask_attach_parallel | -no_attach_parallel]
          [-replay | -no_replay]
          [-go | -halt] [-rank num]
          [-e executable [ pid-list ]
          [-c core-file | recording-file]
          [-parallel_attach_subset subset-specification]
```

**Arguments**

- **-g gid**
  
  Sets the control group for the processes being added to group *gid*. This group must already exist. (The CLI GROUPS variable contains a list of all groups. See “GROUPS” on page 256 for more information.)

- **-r hname**
  
  The host on which the process is running. The CLI launches a TotalView Server on the host machine if one is not already running. See the Setting Up Parallel Debugging Sessions chapter of the TotalView for HPC User Guide for information on the launch command used to start this server.

  Setting a host sets it for all PIDs attached to in this command. If you do not name a host machine, the CLI uses the local host.

- **-ask_attach_parallel**
  
  Specifies that TotalView should ask before attaching to parallel processes of a parallel job. The default is to automatically attach to processes. For additional information, see the Parallel Page in the File > Preferences Dialog Box in the in-product help.

- **-no_attach_parallel**
  
  Does not attach to any additional parallel processes in a parallel job. For additional information, see the Parallel Page in the File > Preferences Dialog Box in the in-product help.

- **-replay | -no_replay**
  
  Enables or disables the ReplayEngine the next time the program is restarted.

- **-go | -halt**
  
  Specifies to explicitly continue or halt target execution after attaching. The default is to leave the target’s run state as it was before the attach.

- **-rank num**
  
  Specifies the rank associated with the executable being loaded. While this can be used independently, this option is best used with core files.

- **-e**
  
  Tells the CLI that the next argument is an executable file name. You need to use -e if the executable name begins with a dash (-) or consists of only numeric characters. Otherwise, you can just provide the executable file name.
**executable**

The name of the executable. Setting an executable here sets it for all PIDs being attached to in this command. If you do not include this argument, the CLI tries to determine the executable file from the process. Some architectures do not allow this to occur.

**pid-list**

A list of system-level process identifiers (such as a UNIX PID) naming the processes that TotalView controls. All PIDs must reside on the same system, and they are placed in the same control group. If you need to place the processes in different groups or attach to processes on more than one system, you must use multiple `dattach` commands.

**-c core-file | recording-file**

Loads the core file `core-file` or the ReplayEngine `recording-file`, which restores a previous ReplayEngine debugging session. If you use this option, you must also specify an executable name (`executable`).

**-parallel_attach_subset subset_specification**

Defines a list of MPI ranks to attach to when an MPI job is created or attached to. The list is space-separated; each element can have one of three forms:

- **rank**: specifies that rank only
- **rank1-rank2**: specifies all ranks between rank1 and rank2, inclusive
- **rank1-rank2:stride**: specifies every strideth rank between rank1 and rank2

A rank must be either a positive decimal integer or max (the last rank in the MPI job). A **subset_specification** that is the empty string (""') is equivalent to 0-max.

For example:

```
dattach -parallel_attach_subset {1 2 4-6 7-max:2} mpirun
```

attaches to ranks 1, 2, 4, 5, 6, 7, 9, 11, 13,....

**Description**

The `dattach` command attaches to one or more processes, making it possible to continue process execution under TotalView control.

**NOTE >> TotalView Individual: You can attach only to processes running on the computer upon which you installed TotalView Individual.**

This command returns the TotalView process ID (DPID) as a string. If you specify more than one process in a command, the `dattach` command returns a list of DPIDs instead of a single value.

TotalView places all processes to which it attaches in one `dattach` command in the same control group. This lets you place all processes in a multiprocess program executing on the same system in the same control group.

If a program has more than one executable, you must use a separate `dattach` command for each one.

If you have not loaded `executable` already, the CLI searches for it. The search includes all directories in the `-EXECUTABLE_PATH` CLI variable.
The process identifiers specified in the *pid-list* must refer to existing processes in the runtime environment. TotalView attaches to the processes, regardless of their execution states.

### Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>at</td>
<td>dattach</td>
<td>Brings the process under TotalView control</td>
</tr>
</tbody>
</table>

### Examples

**dattach mysys 10020**

Loads debugging information for *mysys* and brings the process known to the run-time system as PID 10020 under TotalView control.

**dattach -e 123 10020**

Loads file 123 and brings the process known to the run-time system by PID 10020 under TotalView control.

**dattach -g 4 -r Enterprise myfile 10020**

Loads *myfile* that is executing on the host named *Enterprise* into group 4, and brings the process known to the run-time system by PID 10020 under TotalView control. If a TotalView Server (*tvdsvr*) is not running on *Enterprise*, the CLI will start it.

**dattach my_file 51172 52006**

Loads debugging information for *my_file* and brings the processes corresponding to PIDs 51172 and 52006 under TotalView control.

**set new_pid [dattach -e mainprog 123]**

**dattach -r otherhost -g $CGROUP($new_pid) -e slave 456**

Begins by attaching to *mainprog* running on the local host; then attaches to *slave* running on the *otherhost* host and inserts them both in the same control group.

### RELATED TOPICS

- Using the Root Window in the *TotalView for HPC User Guide*
- Attaching to Processes in the *TotalView for HPC User Guide*
- Examining Core Files in the *TotalView for HPC User Guide*
- File > New Program Command in the online Help
- *ddetach* Command
- TV::parallel_attach Variable
**dbarrier**

Defines a process or thread barrier breakpoint

**Format**

Creates a barrier breakpoint at a source location

```
 dbarrier breakpoint-expr [ -stop_when_hit width ] [ -stop_when_done width ] [ -pending ]
```

Creates a barrier breakpoint at an address

```
 dbarrier -address addr [ -stop_when_hit width ] [ -stop_when_done width ] [ -pending ]
```

**Arguments**

**breakpoint-expr**

This argument can be entered in more than one way, usually using a line number or a pathname containing a file name, function name, and line number, each separated by # characters (for example, <file#line>). For more information, see Chapter 9, “Qualifying Symbol Names” in the TotalView for HPC User Guide. For more information on breakpoint expressions, see dbreak on page 42, particularly Breakpoint Expressions.

**-address addr**

The barrier breakpoint location as an absolute address in the address space of the program.

**-stop_when_hit width**

Identifies, using the width argument, any additional processes or threads to stop when stopping the thread that arrives at a barrier point.

If you do not use this option, the value of BARRIER_STOP_ALL indicates what to stop.

The argument width may have one of the following three values:

- **group**
  
  Stops all processes in the control group when the execution reaches the barrier point.

- **process**
  
  Stops the process that hit the barrier.

- **none**
  
  Stops only the thread that hit the barrier; that is, the thread is held and all other threads continue running. If you apply this width to a process barrier breakpoint, TotalView stops the process that hit the breakpoint.

**-stop_when_done width**

After all processes or threads reach the barrier, releases all processes and threads held at the barrier. (Released means that these threads and processes can run.) Setting this option stops additional threads contained in the same group or process.

If you do not use this option, the value of BARRIER_STOP_WHEN_DONE indicates any other processes or threads to stop.
Use the \texttt{width} argument indicates other stopped processes or threads. You can enter one of the following three values:

\textbf{group}

Stops the entire control group when the barrier is satisfied.

\textbf{process}

Stops the processes that contain threads in the satisfaction set when the barrier is satisfied.

\textbf{none}

Stops the satisfaction set. For process barriers, \texttt{process} and \texttt{none} have the same effect. This is the default if the \texttt{BARRIER_STOP_WHEN_DONE} variable is \texttt{none}.

\texttt{--pending}

If TotalView cannot find a location to set the barrier, adding this option creates the barrier anyway. As shared libraries are read, TotalView checks to see if it can be set in the newly loaded library. For more information on this option, see \texttt{dbreak} on page 42.

\textbf{Description}

The \texttt{dbarrier} command sets a process or thread barrier breakpoint that triggers when execution arrives at a location. This command returns the ID of the newly created breakpoint.

The \texttt{dbarrier} command is most often used to synchronize a set of threads. The P/T set defines which threads the barrier affects. When a thread reaches a barrier, it stops, just as it does for a breakpoint. The difference is that TotalView prevents—that is, holds—each thread that reaches the barrier from responding to resume commands (for example, \texttt{dstep}, \texttt{dnext}, and \texttt{dgo}) until all threads in the affected set arrive at the barrier. When all threads reach the barrier, TotalView considers the barrier to be satisfied and releases these threads. Note that they are just released, not continued. That is, TotalView leaves them stopped at the barrier. If you continue the process, those threads stopped at the barrier also run along with any other threads that were not participating with the barrier. After the threads are released, they can respond to resume commands.

If the process is stopped and then continued, the held threads, including the ones waiting on an unsatisfied barrier, do not run. Only unheld threads run.

The satisfaction set for the barrier is determined by the current focus. If the focus group is a thread group, TotalView creates a thread barrier:

- When a thread hits a process barrier, TotalView holds the thread's process.
- When a thread hits a thread barrier, TotalView holds the thread; TotalView might also stop the thread's process or control group. While they are stopped, neither is held.

TotalView determines the default focus width based on the setting of the \texttt{SHARE_ACTION_POINT} variable. If it is set to true, the default is group. Otherwise, it is process.

TotalView determines the processes and threads that are part of the satisfaction set by taking the intersection of the share group with the focus set. (Barriers cannot extend beyond a share group.)
The CLI displays an error message if you use an inconsistent focus list.

**NOTE >>** Barriers can create deadlocks. For example, if two threads participate in two different barriers, each could be left waiting at different barriers that can never be satisfied. A deadlock can also occur if a barrier is set in a procedure that is never invoked by a thread in the affected set. If a deadlock occurs, use the **ddelete** command to remove the barrier, since deleting the barrier also releases any threads held at the barrier.

The **-stop_when_hit** option specifies if other threads should stop when a thread arrives at a barrier.

The **-stop_when_done** option controls the set of additional threads that are stopped when the barrier is finally satisfied. That is, you can also stop an additional collection of threads after the last expected thread arrives, and all the threads held at the barrier are released. Normally, you want to stop the threads contained in the control group.

If you omit a **stop** option, TotalView sets the default behavior by using the BARRIER_STOP_ALL and BARRIER_STOP_WHEN_DONE variables. For more information, see the **dset** command.

Use the **none** argument for these options to not stop additional threads.

- If **-stop_when_hit** is **none** when a thread hits a thread barrier, TotalView stops only that thread; it does not stop other threads.
- If **-stop_when_done** is **none**, TotalView does not stop additional threads, aside from the ones that are already stopped at the barrier.

TotalView places the barrier point in the processes or groups specified in the current focus, as follows:

- If the current focus does not indicate an explicit group, the CLI creates a process barrier across the share group.
- If the current focus indicates a process group, the CLI creates a process barrier that is satisfied when all members of that group reach the barrier.
- If the current focus indicates a thread group, TotalView creates a thread barrier that is satisfied when all members of the group arrive at the barrier.

The following example illustrates these differences. If you set a barrier with the focus set to a control group (the default), TotalView creates a process barrier. This means that the **-stop_when_hit** value is set to **process** even though you specified **thread**.

```plaintext
d1.<> dbarrier 580 -stop_when_hit thread
2
d1.<> ac 2
1 shared action point for group 3:
   2 addr=0x120005598  [../regress/fork_loop.cxx#580] Enabled (barrier)
   Share in group: true
```

---

**RogueWave**

CLI Commands / dbarrier 39
Stop when hit: process
Stop when done: process
process barrier; satisfaction set = group 1

However, if you create the barrier with a specific workers focus, the stop when hit property remains set to thread:

1.<> baw 580 -stop_when_hit thread
1
d1.<> ac 1

1 unshared action point for process 1:
1 addr=0x120005598 [../regress/fork_loop.cxx#580]
   Enabled (barrier)
Share in group: false
Stop when hit: thread
Stop when done: process
thread barrier; satisfaction set = group 2

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ba</td>
<td>dbARRIER</td>
<td>Defines a barrier.</td>
</tr>
<tr>
<td>baw</td>
<td>{dfocus pW dbARRIER -stop_when_done process}</td>
<td>Creates a thread barrier across the worker threads in the process of interest (POI). TotalView sets the set of threads stopped when the barrier is satisfied to the process that contains the satisfaction set.</td>
</tr>
<tr>
<td>BAW</td>
<td>{dfocus gW dbARRIER -stop_when_done group}</td>
<td>Creates a thread barrier across the worker threads in the share group of interest. The set of threads stopped when the barrier is satisfied is the entire control group.</td>
</tr>
</tbody>
</table>

Examples

dbARRIER 123

Stops each process in the control group when it arrives at line 123. After all processes arrive, the barrier is satisfied, and TotalView releases all processes.

dfocus {p1 p2 p3} dbARRIER my Proc

Holds each thread in processes 1, 2, and 3 as it arrives at the first executable line in procedure my_proc. After all threads arrive, the barrier is satisfied and TotalView releases all processes.

dfocus gW dbARRIER 642 -stop_when_hit none

Sets a thread barrier at line 642 in the workers group. The process is continued automatically as each thread arrives at the barrier. That is, threads that are not at this line continue running.
RELATED TOPICS

Setting Breakpoints and Barriers in the TotalView for HPC User Guide
Barrier Points in the TotalView for HPC User Guide
Using Groups, Processes, and Threads in the TotalView for HPC User Guide
Creating a Satisfaction Set in the TotalView for HPC User Guide
Holding and Releasing Processes and Threads in the TotalView for HPC User Guide
Action Point > Set Barrier Command in the online Help
dactions Command
dbreak Command
denable Command
ddisable Command
**dbreak**

Defines a breakpoint

**Format**

Creates a breakpoint at a source location

```
- dbreak breakpoint-expr [-p | -g | -t] [[ -l lang ] -e expr] [-pending ]
```

Creates a breakpoint at an address

```
- dbreak -address addr [-p | -g | -t] [[ -l lang ] -e expr] [-pending ]
```

**Arguments**

- **breakpoint-expr**
  
  This argument can be entered in more than one way, usually using a line number or a pathname containing a file name, function name, and line number, each separated by # characters (for example, `file#line`). For more information, see Chapter 9, “Qualifying Symbol Names” in the TotalView for HPC User Guide.

  Breakpoint expressions are discussed later in this section.

- **-address addr**
  
  The breakpoint location specified as an absolute address in the address space of the program.

- **-p**
  
  Stops the process that hit this breakpoint. You can set this option as the default by setting the `STOP_ALL` variable to `process`. See `dset` on page 136 for more information.

- **-g**
  
  Stops all processes in the process's control group when execution reaches the breakpoint. You can set this option as the default by setting the `STOP_ALL` variable to `group`. See `dset` on page 136 for more information.

- **-t**
  
  Stops the thread that hit this breakpoint. You can set this option as the default by setting the `STOP_ALL` variable to `thread`. See `dset` on page 136 for more information.

- **-l lang**
  
  Sets the programming language used when you are entering expression `expr`. Enter either: `c`, `c++`, `f7`, `f9`, or `asm` (for C, C++, FORTRAN 77, Fortran 9x, and assembler, respectively). If you do not specify a language, TotalView assumes the language in which the routine at the breakpoint was written.

- **-e expr**
  
  When the breakpoint is hit, TotalView evaluates expression `expr` in the context of the thread that hit the breakpoint. The language statements and operators you can use are described in Chapter 8, “Setting Action Points” in the TotalView for HPC User Guide.

- **-pending**
  
  If TotalView cannot find a location to set the breakpoint, adding this option creates the breakpoint anyway. As shared libraries are read, TotalView checks to see if it can be set in the newly loaded library.
**Description**

The `dbreak` command defines a breakpoint or evaluation point triggered when execution arrives at the specified location, stopping each thread that arrives at a breakpoint. This command returns the ID of the new breakpoint. If a line does not contain an executable statement, the CLI cannot set a breakpoint.

If you try to set a breakpoint at a line at which TotalView cannot stop execution, it sets one at the nearest following line where it can halt execution.

Specifying a procedure name without a line number sets an action point at the beginning of the procedure. If you do not name a file, the default is the file associated with the current source location.

**The -pending Option**

If, after evaluating the breakpoint expression, TotalView determines the location represented by the expression does not exist, it can still set a breakpoint if you use the `--pending` option. This option checks shared libraries that are subsequently loaded to see if a breakpoint can be set. If a location is found, it is set. Stated in a different way, TotalView normally creates and sets a breakpoint at the same time. The option tells it to separate these two actions.

When displaying information on a breakpoint's status, the CLI displays the location where execution actually stops.

**A stop group Breakpoint**

If the CLI encounters a `stop group` breakpoint, it suspends each process in the group as well as the process that contains the triggering thread. The CLI then shows the identifier of the triggering thread, the breakpoint location, and the action point identifier.

**Default Focus Width**

TotalView determines the default focus width based on the setting of the `SHARE_ACTION_POINT` variable. If set to `true`, the default is group. Otherwise, it is process.

**Breakpoint Expressions**

Breakpoint expressions, also called breakpoint specifications, are used in both breakpoints and barrier points, so this discussion is relevant to both.

One possibly confusing aspect of using expressions is that their syntax differs from that of Tcl. This is because you need to embed code written in Fortran, C, or assembler in Tcl commands. In addition, your expressions often include TotalView built-in functions. For example, if you want to use the TotalView `$tid` built-in function, you need to type it as \$tid.

A breakpoint expression can evaluate to more than one source line. If the expression evaluates to a function that has multiple overloaded implementations, TotalView sets a breakpoint on each of the overloaded functions.
Set a breakpoint at the line specified by `breakpoint-expr` or the absolute address `addr`. You can enter a breakpoint expression that are sets of addresses at which the breakpoint is placed, and are as follows:

- `[[##image##]filename##]line_number`
  Indicates all addresses at this line number.

- A function signature; this can be a partial signature.
  Indicates all addresses that are the addresses of functions matching `signature`. If parts of a function signature are missing, this expression can match more than one signature. For example, “`f`” matches “`f(void)`” and “`A::f(int)`”. You cannot specify a return type in a signature.

- `class class_name`
  Specifies that the breakpoint should be planted in all member functions of class `class_name`.

- `virtual class::signature`
  Specifies that the breakpoint should be planted in all virtual member functions that match `signature` and are in the class or derived from the class.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>break</td>
<td>Sets a breakpoint</td>
</tr>
<tr>
<td>bt</td>
<td><code>{dbreak t}</code></td>
<td>Sets a breakpoint only on the thread of interest</td>
</tr>
</tbody>
</table>

**Examples**

For all examples, assume that the current process set is `d2.<` when the breakpoint is defined.

`dbreak 12`

Suspends process 2 when it reaches line 12. However, if the STOP_ALL variable is set to `group`, all other processes in the group are stopped. In addition, if SHARE_ACTION_POINT is `true`, the breakpoint is placed in every process in the group.

`dbreak -address 0x1000764`

Suspends process 2 when execution reaches address 0x1000764.

`b 12 -g`

Suspends all processes in the current control group when execution reaches line 12.

`dbreak 57 -l f9 -e {goto $63}`

Causes the thread that reaches the breakpoint to transfer to line 63. The host language for this statement is Fortran 90 or Fortran 95.

`dfocus p3 b 57 -e {goto $63}`

In process 3, sets the same evaluation point as the previous example.
RELATED TOPICS

- Action Point > Properties Command in the online Help
- Action Point > At Location Command in the online Help
- Setting Breakpoints and Barriers in the TotalView User Guide
- Defining Evaluation Points and Conditional Breakpoints in the TotalView User Guide
- Using Groups, Processes, and Threads in the TotalView for HPC User Guide
- dactions Command
- dbreak Command
- denable Command
- ddisable Command
dcache

Clears the remote library cache

Format
dcache -flush

Arguments
-flush

Deletes all files from the library cache that are not currently being used.

Description
The dcache -flush command removes the library files that it places in your cache, located in the .TotalView/lib_cache subdirectory in your home directory.

When you are debugging programs on remote systems that use libraries that either do not exist on the host or whose version differ, TotalView copies the library files into your cache. This cache can become large.

TotalView automatically deletes cached library files that it hasn't used in the last week. If you need to reclaim additional space at any time, use this command to remove files not currently being used.

RELATED TOPICS

Initializing TotalView in the TotalView for HPC User Guide
**dcalltree**

Displays parallel backtrace data

**Format**


**Arguments**

- **-data pbv_data_array**
  Captures the data from calling `dcalltree` in an associative Tcl array rather than writing the data to the console.

- **-show_details**
  Displays the data with all processes and threads displayed.

- **-hide_backtrace**
  Displays the data with only root and leaf nodes displayed.

- **-sort column**
  Sorts the data display based on the data in a particular column. The possible arguments are Processes, Location, PC, Host, Rank, ID, and Status.

- **-save_as_csv filename**
  Saves the backtrace data as a file of comma-separated values under the name `filename`.

- **-save_as_dot filename**
  Saves the backtrace data as a dot file under the name `filename`. Dot is a plain text graph description language.

**Description**

The TotalView GUI has a Parallel Backtrace View window that displays the state of every process and thread in a parallel job. The `dcalltree` command makes this same data available either in the console window, or, with the `-data` switch, as a Tcl associative array.

The associative array has the following format:

```
{
  {
    Key    <value>
    Level  <value>
    Processes <value>
    Location <value>
    PC      <value>
    Host    <value>
    Rank    <value>
    ID      <value>
    Status  <value>
  }
  {
    ...   
  }
}
```
The `-show_details` and `-hide_backtrace` switches pull in opposite directions. The `-show_details` switch shows the maximum data, including all processes and threads. The `-hide_backtrace` command hides any intermediate nodes, displaying only the root and leaf nodes. If used together, this results in a display of root and leaf nodes and all threads. This reduction can help to de-clutter the data display if the number of processes and threads is large.

**Examples**

**dfocus group dcalltree**

This example first changes the focus to the group using `dfocus`, then calls `dcalltree` with no switches. Note that the ID column is a compressed `ptlist` describing process and thread count, range, and IDs. See Compressed List Syntax (`ptlist`) for more information.

<table>
<thead>
<tr>
<th>Processes</th>
<th>Location</th>
<th>PC</th>
<th>Host</th>
<th>Rank ID</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>/</td>
<td>...</td>
<td>&lt;local&gt;</td>
<td>-1 4:12[p1-4.1-3]</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>_start</td>
<td>0x004011b9</td>
<td>&lt;local&gt;</td>
<td>-1 4:4[p1-4.1]</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>__libc_start_main</td>
<td>0x2b3425358184</td>
<td>&lt;local&gt;</td>
<td>-1 4:4[p1-4.1]</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>main</td>
<td>0x004035bf</td>
<td>&lt;local&gt;</td>
<td>-1 4:4[p1-4.1]</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>fork_wrapper</td>
<td>0x00402790</td>
<td>&lt;local&gt;</td>
<td>-1 4:4[p1-4.1]</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>fork</td>
<td>0x0040274b</td>
<td>&lt;local&gt;</td>
<td>-1 4:4[p1-4.1]</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>snore</td>
<td>0x00401c11</td>
<td>&lt;local&gt;</td>
<td>-1 4:4[p1-4.1]</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>snore#681</td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1 2.1 - 47502964801120</td>
<td>Stopped</td>
</tr>
<tr>
<td>1</td>
<td>snore#705</td>
<td>0x00401c9b</td>
<td>&lt;local&gt;</td>
<td>-1 4.1 - 47502964801120</td>
<td>Stopped</td>
</tr>
<tr>
<td>2</td>
<td>wait_a_while</td>
<td>0x00401a09</td>
<td>&lt;local&gt;</td>
<td>-1 2:2[p1.1, p3.1]</td>
<td>Stopped</td>
</tr>
<tr>
<td>8</td>
<td>start_thread</td>
<td>0x2b3424db1143</td>
<td>&lt;local&gt;</td>
<td>-1 4:12[p1-4.1-3]</td>
<td>...</td>
</tr>
<tr>
<td>8</td>
<td>snore_or_leave</td>
<td>0x004021cb</td>
<td>&lt;local&gt;</td>
<td>-1 4:8[p1-4.2-3]</td>
<td>...</td>
</tr>
<tr>
<td>8</td>
<td>snore</td>
<td>...</td>
<td>&lt;local&gt;</td>
<td>-1 4:8[p1-4.2-3]</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>snore#681</td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1 1.2 - 1082132800</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>1</td>
<td>snore#681</td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1 1.3 - 1090525504</td>
<td>Stopped</td>
</tr>
<tr>
<td>1</td>
<td>snore#705</td>
<td>0x00401c9b</td>
<td>&lt;local&gt;</td>
<td>-1 2.2 - 1082132800</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>1</td>
<td>snore#681</td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1 2.3 - 1090525504</td>
<td>Stopped</td>
</tr>
<tr>
<td>1</td>
<td>snore#681</td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1 4.2 - 1082132800</td>
<td>Stopped</td>
</tr>
<tr>
<td>1</td>
<td>snore#681</td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1 4.3 - 1090525504</td>
<td>Stopped</td>
</tr>
<tr>
<td>2</td>
<td>wait_a_while</td>
<td>...</td>
<td>&lt;local&gt;</td>
<td>-1 1:2[p3.2-3]</td>
<td>...</td>
</tr>
</tbody>
</table>

**dcalltree -show_details**

By adding the `-show_details`, switch, you get more complete output:

<table>
<thead>
<tr>
<th>Processes</th>
<th>Location</th>
<th>PC</th>
<th>Host</th>
<th>Rank ID</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>/</td>
<td>...</td>
<td>&lt;local&gt;</td>
<td>-1 4:12[p1-4.1-3]</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>_start</td>
<td>0x004011b9</td>
<td>&lt;local&gt;</td>
<td>-1 4:4[p1-4.1]</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>__libc_start_main</td>
<td>0x2b3425358184</td>
<td>&lt;local&gt;</td>
<td>-1 4:4[p1-4.1]</td>
<td>...</td>
</tr>
<tr>
<td>Processes</td>
<td>Location</td>
<td>PC</td>
<td>Host</td>
<td>Rank ID</td>
<td>Status</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>----------------</td>
<td>---------</td>
<td>---------</td>
<td>--------------------</td>
</tr>
<tr>
<td>main</td>
<td></td>
<td>0x004035bf</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>4:4[p1-4.1]</td>
</tr>
<tr>
<td>fork_wrapper</td>
<td></td>
<td>0x00402790</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>4:4[p1-4.1]</td>
</tr>
<tr>
<td>forker</td>
<td></td>
<td>0x0040274b</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>4:4[p1-4.1]</td>
</tr>
<tr>
<td>snore</td>
<td></td>
<td>0x00401c11</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>4:4[p1-4.1]</td>
</tr>
<tr>
<td>snore#681</td>
<td></td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>2.1 - 47502964801120 Stopped</td>
</tr>
<tr>
<td>snore#705</td>
<td></td>
<td>0x00401c9b</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>4.1 - 47502964801120 Breakpoint</td>
</tr>
<tr>
<td>wait_a_while</td>
<td></td>
<td>0x00401a09</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>2:2[p1.1, p3.1]   Stopped</td>
</tr>
<tr>
<td>__select_nocancel</td>
<td>0x2b34253f56e2</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>2:2[p1.1, p3.1]</td>
<td>Stopped</td>
</tr>
<tr>
<td>__select_nocancel</td>
<td>0x2b34253f56e2</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>1.1 - 47502964801120</td>
<td>Stopped</td>
</tr>
<tr>
<td>__select_nocancel</td>
<td>0x2b34253f56e2</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>3.1 - 47502964801120</td>
<td>Stopped</td>
</tr>
<tr>
<td>start_thread</td>
<td></td>
<td>0x02b3424db1143</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>4:12[p1-4.1-3]</td>
</tr>
<tr>
<td>snore#681</td>
<td></td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>1.2 - 1082132800  Breakpoint</td>
</tr>
<tr>
<td>snore#681</td>
<td></td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>1.3 - 1090525504  Stopped</td>
</tr>
<tr>
<td>snore#705</td>
<td></td>
<td>0x00401c9b</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>2.2 - 1082132800  Breakpoint</td>
</tr>
<tr>
<td>snore#681</td>
<td></td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>2.3 - 1090525504  Stopped</td>
</tr>
<tr>
<td>snore#681</td>
<td></td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>4.2 - 1082132800  Stopped</td>
</tr>
<tr>
<td>snore#681</td>
<td></td>
<td>0x00401c05</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>4.3 - 1090525504  Stopped</td>
</tr>
<tr>
<td>wait_a_while</td>
<td></td>
<td>0x00401a09</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>1:2[p3.2-3]</td>
</tr>
<tr>
<td>__select_nocancel</td>
<td>0x2b34253f56e2</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>3.3 - 1090525504</td>
<td>Stopped</td>
</tr>
<tr>
<td>wait_a_while#580</td>
<td></td>
<td>0x004019e9</td>
<td>&lt;local&gt;</td>
<td>-1</td>
<td>3.2 - 1082132800</td>
</tr>
</tbody>
</table>

Adding the `-hide_backtrace` switch reduces the clutter somewhat:

```
dcalltree -show_details -hide_backtrace
```

Here is code to get the location of all threads that are at a breakpoint:

```
dcalltree -data pbv_data_array -show_details
foreach { data_record } [array get pbv_data_array] {
```
set print_location 0
set break_location
foreach {title value} $data_record {
    if {$title == "Location"} {
        set break_location $value
    }
    if {$value == "Breakpoint"} {
        set print_location 1
    }
    if {1 == $print_location} {
        puts stdout "Breakpoint found at $break_location"
        set print_location 0
    }
}

RELATED TOPICS

Parallel Backtrace View in the TotalView User Guide
dcheckpoint

Creates a checkpoint image of processes (IBM RS6000 only)

Format

Creates a checkpoint on IBM RS6000 machines.

```
dcheckpoint [ -by process_set ] [ -delete | -halt ]
```

Arguments

- **-by process_set**
  
  This option can take two possible values:
  
  - **pe**
    
    Checkpoint the Parallel Environment job. This value is the default.
  
  - **pid**
    
    Checkpoint the focus process.
  
- **-delete**
  
  Processes exit after the checkpoint occurs.

- **-halt**
  
  Processes halt after the checkpoint occurs.

Description

The `dcheckpoint` command saves program and process information to a file. This information includes process and group IDs. Later, use the `drestart` command to restart the program.

**NOTE >>** This command does not save TotalView breakpoint information. To save breakpoints, use the `dactions` command.

By default, TotalView checkpoints the Parallel Environment job. To checkpoint a particular process, make that process the focus and use the `pid` argument to `-by`. If the focus is a group that contains more than one process, the CLI displays an error message.

By default, the checkpointed processes stop, allowing you to investigate a program's state at the checkpointed position. You can modify this behavior with the `-delete` and `-halt` options.

When you request a checkpoint:

- TotalView temporarily stops (that is, *parks*) the processes that are being checkpointed. Parking ensures that the processes do not run freely after a `dcheckpoint` or `drestart` operation. (If they did, your code would begin running before you could control it.)

- The CLI detaches from processes before they are checkpointed. After checkpointing, the CLI automatically reattaches to them.
Examples

dcheckpoint
  Checkpoints the Parallel Environment job. All associated processes stop.
f3 dcheckpoint -by pid
dcheckpoint -by pe -halt
  Checkpoints the Parallel Environment job. All associated processes halt.

RELATED TOPICS

Tools > Create Checkpoint Command in the online Help
Tools > Restart Checkpoint Command in the online Help
drestart Command
dcont

Continues execution and waits for execution to stop

Format

dcont

Arguments

This command has no arguments

Description

The `dcont` command continues all processes and threads in the current focus, and then waits for all of them to stop.

This command is a Tcl macro, with the following definition:

```
proc dcont {args} {uplevel dgo; "dwait $args" }
```

You often want this behavior in scripts. You seldom want to do interactively.

---

**NOTE >>** You can interrupt this action using Ctrl+C to stop process execution.

---

A `dcont` command completes when all threads in the focus set of processes stop executing. If you do not indicate a focus, the default focus is the process of interest (POI).

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>co</td>
<td>dcont</td>
<td>Resume</td>
</tr>
<tr>
<td>CO</td>
<td>{dfocus g dcont}</td>
<td>Resume at group-level</td>
</tr>
</tbody>
</table>

Examples

**dcont**

Resumes execution of all stopped threads that are not held and which belong to processes in the current focus. (This command does not affect threads that are held at barriers.) The command blocks further input until all threads in all target processes stop. After the CLI displays its prompt, you can enter additional commands.

**dfocus p1 dcont**

Resumes execution of all stopped threads that are not held and that belong to process 1. The CLI does not accept additional commands until the process stops.

**dfocus {p1 p2 p3} co**

Resumes execution of all stopped threads that are not held and that belong to processes 1, 2, and 3.
co

Resumes execution of all stopped threads that are not held and that belong to the current group.

RELATED TOPICS

Starting Processes and Threads in the TotalView for HPC User Guide
dgo Command
dwait Command
**dcuda**

**Manages GPU threads**

**Format**

- `dcuda block [(Bx,By,Bz)]`
- `dcuda thread [(Tx,Ty,Tz)]`
- `dcuda kernel`
- `dcuda device [<>n>]
- `dcuda sm [<>n>]
- `dcuda warp [<>n>]
- `dcuda info-system`
- `dcuda info-device`
- `dcuda info-sm`
- `dcuda info-warp`
- `dcuda info-lane`
- `dcuda focus (Bx,By,Bz),(Tx,Ty,Tz)
- `dcuda hwfocus <D/S/W/L>

**Arguments**

Bx, By, Bz
- The x, y and z block indices

Tx, Ty, Tz
- The x,y, and z thread indices

D/S/W/L
- The coordinates defining the physical space of the hardware:
  - D: device number
  - S: streaming multiprocessor (SM)
  - W: warp (WP) number on the SM
  - L: lane (LN) number on the warp

**Description**

The dcuda commands allow you to manage and view GPU threads, in either the logical coordinate space of block and thread indices (<<<(Bx,By,Bz),(Tx,Ty,Tz)>>>) or the physical coordinate space that defines the hardware (the device number, the streaming multiprocessor number on the device, the warp number on the SM, and lane number on the warp).

**dcuda block [(Bx,By,Bz)]**

- With no arguments, shows the current CUDA block
- With a block argument of the form (Bx,By,Bz), changes the CUDA focus to that block. Omitted parameters (i.e., Bz) are unchanged.
dcuda thread [(Tx,Ty,Tz)]
- With no arguments, shows the current CUDA thread.
- With a thread argument of the form (Tx,Ty,Tz), changes the CUDA focus to that thread. Omitted parameters (i.e., Ty and Tz, or just Tz) are unchanged.

dcuda kernel
Displays the logical and hardware coordinates of the current CUDA context.

dcuda device [<n>]
- With no arguments, shows the current CUDA device.
- With a numeric argument, changes the CUDA device focus to that device.

dcuda sm [<n>]
- With no arguments, shows the current CUDA SM (streaming multiprocessor).
- With a numeric argument, changes the CUDA SM focus to that SM.

dcuda warp [<n>]
- With no arguments, shows the current CUDA warp.
- With a numeric argument, changes the CUDA warp focus to that warp.

dcuda lane [<n>]
- With no arguments, shows the current CUDA lane.
- With a numeric argument, changes the CUDA lane focus to that lane.

dcuda info-system
Displays the CUDA devices in the system.

dcuda info-device
Displays currently running SMs in the current device.

dcuda info-sm
Displays valid warps in the current SM.

dcuda info-warp
Displays valid lanes in the current warp.

dcuda info-lane
Displays the current lane.
**dcuda focus** (Bx, By, Bz), (Tx, Ty, Tz)

Changes the focus via CUDA logical coordinates of the form <<<(Bx, By, Bz), (Tx, Ty, Tz)>>. The following abbreviations are also accepted:

<<<Tx>>>  <<<(Tx)>>>  <<<(Tx, Ty)>>>  <<<(Tx, Ty, Tz)>>>  <<<(Bx), (Tx)>>>  <<<(Bx), (Tx, Ty)>>>  <<<(Bx), (Tx, Ty, Tz)>>>  <<<(Bx, By), (Tx)>>>  <<<(Bx, By), (Tx, Ty)>>>  <<<(Bx, By), (Tx, Ty, Tz)>>>  <<<(Bx, By, Bz), (Tx)>>>  <<<(Bx, By, Bz), (Tx, Ty)>>>  <<<(Bx, By, Bz), (Tx, Ty, Tz)>>>  <<<(Bx, By, Bz), (Tx, Ty, Tz)>>>  

Angle brackets are optional, but must be balanced.

**dcuda hwfocus** <D/S/W/L>

Changes the focus via CUDA hardware coordinates of the form D/S/W/L, S/W/L, W/L, or L.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuda</td>
<td>dcuda</td>
<td>Writes out the focus thread, as in dcuda kernel.</td>
</tr>
</tbody>
</table>

**Examples**

**Displaying device information**

dcuda info-device

**Output:**

DEV: 0/1  Device Type: gt200  SM Type: sm_13  SM/WP/LN: 30/32/32  Regs/LN: 128
SM: 0/30  valid warps: 0x0000000000000001


dcuda info-sm

**Output:**

DEV: 0/1  Device Type: gt200  SM Type: sm_13  SM/ WP/ LN: 30/32/32  Regs/LN: 128
SM: 0/30  valid warps: 0x0000000000000001
WP: 0/32  valid/active/divergent lanes: 0x00000000f/0x00000000f/0x00000000 block: (0, 0, 0)


dcuda info-warp

**Output:**
DEV: 0/1   Device Type: gt200   SM Type: sm_13   SM/WP/LN: 30/32/32   Regs/LN: 128
SM:  0/30  valid warps: 0x0000000000000001
WP:  0/32  valid/active/divergent lanes: 0x0000000f/0x0000000f/0x00000000  block: (0,0,0)
LN:  0/32  pc=0x000000001ef2efa8  thread: (0,0,0)
LN:  1/32  pc=0x000000001ef2efa8  thread: (1,0,0)
LN:  2/32  pc=0x000000001ef2efa8  thread: (0,1,0)
LN:  3/32  pc=0x000000001ef2efa8  thread: (1,1,0)

dcuda info-lane

Output:
DEV: 0/1   Device Type: gt200   SM Type: sm_13   SM/WP/LN: 30/32/32   Regs/LN: 128
SM:  0/30  valid warps: 0x0000000000000001
WP:  0/32  valid/active/divergent lanes: 0x0000000f/0x0000000f/0x00000000  block: (0,0,0)

Displaying the focus
dcuda warp sm

Output:
sm 0 warp 0
dcuda lane device

Output:
device 0 lane 3
dcuda thread

Output:
thread (1,1,0)
dcuda kernel

Output:
device 0, sm 0, warp 0, lane 3, block (0,0,0), thread (1,1,0)

Changing the focus

In these commands, note that TotalView assigns CUDA threads a negative thread ID. In the examples here, the CUDA thread is labeled “1.-1”.

dcuda thread (1,1,0)

Changes the CUDA focus to the thread represented by logical coordinates 1,1,0.
New CUDA focus (1.-1): device 0, sm 0, warp 0, lane 3, (1,1,0)

dcuda lane 2
Changes the CUDA focus to lane 2.

New CUDA focus (1.-1): device 0, sm 0, warp 0, lane 2, (0,1,0)

dcuda lane 1 sm 0
Changes the CUDA focus to lane 1 and to SM 0.

New CUDA focus (1.-1): device 0, sm 0, warp 0, lane 1, (1,0,0)

dcuda thread 0,0,0
Changes the CUDA focus to thread 0,0,0.

New CUDA focus (1.-1): device 0, sm 0, warp 0, lane 0, (0,0,0)

dcuda thread 1
Changes the CUDA focus to thread 1,0,0.

New CUDA focus (1.-1): device 0, sm 0, warp 0, lane 1, (1,0,0)

RELATED TOPICS

Using the CUDA Debugger in the TotalView for HPC User Guide
**ddelete**

Deletes action points

**Format**

Deletes the specified action points

```
ddelete action-point-list
```

Deletes all action points

```
ddelete -a
```

**Arguments**

*action-point-list*

A list of the action points to delete.

*-a*

Deletes all action points in the current focus.

**Description**

The ddelete command permanently removes one or more action points. If you delete a barrier point, the CLI releases the processes and threads held at it.

If you do not indicate a focus, the default focus is the process of interest (POI).

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>de</td>
<td>ddelete</td>
<td>Deletes action points</td>
</tr>
</tbody>
</table>

**Examples**

```
ddelete 1 2 3
```

Deletes action points 1, 2, and 3.

```
ddelete -a
```

Deletes all action points associated with processes in the current focus.

```
dfocus {p1 p2 p3 p4} ddelete -a
```

Deletes all the breakpoints associated with processes 1 through 4. Breakpoints associated with other threads are not affected.

```
dfocus a de -a
```

Deletes all action points known to the CLI.
RELATED TOPICS

Action Point > Delete All Command in the online Help
**ddetach**

**Detaches from processes**

**Format**

```
ddetach
```

**Arguments**

This command has no arguments.

**Description**

The `ddetach` command detaches the CLI from all processes in the current focus. This undoes the effects of attaching the CLI to a running process; that is, the CLI releases all control over the process, eliminates all debugger state information related to it (including action points), and allows the process to continue executing in the normal run-time environment.

You can detach any process controlled by the CLI; the process being detached need not have been loaded with a `dattach` command.

After this command executes, you are no longer able to access program variables, source location, action point settings, or other information related to the detached process.

If a single thread serves as the set, the CLI detaches the process that contains the thread. If you do not indicate a focus, the default focus is the process of interest (POI).

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>det</td>
<td>ddetach</td>
<td>Detaches from processes</td>
</tr>
</tbody>
</table>

**Examples**

```
ddetach
```

Detaches the process or processes that are in the current focus.

```
dfocus {p4 p5 p6} det
```

Detaches processes 4, 5, and 6.

```
dfocus g2 det
```

Detaches all processes in the control group associated with process 2.
RELATED TOPICS

Detaching from Processes in the TotalView for HPC User Guide

dattach Command
ddisable  

Temporarily disables action points

**Format**

Disables the specified action points

    ddisable action-point-list [-block number-list ]

Disables all action points

    ddisable -a

**Arguments**

*action-point-list*

A list of the action points to disable.

*block number-list*

If you set a breakpoint on a line that is ambiguous, use this option to identify the instances to disable. Obtain a list of these numbers using the `dactions` command.

*-a*

Disables all action points.

**Description**

The `ddisable` command temporarily deactivates action points. To delete an action point, use `ddelete`.

You can explicitly name the IDs of the action points to disable or you can disable all action points.

If you do not indicate a focus, the default focus is the process of interest (POI).

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>di</td>
<td>ddisable</td>
<td>Temporarily disables action points</td>
</tr>
</tbody>
</table>

**Examples**

    ddisable 3 7

    Disables the action points with IDs 3 and 7.
    
    di -a

    Disables all action points in the current focus.
    
    dfocus {p1 p2 p3 p4} ddisable -a

    Disables all action points associated with processes 1 through 4. Action points associated with other processes are not affected.
\texttt{ddi 1 -block 3 4}

Disables the action points associated with blocks 3 and 4. That is, one logical action point can map to more than one actual action point if you set the action point at an ambiguous location.

\texttt{ddisable 1 2 -block 3 4}

Disables the action points associated with blocks 3 and 4 in action points 1 and 2.

**RELATED TOPICS**

- \texttt{Action Point > Disable Command} in the online Help
**ddlopen**

Dynamically loads shared object libraries

**Format**

Dynamically loads a shared object library

```
  ddlopen [ -now | -lazy ] [ -local | -global ] [ -mode int ] filespec
```

Displays information about shared object libraries

```
  ddlopen [ -list { dll-ids... } ]
```

**Arguments**

- **-now**
  
  Includes `RTLD_NOW` in the `dlopen` command's mode argument. (Now immediately resolves all undefined symbols.)

- **-lazy**

  Includes `RTLD_LAZY` in the `dlopen` command's mode argument. (Lazy tries to resolve unresolved symbols as code is executed, rather than now.)

- **-local**

  Includes `RTLD_GLOBAL` in the `dlopen` command's mode argument. (Local makes library symbols unavailable to libraries that the program subsequently loads.) This argument is the default.

- **-global**

  Includes `RTLD_LOCAL` in the `dlopen` command's mode argument. (Global makes library symbols available to libraries that the program subsequently loads.)

- **-mode int**

  The integer arguments are ORed into the other mode flags passed to the `dlopen()` function. (See your operating system's documentation for information on these flags.)

- **filespec**

  The shared library to load.

- **-list**

  Displays information about the listed DLL IDs. If you omit this option or use the `-list` without a DLL ID list, TotalView displays information about all DLL IDs.

- **dll-ids**

  A list of one or more DLL IDs.

**Description**

The **ddlopen** command dynamically loads shared object libraries, or lists the shared object libraries loaded using this or the **Tools > Debugger Loaded Libraries** command.

For a **filespec** argument, TotalView performs a `dlopen` operation on this file in each process in the current P/T set. On the IBM AIX operating system, you can add a parenthesized library module name to the end of the **filespec** argument.
NOTE >> dlopen(3), dlerror(3), and other related routines are not part of the default runtime libraries on AIX, Solaris, and Red Hat Linux. Instead, they are in the libdl system library. Consequently, you must link your program using the -ldl option if you want to use the ddlopen command.

The -now and -lazy options indicate whether dlopen immediately resolves unresolved symbol references or defers resolving them until the target program references them. If you don't use either option, TotalView uses your operating system's default. (Not all platforms support both alternatives. For example, AIX treats RTLD_LAZY the same as RTLD_NOW).

The -local and -global options determine if symbols from the newly loaded library are available to resolve references. If you don't use either option, TotalView uses the target operating system's default. (Linux supports only the -global option; if you don't specify an option, the default is the -local option.)

After entering this command, the CLI waits until all dlopen calls complete across the current focus. The CLI then returns a unique dll-id and displays its prompt, which means that you can enter additional CLI commands. However, if an event occurs (for example, a $stop, a breakpoint in user function called by static object constructors, a SEGV, and so on), the ddlopen command throws an exception that describes the event. The first exception sub-code in the errorCode variable is the DLL ID for the suspended dlopen() function call.

If an error occurs while executing the dlopen() function, TotalView calls the dlerror() function in the target process, and then prints the returned string.

A DLL ID describes a shareable object that was dynamically loaded by the ddlopen command. Use the TV:dl command to obtain information about and delete these objects. If all dlopen() calls return immediately, the ddlopen command returns a unique DLL ID that you can also use with the TV::dl command.

Every DLL ID is also a valid breakpoint ID, representing the expressions used to load and unload DLLs; you can manipulate these breakpoints using the TV::expr command.

If you do not use a filespec argument or if you use the -list option without using a DLL ID argument, TotalView prints information about objects loaded using ddlopen. If you do use a DLL ID argument, TotalView prints information about DLLs loaded into all processes in the focus set; otherwise, TotalView prints information about just those DLLs. The ddlopen command prints its output directly to the console.

The ddlopen command calls the dlopen() function and it can change the string returned by the dlerror() function. It can also change the values returned to the application by any subsequent dlerror() call.

Examples

```
   ddlopen "mpistat.so"
```

Loads mpistat.so library file. The returned argument lists the process into which TotalView loaded the library.

```
   dfocus g ddlopen "mpistat.so(mpistat.o)"
```

Loads the module mpistat.o in the AIX DLL library mpistat.so into all members of the current process's control group.
ddlopen -lazy -global "mpistat.so"

Loads mpistat.so into process 1, and does not resolve outstanding application symbol requests to point to mpistat. However, TotalView uses the symbols in this library if it needs them.

ddlopen

Prints the list of shared objects dynamically loaded by the ddlopen command.

RELATED TOPICS

- Preloading Shared Libraries in the TotalView for HPC User Guide
- Tools > Debugger Loaded Libraries Command in the online Help
- TV::dll Command
ddown

Moves down the call stack

**Format**

```
$ ddown [ num-levels ]
```

**Arguments**

`num-levels`

Number of levels to move down. The default is 1.

**Description**

The `ddown` command moves the selected stack frame down one or more levels and prints the new frame’s number and function name.

Call stack movements are all relative, so using the `ddown` command effectively moves down in the call stack. (If up is in the direction of the `main()` function, then down is back to where you were before you moved through stack frames.)

Frame 0 is the most recent—that is, the currently executing—frame in the call stack, frame 1 corresponds to the procedure that invoked the currently executing frame, and so on. The call stack’s depth is increased by one each time a procedure is entered, and decreased by one when it is exited.

The command affects each thread in the focus. That is, if the current width is process, the `ddown` command acts on each thread in the process. You can specify any collection of processes and threads as the target set.

In addition, the `ddown` command modifies the current list location to be the current execution location for the new frame; this means that a `dlist` command displays the code that surrounds this new location.

The context and scope changes made by this command remain in effect until the CLI executes a command that modifies the current execution location (for example, the `dstep` command), or until you enter either a `dup` or `ddown` command.

If you tell the CLI to move down more levels than exist, the CLI simply moves down to the lowest level in the stack, which was the place where you began moving through the stack frames.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td><code>ddown</code></td>
<td>Moves down the call stack</td>
</tr>
</tbody>
</table>

**Examples**

```
$ ddown
```

Moves down one level in the call stack. As a result, for example, `dlist` commands that follow refers to the procedure that invoked this one. The following example shows what prints after you enter this command:
check_fortran_arrays_ PC=0x10001254,
FP=0x7fff2ed0 [arrays.F#48]

\texttt{d \ 5}

Moves the current frame down five levels in the call stack.

**RELATED TOPICS**

- \texttt{dup Command}
denable

Enables action points

Format
Enables some action points

\texttt{denable action-point-list \ [-\textbf{block} number-list]}

Enables all disabled action points in the current focus

\texttt{denable -a}

Arguments
\textit{action-point-list}

The identifiers of the action points being enabled.

\textbf{-a}

Enables all action points.

\textbf{-block number-list}

If you set a breakpoint on a line that is ambiguous, this option names which instances to enable. Use the \texttt{dactions} command to obtain a list of these numbers.

Description
The \texttt{denable} command reactivates action points that you previously disabled with the \texttt{ddisable} command. The \textbf{-a} option enables all action points in the current focus.

If you did not save the ID values of disabled action points, use \texttt{dactions} to obtain a list of this information.

If you do not indicate a focus, the default focus is the process of interest (POI).

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>en</td>
<td>\texttt{denable}</td>
<td>Enables action points</td>
</tr>
</tbody>
</table>

Examples

\texttt{denable 3 4}

Enables two previously identified action points.

\texttt{dfocus \{p1 p2\} denable -a}

Enables all action points associated with processes 1 and 2. This command does not affect settings associated with other processes.

\texttt{en -a}

Enables all action points associated with the current focus.
fa en -a

Enables all actions points in all processes.

en 1 -block 3 4

Enables the action points associated with blocks 3 and 4. That is, one logical action point can map to more than one actual action point if you set the action point at an ambiguous location.

denable 1 2 -block 3 4

Enables the action points associated with blocks 3 and 4 in action points 1 and 2.

RELATED TOPICS

Enabling Action Points in the TotalView for HPC User Guide
Action Points > Enable Command in the online Help
ddisable Command
dbarrier Command
dbreak Command
dwatch Command
dexamine

Displays memory contents

Format

dexamine [ -column_count cnt ] [ -count cnt ] [ -data_only ] [ -show_chars ] [ -string_length len ] [ -format fmt ]
[ -memory_info ] [ -wordsize size ] variable_or_expression

Arguments

-cols cnt
-Column_count cnt

Specifies the number of columns to display. Without this option, the CLI determines this number of columns
based on the data's wordactid size and format.

-c cnt
-count cnt

Specifies the number of elements to examine. Without this option, the CLI displays the entire object. This num-
ber is determined by the object's datatype. If no type is available, the default value for cnt is 1 element.

-d
-data_only

Does not display memory values with a prefixed address: field or address annotations. This option is incompatible
with -memory_info.

-f fmt
-format fmt

Specifies the format to use when displaying memory. The default format is hex. You can abbreviate each of
these to the first character in the format's name.

address

Interprets memory as addresses; the word size is always the size of a pointer

binary

Binary; this can also be abbreviated to \( t \)

char

Unsigned character

dec

Signed decimal value of size 1, 2, 4, or 8 bytes

float

Signed float value, either 4 or 8 byte word size

hex

Unsigned hexadecimal value of size 1, 2, 4, or 8 bytes

instruction

Sequence of instructions
oct
  Unsigned octal value of size 1, 2, 4, or 8 bytes

string
  String

-m
-memory_info
  Shows information about the type of memory associated with the address. Without this option, the CLI does not
display this information. This argument is incompatible with -data_only. When you use this option, the CLI an-
notates address each line in the dump as follows:

  [d]: .data
  [t]: .text
  [p]: .plt
  [b]: .bss
  [?]: Another type of memory (such as stack address)

If you have enabled memory debugging, the following annotations can also appear:

  [A]: Allocated block of memory
  [D]: Deallocated block of memory
  [G]: Address is a guard region
  [C]: Address is a corrupted guard region

If the address being examined is within an allocated block, this option tells the Memory Debugger to automati-
cally include the pre-guard region if the user specified guards in the memory debugging configuration.

-sc
-show_chars
  Shows a trailing character dump for each line. Without this option, the CLI does not show the trailing characters.

-sl len
-string_length len
  Specifies the maximum size string to display. Without this option, the length is all characters up to the first null
character.

-w size
-wordsize size
  Specifies the “word size” to apply to the format. The default word size is ‘1’ for most formats. For ‘address’ format,
the word size is always the size of a target pointer. The values can be 1, 2, 4, 8 or one of the following: b (byte), h
(half word), w (word), or g (giant).

variable_or_expression
  A variable or an expression that can be resolved into a memory address.

Description
Examines memory at the address of the specified variable or the address resulting from the evaluation of an
expression. If you specify an expression, the result of the evaluation must be an lvalue.
In most cases, you will enclose the expression in `{}` symbols.

**NOTE >>** Instead of using the listed *dexamine* options, you can instead use the *gdb examine* command syntax.

## Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>dexamine</td>
<td>Examines (dumps) memory</td>
</tr>
</tbody>
</table>
**dflush**

Unwinds stack from suspended computations

**Format**

Removes the top-most suspended expression evaluation.

```
dflush
```

Removes the computation indicated by a suspended evaluation ID and all those that precede it

```
dflush susp-eval-id
```

Removes all suspended computations

```
dflush -all
```

**Arguments**

- `susp-eval-id`
  
  The ID returned or thrown by the `dprint` command or which is printed by the `dwhere` command.

- `-all`
  
  Flushes all suspended evaluations in the current focus.

**Description**

The `dflush` command unwinds the stack to eliminate frames generated by suspended computations. Typically, these frames can occur when using the `dprint -nowait` command. Other possibilities are if an error occurred in a function call in an eval point, in an expression in a Tools > Evaluate window, or if you use a `$stop` function.

Use this command as follows:

- If you don't use an argument, the CLI unwinds the top-most suspended evaluation in all threads in the current focus.

- If you use a `susp-eval-id`, the CLI unwinds each stack of all threads in the current focus, flushing all pending computations up to and including the frame associated with the ID.

- If you use the `-all` option, the CLI flushes all suspended evaluations in all threads in the current focus.

If no evaluations are suspended, the CLI ignores this command. If you do not indicate a focus, the default focus is the thread of interest.

**Examples**

The following example uses the `dprint` command to place five suspended routines on the stack. It then uses the `dflush` command to remove them. This example uses the `dflush` command in three different ways.

```
# Create 5 suspended functions
#
dl.<> dprint -nowait nothing2(7)
```
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
```
d1.<> dprint -nowait nothing2(8)
```
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
```
d1.<> dprint -nowait nothing2(9)
```
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
```
d1.<> dprint -nowait nothing2(10)
```
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
```
d1.<> dprint -nowait nothing2(11)
```
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
```
...  
```
#  
# The top of the call stack looks like:
```
d1.<> dwhere 0 nothing2 PC=0x00012520, FP=0xffbef130 [fork.cxx#310]
```
```
1 ***** Eval Function Call (11) ****************
2 nothing2 PC=0x00012520, FP=0xffbef220 [fork.cxx#310]
```
```
3 ***** Eval Function Call (10) ****************
4 nothing2 PC=0x00012520, FP=0xffbef310 [fork.cxx#310]
```
```
5 ***** Eval Function Call (9) ****************
6 nothing2 PC=0x00012520, FP=0xffbef400 [fork.cxx#310]
```
```
7 ***** Eval Function Call (8) ****************
8 nothing2 PC=0x00012520, FP=0xffbef4f0 [fork.cxx#310]
```
```
9 ***** Eval Function Call (7) ****************
10 forker PC=0x00013fd8, FP=0xffbef648 [fork.cxx#1120]
```
```
11 fork_wrap PC=0x00014780, FP=0xffbef6c8 [fork.cxx#1278] ...
```
```
#  
# Use the dflush command to remove the last item pushed
# onto the stack. Notice the frame associated with "11"
# is no longer there.
```
d1.<> dflush
```
d1.<> dwhere
```
0 nothing2 PC=0x00012520, FP=0xffbef220 [fork.cxx#310]
```
```
1 ***** Eval Function Call (10) ****************
2 nothing2 PC=0x00012520, FP=0xffbef310 [fork.cxx#310]
```
```
3 ***** Eval Function Call (9) ****************
4 nothing2 PC=0x00012520, FP=0xffbef400 [fork.cxx#310]
```
```
5 ***** Eval Function Call (8) ****************
6 nothing2 PC=0x00012520, FP=0xffbef4f0 [fork.cxx#310]
```
```
7 ***** Eval Function Call (7) ****************
8 forker PC=0x00013fd8, FP=0xffbef648 [fork.cxx#1120]
```
```
9 fork_wrap PC=0x00014780, FP=0xffbef6c8 [fork.cxx#1278]
Use the dflush command with a suspended ID argument to remove all frames up to and including the one associated with suspended ID 9. This means that IDs 7 and 8 remain.

d1.<> dflush 9
# Top of call stack after dflush 9

```
d1.<> dwhere
  0 nothing2   PC=0x00012520, FP=0xffbef400 [fork.cxx#310]
  1 ***** Eval Function Call (8) *********************
  2 nothing2   PC=0x00012520, FP=0xffbef4f0 [fork.cxx#310]
  3 ***** Eval Function Call (7) *********************
  4 forker     PC=0x00013fd8, FP=0xffbef648 [fork.cxx#1120]
  5 fork_wrap  PC=0x00014780, FP=0xffbef6c8 [fork.cxx#1278]
```

Use dflush -all to remove all frames. Only the frames associated with the program remain.

```
d1.<> dflush -all
# Top of call stack after dflush -all

```
d1.<> dwhere
  0 forker     PC=0x00013fd8, FP=0xffbef648 [fork.cxx#1120]
  1 fork_wrap  PC=0x00014780, FP=0xffbef6c8 [fork.cxx#1278]
```

RELATED TOPICS

Tools > Evaluate Command in the online Help
Built-In Statements in the TotalView for HPC User Guide
**dfocus**

Changes the current (Process/Thread P/T) set

**Format**

Changes the target of future CLI commands to this P/T set or returns the value of the current P/T set

```
dfocus [ p/t-set ]
```

Executes a command in this P/T set

```
dfocus p/t-set command
```

**Arguments**

- `p/t-set`
  - A set of processes and threads to be the target of subsequent CLI commands.

- `command`
  - A CLI command that operates on its own local focus.

**Description**

The `dfocus` command changes the set of processes, threads, and groups upon which a command acts. This command can change the focus for all commands that follow, or just the command that immediately follows.

The `dfocus` command always expects a P/T value as its first argument. This value can be either a single arena specifier or a list of arena specifiers. The default focus is `d1.<`, which selects the first user thread. The `d` (for default) indicates that each CLI command is free to use its own default width.

If you enter an optional `command`, the focus is set temporarily, and the CLI executes the `command` in the new focus. After the `command` executes, the CLI restores focus to its original value. The `command` argument can be a single command or a list.

If you use a `command` argument, the `dfocus` command returns the result of this command's execution. If you do not enter use a `command` argument, the `dfocus` command returns the focus as a string value.

**NOTE >>** Instead of a P/T set, you can enter a P/T set expression. These expressions are described in “Using P/T Set Operators” in the TotalView for HPC User Guide.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>dfocus</td>
<td>Changes the object upon which a command acts</td>
</tr>
</tbody>
</table>

**Examples**

```
dfocus g dgo
```

Continues the TotalView group that contains the focus process.
dfocus p3 {dhalt; dwhere}

    Stops process 3 and displays backtraces for each of its threads.

dfocus 2.3

    Sets the focus to thread 3 of process 2, where 2 and 3 are TotalView process and thread identifier values. The focus becomes d2.3.

dfocus 3.2

dfocus .5

    Sets and then resets command focus. A focus command that includes a dot and omits the process value uses the current process. Thus, this sequence of commands changes the focus to process 3, thread 5 (d3.5).

dfocus g dstep

    Steps the current group. Although the thread of interest (TOI) is determined by the current focus, this command acts on the entire group that contains that thread.

dfocus {p2 p3} {dwhere ; dgo}

    Performs a backtrace on all threads in processes 2 and 3, and then tells these processes to execute.

f 2.3 {f p w; f t s; g}

    Executes a backtrace (dwhere) on all the threads in process 2, steps thread 3 in process 2 (without running any other threads in the process), and continues the process.

dfocus p1

    Changes the current focus to include just those threads currently in process 1. The width is set to process. The CLI sets the prompt to p1.<.

dfocus a

    Changes the current set to include all threads in all processes. After you execute this command, your prompt changes to a1.<. This command alters CLI behavior so that actions that previously operated on a thread now apply to all threads in all processes.

dfocus gW dstatus

    Displays the status of all worker threads in the control group. The width is group level and the target is the workers group.

dfocus pW dstatus

    Displays the status of all worker threads in the current focus process. The width is process level and the target is the workers group.

f {breakpoint(a) | watchpoint(a)} st

    Shows all threads that are stopped at breakpoints or watchpoints.

f {stopped(a) - breakpoint(a)} st

    Shows all stopped threads that are not stopped at breakpoints.

Chapter 21 of the TotalView for HPC User Guide contains additional dfocus examples.
RELATED TOPICS

Using Groups, Processes, and Threads in the TotalView for HPC User Guide
**dga**

Displays Global Array variables

**Format**

\[ dga \ [ -lang \ lang\_type \ ] \ [ handle\_or\_name \ ] \ [ slice \ ] \]

**Arguments**

- **-lang**
  Specifies the language conventions to use. Without this option, TotalView uses the language used by the *thread of interest* (TOI).

  **lang_type**
  Specifies the language type to use when displaying a global array. The type must be either "c" or "f".

- **handle_or_name**
  Displays an array. This can be either a numeric handle or the name of the array. Without this argument, TotalView displays a list of all Global Arrays.

- **slice**
  Displays only a slice (that is, part of an array). If you are using C, you must place the array designators within braces \{\} because square brackets ([]) have special meaning in Tcl.

**Description**

The **dga** command displays information about Global Arrays.

If the focus includes more than one process, TotalView prints a list for each process in the focus. Because the arrays are global, each list is identical. If there is more than one thread in the focus, the CLI prints the value of the array as seen from that thread.

In almost all cases, you should change the focus to **d2.<** so that you don't include a starter process such as **prun**.

**Examples**

**dga**

Displays a list of Global Arrays, for example:

```
lb_dist
  Handle -1000
  Ghosts yes
  C type $double[129][129][27]
  Fortran Type $double_precision(27,129,129)
```

```
bc_mask
  Handle -999
  Ghosts yes
  C type long[129][129]
  Fortran Type $integer(129,129)
```
dga bc_mask (:2,:2)

Displays a slice of the bc_mask variable, for example:

\[
bc\_mask(:2,:2) = \{
(1,1) = 1 \text{ (0x00000001)} \\
(2,1) = 1 \text{ (0x00000001)} \\
(1,2) = 1 \text{ (0x00000001)} \\
(2,2) = 0 \text{ (0x00000000)}
\}
\]

dga --lang c -998 [:1][:1]

Displays the same bc_mask variable as in the previous example in C format. In this case, the command refers to the variable by its handle.

RELATED TOPICS

Tools > Global Arrays Command in the online Help
Debugging Global Arrays Applications in the TotalView for HPC User Guide
dgo

Resumes execution of processes

Format

```
dgo
```

Arguments

```
-back
```

(ReplayEngine only) Runs the nonheld process in the current focus backward until it hits some action point or the beginning of recorded Replay history. This option can be abbreviated to `-b`.

Description

The `dgo` command resumes execution of all nonheld processes and threads in the current focus. If the process does not exist, this command creates it, passing it the default command arguments. These can be arguments passed into the CLI, or they can be the arguments set with the `drerun` command. If you are also using the TotalView GUI, you can set this value by using the `Process > Startup Parameters` command.

If a process or thread is held, it ignores this command.

You cannot use a `dgo` command when you are debugging a core file, nor can you use it before the CLI loads an executable and starts executing it.

If you do not indicate a focus, the default focus is the process of interest (POI).

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td><code>dgo</code></td>
<td>Resumes execution</td>
</tr>
<tr>
<td>G</td>
<td><code>{dfocus g dgo}</code></td>
<td>Resumes group</td>
</tr>
</tbody>
</table>

Examples

```
dgo
```

Resumes execution of all stopped threads that are not held and which belong to processes in the current focus. (Threads held at barriers are not affected.)

```
G
```

Resumes execution of all threads in the current control group.

```
f p g
```

Continues the current process. Only threads that are not held can run.

```
f q g
```

Continues all processes in the control group. Only processes and threads that are not held are allowed to run.

```
f qL g
```

Continues all threads in the share group that are at the same PC as the thread of interest (TOI).
CLI Commands

f pL g
Continues all threads in the current process that are at the same PC as the TOI.

f t g
Continues a single thread.

RELATED TOPICS

Starting Processes and Threads in the TotalView for HPC User Guide
Process > Go Command in the online Help
Thread > Go Command in the online Help
dcont Command
dgroups

Manipulates and manages groups

**Format**

Adds members to thread and process groups

```
dgroups -add [-g gid] [id-list]
```

Deletes groups

```
dgroups -delete [-g gid]
```

Intersects a group with a list of processes and threads

```
dgroups -intersect [-g gid] [id-list]
```

Prints process and thread group information

```
dgroups [-list] [pattern-list]
```

Creates a new thread or process group

```
dgroups -new [thread_or_process] [-g gid] [id-list]
```

Removes members from thread or process groups

```
dgroups -remove [-g gid] [id-list]
```

**Arguments**

- `-g gid`
  The group ID on which the command operates. The `gid` value can be an existing numeric group ID, an existing group name, or, if you are using the `-new` option, a new group name.

- `id-list`
  A Tcl list that contains process and thread IDs. Process IDs are integers; for example, 2 indicates process 2. Thread IDs define a `pid tid` pair and look like decimal numbers; for example, 2.3 indicates process 2, thread 3. If the first element of this list is a group tag, such as the word `control`, the CLI ignores it. This makes it easy to insert all members of an existing group as the items to be used in any of these operations. (See the `dset` command's discussion of the `GROUP(gid)` variable for information on group designators.) These words appear in some circumstances when the CLI returns lists of elements in P/T sets.

- `pattern-list`
  A pattern to be matched against group names. The pattern is a Tcl regular expression.

- `thread_or_process`
  Keywords that create a new process or thread group. You can specify one of the following arguments: `t`, `thread`, `p`, or `process`.

**Description**

The `dgroups` command supports the following functions:

- Adding members to process and thread groups.
- Creating a group.
• Intersecting a group with a set of processes and threads.
• Deleting groups.
• Displaying the name and contents of groups.
• Removing members from a group.

**dgroups -add**

Adds members to one or more thread or process groups. The CLI adds each of these threads and processes to the group. If you add a:

• Process to a thread group, the CLI adds all of its threads.
• Thread to a process group, the CLI adds the thread’s parent process.

You can abbreviate the `-add` option to `-a`.

The CLI returns the ID of this group.

You can explicitly name the items being added by using an *id-list* argument. Without an *id-list* argument, the CLI adds the threads and processes in the current focus. Similarly, you can name the group using the `-g` option. Without the `-g` option, the CLI uses the groups in the current focus.

If the *id-list* argument contains processes, and the target is a thread group, the CLI adds all threads from these processes. If it contains threads and the target is a process group, the CLI adds the parent process for each thread.

**NOTE >>** If you specify an *id-list* argument and you also use the `-g` option, the CLI ignores the focus. You can use two *dgroups -add* commands instead.

If you try to add the same object more than once to a group, the CLI adds it only once.

You cannot use this command to add a process to a control group. Instead, use the `CGROUP(dpid)` variable; for example:

```
  dset CGROUP($mypid) $new_group_id
```

**dgroups -delete**

Deletes the target group. You can delete only groups that you create; you cannot delete groups that TotalView creates.

**dgroups -intersect**

Intersects a group with a set of processes and threads. If you intersect a thread group with a process, the CLI includes all of the process’s threads. If you intersect a process group with a thread, the CLI uses the thread’s process.
After this command executes, the group no longer contains members that were not in this intersection.

You can abbreviate the `-intersect` option to `-i`.

**dgroups -list**

Prints the name and contents of process and thread groups. If you specify a `pattern-list` as an argument, the CLI only prints information about groups whose names match this pattern. When entering a list, you can specify a `pattern`. The CLI matches this pattern against the list of group names by using the Tcl `regex` command.

**NOTE >>** If you do not enter a pattern, the CLI displays only groups that you have created with nonnumeric names.

You can abbreviate `-list` to `-l`.

**dgroups -new**

Creates a new thread or process group and adds threads and processes to it. If you use a name with the `-g` option, the CLI uses that name for the group ID; otherwise, it assigns a new numeric ID. If the group you name already exists, the CLI replaces it with the newly created group.

The CLI returns the ID of the newly created group.

You can explicitly name the items being added with an `id-list` argument. If you do not use an `id-list` argument, the CLI adds the threads and processes in the current focus.

If the `id-list` argument contains processes, and the target is a thread group, the CLI adds all threads from these processes. If it contains threads and the target is a process group, TotalView adds the parent process for each thread.

**NOTE >>** If you use an `id-list` argument and also use the `-g` option, the CLI ignores the focus. You can use two `dgroups -add` commands instead.

If you are adding more than one object and one of these objects is a duplicate, The CLI adds the nonduplicate objects to the group.

You can abbreviate the `-new` option to `-n`.

**dgroups -remove**

Removes members from one or more thread or process groups. If you remove a process from a thread group, The CLI removes all of its threads. If remove a thread from a process group, The CLI removes its parent process.

You cannot remove processes from a control group. You can, however, move a process from one control group to another by using the `dset` command to assign it to the CGROUP(dpid) variable group.

Also, you cannot use this command on read-only groups, such as share groups.
You can abbreviate the `-remove` option to `-r`.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr</td>
<td>dgroups</td>
<td>Manipulates a group</td>
</tr>
</tbody>
</table>

**Examples**

dgroups -add
f tW gr -add

  Adds the focus thread to its workers group.

dgroups -add

  Adds the current focus thread to the current focus group.

  ```
  set gid [dgroups -new thread ($CGROUP(1))]
  ```

  Creates a new thread group that contains all threads from all processes in the control group for process 1.

  ```
  f $a_group/9 dgroups -add
  ```

  Adds process 9 to a user-defined group.

  **dgroups -delete**
  **gr -delete -g mygroup**

  Deletes the `mygroup` group.

  **dgroups -intersect**
  **dgroups -intersect -g 3 3.2**

  Intersects thread 3.2 with group 3. If group 3 is a thread group, this command removes all threads except 3.2 from the group; if it is a process group, this command removes all processes except process 3 from it.

  ```
  f tW gr -i
  ```

  Intersects the focus thread with the threads in its workers group.

  ```
  f gW gr -i -g mygroup
  ```

  Removes all nonworker threads from the `mygroup` group.

  **dgroups -list**
  **dgroups -list**

  Displays information about all named groups; for example:

  ```
  ODD_P: {process 1 3}
  EVEN_P: {process 2 4}
  ```

  ```
  gr -l *
  ```

  Displays information about groups in the current focus.

  ```
  1: {control 1 2 3 4}
  2: {workers 1.1 1.2 1.3 1.4 2.1 2.2 2.3 2.4 3.1
    3.2 3.3 3.4 4.1 4.2 4.3 4.4}
  ```
3: {share 1 2 3 4}
ODD_P: {process 1 3}
EVEN_P: {process 2 4}

\textbf{dgroups -new}
\begin{verbatim}
gr -n t -g mygroup $GROUP($CGROUP(1))
\end{verbatim}
Creates a new thread group named \texttt{mygroup} that contains all threads from all processes in the control group for process 1.
\begin{verbatim}
set mygroup [dgroups -new]
\end{verbatim}
Creates a new process group that contains the current focus process.

\textbf{dgroups -remove}
\textbf{dgroups -remove -g 3 3.2}
Removes thread 3.2 from group 3.
\begin{verbatim}
f W dgroups -add
\end{verbatim}
Marks the current thread as being a worker thread.
\begin{verbatim}
f W dgroups -r
\end{verbatim}
Indicates that the current thread is not a worker thread.

\textbf{RELATED TOPICS}

- Using the Group Editor in the \textit{TotalView for HPC User Guide}
- Using Groups, Processes, and Threads in the \textit{TotalView for HPC User Guide}
- Setting Group Focus in the \textit{TotalView for HPC User Guide}
- Group > Edit Group Command in the online Help
dhalt

Suspects execution of processes

Format

dhalt

Arguments

This command has no arguments

Description

The dhalt command stops all processes and threads in the current focus.
If you do not indicate a focus, the default focus is the process of interest (POI).

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>dhalt</td>
<td>Suspends execution</td>
</tr>
<tr>
<td>H</td>
<td>{dfocus g dhalt}</td>
<td>Suspends group execution</td>
</tr>
</tbody>
</table>

Examples

dhalt

Suspends execution of all running threads belonging to processes in the current focus. (This command does not affect threads held at barriers.)

f t 1.1 h

Suspends execution of thread 1 in process 1. Note the difference between this command and f 1.< dhalt. If the focus is set as thread level, this command halts the first user thread, which is probably thread 1.

RELATED TOPICS

Stopping Processes and Threads in the TotalView for HPC User Guide
Updating Process Information in the TotalView for HPC User Guide
Group > Halt Command in the online Help
Process > Halt Command in the online Help
Thread > Halt Command in the online Help
**dheap**

Controls heap debugging

**Format**

Shows Memory Debugger state

```
dheap [-status ]
```

Applies a saved configuration file

```
dheap -apply_config { default | filename }
```

Shows information about a backtrace

```
dheap -backtrace [ subcommands ]
```

Compares memory states

```
dheap -compare subcommands [ optional_subcommands ]
[ process | filename [ process | filename ] ]
```

Enables or disables the Memory Debugger

```
dheap { -enable | -disable } 
```

Enables or disables event notification

```
dheap -event_filter subcommands
```

Writes memory information

```
dheap -export subcommands
```

Specifies the filters the Memory Debugger uses

```
dheap -filter subcommands
```

Writes guard blocks (memory before and after an allocation)

```
dheap -guard [ subcommands ]
```

Enables or disables the retaining (hoarding) of freed memory blocks

```
dheap -hoard [ subcommands ]
```

Displays Memory Debugger information

```
dheap -info [ subcommands ]
```

Indicates whether an address is in a deallocated block

```
dheap -is_dangling address
```

Locates memory leaks

```
dheap -leaks [-check_interior ]
```

Enables or disables Memory Debugger event notification

```
dheap [-no]notify
```

Paints memory with a distinct pattern

```
dheap -paint [ subcommands ]
```
Enables or disables the ability to catch bounds errors and use-after-free errors retaining freed memory blocks

\texttt{\textit{dheap\_red\_zones [ subcommands ]}}

Enables or disables allocation and reallocation notification

\texttt{\textit{dheap\_tag\_alloc subcommand [ start\_address [ end\_address ] ]}}

Displays the Memory Debugger version number

\texttt{\textit{dheap\_version}}

\textbf{Description}

The \texttt{dheap} command is described in the \textit{Batch Scripting} section of “Locating Memory Problems” in the \textit{MemoryScape} documentation.
dhistory

Performs actions upon ReplayEngine

**Format**


**Arguments**

- **-info**  
  Writes ReplayEngine information including the current time, the live time, and whether the process is in replay or record mode. If you do not enter any arguments, **-info** is the default.

- **-get_time**  
  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**  
  Places the process back to the virtual time specified by the **time** integer 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.

- **-go_live**  
  Places the process back to the PC and back into record mode. You can resume your “regular” debugging session.

- **-enable**  
  Enables ReplayEngine, starting the history

- **-disable**  
  Disables ReplayEngine, ending the history.

- **-save file**  
  Saves the current replay history to a file. There is an optional argument to specify the name of the file to save to. The file specification can be a path or a simple file name, in which case it is saved in the current working directory. If no file is specified, the recording is saved in the current working directory with the file name `replay_pid_hostname.recording`.

  To reload the recording file, use one of the following commands based on the functionality for loading core files. TotalView recognizes the recording file for what it is and acts appropriately.

  To reload a recording at startup:

  ```sh
  totalview executable recording-file
  ```

  To reload a recording file when TotalView is running:
detach filename -c recording-file

The recording-file argument can be either a path or a simple file name, in which case the current working directory is assumed.

Description

The dhistory command displays information about the current process either by default or when using the -info argument. In addition, options to this command can obtain a debugging time, which can be stored in a variable to go back to a particular time.

In addition, you can enable and display ReplayEngine as well as put it back into regular debugging mode using the -go_live option. You'll need to do this after your program is placed into replay mode. This occurs whenever you use any GUI or CLI command that moves to replay mode. For example, in the CLI, this can occur when you execute such commands as dnext or dout.

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay</td>
<td>dhistory</td>
<td>Performs actions upon ReplayEngine.</td>
</tr>
</tbody>
</table>

Examples

dhistory

Typing this command displays the following information:

History info for process 1

  Live time: 421 0x80485d6
  Current time: 421 0x80485d6
  Live PC: 0x80485d6
  Record Mode: True
  Replay Wanted: True
  Stop Reason: Normal result [waitpid, search, or goto_time
  Temp directory: /tmp/replay_jsm_local/replay_session_pZikY9
  Event log mode: circular
  Event log size: 268435456
**dhold**

Holds threads or processes

**Format**

Holds processes

```
dhold -process
```

Holds threads

```
dhold -thread
```

**Arguments**

- **-process**
  
  Holds processes in the current focus. Can be abbreviated to `-p`.

- **-thread**
  
  Holds threads in the current focus. Can be abbreviated to `-t`.

**Description**

The `dhold` command holds the threads and processes in the current focus.

**NOTE >>**

You cannot hold system manager threads. In all cases, holding threads that aren't part of your program always involves some risk.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp</td>
<td><code>{dhold -process}</code></td>
<td>Holds the focus process</td>
</tr>
<tr>
<td>HP</td>
<td><code>{f g dhold -process}</code></td>
<td>Holds all processes in the focus group</td>
</tr>
<tr>
<td>ht</td>
<td><code>{f t dhold -thread}</code></td>
<td>Holds the focus thread</td>
</tr>
<tr>
<td>HT</td>
<td><code>{f g dhold -thread}</code></td>
<td>Holds all threads in the focus group</td>
</tr>
<tr>
<td>htp</td>
<td><code>{f p dhold -thread}</code></td>
<td>Holds all threads in the focus process</td>
</tr>
</tbody>
</table>

**Examples**

```
f W HT
```

Holds all worker threads in the focus group.

```
f s HP
```

Holds all processes in the share group.

```
f $mygroup/ HP
```

Holds all processes in the group identified by the contents of `mygroup`.
RELATED TOPICS

Group > Hold Command in the online Help
Process > Hold Command in the online Help
Thread > Hold Command in the online Help
Group > Release Command in the online Help
Process > Release Threads Command in the online Help
dunhold Command
**dkill**

Terminates execution of processes

**Format**

dkill [ -remove ]

**Arguments**

- remove

Removes all knowledge of the process from its internal tables. If you are using TotalView Team, this frees a token so that you can reuse it.

**Description**

The `dkill` command terminates all processes in the current focus.

Because the executables associated with the defined processes are still loaded, using the `drun` command restarts the processes.

The `dkill` command alters program state by terminating all processes in the affected set. In addition, TotalView destroys any spawned processes when the process that created them is killed. The `drun` command can restart only the initial process.

If you do not indicate a focus, the default focus is the process of interest (POI). If, however, you kill the primary process for a control group, all of the slave processes are killed.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>dkill</td>
<td>Terminates a process's execution</td>
</tr>
</tbody>
</table>

**Examples**

dkill

Terminates all threads belonging to processes in the current focus.

dfocus {p1 p3} dkill

Terminates all threads belonging to processes 1 and 3.

**RELATED TOPICS**

- Starting Your Program in the CLI in the TotalView for HPC User Guide
- Restarting and Deleting Programs in the TotalView for HPC User Guide
- Group > Delete Command in the online Help
- Group > Restart Command in the online Help


**dlappend**

Appends list elements to a TotalView variable

**Format**

dlappend variable-name value [ ... ]

**Arguments**

- **variable-name**
  
The variable to which values are appended.

- **value**
  
The values to append.

**Description**

The `dlappend` command appends list elements to a TotalView variable. This command performs the same function as the Tcl `lappend` command, differing in that `dlappend` does not create a new debugger variable. That is, the following Tcl command creates a variable named `foo`:

```tcl
lappend foo 1 3 5
```

In contrast, the CLI command displays an error message:

```text
dlappend foo 1 3 5
```

**Examples**

dlappend TV::process_load_callbacks my_load_callback

  Adds the `my_load_callback` function to the list of functions in the TV::process_load_callbacks variable.

**RELATED TOPICS**

dset Command
dlist

**Format**
Displays source code relative to the current list location
```
dlist [-n num-lines ]
```
Displays source code relative to a named place
```
dlist breakpoint-expr [-n num-lines ]
```
Displays source code relative to the current execution location
```
dlist -e [-n num-lines ]
```

**Arguments**

- `-n num-lines`
  Displays this number of lines rather than the default number. (The default is the value of the MAX_LIST variable.) If `num-lines` is negative, the CLI displays lines before the current location, and additional `dlist` commands show preceding lines in the file rather than following lines.

  This option also sets the value of the `MAX_LIST` variable to `num-lines`.

- `breakpoint-expr`
  The location at which the CLI begins displaying information. In most cases, specify this location as a line number or as a string that contains a file name, function name, and line number, each separated by `#` characters; for example: `file#func#line`.

  For more information, see Chapter 9, “Qualifying Symbol Names” in the TotalView for HPC User Guide. The CLI creates defaults if you omit parts of this specification.

  If you enter a different file, it is used for future display. This means that if you want to display information relative to the current thread's execution point, use the `-e` option to `dlist`.

  If the breakpoint expression evaluates to more than one location, TotalView chooses one.

  For other ways to enter these expressions, see “Breakpoint Expressions” on page 43. If you name more than one address, TotalView picks one.

- `-e`
  Sets the display location to include the current execution point of the thread of interest (TOI). If you use `dup` and `ddown` commands to select a buried stack frame, this location includes the PC (program counter) for that stack frame.

**Description**
The `dlist` command displays source code lines relative to a source code location, called the list location. The CLI prints this information; it is not returned. If you do not specify `source-loc` or `-e`, the command continues where the previous list command stopped. To display the thread's execution point, use the `dlist -e` command.

If you enter a file or procedure name, the listing begins at the file or procedure's first line.
The default focus for this command is thread level. If your focus is at process level, TotalView acts on each thread in the process.

The first time you use the `dlist` command after you focus on a different thread—or after the focus thread runs and stops again—the location changes to include the current execution point of the new focus thread.

Tabs in the source file are expanded as blanks in the output. The TAB_WIDTH variable controls the tab stop width, which defaults to 8. If `TAB_WIDTH` is set to -1, no tab processing is performed, and the CLI displays tabs using their ASCII value.

All lines appear with a line number and the source text for the line. The following symbols are also used:

- `@` — An action point is set at this line.
- `>` — The PC for the current stack frame is at the indicated line and this is the leaf frame.
- `=` — The PC for the current stack frame is at the indicated line and this is a buried frame; this frame has called another function so that this frame is not the active frame.

These correspond to the marks shown in the backtrace displayed by the `dwhere` command that indicates the selected frame.

Here are some general rules:

- The initial display location is `main()`.
- The CLI sets the display location to the current execution location when the focus is on a different thread.

If the `source-loc` argument is not fully qualified, the CLI looks for it in the directories named in the CLI EXECUTABLE_PATH variable.

### Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td><code>dlist</code></td>
<td>Displays lines</td>
</tr>
</tbody>
</table>

### Examples

The following examples assume that the MAX_LIST variables equals 20, which is its initial value.

```bash
dlist
```

Displays 20 lines of source code, beginning at the current list location. The list location is incremented by 20 when the command completes.
dlist 10

Displays 20 lines, starting with line 10 of the file that corresponds to the current list location. Because this uses an explicit value, the CLI ignores the previous command. The list location is changed to line 30.

dlist -n 10

Displays 10 lines, starting with the current list location. The value of the list location is incremented by 10.

dlist -n -50

Displays source code preceding the current list location; shows 50 lines, ending with the current source code location. The list location is decremented by 50.

dlist do_it

Displays 20 lines in procedure do_it. Changes the list location to be the 20th line of the procedure.

dfocus 2.< dlist do_it

Displays 20 lines in the do_it routine associated with process 2. If the current source file is named foo, you can also specify this as dlist foo#do_it, naming the executable for process 2.

dlist -e

Displays 20 lines starting 10 lines above the current execution location.

f 1.2 l -e

Lists the lines around the current execution location of thread 2 in process 1.

dfocus 1.2 dlist -e -n 10

Produces essentially the same listing as the previous example, differing in that it displays 10 lines.

dlist do_it.f#80 -n 10

Displays 10 lines, starting with line 80 in file do_it.f. Updates the list location to line 90.
dload Loads debugging information

Format

dload [-g gid] [-r hname]
  [{ -np | -procs | -tasks } num]
  [-nodes num]
  [-replay | -no_replay]
  [-mpi starter]
  [-starter_args argument]
  [-env variable=value] ...
  [-e executable]
  [-parallel_attach_subset subset_specification]

Arguments

-g gid
  Sets the control group for the process being added to the group ID specified by gid. This group must already exist. (The CLI GROUPS variable contains a list of all groups.)

-r hname
  The host on which the process will run. The CLI launches a TotalView Debugger Server on the host machine if one is not already running there. (See “Setting Up Remote Debugging Sessions” in the TotalView for HPC User Guide for information on the server launch commands.)

{ -np | -procs | -tasks } num
  Indicates the number of processes or tasks that the starter program creates.

-nodes num
  Indicates the number of nodes upon which your program will execute.

-replay | -no_replay
  These options enable and disable the ReplayEngine the next time the program is restarted.

-starter_args argument
  Indicates additional arguments to be passed to the starter program.

-env variable=value
  Sets a variable that is added to the program's environment.

-e
  Indicates that the next argument is an executable file name. You need to use -e if the executable name begins with a dash (-) or consists of only numeric characters. Otherwise, just provide the executable file name.

eexecutable
  A fully or partially qualified file name for the file corresponding to the program.
-parallel_attach_subset subset_specification

Defines a list of MPI ranks to attach to when an MPI job is created or attached to. The list is space-separated; each element can have one of three forms:

rank: specifies that rank only

rank1-rank2: specifies all ranks between rank1 and rank2, inclusive

rank1-rank2:stride: specifies every strideth rank between rank1 and rank2

A rank must be either a positive decimal integer or max (the last rank in the MPI job).

A subset_specification that is the empty string (""") is equivalent to 0-max.

For example:

dload -parallel_attach_subset {1 2 4-6 7-max:2} mpirun

will attach to ranks 1, 2, 4, 5, 6, 7, 9, 11, 13,...

Description

The dload command creates a new TotalView process object for the executable file and returns its TotalView ID.

NOTE >>

Your license limits the number of processes that you can run at the same time. For example, the maximum number of processes for TotalView Individual is 16. As some systems and run time environments create threads to manage a process, you may not be able to get this many processes running at the same time. (Only TotalView Individual counts threads against your license. TotalView Enterprise and Team allows an unlimited number of threads to run at the same time.)

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lo</td>
<td>dload</td>
<td>Loads debugging information</td>
</tr>
</tbody>
</table>

Examples

dload do_this

Loads the debugging information for the do_this executable into the CLI. After this command completes, the process does not yet exist and no address space or memory is allocated to it.

dload -mpi POE -starter_args "hfile=~/my_hosts" \
    -np 2 -nodes

Loads an MPI job using the POE configuration. Two processes will be used across nodes. The hfiles starter argument is used.

lo -g 3 -r other_computer do_this

Loads the debugging information for the do_this executable that is executing on the other_computer machine into the CLI. This process is placed into group 3.
f g3 lo -r other_computer do_this

Does not do what you would expect it to do because the dload command ignores the focus command. Instead, this does exactly the same thing as the previous example.

dload -g $CGROUP(2) -r slowhost foo

Loads another process based on image foo on machine slowhost. The CLI places this process in the same group as process 2.

dload -env DISPLAY=aurora:0.0
   -env STARTER=/~starter myprog

Sets up two environment variables $DISPLAY and $STARTER for the program myprog and loads myprog's debugging information.

RELATED TOPICS

Loading Executables in the TotalView for HPC User Guide
File > New Program Command in the online Help
dattach Command
drun Command
dmstat

Displays memory use information

Format

dmstat

Arguments

This command has no arguments

Description

The dmstat command displays information on your program's memory use, returning information in three parts:

• **Memory usage summary**: The minimum and maximum amounts of memory used by the text and data segments, the heap, and the stack, as well as the virtual memory stack usage and the virtual memory size.

• **Individual process statistics**: The amount of memory that each process is currently using.

• **Image information**: The name of the image, the image's text size, the image's data size, and the set of processes using the image.

The following table describes the displayed columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The amount of memory used to store your program's machine code instructions. The text segment is sometimes called the code segment.</td>
</tr>
<tr>
<td>data</td>
<td>The amount of memory used to store initialized and uninitialized data.</td>
</tr>
<tr>
<td>heap</td>
<td>The amount of memory currently used for data created at run time; for example, calls to the malloc() function allocate space on the heap while the free() function releases it.</td>
</tr>
<tr>
<td>stack</td>
<td>The amount of memory used by the currently executing routine and all the routines in its backtrace. If this is a multithreaded process, TotalView shows only information for the main thread's stack. Note that the stacks of other threads might not change over time on some architectures. On some systems, the space allocated for a thread is considered part of the heap. For example, if your main routine invokes function foo(), the stack contains two groups of information—these groups are called frames. The first frame contains the information required for the execution of your main routine, and the second, which is the current frame, contains the information needed by the foo() function. If foo() invokes the bar() function, the stack contains three frames. When foo() finishes executing, the stack contains only one frame.</td>
</tr>
</tbody>
</table>
CLI Commands

dmstat

dmstat is sensitive to the focus. Note this four-process program:

<table>
<thead>
<tr>
<th>process</th>
<th>text</th>
<th>data</th>
<th>heap</th>
<th>stack</th>
<th>[stack_vm]</th>
<th>vm_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(9271)</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10432</td>
<td>[16384]</td>
</tr>
</tbody>
</table>

image information:

<table>
<thead>
<tr>
<th>image_name</th>
<th>text</th>
<th>data</th>
<th>dpids</th>
</tr>
</thead>
<tbody>
<tr>
<td>...ry/forked_mem_exampleLINUX</td>
<td>2524</td>
<td>16778479</td>
<td>1</td>
</tr>
<tr>
<td>/lib/i686/libpthread.so.0</td>
<td>32172</td>
<td>27948</td>
<td>1</td>
</tr>
<tr>
<td>/lib/i686/libc.so.6</td>
<td>1050688</td>
<td>122338</td>
<td>1</td>
</tr>
<tr>
<td>/lib/ld-linux.so.2</td>
<td>70240</td>
<td>10813</td>
<td>1</td>
</tr>
</tbody>
</table>

The CLI prints the following for a four-process program:

<table>
<thead>
<tr>
<th>process</th>
<th>text</th>
<th>data</th>
<th>heap</th>
<th>stack</th>
<th>[stack_vm]</th>
<th>vm_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(9979)</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>17.44M</td>
</tr>
<tr>
<td>5</td>
<td>(9982)</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>17.44M</td>
</tr>
<tr>
<td>6</td>
<td>(9983)</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>17.44M</td>
</tr>
<tr>
<td>7</td>
<td>(9984)</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>17.44M</td>
</tr>
</tbody>
</table>

maximum:

<table>
<thead>
<tr>
<th>process</th>
<th>text</th>
<th>data</th>
<th>heap</th>
<th>stack</th>
<th>[stack_vm]</th>
<th>vm_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(9979)</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10432</td>
<td>17.69M</td>
</tr>
</tbody>
</table>

minimum:

<table>
<thead>
<tr>
<th>process</th>
<th>text</th>
<th>data</th>
<th>heap</th>
<th>stack</th>
<th>[stack_vm]</th>
<th>vm_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>(9982)</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10432</td>
<td>17.44M</td>
</tr>
</tbody>
</table>

image information:

<table>
<thead>
<tr>
<th>image_name</th>
<th>text</th>
<th>data</th>
<th>dpids</th>
</tr>
</thead>
<tbody>
<tr>
<td>...ry/forked_mem_exampleLINUX</td>
<td>2524</td>
<td>16778479</td>
<td>1 5 6 7</td>
</tr>
<tr>
<td>/lib/i686/libpthread.so.0</td>
<td>32172</td>
<td>27948</td>
<td>1 5 6 7</td>
</tr>
<tr>
<td>/lib/i686/libc.so.6</td>
<td>1050688</td>
<td>122338</td>
<td>1 5 6 7</td>
</tr>
<tr>
<td>/lib/ld-linux.so.2</td>
<td>70240</td>
<td>10813</td>
<td>1 5 6 7</td>
</tr>
</tbody>
</table>
RELATED TOPICS

Generate a Memory Usage Report in the MemoryScape in-product help
Opening MemoryScape to examine memory usage in the “Creating Programs for Memory Debugging” chapter of Debugging Memory Problems with MemoryScape™
dnext

Steps source lines, stepping over subroutines

Format

dnext [ -back ] [ num-steps ]

Arguments

- back
  (ReplayEngine only) Steps to the previous source line, stepping over subroutines. This option can be abbreviated to -b.

num-steps
  An integer greater than 0, indicating the number of source lines to be executed.

Description

The dnext command executes source lines; that is, it advances the program by steps (source line statements). However, if a statement in a source line invokes a routine, the dnext command executes the routine as if it were one statement; that is, it steps over the call.

The optional num-steps argument defines how many dnext operations to perform. If you do not specify num-steps, the default is 1.

The dnext command iterates over the arenas in its focus set, performing a thread-level, process-level, or group-level step in each arena, depending on the width of the arena. The default width is process (p).

For more information on stepping in processes and threads, see dstep on page 142.
## Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>dnext</td>
<td>Runs the <em>thread of interest</em> (TOI) one statement, while allowing other threads in the process to run.</td>
</tr>
<tr>
<td>N</td>
<td>{dfocus g dnext}</td>
<td>A group stepping command. This searches for threads in the share group that are at the same PC as the TOI, and steps one such aligned thread in each member one statement. The rest of the control group runs freely.</td>
</tr>
<tr>
<td>nl</td>
<td>{dfocus L dnext}</td>
<td>Steps the process threads in lockstep. This steps the TOI one statement and runs all threads in the process that are at the same PC as the TOI to the same statement. Other threads in the process run freely. The group of threads that is at the same PC is called the lockstep group. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>NL</td>
<td>{dfocus gl dnext}</td>
<td>Steps lockstep threads in the group. This steps all threads in the share group that are at the same PC as the TOI one statement. Other threads in the control group run freely.</td>
</tr>
<tr>
<td>nw</td>
<td>{dfocus W dnext}</td>
<td>Steps worker threads in the process. This steps the TOI one statement, and runs all worker threads in the process to the same (goal) statement. The nonworker threads in the process run freely. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>NW</td>
<td>{dfocus gW dnext}</td>
<td>Steps worker threads in the group. This steps the TOI one statement, and runs all worker threads in the same share group to the same statement. All other threads in the control group run freely.</td>
</tr>
</tbody>
</table>

### Examples

- **dnext**
  - Steps one source line.
- **n 10**
  - Steps ten source lines.
- **N**
  - Steps one source line. It also runs all other processes in the group that are in the same lockstep group to the same line.
- **f t n**
  - Steps the thread one statement.
- **dfocus 3. dnext**
  - Steps process 3 one step.
RELATED TOPICS

Creating a Process by Single Stepping in the TotalView for HPC User Guide
Stepping and Setting Breakpoints in the TotalView for HPC User Guide
Creating a Process by Single Stepping in the TotalView for HPC User Guide
Stepping and Setting Breakpoints in the TotalView for HPC User Guide
Using Stepping Commands in the TotalView for HPC User Guide
Group > Next Command in the online Help
Process > Next Command in the online Help
Thread > Next Command in the online Help

dnexti Command
dstep Command
dfocus Command
**dnexti**

Steps machine instructions, stepping over subroutines

**Format**

```plaintext
dnexti [-back] [num-steps]
```

**Arguments**

- **-back**
  (ReplayEngine only) Steps a machine instruction back to the previous instruction, stepping over subroutines. This option can be abbreviated to `-b`.

- **num-steps**
  An integer greater than 0, indicating the number of instructions to be executed.

**Description**

The `dnexti` command executes machine-level instructions; that is, it advances the program by a single instruction. However, if the instruction invokes a subfunction, the `dnexti` command executes the subfunction as if it were one instruction; that is, it steps over the call. This command steps the thread of interest (TOI) while allowing other threads in the process to run.

The optional `num-steps` argument defines how many `dnexti` operations to perform. If you do not specify `num-steps`, the default is 1.

The `dnexti` command iterates over the arenas in the focus set, performing a thread-level, process-level, or group-level step in each arena, depending on the width of the arena. The default width is `process (p)`.

For more information on stepping in processes and threads, see `dstep` on page 142.
### Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ni</td>
<td>dnexti</td>
<td>Runs the TOI one instruction while allowing other threads in the process to run.</td>
</tr>
<tr>
<td>NI</td>
<td>{dfocus g dnexti}</td>
<td>A group stepping command. This searches for threads in the share group that are at the same PC as the TOI, and steps one such aligned thread in each member one instruction. The rest of the control group runs freely.</td>
</tr>
<tr>
<td>nil</td>
<td>{dfocus L dnexti}</td>
<td>Steps the process threads in lockstep. This steps the TOI one instruction, and runs all threads in the process that are at the same PC as the TOI to the same statement. Other threads in the process run freely. The group of threads that is at the same PC is called the lockstep group. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>NIL</td>
<td>{dfocus gL dnexti}</td>
<td>Steps lockstep threads in the group. This steps all threads in the share group that are at the same PC as the TOI one instruction. Other threads in the control group run freely.</td>
</tr>
<tr>
<td>niw</td>
<td>{dfocus W dnexti}</td>
<td>Steps worker threads in the process. This steps the TOI one instruction, and runs all worker threads in the process to the same (goal) statement. The nonworker threads in the process run freely. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>NIW</td>
<td>{dfocus gW dnexti}</td>
<td>Steps worker threads in the group. This steps the TOI one instruction, and runs all worker threads in the same share group to the same statement. All other threads in the control group run freely.</td>
</tr>
</tbody>
</table>

### Examples

**dnexti**

Steps one machine-level instruction.

**ni 10**

Steps ten machine-level instructions.

**NI**

Steps one instruction and runs all other processes in the group that were executing at that instruction to the next instruction.

**f t n**

Steps the thread one machine-level instruction.

**dfocus 3. dnexti**

Steps process 3 one machine-level instruction.
RELATED TOPICS

Creating a Process by Single Stepping in the TotaView for HPC User Guide
Stepping and Setting Breakpoints in the TotaView for HPC User Guide
Using Stepping Commands in the TotaView for HPC User Guide
Process > Next Instruction Command in the online Help
Thread > Next Instruction Command in the online Help
dnexti Command
dstep Command
dfocus Command
**dout**

Executes until just after the place that called the current routine

**Format**

dout [-back] [frame-count]

**Arguments**

- **-back**
  (ReplayEngine only) Returns to the function call that placed the PC into the current routine. This option can be abbreviated to -b.

- **frame-count**
  An integer that specifies that the thread returns out of this many levels of subroutine calls. Without this number, the thread returns from the current level.

**Description**

The **dout** command runs a thread until it returns from either of the following:

- The current subroutine
- One or more nested subroutines

When you specify process width, TotalView allows all threads in the process that are not running to this goal to run free. (Specifying process width is the default.)
**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ou</td>
<td>dout</td>
<td>Runs the <em>thread of interest</em> (TOI) out of the current function, while allowing other threads in the process to run.</td>
</tr>
<tr>
<td>OU</td>
<td>{dfocus g dout}</td>
<td>Searches for threads in the share group that are at the same PC as the TOI, and runs one such aligned thread in each member out of the current function. The rest of the control group runs freely. This is a group stepping command.</td>
</tr>
<tr>
<td>oul</td>
<td>{dfocus L dout}</td>
<td>Runs the process threads in lockstep. This runs the TOI out of the current function, and also runs all threads in the process that are at the same PC as the TOI out of the current function. Other threads in the process run freely. The group of threads that is at the same PC is called the lockstep group. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>OUL</td>
<td>{dfocus gL dout}</td>
<td>Runs lockstep threads in the group. This runs all threads in the share group that are at the same PC as the TOI out of the current function. Other threads in the control group run freely.</td>
</tr>
<tr>
<td>ouw</td>
<td>{dfocus W dout}</td>
<td>Runs worker threads in the process. This runs the TOI out of the current function and runs all worker threads in the process to the same (goal) statement. The nonworker threads in the process run freely. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>OUW</td>
<td>{dfocus gW dout}</td>
<td>Runs worker threads in the group. This runs the TOI out of the current function and also runs all worker threads in the same share group out of the current function. All other threads in the control group run freely.</td>
</tr>
</tbody>
</table>

For additional information on the different kinds of stepping, see the *dstep* on page 142 command information.

**Examples**

```plaintext
f t ou

Runs the current TOI out of the current subroutine.

f p dout 3

Unwinds the process in the current focus out of the current subroutine to the routine three levels above it in the call stack.
```

**RELATED TOPICS**

- Executing to the Completion of a Function in the *TotalView User Guide*
Group > Out Command in the online Help
Process > Out Command in the online Help
Thread > Out Command in the online Help
**dprint**

Evaluates and displays information

**Format**

Prints the value of a variable

```bash
dprint [-nowait] [-slice "slice_expr"] variable
```

Prints the value of an expression

```bash
dprint [-nowait] [-slice "slice_expr"] [-stats [-data]] expression
```

**Arguments**

- **-nowait**
  
  Tells TotalView to evaluate the expression in the background. Use `TV::expr` to obtain the results, as they are not displayed.

- **-slice "slice_expr"**
  
  Defines an array slice—that is, a portion of the array—to print. If the programming language is C or C++, use a backslash (\) when you enter the array subscripts. For example, "\[100:110\]."

- **-stats**
  
  Displays statistical data about an array. When using this switch, the expression provided to `dprint` must resolve to an array. The `-slice` switch may be used with `-stats` to select a subset of values from the array to calculate statistics on.

- **-data**
  
  Returns the results of `dprint -stats` as data in the form of a Tcl nested associative array rather than as output to the console. See the description section for the structure of the array.

  **Note:** This switch can be used only in conjunction with the `-stats` switch.

- **variable**
  
  A variable whose value is displayed. The variable can be local to the current stack frame or it can be global. If the displayed variable is an array, you can qualify the variable's name with a slice that displays a portion of the array,

- **expression**
  
  A source-language expression to evaluate and print. Because `expression` must also conform to Tcl syntax, you must enclose it within quotation marks if it includes any blanks, and in braces `{}` if it includes brackets `[]`, dollar signs `($)`, quotation marks `"`), or other Tcl special characters.

**Description**

The `dprint` command evaluates and displays a variable or an expression. The CLI interprets the expression by looking up the values associated with each symbol and applying the operators. The result of an expression can be a scalar value or an aggregate (array, array slice, or structure).

If an event such as a `$stop`, SEGV, breakpoint occurs, the `dprint` command throws an exception that describes the event. The first exception subcode returned by `TV::errorCodes` is the `susp-eval-id` (a suspension-evaluation-ID). You can use this to manipulate suspended evaluations with the `dflush` and `TV::expr` commands. For example:
dfocus tdpid.dtdid TV::expr get susp-eval-id

NOTE >> If the expression calls a function, the focus must not specify more than one thread for each process.

If you use the -nowait option, TotalView evaluates the expression in the background. It also returns a susp-eval-id that you can use to obtain the results of the evaluation using TV::expr.

As the CLI displays data, it passes the data through a simple more processor that prompts you after it displays each screen of text. At this time, you can press the Enter key to tell the CLI to continue displaying information. Entering q stops printing.

Since the dprint command can generate a considerable amount of output, you might want to use the capture on page 25 command to save the output to a variable.

Structure output appears with one field printed per line; for example:

```
sbfo = {
    f3 = 0x03 (3)
    f4 = 0x04 (4)
    f5 = 0x05 (5)
    f20 = 0x000014 (20)
    f32 = 0x00000020 (32)
}
```

Arrays print in a similar manner; for example:

```
foo = {
    [0][0] = 0x00000000 (0)
    [0][1] = 0x00000004 (4)
    [1][0] = 0x00000001 (1)
    [1][1] = 0x00000005 (5)
    [2][0] = 0x00000002 (2)
    [2][1] = 0x00000006 (6)
    [3][0] = 0x00000003 (3)
    [3][1] = 0x00000007 (7)
}
```

You can append a slice to the variable's name to tell the CLI to display a portion of an array; for example:

```
d.1<> p -slice "[10:20]" random
random slice: (10:30) = {
    (10) = 0.479426
    (11) = 0.877583
    (12) = 0.564642
    (13) = 0.825336
    (14) = 0.644218
    (15) = 0.764842
    (16) = 0.717356
    (17) = 0.696707
```
The following is another way of specifying the same slice:

d.1<> set my_var \[10:20\]
d.1<> p -slice $my_var random
random slice:(10:30) = {

The following example illustrates the output from `dprint -stats` command:

d1.<> dprint -stats twod_array

Count: 2500
Zero Count: 1
Sum: 122500
Minimum: 0
Maximum: 98
Median: 49
Mean: 49
Standard Deviation: 20.4124145231932
First Quartile: 34
Third Quartile: 64
Lower Adjacent: 0
Upper Adjacent: 98

NaN Count: N/A
Infinity Count: N/A
Denormalized Count: N/A

Checksum: 41071

By adding the `-data` switch,

d1.<> dprint -stats -data twod_array

the statistics are returned in a Tcl nested associative array, which has the following structure:

```
{<dpid.dtid>
  {Count <value>
   ZeroCount <value>
   Sum <value>
   Minimum <value>
   Maximum <value>
   Median <value>
   Mean <value>
   StandardDeviation <value>
   FirstQuartile <value>}
  }
```
ThirdQuartile <value>
LowerAdjacent <value>
UpperAdjacent <value>
NaNCount <value>
InfinityCount <value>
DenormalizedCount <value>
Checksum <value>

} <dpid.dtid>
{
  ...
}

To access data for a single process/thread, use the following Tcl commands:
array set stats_data [dprint -stats -data <arrayexpression>]
array set stats $stats_data([lindex [array names stats_data] 0])
puts "Array Sum: $stats(Sum)"

The CLI evaluates the expression or variable in the context of each thread in the target focus. Thus, the overall format of dprint output is as follows:

first process or thread:
  expression result

second process or thread:
  expression result
...

last process or thread:
  expression result

TotalView lets you cast variables and cast a variable to an array. If you are casting a variable, the first array address is the address of the variable. For example, assume the following declaration:
float bint;
The following statement displays the variable as an array of one integer:
dprint {(int \[1\])bint:

If the expression is a pointer, the first addresses is the value of the pointer. Here is an array declaration:
float bing[2], *bp = bint;

TotalView assumes the first array address is the address of what bp is pointing to. So, the following command displays the array:
dprint {(int \[2\])bp}

You can also use the dprint command to obtain values for your computer’s registers. For example, on most architectures, $r1 is register 1. To obtain the contents of this register, type:
dprint \$r1
NOTE >> Do not use a $ when asking the dprint command to display your program’s variables.

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>dprint</td>
<td>Evaluates and displays information</td>
</tr>
</tbody>
</table>

Examples

dprint scalar_y
  Displays the values of variable scalar_y in all processes and threads in the current focus.
p argc
  Displays the value of argc.
p argv
  Displays the value of argv, along with the first string to which it points.
p {argv[argc-1]}
  Prints the value of argv[argc-1]. If the execution point is in main(), this is the last argument passed to main().
dfocus p1 dprint scalar_y
  Displays the values of variable scalar_y for the threads in process 1.
f 1.2 p arrayx
  Displays the values of the array arrayx for the second thread in process 1.
for {set i 0} {$i < 100} {incr i} {p argv[$i]}
  If main() is in the current scope, prints the program’s arguments followed by the program’s environment strings.
f {t1.1 t2.1 t3.1} dprint {f()}
  Evaluates a function contained in three threads. Each thread is in a different process:
  Thread 1.1:
  f(): 2 Thread 2.1:
  f(): 3 Thread 3.1:
  f(): 5
f {t1.1 t2.1 t3.1} dprint -nowait {f()}
  Evaluates a function without waiting. Later, you can obtain the results using TV::expr. The number displayed immediately after the command, which is “1”, is the susp-eval-id. The following example shows how to get this result:
  f t1.1 TV::expr get 1 result
  2
CLI Commands

dprint
123

f t2.1 TV::expr get 1 result
Thread 1.1:
f(): 2
Thread 2.1:
f(): 3
Thread 3.1:
f(): 5
3

f t3.1 TV::expr get 1 result
5

RELATED TOPICS

Examining and Editing Data and Program Elements in the TotalView for HPC User Guide
Examining Arrays in the TotalView for HPC User Guide
Evaluating Expressions in the TotalView for HPC User Guide
Tools > Evaluate Command in the online Help
TV::errorCodes Command
TV::expr Command
dptsets

Shows the status of processes and threads

Format

dptsets [ptset_array]...

Arguments

ptset_array

An optional array that indicates the P/T sets to show. An element of the array can be a number or it can be a more complicated P/T expression. (For more information, see “Using P/T Set Operators” in Chapter 21, “Group, Process, and Thread Control” of the TotalView for HPC User Guide.)

Description

The dptsets command shows the status of each process and thread in a Tcl array of P/T expressions. These array elements are P/T expressions (see “Using P/T Set Operators” in Chapter 21, “Group, Process, and Thread Control” of the TotalView for HPC User Guide, and the elements’ array indices are strings that label each element’s section in the output.

If you do not use the optional ptset_array argument, the CLI supplies a default array that contains all P/T set designators: error, existent, held, running, stopped, unheld, and watchpoint.

Examples

The following example displays information about processes and threads in the current focus:

d.1<> dptsets
unheld:
  1: 808694 Stopped [fork_loopSGI]
      1.1: 808694.1 Stopped PC=0x0d9cae64
      1.2: 808694.2 Stopped PC=0x0d9cae64
      1.3: 808694.3 Stopped PC=0x0d9cae64
      1.4: 808694.4 Stopped PC=0x0d9cae64

existent:
  1: 808694 Stopped [fork_loopSGI]
      1.1: 808694.1 Stopped PC=0x0d9cae64
      1.2: 808694.2 Stopped PC=0x0d9cae64
      1.3: 808694.3 Stopped PC=0x0d9cae64
      1.4: 808694.4 Stopped PC=0x0d9cae64

watchpoint:

running:

held:

error:
   stopped: 1: 808694 Stopped [fork_loopSGI]
1.1: 808694.1 Stopped PC=0x0d9cae64
1.2: 808694.2 Stopped PC=0x0d9cae64
1.3: 808694.3 Stopped PC=0x0d9cae64
1.4: 808694.4 Stopped PC=0x0d9cae64
...

The following example creates a two-element P/T set array, and then displays the results. Notice the labels in this example.

d1.<> set set_info(0) breakpoint(1)  
breakpoint(1)  
d1.<> set set_info(1) stopped(1)  
stopped(1)  
d1.<> dptsets set_info  
0:  
l: 892484 Breakpoint [arraySGI]  
  1.1: 892484.1 Breakpoint PC=0x10001544, [array.F#81]  

l:  
l: 892484 Breakpoint [arraySGI]  
  1.1: 892484.1 Breakpoint PC=0x10001544, [array.F#81]  

The array index to set_info becomes a label identifying the type of information being displayed. In contrast, the information within parentheses in the breakpoint and stopped functions identifies the arena for which the function returns information.

If you use a number as an array index, you might not remember what is being printed. The following very similar example shows a better way to use these array indices:

d1.<> set set_info(my_breakpoints) breakpoint(1)  
breakpoint(1)  
d1.<> set set_info(my_stopped) stopped(1)  
stopped(1)  
d1.<> dptsets set_info  
my_stopped:  
l: 882547 Breakpoint [arraysSGI]  
  1.1: 882547.1 Breakpoint PC=0x10001544, [arrays.F#81]  

my_breakpoints:  
l: 882547 Breakpoint [arraysSGI]  
  1.1: 882547.1 Breakpoint PC=0x10001544, [arrays.F#81]  

The following commands also create a two-element array. This example differs in that the second element is the difference between three P/T sets.

d.1< set mystat(system) a-gW  
d.1< set mystat(reallystopped)  
  stopped(a)-breakpoint(a)-watchpoint(a)  
d.1< dptsets t mystat  
system:  
Threads in process 1 [regress/fork_loop]:
1.-1:  21587.[-1] Running PC=0x3ff805c6998
1.-2:  21587.[-2] Running PC=0x3ff805c669c
...
Threads in process 2 [regress/fork_loop.1]:
2.-1:  15224.[-1] Stopped PC=0x3ff805c6998
2.-2:  15224.[-2] Stopped PC=0x3ff805c669c
...
reallystopped:
2.2  224.2 Stopped PC=0x3ff800d5758
2.-1  5224.[-1] Stopped PC=0x3ff805c6998
2.-2:  15224.[-2] Stopped PC=0x3ff805c669c
...
**drerun**

**Reverts processes**

**Format**

```
drerun [cmd_args] [in_operation]
    [out_operations]
    [error_operations]
```

**Arguments**

*cmd_args*

The arguments to be used for restarting a process.

*in_operation*

Names the file from which the CLI reads input.

`< infile`

Reads from *infile* instead of *stdin* *infile* indicates a file from which the launched process reads information.

*out_operations*

Names the file to which the CLI writes output. In the following, *outfile* indicates the file into which the launched processes writes information.

`> outfile`

Sends output to *outfile* instead of *stdout*.

`>& outfile`

Sends output and error messages to *outfile* instead of *stdout* and *stderr*.

`>>& outfile`

Appends output and error messages to *outfile*.

`>> outfile`

Appends output to *outfile*.

*error_operations*

Names the file to which the CLI writes error output. In the following, *errfile* indicates the file into which the launched processes writes error information.

`2> errfile`

Sends error messages to *errfile* instead of *stderr*.

`2>> errfile`

Appends error messages to *errfile*. 


Description

The `drerun` command restarts the process that is in the current focus set from its beginning. The `drerun` command uses the arguments stored in the `ARGS(dpmid)` and `ARGS_DEFAULT` variables. These are set every time you run the process with different arguments. Consequently, if you do not specify the arguments that the CLI uses when restarting the process, it uses the arguments you used when the CLI previously ran the process. (See `drun` on page 132 for more information.)

The `drerun` command differs from the `drun` command in that:

- If you do not specify an argument, the `drerun` command uses the default values. In contrast, the `drun` command clears the argument list for the program. This means that you cannot use an empty argument list with the `drerun` command to tell the CLI to restart a process and expect that it does not use any arguments.

- If the process already exists, the `drun` command does not restart it. (If you must use the `drun` command, you must first kill the process.) In contrast, the `drerun` command kills and then restarts the process.

The arguments to this command are similar to the arguments used in the Bourne shell.

Issues When Using Starter Programs

Starter programs such as `poe` or `aprun` and the CLI can interfere with one another because each believes that it owns `stdin`. Because the starter program is trying to manage `stdin` on behalf of your processes, it continually reads from `stdin`, acquiring all characters that it sees. This means that the CLI never sees these characters. If your target process does not use `stdin`, you can use the `-stdinmode none` option. Unfortunately, this option is incompatible with `poe -cmdfile` option that is used when specifying `-pgmmodel mpmd`.

If you encounter these problems, try redirecting `stdin` within the CLI; for example:

```
drun < in.txt
```

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rr</td>
<td><code>{drerun}</code></td>
<td>Restarts processes</td>
</tr>
</tbody>
</table>

Examples

`drerun`

Reruns the current process. Because it doesn't use arguments, the process restarts using its previous values.

`rr -firstArg an_argument -aSecondArg a_second_argument`

Reruns the current process. The CLI does not use the process's default arguments because replacement arguments exist.
RELATED TOPICS

- Starting Processes and Threads in the TotalView for HPC User Guide
- Command Arguments in the TotalView for HPC User Guide
- Process > Startup Parameters in the online Help
  - drun Command
  - dgo Command
  - capture Command
**drestart**

Restarts a checkpoint (IBM RS6000 machines only)

**Format**

Restarts a checkpoint on IBM AIX

```
drestart [ -halt ] [ -g gid ] [ -r host ] [ -no_same_hosts ]
```

**Arguments**

- **-halt**
  
  TotalView stops checkpointed processes after it restarts them.

- **-g gid**
  
  Names the control group into which TotalView places all created processes.

- **-r host**
  
  Names the remote host upon which the restart occurs.

- **-no_same_hosts**
  
  Restart can use any available hosts. If you do not use this option, the restart occurs on the same hosts upon which the program was executing when the checkpoint file was made. If these hosts are not available, the restart operation fails.

**Description**

The **drestart** command restores and restarts all of the checkpointed processes. The CLI attaches to the base process, and if there are parallel processes related to this base process, TotalView then attaches to them.

**Restarting using LoadLeveler**

If you checkpointed a LoadLeveler POE job, you cannot restart it with this command. You must resubmit the program as a LoadLeveler job to restart the checkpoint. You also need to set the **MP_POE_RESTART_SLEEP** environment variable to an appropriate number of seconds. After you restart POE, start TotalView and attach to POE. POE tells TotalView when it is time to attach to the parallel task so that it can complete the restart operation.

**NOTE >>** When attaching to POE, parallel tasks will not have been created yet, so you should avoid trying to attach to them. Therefore, use the **-no_attach_parallel** option when using the **dattach** command to attach to POE.

**Examples**

```
drestart
```

Restarts the checkpointed processes. The CLI automatically attaches to parallel processes.

```
drestart -halt -no_same_hosts
```

Restarts the checkpointed processes using available hosts. Stops checkpointed processes after restoring them.
RELATED TOPICS

dcalltree Command
Tools > Create Checkpoint Command in the online Help
Tools > Restart Checkpoint Command in the online Help
drun  

Starts or restarts processes

Format

```
drun [ cmd_arguments ][ in_operation infile ] [ out_operations outfile ] [ error_operations errfile ]
```

Arguments

**cmd_arguments**

The argument list passed to the process.

**in_operation**

Names the file from which the CLI reads input.

<infile

Reads from infile instead of stdin. infile indicates a file from which the launched process reads information.

**out_operations**

Names the file to which the CLI writes output. In the following, outfile indicates the file into which the launched processes writes information.

>outfile

Sends output to outfile instead of stdout.

>& outfile

Sends output and error messages to outfile instead of stdout and stderr.

>>>& outfile

Appends output and error messages to outfile.

>> outfile

Appends output to outfile.

**error_operations**

Names the file to which the CLI writes error output. In the following, errfile indicates the file into which the launched processes writes error information.

2>errfile

Sends error messages to errfile instead of stderr.

2>>errfile

Appends error messages to errfile.

Description

The drun command launches each process in the current focus and starts it running. The CLI passes the command arguments to the processes. You can also indicate I/O redirection for input and output information. Later in the session, you can use the drerun command to restart the program.
The arguments to this command are similar to the arguments used in the Bourne shell.

In addition, the CLI uses the following variables to hold the default argument list for each process:

**ARGS_DEFAULT**

The CLI sets this variable if you use the `-a` command-line option when you started the CLI or TotalView. (This option passes command-line arguments that TotalView uses when it invokes a process.) This variable holds the default arguments that TotalView passes to a process when the process has no default arguments of its own.

**ARGS(dpmid)**

An array variable that contains the command-line arguments. The index `dpid` is the process ID. This variable holds a process's default arguments. It is always set by the `drun` command, and it also contains any arguments you used when executing a `drerun` command.

If more than one process is launched with a single `drun` command, each receives the same command-line arguments.

In addition to setting these variables by using the `-a command-line option or specifying `cmd_arguments` when you use this or the `drerun` command, you can modify these variables directly with the `dset` and `dunset` commands.

You can only use this command to tell TotalView to execute initial processes, because TotalView cannot directly run processes that your program spawns. When you enter this command, the initial process must have terminated; if it was not terminated, you are told to kill it and retry. (You could, use the `drerun` command instead because the `drerun` commands first kills the process.)

The first time you use the `drun` command, TotalView copies arguments to program variables. It also sets up any requested I/O redirection. If you re-enter this command for processes that TotalView previously started—or use it when you use the `dattach` command to attach to a process—the CLI reinitializes your program.

**Issues When Using Starter Programs**

Starter programs such as `poe` or `aprun` and the CLI can interfere with one another because each believes that it owns `stdin`. Because the starter program is trying to manage `stdin` on behalf of your processes, it continually reads from `stdin`, acquiring all characters that it sees. This means that the CLI never sees these characters. If your target process does not use `stdin`, you can use the `stdinmode none` option. Unfortunately, this option is incompatible with the `poe -cmdfile` option that is used when specifying `-pgmmodel mpmd`.

If you encounter these problems, try redirecting `stdin` within the CLI; for example:

```
  drun < in.txt
```
# CLI Commands: `drun`

## Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td><code>drun</code></td>
<td>Starts or restarts processes</td>
</tr>
</tbody>
</table>

## Examples

```plaintext
drun
  Begins executing processes represented in the current focus.

f {p2 p3} drun
  Begins execution of processes 2 and 3.

f 4.2 r
  Begins execution of process 4. This is the same as `f 4 drun`.

dfocus a drun
  Restarts execution of all processes known to the CLI. If they were not previously killed, you are told to use the `dkill` command and then try again.

drun < in.txt
  Restarts execution of all processes in the current focus, setting them up to get standard input from `in.txt` file.
```

## RELATED TOPICS

- [Starting Processes and Threads](#) in the TotalView for HPC User Guide
- [Command Arguments](#) in the "Using the Command Line Interface (CLI)" chapter in the TotalView for HPC User Guide
- [Process > Startup Parameters](#) in the online Help
- [drerun Command](#)
- [dgo Command](#)
**dsession**

*Loads a session*

**Format**

Loads a session.

\[ \text{dsession [ -load session\_name ]} \]

**Arguments**

- **-load session\_name**
  
  Loads the session with the given *session\_name*.

**Description**

Loads a previously created session. The session attributes are applied to the TotalView process object created for the executable named in the session. Returns the TotalView ID for the new object as a string value. A *session\_name* that contains a space must be surrounded by quotes.

Sessions that attach to an existing process cannot be loaded this way; use the *dattach* command instead.

**RELATED TOPICS**

- Loading a Session Using the Sessions Manager in the *TotalView for HPC User Guide*
- Managing Sessions in the *TotalView for HPC User Guide*
- *dattach* Command
dset

Changes or views CLI variables

**Format**

Changes a CLI variable

```
dset debugger-var value
```

Views current CLI variables

```
dset [ debugger-var ]
```

Sets the default for a CLI variable

```
dset -set_as_default debugger-var value
```

**Arguments**

`debugger-var`

Name of a CLI variable.

`value`

Value to be assigned to `debugger-var`.

`-set_as_default`

Sets the value to use as the variable's default. This option is most often used by system administrators to set site-specific defaults in the global `.tvdrc` startup script. Values set using this option replace the CLI built-in default.

**Description**

The `dset` command sets the value of CLI debugger variables. CLI and TotalView variables are described in Chapter 5, “TotalView Variables,” on page 252.

If you use the `dset` command with no arguments, the CLI displays the names and current values for all CLI variables in the global namespace. If you use only one argument, the CLI returns and displays that variable's value.

The second argument defines the value that replaces a variable's previous value. You must enclose it in quotation marks if it contains more than one word.

If you do not use an argument, the CLI only displays variables in the current namespace. To show all variables in a namespace, enter the namespace name immediately followed by a double colon; for example, `TV::`

You can use an asterisk (`*`) as a wildcard character to tell the CLI to match more than one string; for example, `TV::g*` matches all variables in the `TV::` namespace beginning with `g`. For example, to view all variables in the `TV::` namespace, enter the following:

```
dset TV::
```

or:

```
dset TV::GUI::
```
You need to type the double colons at the end of this example when obtaining listings for a namespace. Without them, Tcl assumes that you are requesting information on a variable. For example, `dset TV::GUI` looks for a variable named GUI in the TV namespace.

**Using -set_as_default**

When you press a default button within a **File > Preferences** dialog box, TotalView reinitializes some settings to their original values. However, what happens if you set a value in your tvdrc file when you press a default button? In this case, setting a variable doesn't change what TotalView thinks the default is, so it still changes the setting back to its defaults.

The next time you invoke TotalView, TotalView will again use the value in your tvdrc.

You can tell TotalView that the value set in your tvdrc file is the default if you use the **-set_as_default** option. Now when you press a default button, it will use your value instead of its own.

If your TotalView administrator sets up a global .tvdrc file, TotalView reads values from that file and merges them with your preferences and other settings. If the value in the .tvdrc file changes, TotalView ignores the change because it has already set a value in your local preferences file. If the administrator uses the **-set_as_default** option, you can be told to press the default button to get the changes. If, however, the administrator doesn't use this option, the only way to get changes is by deleting your preferences file.

**Examples**

```
dset PROMPT "Fixme% 

Sets the prompt to **Fixme%** followed by a space.

dset *

Displays all CLI variables and their current settings.

dset VERBOSE

Displays the current setting for output verbosity.

dset EXECUTABLE_PATH ../_test_dir;$_EXECUTABLE_PATH

Places ../_test_dir at the beginning of the previous value for the executable path.

dset -set_as_default TV::server_launch_string \
   {/use/this/one/tvdsrv}

Sets the default value of the **TV::server_launch_string**. If you change this value, you can later select the **Defaults** button within the **File > Preferences** Launch String page to reset it to its original value.

dset TV::GUI::fixed_font_size 12

Sets the TotalView GUI to display information using a 12-point, fixed-width font. Commands such as this are often found in a startup file.
```
RELATED TOPICS

dlappend Command
**dstatus**

Shows current status of processes and threads

**Format**

dstatus

dstatus [ -g ]
dstatus [ -group_by process_state | replay | pheld | thread_state | pc, | function | line | apid | theld | stop_reason ]

**Arguments**

-g
   Alias for -group_by.

-group_by
   Reduces the display based on the following process-level or thread-level arguments. The reduction is shown using either a compressed process list for process-level properties (plist) or a compressed thread list for thread-level properties (ptlist). See Compressed List Syntax (ptlist) for a description of a ptlist.

**Process level:**

process_state
   Limits the display to the state of the process.

replay
   Groups by replay mode. A process can be in three replay states: Replay Unavailable, Replay, or -Record.

pheld
   Groups the processes as either Held or UnHeld.

**Thread level:**

thread_state
   The state of the thread

pc
   The Program Counter of the thread

function
   The function where the thread's pc is currently.

line
   The line number for the current thread's pc

apid
   The action point identifier that the thread's pc is on. If the thread is not at an action point, it will be grouped as ap (none).

theld
   Threads grouped as either Held or UnHeld.
stop_reason
The stop code and stop message for a stopped thread.

-pcount
Alias for the -ptlist_element_count argument

-ptlist_element_count number
Displays, at maximum, number elements (comma separated plists or ptlists) in the process/thread compressed list that is shown in a reduced dstatus display. If a reduction results in exceeding the ptlist_element_count, an ellipsis is appended. For instance, if ptlist_element_count is set to 5:

[p1-4.1, p2.2, p3-4.3, p5.4, p6.1-2, ...]

To change the default value, use the TotalView State variable ptlist_element_threshold. For example:

dset TV::ptlist_element_threshold 10

-levels
The number of levels to show for a set of properties. If no levels are specified, then each property is reduced on a new line with indentation. If the number of levels is less than the number of specified properties, then the remaining properties are shown in a single reduction on one line.

-v
Show verbose output in the reduced display. Without -v, full paths of filenames and line numbers are not displayed.

-detail
Force full detailed information for the current state of each process and thread in the current focus. This option affects the amount of information displayed from grouping by function.

Description
With the -group_by option, the dstatus command displays an aggregated view of the process and thread state in the current focus. To make the display more useful, you can reduce it based on specific properties, provided as arguments as described above. The full detail shows the current state of each process and thread in the current focus. ST is aliased to dfocus g dstatus and acts as a group-status command. Type help ptset for more information.

If you have not changed the focus, the default is process. In this case, the dstatus command shows the status for each thread in process 1. In contrast, if you set the focus to g1.<, the CLI displays the status for every thread in the control group. When you limit thread state display by certain properties, the output is displayed as a compressed thread list, or ptlist.

Compressed List Syntax (ptlist)
A compressed ptlist consists of a process and thread count, followed by square-bracket-enclosed list of process and thread ranges separated by dot (.). If the thread range is missing, it’s merely a compressed list of processes and it is referred to as a plist.

If the process range starts with the letter p, the process IDs are TotalView DPIDs (debugger unique process identifiers); otherwise, they are the MPI rank for the process, MPI_COMM_WORLD.
The thread IDs are always TotalView DTIDs (debugger unique thread identifiers). For example, the compressed `ptlist 5:13[0-3.1-3, p1.1]` indicates that there are five processes and 13 threads in the list. The process and thread range `0-3.1-3` indicates MPI rank processes 0 through 3, each with DTIDs 1 through 3. The process range `p1.1` indicates process DPID 1 and thread DTID 1, normally the MPI starter process named `mpirun`.

### Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>st</td>
<td><code>dstatus</code></td>
<td>Shows current status</td>
</tr>
<tr>
<td>ST</td>
<td><code>{dfocus g dstatus}</code></td>
<td>Shows group status</td>
</tr>
</tbody>
</table>

### Examples

**dstatus**

Displays the status of all processes and threads in the current focus; for example:

```
1:     42898    Breakpoint  [arraysAIX]
  1.1: 42898.1  Breakpoint  
    PC=0x100006a0, [./arrays.F#87]
```

**f a st**

Displays the status for all threads in all processes.

**f p1 st**

Displays the status of the threads associated with process 1. If the focus is at its default (d1.<), this is the same as typing `st`.

**ST**

Displays the status of all processes and threads in the control group having the focus process; for example:

```
1:     773686    Stopped  [fork_loop_64]
  1.1:773686.1  Stopped  PC=0x0d9cae64
  1.2:773686.2  Stopped  PC=0x0d9cae64
  1.3:773686.3  Stopped  PC=0x0d9cae64
  1.4:773686.4  Stopped  PC=0x0d9cae6
  2:     779490  Stopped  [fork_loop_64.1]
  2.1:779490.1  Stopped  PC=0x0d9cae64
  2.2:779490.2  Stopped  PC=0x0d9cae64
  2.3:779490.3  Stopped  PC=0x0d9cae64
  2.4:779490.4  Stopped  PC=0x0d9cae64
```

**f W st**

Shows status for all worker threads in the focus set. If the focus is set to d1.<, the CLI shows the status of each worker thread in process 1.

**f W ST**
Shows status for all worker threads in the control group associated with the current focus.

In this case, TotalView merges the \texttt{W} and \texttt{g} specifiers in the \texttt{ST} alias. The result is the same as if you had entered \texttt{f gW st}.

\texttt{f L ST}

Shows status for every thread in the share group that is at the same PC as the \textit{thread of interest} (TOI).

\texttt{d1.<> dfocus g dstatus -group\_by thread\_state, function}

First reduces the focus by \texttt{thread\_state}, then further breaks down and reduces the results according to the function the threads are in within each thread state. This call might output this reduced display:

\textbf{Focus: 4:20 [p1-4.1-5]}
- \texttt{Breakpoint : 4:4[p1.2, p3-4.2, p2.3]}
- \texttt{snore : 4:4[p1.2, p3-4.2, p2.3]}
- \texttt{Stopped : 4:16[p1-4.1, p2.2, p1.3, p3-4.3, p1-4.4-5]}
  - \texttt{___newselect_nocancel : 4:13[p1-4.1, p2.2, p3-4.3, p1.4]}
  - \texttt{snore : 2:3[p1.3, p2.4-5]}

The above output displays the reduction produced by the \texttt{group\_by} command as a series of \texttt{ptlists}. (See above, \textit{Compressed List Syntax (ptlist)}).

\texttt{dfocus group dwhere -group\_by function}

This \texttt{dwhere} call output shows that all the processes have the first three frames in their backtrace but then they diverge and one process is in function \texttt{rank0} while the other three processes are in \texttt{rankn}.

\texttt{+/ : 10:10[0-9.1]}
\texttt{+_start}
\texttt{+__libc_start_main}
\texttt{+main}
  - \texttt{+rank0 : 1:1[0.1]}
  - \texttt{+rankn : 3:3[1.1, 5.1, 8.1]}

\section*{RELATED TOPICS}

- \textbf{Using the Root Window} in the \textit{TotalView for HPC User Guide}
- \textbf{Viewing Process and Thread State} in the \textit{TotalView for HPC User Guide}
- \textbf{The Root Window} in the online Help
- \texttt{dwhat Command}
- \texttt{dwhere Command}
dstep

Steps lines, stepping into subfunctions

Format

dstep [ -back ] [ num-steps ]

Arguments

- back

(ReplayEngine only) Steps to the previous source line, moving into subroutines that called the current function. This option can be abbreviated to -b.

num-steps

An integer greater than 0, indicating the number of source lines to execute.

Description

The dstep command executes source lines; that is, it advances the program by steps (source lines). If a statement in a source line invokes a subfunction, the dstep command steps into the function.

The optional num-steps argument defines the number of dstep operations to perform. If you do not specify num-steps, the default is 1.

The dstep command iterates over the arenas in the focus set by doing a thread-level, process-level, or group-level step in each arena, depending on the width of the arena. The default width is process (p).

If the width is process, the dstep command affects the entire process that contains the thread being stepped. Thus, although the CLI is only stepping one thread, all other threads in the same process also resume executing. In contrast, the dfocus t dstep command steps only the thread of interest (TOI).

NOTE >> On systems having identifiable manager threads, the dfocus t dstep command allows the manager threads as well as the TOI to run.

The action taken on each term in the focus list depends on whether its width is thread, process, or group, and on the group specified in the current focus. (If you do not explicitly specify a group, the default is the control group.)

If some thread hits an action point other than the goal breakpoint during a step operation, that ends the step.

Group Width

The behavior depends on the group specified in the arena:

Process group

TotalView examines that group and identifies each process having a thread stopped at the same location as the TOI. TotalView selects one matching thread from each matching process. TotalView then runs all processes in the group and waits until the TOI arrives at its goal location; each selected thread also arrives there.
Thread group

The behavior is similar to process width behavior except that all processes in the program control group run, rather than just the process of interest (POI). Regardless of which threads are in the group of interest, TotalView only waits for threads that are in the same share group as the TOI. This is because it is not useful to run threads executing in different images to the same goal.

Process Width (default)

The behavior depends on the group specified in the arena. Process width is the default.

Process group

TotalView allows the entire process to run, and execution continues until the TOI arrives at its goal location. TotalView plants a temporary breakpoint at the goal location while this command executes. If another thread reaches this goal breakpoint first, your program continues to execute until the TOI reaches the goal.

Thread group

TotalView runs all threads in the process that are in that group to the same goal as the TOI. If a thread arrives at the goal that is not in the group of interest, this thread also stops there. The group of interest specifies the set of threads for which TotalView waits. This means that the command does not complete until all threads in the group of interest are at the goal.

Thread Width

Only the TOI is allowed to run. (This is not supported on all systems.)
Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>dstep</td>
<td>Runs the TOI one statement, while allowing other threads in the process to run.</td>
</tr>
<tr>
<td>S</td>
<td>{dfocus g dstep}</td>
<td>Searches for threads in the share group that are at the same PC as the TOI, and steps one such aligned thread in each member one statement. The rest of the control group runs freely. This is a group stepping command.</td>
</tr>
<tr>
<td>sl</td>
<td>{dfocus L dstep}</td>
<td>Steps the process threads in lockstep. This steps the TOI one statement, and runs all threads in the process that are at the same PC as the TOI to the same (goal) statement. Other threads in the process run freely. The group of threads that is at the same PC is called the lockstep group. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>SL</td>
<td>{dfocus gL dstep}</td>
<td>Steps lockstep threads in the group. This steps all threads in the share group that are at the same PC as the TOI one statement. Other threads in the control group run freely.</td>
</tr>
<tr>
<td>sw</td>
<td>{dfocus W dstep}</td>
<td>Steps worker threads in the process. This steps the TOI one statement, and runs all worker threads in the process to the same (goal) statement. The nonworker threads in the process run freely. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>SW</td>
<td>{dfocus gW dstep}</td>
<td>Steps worker threads in the group. This steps the TOI one statement, and runs all worker threads in the same share group to the same (goal) statement. All other threads in the control group run freely.</td>
</tr>
</tbody>
</table>

Examples

dstep

Executes the next source line, stepping into any procedure call it encounters. Although the CLI only steps the current thread, other threads in the process run.

s 15

Executes the next 15 source lines.

f p1.2 dstep

Steps thread 2 in process 1 by one source line. This also resumes execution of all threads in process 1; they halt as soon as thread 2 in process 1 executes its statement.

f t1.2 s

Steps thread 2 in process 1 by one source line. No other threads in process 1 execute.
RELATED TOPICS

- **Creating a Process by Single Stepping** in the *TotalView for HPC User Guide*
- **Stepping and Setting Breakpoints** in the *TotalView for HPC User Guide*
- **Using Stepping Commands** in the *TotalView for HPC User Guide*
- **Group > Step Command** in the online Help
- **Process > Step Command** in the online Help
- **Thread > Step Command** in the online Help

- `dstepi` Command
- `dnext` Command
- `dfocus` Command
**dstepi**

Steps machine instructions, stepping into subfunctions

**Format**

dstepi [-back] [num-steps]

**Arguments**

- **-back**
  
  (ReplayEngine only) Steps to the previous instruction, moving into subroutines that called the current function. This option can be abbreviated to -b.

- **num-steps**
  
  An integer greater than 0, indicating the number of instructions to execute.

**Description**

The **dstepi** command executes assembler instruction lines; that is, it advances the program by single instructions.

The optional **num-steps** argument defines the number of **dstepi** operations to perform. If you do not specify **num-steps**, the default is 1.

For more information, see **dstep** on page 142.
# Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>si</td>
<td>dstepi</td>
<td>Runs the <em>thread of interest</em> (TOI) one instruction while allowing other threads in the process to run.</td>
</tr>
<tr>
<td>SI</td>
<td>{dfocus g dstepi}</td>
<td>Searches for threads in the share group that are at the same PC as the TOI, and steps one such aligned thread in each member one instruction. The rest of the control group runs freely. This is a group stepping command.</td>
</tr>
<tr>
<td>sil</td>
<td>{dfocus L dstepi}</td>
<td>Steps the process threads in lockstep. This steps the TOI one instruction, and runs all threads in the process that are at the same PC as the TOI to the same instruction. Other threads in the process run freely. The group of threads that is at the same PC is called the lockstep group. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>SIL</td>
<td>{dfocus gL dstepi}</td>
<td>Steps lockstep threads in the group. This steps all threads in the share group that are at the same PC as the TOI one instruction. Other threads in the control group run freely.</td>
</tr>
<tr>
<td>siw</td>
<td>{dfocus W dstepi}</td>
<td>Steps worker threads in the process. This steps the TOI one instruction, and runs all worker threads in the process to the same (goal) statement. The nonworker threads in the process run freely. This alias does not force process width. If the default focus is set to group, this steps the group.</td>
</tr>
<tr>
<td>SIW</td>
<td>{dfocus gW dstepi}</td>
<td>Steps worker threads in the group. This steps the TOI one instruction, and runs all worker threads in the same share group to the same statement. All other threads in the control group run freely.</td>
</tr>
</tbody>
</table>

## Examples

**dstepi**

Executes the next machine instruction, stepping into any procedure call it encounters. Although the CLI only steps the current thread, other threads in the process run.

**si 15**

Executes the next 15 instructions.

**f p1.2 dstepi**

Steps thread 2 in process 1 by one instruction, and resumes execution of all other threads in process 1; they halt as soon as thread 2 in process 1 executes its instruction.

**f t1.2 si**

Steps thread 2 in process 1 by one instruction. No other threads in process 1 execute.
RELATED TOPICS

- Creating a Process by Single Stepping in the TotalView for HPC User Guide
- Stepping and Setting Breakpoints in the TotalView for HPC User Guide
- Using Stepping Commands in the TotalView for HPC User Guide
- Group > Step Instruction Command in the online Help
- Process > Step Instruction Command in the online Help
- Thread > Step Instruction Command in the online Help
- dstep Command
- dnext Command
- dfocus Command
**dunhold**

Releases a held process or thread

**Format**

Releases a process

```
dunhold -process
```

Releases a thread

```
dunhold -thread
```

**Arguments**

- **-process**
  
  Releases processes in the current focus. You can abbreviate the `-process` option argument to `-p`.

- **-thread**
  
  Releases threads in the current focus. You can abbreviate the `-thread` option to `-t`.

**Description**

The `dunhold` command releases the threads or processes in the current focus. You cannot hold or release system manager threads.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uhp</td>
<td><code>{dfocus p dunhold -process}</code></td>
<td>Releases the process of interest (POI)</td>
</tr>
<tr>
<td>UHP</td>
<td><code>{dfocus g dunhold -process}</code></td>
<td>Releases the processes in the focus group</td>
</tr>
<tr>
<td>uht</td>
<td><code>{dfocus t dunhold -thread}</code></td>
<td>Releases the thread of interest (TOI)</td>
</tr>
<tr>
<td>UHT</td>
<td><code>{dfocus g dunhold -thread}</code></td>
<td>Releases all threads in the focus group</td>
</tr>
<tr>
<td>uhtp</td>
<td><code>{dfocus p dunhold -thread}</code></td>
<td>Releases the threads in the current -process</td>
</tr>
</tbody>
</table>

**Examples**

```
f w uhtp
```

Releases all worker threads in the focus process.

```
htp; uht
```

Holds all threads in the focus process except the TOI.

**RELATED TOPICS**

- [Holding and Releasing Processes and Threads](#) in the *TotalView for HPC User Guide*
- [Starting Processes and Threads](#) in the *TotalView for HPC User Guide*
- Group > Release Command in the online Help
Process > Release Threads Command in the online Help
Thread > Hold Command in the online Help
dhold Command
dunset

Restores default settings for variables

Format
Restores a CLI variable to its default value

    dunset debugger-var

Restores all CLI variables to their default values

    dunset -all

Arguments

    debugger-var
        Name of the CLI variable whose default setting is being restored.

    -all
        Restores the default settings of all CLI variables.

Description

The dunset command reverses the effects of any previous dset commands, restoring CLI variables to their default settings. See Chapter 5, “TotalView Variables,” on page 252 for information on these variables.

Tcl variables (those created with the Tcl set command) are not affected by this command.

If you use the -all option, the dunset command affects all changed CLI variables, restoring them to the settings that existed when the CLI session began. Similarly, specifying debugger-var restores that one variable.

Examples

    dunset PROMPT
        Restores the prompt string to its default setting; that is, {[dfocus]>}.

    dunset -all
        Restores all CLI variables to their default settings.

RELATED TOPICS

    dset Command
**duntil**

Runs the process until a target place is reached

**Format**

Runs to a line

```
duntil [ -back ] line-number
```

Runs to an address

```
duntil [ -back ] -address addr
```

Runs into a function

```
duntil [ -back ] proc-name
```

**Arguments**

- **-back**
  
  (ReplayEngine only) Steps to the previous instruction, moving into subroutines that called the current function. This option can be abbreviated to **-b**.

  **line-number**
  
  A line number in your program.

- **-address addr**
  
  An address in your program.

- **proc-name**
  
  The name of a procedure, function, or subroutine in your program.

**Description**

The **duntil** command runs the thread of interest (TOI) until execution reaches a line or absolute address, or until it enters a function.

If you use a process or group width, all threads in the process or group not running to the goal are allowed to run. If a secondary thread arrives at the goal before the TOI, the thread continues running, ignoring this goal. In contrast, if you specify thread width, only the TOI runs.

The **duntil** command differs from other step commands when you apply it to a group, as follows:

**Process group**

Runs the entire group, and the CLI waits until all processes in the group contain at least one thread that has arrived at the goal breakpoint. This lets you **sync** all the processes in a group in preparation for group-stepping them.

**Thread group**

Runs the process (for **p** width) or the control group (for **g** width) and waits until all the running threads in the group of interest arrive at the goal.

There are some differences in the way processes and threads run using the **duntil** command and other stepping commands:
• **Process Group Operation**: TotalView examines the TOI to see if it is already at the goal. If it is, TotalView does not run the POI. Similarly, TotalView examines all other processes in the share group, and runs only processes without a thread at the goal. It also runs members of the control group not in the share group.

• **Group-Width Thread Group Operation**: TotalView identifies all threads in the entire control group that are not at the goal. Only those threads run. Although TotalView runs share group members in which all worker threads are already at the goal, it does not run the workers. TotalView also runs processes in the control group outside the share group. The duntil command operation ends when all members of the focus thread group are at the goal.

• **Process-Width Thread Group Operation**: TotalView identifies all threads in the entire focus process not already at the goal. Only those threads run. The duntil command operation ends when all threads in the process that are also members of the focus group arrive at the goal.

### Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>un</td>
<td>duntil</td>
<td>Runs the TOI until it reaches a target, while allowing other threads in the process to run.</td>
</tr>
<tr>
<td>UN</td>
<td>{dfocus g duntil}</td>
<td>Runs the entire control group until every process in the share group has at least one thread at the goal. Processes that have a thread at the goal do not run.</td>
</tr>
<tr>
<td>unl</td>
<td>{dfocus L duntil}</td>
<td>Runs the TOI until it reaches the target, and runs all threads in the process that are at the same PC as the TOI to the same target. Other threads in the process run freely. The group of threads that is at the same PC is called the lockstep group. This does not force process width. If the default focus is set to group, this runs the group.</td>
</tr>
<tr>
<td>UNL</td>
<td>{dfocus gL duntil}</td>
<td>Runs lockstep threads in the share group until they reach the target. Other threads in the control group run freely.</td>
</tr>
<tr>
<td>unw</td>
<td>{dfocus W duntil}</td>
<td>Runs worker threads in the process to a target. The nonworker threads in the process run freely. This does not force process width. If the default focus is set to group, this runs the group.</td>
</tr>
<tr>
<td>UNW</td>
<td>{dfocus gW duntil}</td>
<td>Runs worker threads in the same share group to a target. All other threads in the control group run freely.</td>
</tr>
</tbody>
</table>

### Examples

**UNW 580**

Runs all worker threads to line 580.

**un buggy_subr**

Runs to the start of the **buggy_subr** routine.
RELATED TOPICS

Executing to a Selected Line in the TotalView for HPC User Guide
Using Groups, Processes, and Threads in the TotalView for HPC User Guide
Using Run To and duntil Commands in the TotalView for HPC User Guide
Group > Run To Command in the online Help
Process > Run to Command in the online Help
Thread > Run To Command in the online Help
**CLI Commands**

## dup

Moves up the call stack

**Format**

```latex
dup [ num-levels ]
```

**Arguments**

`num-levels`

Number of levels to move up. The default is 1.

**Description**

The `dup` command moves the current stack frame up one or more levels. It also prints the new frame number and function.

Call stack movements are all relative, so `dup` effectively “moves up” in the call stack. (“Up” is in the direction of `main()`.)

Frame 0 is the most recent—that is, currently executing—frame in the call stack; frame 1 corresponds to the procedure that invoked the currently executing frame, and so on. The call stack’s depth is increased by one each time a program enters a procedure, and decreases by one when the program exits from it. The effect of the `dup` command is to change the context of commands that follow. For example, moving up one level allows access to variables that are local to the procedure that called the current routine.

Each `dup` command updates the frame location by adding the appropriate number of levels.

The `dup` command also modifies the current list location to be the current execution location for the new frame, so a subsequent `dlist` command displays the code surrounding this location. Entering the `dup` 2 command (while in frame 0) followed by a `dlist` command, for instance, displays source lines centered around the location from which the current routine’s parent was invoked. These lines are in frame 2.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td><code>dup</code></td>
<td>Moves up the call stack</td>
</tr>
</tbody>
</table>

**Examples**

```text
dup
```

Moves up one level in the call stack. As a result, subsequent `dlist` commands refer to the procedure that invoked this one. After this command executes, it displays information about the new frame; for example:

```text
1 check_fortran_arrays_ PC=0x10001254, 
FP=0x7fff2ed0 [arrays.F#48]
```

d`focus p1 u 5

---

[Image: RogueWave Logo]
Moves up five levels in the call stack for each thread involved in process 1. If fewer than five levels exist, the CLI moves up as far as it can.

RELATED TOPICS

ddown Command
**dwait**

Blocks command input until the target processes stop

**Format**

dwait

**Arguments**

This command has no arguments

**Description**

The `dwait` command waits for all threads in the current focus to stop or exit. Generally, this command treats the focus the same as other CLI commands.

If you interrupt this command—typically by entering Ctrl+C—the CLI manually stops all processes in the current focus before it returns.

Unlike most other CLI commands, this command blocks additional CLI input until the blocking action is complete.

**Examples**

*dwait*

Blocks further command input until all processes in the current focus have stopped (that is, none of their threads are still running).

*dfocus {p1 p2} dwait*

Blocks command input until processes 1 and 2 stop.
**dwatch**

Defines a watchpoint

**Format**

Defines a watchpoint for a variable

```
dwatch variable [-length byte-count ] [ -g | -p | -t ] [ [ [ -l lang ] -e expr ] [ -t type ]
```

Defines a watchpoint for an address

```
dwatch -address addr -length byte-count [ -g | -p | -t ] [ [ -l lang ] -e expr ] [ -t type ]
```

**Arguments**

`variable`

A symbol name corresponding to a scalar or aggregate identifier, an element of an aggregate, or a dereferenced pointer.

`-address addr`

An absolute address in the file.

`-length byte-count`

The number of bytes to watch. If you enter a variable, the default is the variable's byte length.

If you are watching a variable, you need to specify only the amount of storage to watch if you want to override the default value.

`-g`

Stops all processes in the process's control group when the watchpoint triggers.

`-p`

Stops the process that hit this watchpoint.

`-t`

Stops the thread that hit this watchpoint.

`-l lang`

Specifies the language in which you are writing an expression. The values you can use for `lang` are **c**, **c++**, **f7**, **f9**, and **asm**, for C, C++, FORTRAN 77, Fortran-9x, and assembler, respectively. If you do not use a language code, TotalView picks one based on the variable's type. If you specify only an address, TotalView uses the C language.

Not all languages are supported on all systems.

`-e expr`

When the watchpoint is triggered, evaluates `expr` in the context of the thread that hit the watchpoint. In most cases, you need to enclose the expression in braces `{ }`.

`-t type`

The data type of `$oldval/$newval` in the expression. If you do not use this option, TotalView uses the variable's datatype. If you specify an address and you also use an expression, you must use this option.
Description

The `dwatch` command defines a watchpoint on a memory location where the specified variables are stored. The watchpoint triggers whenever the value of the variable changes. The CLI returns the ID of the newly created watchpoint.

**NOTE >>** Watchpoints are not available on Macintosh computers running OS X, and IBM PowerPC computers running Linux Power.

The value set in the STOP_ALL variable indicates which processes and threads stop executing.

The watched variable can be a scalar, array, record, or structure object, or a reference to a particular element in an array, record, or structure. It can also be a dereferenced pointer variable.

To obtain a variable’s address if your application demands that you specify a watchpoint with an address instead of a variable name:

- `dprint &variable`
- `dwhat variable`

The `dprint` command displays an error message if the variable is in a register.

See Chapter 8, “Using Watchpoints” in the *TotalView for HPC User Guide* for additional information on watchpoints.

If you do not use the `-length` option, the CLI uses the length attribute from the program’s symbol table. This means that the watchpoint applies to the data object named; that is, specifying the name of an array lets you watch all elements of the array. Alternatively, you can watch a certain number of bytes, starting at the named location.

**NOTE >>** In all cases, the CLI watches addresses. If you specify a variable as the target of a watchpoint, the CLI resolves the variable to an absolute address. If you are watching a local stack variable, the position being watched is just where the variable happened to be when space for the variable was allocated.

The focus establishes the processes (not individual threads) for which the watchpoint is in effect.

The CLI prints a message showing the action point identifier, the location being watched, the current execution location of the triggering thread, and the identifier of the triggering threads.

One possibly confusing aspect of using expressions is that their syntax differs from that of Tcl. This is because you need to embed code written in Fortran, C, or assembler within Tcl commands. In addition, your expressions often include TotalView built-in functions.
### Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wa</td>
<td>dwatch</td>
<td>Defines a watchpoint</td>
</tr>
</tbody>
</table>

### Examples

For these examples, assume that the current process set at the time of the `dwatch` command consists only of process 2, and that `ptr` is a global variable that is a pointer.

**dwatch *ptr**

Watches the address stored in pointer `ptr` at the time the watchpoint is defined, for changes made by process 2. Only process 2 is stopped. The watchpoint location does not change when the value of `ptr` changes.

**dwatch {*ptr}**

Performs the same action as the previous example. Because the argument to the `dwatch` command contains a space, Tcl requires you to place the argument within braces.

**dfocus {p2 p3} wa *ptr**

Watches the address pointed to by `ptr` in processes 2 and 3. Because this example does not contain either a `-p` or `-g` option, the value of the STOP_ALL variable lets the CLI know if it should stop processes or groups.

**dfocus {p2 p3 p4} dwatch -p *ptr**

Watches the address pointed to by `ptr` in processes 2, 3, and 4. The `-p` option indicates that TotalView only stops the process triggering the watchpoint.

**wa * aString -length 30 -e {goto $447}**

Watches 30 bytes of data beginning at the location pointed to by `aString`. If any of these bytes change, execution control transfers to line 447.

**wa my_vbl -type long -e {if ($newval == 0x11ffff38) $stop;]**

Watches the `my_vbl` variable and triggers when `0x11ffff38` is stored in it.

**wa my_vbl -e {if (my_vbl == 0x11ffff38) $stop;]**

Performs the same function as the previous example. This example tests the variable directly rather than by using the `$newval` variable.

### RELATED TOPICS

- **Using Watchpoints** in the TotalView for HPC User Guide
- **Writing Code Fragments** in the TotalView for HPC User Guide
- **Tools > Watchpoint Command** in the online Help
- **dactions Command**
dwhat

Determine what a name refers to

Format

dwhat symbol-name

Arguments

symbol-name

Fully or partially qualified name specifying a variable, procedure, or other source code symbol.

Description

The dwhat command displays the name and description of a named entity in a program.

NOTE >> To view information on CLI variables or aliases, use the dset or alias commands.

The focus constrains the query to a particular context.

The default width for this command is thread (t).

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wh</td>
<td>dwhat</td>
<td>Determines what a name refers to</td>
</tr>
</tbody>
</table>

Examples

The following examples the CLI display for various commands.

dprint timeout

timeout = {
    tv_sec = 0xc0089540 (-1073179328)
    tv_usec = 0x000003ff (1023)
}

dwhat timeout

In thread 1.1:

Name: timeout; Type: struct timeval; Size: 8 bytes; Addr: 0x11fffe5c0
Scope: #fork_loop.cxx#snore 
    (Scope class: Any) Address class: auto_var 
    (Local variable)

wh timeval

In process 1: Type name: struct timeval; Size: 8 bytes; 
Category: Structure
Fields in type:
    { tv_sectime_t(32 bits)
tv_usecint(32 bits)


dlist
20    float field3_float;
21    double field3_double;
22    en_check en1;
23
24    }
25
26 main ()
27 {
28    en_check vbl;
29    check_struct s_vbl;
30    vbl = big;
31    s_vbl.field2_char = 3;
32    return (vbl + s_vbl.field2_char);
33 }

p vbl
vbl = big (0)

wh vbl

In thread 2.3:
Name: vbl; Type: enum en_check; \
Size: 4 bytes; Addr: Register 01
Scope: check_structs.cxx#main \
(Scope class: Any)
Address class: register_var (Register \ variable)

wh en_check

In process 2:
Type name: enum en_check; Size: 4 bytes; \
Category: Enumeration
Enumerated values:
   big    = 0
   little = 1
   fat    = 2
   thin   = 3

p s_vbl
s_vbl = { field1_int = 0x800164dc (-2147392292) field2_char = '\377' (0xff, or -1) field2_chars = "\003" <padding> = "\000" (0x00, or 0) field3_int = 0xc0006140 (-1073716928) field2_uchar = '\377' (0xff, or 255) <padding> = "\003" (0x03, or 3) <padding> = "\000" (0x00, or 0) <padding> = "\000" (0x00, or 0)
field_sub = {
    field1_int = 0xc0002980 (-1073731200)
    <padding> = '\377' (0xff, or -1)
    <padding> = '\003' (0x03, or 3)
    <padding> = '\000' (0x00, or 0)
    field2_long = 0x0000000000000000 (0)
    ...
}

wh s_vbl

In thread 2.3

Name: s_vbl; Type: struct check_struct; 
    Size: 80 bytes; Addr: 0x11ffff240
    Scope: #check_structs.cxx#main 
    Scope class: Any)
    Address class: auto_var (Local variable)

wh check_struct

In process 2:

Type name: struct check_struct; 
    Size: 80 bytes; Category: Structure
    Fields in type:
    {
    field1_int (32 bits)
    field2_char (8 bits)
    field2_chars$string[2] (16 bits)
    <padding>$char (8 bits)
    field3_int (32 bits)
    field2_uchar unsigned char (8 bits)
    <padding>$char[3] (24 bits)
    field_substruct sub_st (320 bits){
        field1_int (32 bits)
        <padding>$char[4] (32 bits)
        field2_longlong (64 bits)
        field2_ulong unsigned long (64 bits)
        field3_uint unsigned int (32 bits)
        en1 enum en_check (32 bits)
        field3_double double (64 bits)
    }
    ...
}
RELATED TOPICS

View > Lookup Variable Command in the online Help
dstatus Command
dwhere Command
dwhere

Displays the current execution location and call stack

**Format**

Displays locations in the call stack

```
```

Displays all locations in the call stack

```
```

**Arguments**

- **-all**

  Shows all levels of the call stack. This is the default.

- **-level level-num**

  Sets the level at which `dwhere` starts displaying information.

- **num-levels**

  Restricts output to this number of levels of the call stack. By default, all levels are shown.

- **-args**

  Displays argument names and values in addition to program location information. By default, the arguments are not shown.

- **-locals**

  Displays each frame’s local variables. By default, the local variable information is not shown.

- **-noshow_pc**

  Does not show the PC. By default, the PC value is shown.

- **-noshow_fp**

  Does not show the FP. By default, the FP value is shown.

- **-registers**

  Displays each frame’s registers. By default, the register information is not shown.

- **-show_image**

  Shows the executable name as well as the file name. By default, `dwhere` displays the associated image information if the source line cannot be found.

- **-group_by property**

  Aggregates stack backtraces of the focus threads, outputting a compressed `ptlist` that identifies the processes and threads containing equivalent stack frames in the backtrace. For information on the `ptlist` syntax, see Compressed List Syntax (`ptlist`).

  This option requires a `property` argument to control the “equivalence” relationship of stack frames across the threads. See The `-group_by Option` for more information.
**Description**

The `dwhere` command prints the current execution locations and the call stacks—or sequences of procedure calls—that led to that point. The CLI shows information for threads in the current focus; the default shows information at the thread level.

Arguments control the amount of command output in two ways:

- The `num-levels` argument determines how many levels of the call stacks are displayed, counting from the uppermost (most recent) level. Without this argument, the CLI shows all levels in the call stack, which is the default.
- The `-a` option displays procedure argument names and values for each stack level.

A `dwhere` command with no arguments or options displays the call stacks for all threads in the target set.

The `MAX_LEVELS` variable contains the default maximum number of levels displayed when you do not use the `num-levels` argument.

Output is generated for each thread in the target focus. The output is printed directly to the console.

**The `-group_by` Option**

The `-group_by` option requires a `property` argument, which controls the “equivalence” relationship of stack frames across the threads. When you use the `--group_by` option, `dwhere` aggregates the stack frames of each of the focus threads, forming a tree of equivalent stack frames.

Starting at the base of the stack (closest to `main()` or the thread’s start function), the `dwhere` command assigns each frame a distance from a synthetic root frame indicated by `/`. Two frames are equivalent only if all of the following apply:

- Their distance from the root is equal.
- They have the same parent frame.
- The selected property of frames is equivalent.

The following property values are supported, with their abbreviations in parentheses:

- `function` (f): Equivalence based on the name of the function containing the PC for the frame.
- `function+line` (f+l): Equivalence based on the name of the function and the file and line number containing the PC for the frame.
- `function+offset` (f+o): Equivalence based on the name of the function containing the PC for the frame and offset from the beginning of the function to the PC for the frame.
Looking at backtraces purely by the function property is the most coarse grained grouping of threads. Choosing a more fine grained grouping, such as a line number within the function, provides more detail about where in the code a given thread is executing, but it may also result in a much larger set of equivalent frames.

The dwhere command displays the current execution location(s) and the backtrace(s) for the threads in the current focus. If backtraces for multiple threads are requested, the stack displays are aggregated.

Lines denoting evaluation frames for compiled expressions or interpreted function calls are labeled with a suspended evaluation id. This id can be used to manipulate suspended evaluations with dflush and TV::expr.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>dwhere</td>
<td>Displays the current location in the call stack</td>
</tr>
</tbody>
</table>

**Examples**

dwhere

Displays the call stacks for all threads in the current focus.

dfocus 2.1 dwhere 1

Displays just the most recent level of the call stack corresponding to thread 1 in process 2. This shows just the immediate execution location of a thread or threads.

f p1.< w 5

Displays the most recent five levels of the call stacks for all threads involved in process 1. If the depth of any call stack is less than five levels, all of its levels are shown.

This command is a slightly more complicated way of saying f p1 w 5 because specifying a process width tells the dwhere command to ignore the thread indicator.

w 1 -a

Displays the current execution locations (one level only) of threads in the current focus, together with the names and values of any arguments that were passed into the current process.

**RELATED TOPICS**

dwhat Command
dstatus Command
dworker

Adds or removes a thread from a workers group

Format

dworker { number | boolean }

Arguments

number

If positive, marks the thread of interest (TOI) as a worker thread by inserting it into the workers group.

boolean

If \textbf{true}, marks the TOI as a worker thread by inserting it into the workers group. If \textbf{false}, marks the thread as a nonworker thread by removing it from the workers group.

Description

The \texttt{dworker} command inserts or removes a thread from the workers group.

If \texttt{number} is 0 or \texttt{false}, this command marks the TOI as a nonworker thread by removing it from the workers group. If \texttt{number} is \texttt{true} or is a positive value, this command marks the TOI as a worker thread by inserting it in the workers group.

Moving a thread into or out of the workers group has no effect on whether the thread is a manager thread. Manager threads are threads that are created by the pthreads package to manage other threads; they never execute user code, and cannot normally be controlled individually. TotalView automatically inserts all threads that are not manager threads into the workers group.

Command alias

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wof</td>
<td>{dworker false}</td>
<td>Removes the focus thread from the workers group</td>
</tr>
<tr>
<td>wot</td>
<td>{dworker true}</td>
<td>Inserts the focus thread into the workers group</td>
</tr>
</tbody>
</table>

RELATED TOPICS

- Organizing Chaos \textit{in the TotalView for HPC User Guide}
- Creating Groups \textit{in the TotalView for HPC User Guide}
- Setting Group Focus \textit{in the TotalView for HPC User Guide}
- dgroups Command
exit

Terminates the debugging session

Format
exit [-force ]

Arguments
-force
Exits without asking permission. This is most often used in scripts.

Description
The exit command ends the debugging session.

After you enter this command, the CLI confirms that you wish to exit, then exits. If you entered the CLI from the TotalView GUI, this command also closes the GUI window.

NOTE >> If you invoked the CLI from within the TotalView GUI, pressing Ctrl+D closes the CLI window without exiting from TotalView.

TotalView destroys all processes and threads that it makes. Any processes that existed prior to the debugging session (that is, TotalView attached to them because you used the detach command) are detached and left executing.

The exit and quit commands are interchangeable and do the same thing.

Examples
exit

Exits TotalView, leaving any attached processes running.

RELATED TOPICS
File > Exit Command in the online Help
quit Command
**help**

Displays help information

**Format**

```
help [ topic ]
```

**Arguments**

`topic`

A CLI topic or command.

**Description**

The `help` command prints information about the specified topic or command. With no argument, the CLI displays a list of the topics for which help is available.

If the CLI needs more than one screen to display the help information, it fills the screen with data and then displays a `more` prompt. Press Enter to see more data or `q` to return to the CLI prompt.

When you enter a topic name, the CLI attempts to complete an entry. You can also enter one of the CLI built-in aliases; for example:

```
d1.<> he a
Ambiguous help topic "a". Possible matches:
    alias accessors arguments addressing_expressions
d1.<> he ac
"ac" has been aliased to "dactions":
dactions [ bp-ids ... ] [ -at <source-loc> ] [ -disabled | 
    -enabled ]
    Default alias: ac
...
```

Use the `capture` command to place help information into a variable.

**Command alias**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>he</td>
<td>help</td>
<td>Displays help information</td>
</tr>
</tbody>
</table>

**Examples**

```
help help
```

Prints information about the `help` command.
quit

Terminates the debugging session

**Format**

```
quit [ -force ]
```

**Arguments**

- `-force`

Closes all TotalView processes without asking permission.

**Description**

The `exit` command terminates the TotalView session.

After you enter this command, the CLI confirms that you wish to exit, then exits. If you entered the CLI from the TotalView GUI, this command also closes the GUI window.

**NOTE >>** If you invoked the CLI from within the TotalView GUI, pressing Ctrl+D closes the CLI window without exiting from TotalView.

TotalView destroys all processes and threads that it makes. Any processes that existed prior to the debugging session (that is, TotalView attached to them because you used the `dattach` command) are detached and left executing.

The `exit` and `quit` commands are interchangeable and do the same thing.

**Examples**

```
quit
```

Exits TotalView, leaving any attached processes running.

**RELATED TOPICS**

File > Exit Command in the online Help

exit Command
**spurs**

Returns information on the spurs library use

**Format**
Displays help information
```
spurs [ help ]
```

Adds directories to the OBJECT_SEARCH_PATH variable
```
spurs add [ directory directory ... ]
```

Creates an image-qualified breakpoint
```
spurs break [ spu-image-name spu-source-location-expression ]
```

Deletes breakpoints
```
spurs delete breakpoint-id ... 
```

Shows the directories in which TotalView searches for SPURS SPU ELF executables
```
spurs info [ directory | break ]
```

Prints information about the kernel, the taskset, tasks, and other SPURS objects
```
spurs print [ kernel [ eaSpurs ] | barrier eaBarrier | event_flag eaEventFlagSet | lfqueue eaLFQueue | queue eaQueue | semaphore eaSemaphore | taskset [ eaTaskset ] | task eaTaskset taskId ]
```

Scans for information—this is a no-op
```
spurs scan
```

**Arguments**

**directory**

The directory or directories to be added to the CLI's OBJECT_SEARCH_PATH variable. For example:
```
spurs add directory directory1 directory2
```

Notice that directory names are separated by space characters.

**spu-image-name**

The name of the image that is or will be loaded by TotalView

**spu-source-location-expression**

An expression that resolves to a specific line in the image. For information on location expressions, see `dbreak` on page 42.

**breakpoint-id**

The action point ID to delete
eaSpurs
  The kernel context at this PPU address

eaBarrier
  The barrier object at this PPU address

eaEventFlagSet
  The event flag object at this PPU address

eaLQueue
  The lqueue object at this PPU address

eaQueue
  The queue object at this PPU address

eaSemaphore
  The semaphore object at this PPU address

eaTaskset
  The taskset at this PPU address

taskID
  The task at this index

Description
Modeled after the GDB “spurs” command, the spurs command was created so that developers who are familiar the GDB command have a similar set of commands in TotalView. However, not all GDB “spurs” commands are implemented.

TotalView supports the SPURS library. Here’s this library’s description in the SPURS documentation:
  libspurs is a user-level thread library for SPUs. In a SPURS environment (SPU Runtime System), SPU threads are managed by SPUs. For this reason, thread switching is more efficient than under PPU management and requires no PPU resources. Using SPURS also makes it easier to synchronize threads and adjust the load balance on multiple SPUs. SPURS is furthermore highly extensible and allows users to define their own thread models as necessary.

spurs [ help ]

To access help on the spurs command:

  • Enter spurs to return a one-line description of its commands.
  • Enter spurs help to display more information about each spurs subcommand.

spurs add [ directory directory ... ]

Displays either a one-line description of this command or adds directories to search when TotalView looks for SPURS SPU executables.

  • spurs add writes a one-line description of this command.
• **spurs add directory** `directory` adds a directory or directories to the CLI’s `OBJECT_SEARCH_PATH` variable. This variable contains the path used when searching for SPU ELF executable files. The directories are placed at the beginning of the list in the order in which they are named. If a directory is already in the list, the previously named directory is removed.

This command returns the modified `OBJECT_SEARCH_PATH` variable.

**spurs break [spu-image-name spu-source-location-expression]**

Displays either a one-line description of this command or adds a breakpoint.

• **spurs break** returns a one-line description of this command.

• **spurs break** `spu-image-name spu-source-location-expression` creates an image-qualified breakpoint path. This is identical to the following CLI command:

`dbreak -pending ##spu-image-name##source-loc-expr`

This command creates a pending breakpoint that is located only with the image you name. However, if the image has already been loaded, TotalView sets an ordinary breakpoint rather than a pending breakpoint. The focus must be on an SPU thread.

This command returns the action point ID of the created breakpoint. You can use this ID with other CLI commands that act upon breakpoints; for example, `dactions`, `ddelete`, `ddisable`, `denable`, and others.

**spurs delete breakpoint-id ...**

Permanently removes one or more action points. The argument defines which action points to delete. Unlike **spurs break**, this command does not require that the command focus be set to an SPU thread.

**spurs info [ directory | break ]**

• **spurs info** returns a one-line description of this command.

• **spurs info directory** prints the `OBJECT_SEARCH_PATH` state variable

• **spurs info break** prints action point information about action points in the thread in the current focus.

**spurs print**

The spurs print command can be used in the following ways:

```
spurs print [ kernel [ eaSpurs ] |
  barrier eaBarrier |
  event_flag eaEventFlagSet |
  lqueue eaLFQueue |
  queue eaQueue |
  semaphore eaSemaphore |
  taskset [ eaTaskset ] |
  task eaTaskset taskId ]
```

spurs print
  Displays one line of information on using this command.

spurs print kernel
  Displays the kernel context for the SPU threads in the current or named focus. The focus must be one or more
  SPU threads.

  **cell_spurs_print_kernel** is an alias for this command.

spurs print kernel eaSpurs
  Displays the kernel context at PPU address `eaSpurs`. The command focus must be one or more PPU threads.

  **cell_spurs_print_kernel** is an alias for this command.

spurs print barrier eaBarrier
  Displays the barrier object at PPU address `eaBarrier`. The command focus must be one or more PPU threads.

  **cell_spurs_print_barrier_info** is an alias for this command.

spurs print event_flag eaEventFlagSet
  Displays the event flag object at PPU address `eaEventFlagSet`. The command focus must be one or more PPU
  threads.

  **cell_spurs_print_event_flag_info** is an alias for this command.

spurs print lfqueue eaLFQueue
  Displays the lfqueue object at PPU address `eaLFQueue`. The command focus must be one or more PPU
  threads.

  **cell_spurs_print_lfqueue_info** is an alias for this command.

spurs print queue eaQueue
  Displays the queue object at PPU address `eaQueue`. The command focus must be one or more PPU
  threads.

  **cell_spurs_print_queue_info** is an alias for this command.

spurs print semaphore eaSemaphore
  Displays the semaphore object at PPU address `eaSemaphore`. The command focus must be one or more PPU
  threads.

  **cell_spurs_print_semaphore_info** is an alias for this command.

spurs print taskset
  Prints the taskset for the focus SPU threads. The command focus must be one or more SPU threads.

  **cell_spurs_print_taskset** is an alias for this command

spurs print taskset eaTaskset
  Prints the taskset at PPU address `eaTaskset`. The command focus must be one or more SPU threads.

  **cell_spurs_print_taskset** is an alias for this command

spurs print task eaTaskset taskId
  Prints the task at index `taskId` in the taskset at PPU address `eaTaskset`. The command focus must be one or more
  PPU threads.
**cell_spurs_print_task** is an alias for this command.

**spurs scan**

This command is for compatibility with GDB. Unlike the GDB command, this command is a no-op as TotalView has no need to scan for SPU executables because searches for SPU executables happen dynamically.
**stty**

Sets terminal properties

**Format**

```
stty [ stty-args ]
```

**Arguments**

*stty-args*

One or more UNIX *stty* command arguments as defined in the *man* page for your operating system.

**Description**

The CLI *stty* command executes a UNIX *stty* command on the *tty* associated with the CLI window, allowing you to set all your terminal’s properties. However, this is most often used to set erase and kill characters.

If you start the CLI from a terminal using the *totalviewcli* command, the *stty* command alters this terminal’s environment. Consequently, the changes you make using this command are retained in the terminal after you exit.

If you omit the *stty-args* argument, the CLI returns help information on your current settings.

The output from this command is returned as a string.

**Examples**

```
stty
```

Prints information about your terminal settings, equivalent to having entered *stty* while interacting with a shell.

```
stty -a
```

Prints information on all your terminal settings.

```
stty erase ^H
```

Sets the *erase* key to Backspace.

```
stty sane
```

Resets the terminal’s settings to values that the shell thinks they should be. For problems with command-line editing, use this command. (The *sane* argument is not available in all environments.)
unalias

Removes a previously defined alias

Format
Removes an alias
  unalias alias-name
Removes all aliases
  unalias -all

Arguments
  alias-name
    The name of the alias to delete.

  -all
    Removes all aliases.

Description
The unalias command removes a previously defined alias. You can delete all aliases using the -all option. Aliases defined in the tvdinit.tvd file are also deleted.

Examples

unalias step2
  Removes the step2 alias; step2 is undefined and can no longer be used. If step2 was included as part of the definition of another command, that command no longer works correctly. However, the CLI only displays an error message when you try to execute the alias that contains this removed alias.

unalias -all
  Removes all aliases.

RELATED TOPICS
  alias Command
Chapter 3

CLI Namespace Commands

Command Overview

This chapter lists all of CLI commands that are not in the top-level mainspace.

AccessorType Functions

The following functions, all within the TV:: namespace, access and set TotalView properties:

- **actionpoint**: Accesses and sets action point properties.
- **expr**: Manipulates values created by the `dprint -nowait` command.
- **focus_groups**: Returns a list containing the groups in the current focus.
- **focus_processes**: Returns a list of processes in the current focus.
- **focus_threads**: Returns a list of threads in the current focus.
- **group**: Accesses and sets group properties.
- **process**: Accesses and sets process properties.
- **scope**: Accesses and sets scope properties.
- **symbol**: Accesses and sets symbol properties.
- **thread**: Accesses and sets thread properties.
- **type**: Accesses and sets data type properties.
- **type_transformation**: Accesses and defines type transformations.
Helper Functions

The following functions, all within the `TV::` namespace, are most often used in scripts:

- **dec2hex**: Converts a decimal number into hexadecimal format.
- **dll**: Manages shared libraries.
- **errorCodes**: Returns or raises TotalView error information.
- **hex2dec**: Converts a hexadecimal number into decimal format.
- **read_symbols**: Reads shared library symbols.
- **respond**: Sends a response to a command.
- **source_process_startup**: Reads and executes a `.tvd` file when TotalView loads a process.
**actionpoint**

Sets and gets action point properties

**Format**

TV::actionpoint action [ object-id ] [ other-args ]

**Arguments**

action  

The action to perform, as follows:

- **commands**  
  Displays the subcommands that you can use. The CLI responds by displaying these four action subcommands. There are no arguments to this subcommand.

- **get**  
  Retrieves the values of one or more action point properties. The other-args argument can include one or more property names. The CLI returns values for these properties in a list whose order is the same as the names you enter.

  If you use the -all option instead of the object-id, the CLI returns a list containing one (sublist) element for each object.

- **properties**  
  Lists the action point properties that TotalView can access. There are no arguments to this subcommand.

- **set**  
  Sets the values of one or more properties. The other-args argument contains property name and value pairs.

  - **object-id**  
    An identifier for the action point.

  - **other-args**  
    Arguments that the get and set actions use.

**Description**

The TV::actionpoint command lets you examine and set the following action point properties and states:

- **address**  
  The address of the action point.

- **block_count**  
  The number of addresses associated with an actionpoint.

  A single line of code can generate multiple instruction sequences. For example, there may be several entry points to a subroutine, depending on where the caller is. This means that an actionpoint can be set at many addresses even if you are placing it on a single line.

  Internally, a block represents one of these addresses.
block_enabled
Each individual actionpoint block is an instruction that TotalView may replace with a trap instruction. (When a trap instruction is encountered, the operating system passes control to the debugger.)
Each block can be enabled or disabled separately. This property type returns a list with in which 1 indicates if the block is enabled and 0 if it is disabled.
This is the only property that can be set from within TotalView. All others are read-only.
conflicted
Indicates that another action point shares at least one of the action point blocks. If this condition exists, the block is conflicted. If a block is conflicted, TotalView completely disables the action point.
The conflicted property is 1 if the actionpoint is conflicted, and 0 if it is not.
context
A string that totally identifies an action point.
The location of every action point is represented by a string. Even action points set by clicking on a line number are represented by strings. (In this case, the string is the line number.)
Sometimes, this string is all that is needed. Usually, however, more context is needed. For example, a line number needs a file name.
enabled
A value (either 1 or 0) indicating if the action point is enabled. A value of 1 means enabled. (settable)
expression
The expression to execute at an action point. (settable)
id
The ID of the action point.
language
The language in which the action point expression is written.
length
The length in bytes of a watched area. This property is only valid for watchpoints. (settable)
line
The source line at which the action point is set. This property is not valid for watchpoints.
satisfaction_group
The group that must arrive at a barrier for the barrier to be satisfied. (settable)
share
A value (either 1 or 0) indicating if the action point is active in the entire share group. A value of 1 means that it is. (settable)
stop_when_done
A value that indicates what is stopped when a barrier is satisfied (in addition to the satisfaction set). Values are process, group, or none. (settable)
stop_when_hit
A value that indicates what is stopped when an action point is hit (in addition to the thread that hit the action point). Values are process, group, or none. (settable)

type
The object's type. (See type_values for a list of possible types.)

type_values
Lists values that can TotalView can assign to the type property: break, eval, process_barrier, thread_barrier, and watch.

Examples
TV::actionpoint set 5 share 1 enable 1
Shares and enables action point 5.
f p3 TV::actionpoint set -all enable 0
Disables all the action points in process 3.
foreach p [TV::actionpoint properties]
  puts [format "%20s %s" $p: [TV::actionpoint get 1 $p]]
Dumps all the properties for action point 1. Here is what your output might look like:

address: 0x1200019a8
enabled: 0
expression: id: 1
language: length:
line: /temp/arrays.F#84
satisfaction_group: satisfaction_process: satisfaction_width:
share: 1
stop_when_done: stop_when_hit: group
type: break
type_values: break eval process_barrier thread_barrier watch

RELATED TOPICS
dactions Command
dec2hex

Converts a decimal number into hexadecimal

Format
TV::dec2hex number

Arguments
number
A decimal number to convert.

Description
The TV::dec2hex command converts a decimal number into hexadecimal. This command correctly manipulates 64-bit values, regardless of the size of a long value on the host system.

RELATED TOPICS
hex2dec Command
**dll**

Manages shared libraries

**Format**

TV::dll *action* [ *dll-id-list* ] [-all ]

**Arguments**

*action*

The action to perform, as follows:

**close**

Dynamically unloads the shared object libraries that were dynamically loaded by the `ddlopen` commands corresponding to the list of `dll-ids`.

If you use the `-all` option, TotalView closes all of the libraries that it opened.

**commands**

Displays the subcommands that you can use. The CLI responds by displaying these four `action` subcommands. There are no arguments to this subcommand.

**get**

Retrieves the values of one or more TV::dll properties. The other-args argument can include one or more property names.

If you use the `-all` option as the `dll-id-list`, the CLI returns a list containing one (sublist) element for each object.

**properties**

Lists the TV::dll properties that TotalView can access. There are no arguments to this subcommand.

**resolution_urgency_values**

Returns a list of values that this property can take. This list is operating-system specific, but always includes `{lazy now}`.

**symbol_availability_values**

Returns a list of values that this property can take. This list is operating system specific, but always includes `{lazy now}`.

*dll-id-list*

A list of one or more dll-ids. There are the IDs returned by the `ddlopen` command.

*all*

Closes all shared libraries that you opened using the `ddlopen` command.

**Description**

The TV::dll command either closes shared libraries that were dynamically loaded with the `ddlopen` command or obtains information about loaded shared libraries.
Examples

TV::dll close 1

Closes the first shared library that you opened.

RELATED TOPICS

ddlopen Command
**errorCodes**

Returns or raises TotalView error information

**Format**

Returns a list of all error code tags

```
TV::errorCodes
```

Returns or raises error information

```
TV::errorCodes number_or_tag [-raise [ message ] ]
```

**Arguments**

`number_or_tag`

An error code mnemonic tag or its numeric value.

`-raise`

Raises the corresponding error. If you append a `message`, TotalView returns this string. Otherwise, TotalView uses the human-readable string for the error.

`message`

An optional string used when raising an error.

**Description**

The `TV::errorCodes` command lets you manipulate the TotalView error code information placed in the Tcl `errorCodes` variable. The CLI sets this variable after every command error. Its value is intended to be easy to parse in a Tcl script.

When the CLI or TotalView returns an error, `errorCodes` is set to a list with the following format:

```
TOTALVIEW error-code subcodes... string
```

where:

- The first list element is always `TOTALVIEW`.
- The second list element is always the error code.
- The `subcodes` argument is not used at this time.
- The last list element is a string describing the error.

With a tag or number, this command returns a list containing the mnemonic tag, the numeric value of the tag, and the string associated with the error.

The `-raise` option raises an error. If you add a message, that message is used as the return value; otherwise, the CLI uses its textual explanation for the error code. This provides an easy way to return errors from a script.
Examples

```
foreach e [TV::errorCodes] {
    puts [eval format {"%20s %2d %s"} \n        [TV::errorCodes $e]]}
```

Displays a list of all TotalView error codes.

RELATED TOPICS

- dprint Command
- TV::expr Command
**expr**

Manipulates values created by the dprint -nowait command

**Format**

TV::expr action [ susp-eval-id ] [ other-args ]

**Arguments**

*action*

The action to perform, as follows:

- **commands**
  Displays the subcommands that you can use. The CLI responds by displaying the subcommands shown here. Do not use additional arguments with this subcommand.

- **delete**
  Deletes all data associated with a suspended ID. If you use this command, you can specify an *other-args* argument. If you use the *-done* option, the CLI deletes the data for all completed expressions; that is, those expressions for which TV::expr get susp-eval-id done returns 1. If you specify *-all*, the CLI deletes all data for all expressions.

- **get**
  Gets the values of one or more *expr* properties. The *other-args* argument can include one or more values. The CLI returns these values in a list whose order is the same as the property names.

  If you use the *-all* option instead of *suspeval-id*, the CLI returns a list containing one (sublist) element for each object.

- **properties**
  Displays the properties that the CLI can access. Do not use additional arguments with this option.

* susp-eval-id
  The ID returned or thrown by the dprint command, or printed by the dwhere command.

* other-args
  Arguments required by the delete subcommand.

**Description**

The TV::expr command, in addition to showing you command information, returns and deletes values returned by a dprint -nowait command. You can use the following properties for this command:

- **done**
  TV::expr returns 1 if the process associated with *suspeval-id* has finished in all focus threads. Otherwise, it returns 0.

- **expression**
  The expression to execute.

- **focus_threads**
  A list of *dpid dtid* values in which the expression is being executed.
id

The `susp-eval-id` of the object.

initially_suspended_process

A list of dpid IDs for the target processes that received control because they executed the function calls or compiled code. You can wait for processes to complete by entering the following:

```
dfocus p dfocus [TV::expr get \ susp-eval-id \ initially_suspended_processes] dwait
```

result

A list of pairs for each thread in the current focus. Each pair contains the thread as the first element and that thread's result string as the second element; for example:

```
d1.<> dfocus {1.1 2.1} TV::expr \ get susp-eval-id result
{{1.1 2} {2.1 3}} d1.<>
```

The result of expression `susp-eval-id` in thread 1.1 is 2, and in thread 2.1 is 3.

status

A list of pairs for each thread in the current focus. Each pair contains the thread ID as the first element and that thread's status string as the second element. The possible status strings are `done`, `suspended`, and `{error diag}`.

For example, if expression `susp-eval-id` finished in thread 1.1, suspended on a breakpoint in thread 2.1, and received a syntax error in thread 3.1, that expression's status property has the following value when `TV::expr` is focused on threads 1.1, 2.1, and 3.1:

```
d1.<> dfocus {t1.1 t2.1 t3.1} \ TV::expr get 1 status
{1.1 done} {2.1 suspended} {3.1 {error {Symbol nothing2 not found}}}
d1.<>
```

RELATED TOPICS

dprint Command
focus_groups

Returns a list of groups in the current focus

Format
TV::focus_groups

Arguments
This command has no arguments

Description
The TV::focus_groups command returns a list of all groups in the current focus.

Examples
f d1.< TV::focus_groups

Returns a list containing one entry, which is the ID of the control group for process 1.

RELATED TOPICS
Using Groups, Processes, and Threads in the TotalView for HPC User Guide
focus_processes Command
focus_threads Command
dfocus Command
focus_processes

Returns a list of processes in the current focus

Format
TV::focus_processes [ -all | -group | -process | -thread ]

Arguments
- all
  Changes the default width to all.
- group
  Changes the default width to group.
- process
  Changes the default width to process.
- thread
  Changes the default width to thread.

Description
The TV::focus_processes command returns a list of all processes in the current focus. If the focus width is something other than d (default), the focus width determines the set of processes returned. If the focus width is d, the TV::focus_processes command returns process width. Using any of the options changes the default width.

Examples
f g1.< TV::focus_processes
  Returns a list containing all processes in the same control as process 1.

RELATED TOPICS
focus_processes Command
focus_threads Command
dfocus Command
Using Groups, Processes, and Threads in the TotalView for HPC User Guide
**focus_threads**

Returns a list of threads in the current focus

**Format**

```
TV::focus_threads [ -all | -group | -process | -thread ]
```

**Arguments**

- **-all**
  - Changes the default width to **all**.
- **-group**
  - Changes the default width to **group**.
- **-process**
  - Changes the default width to **process**.
- **-thread**
  - Changes the default width to **thread**.

**Description**

The *TV::focus_threads* command returns a list of all threads in the current focus. If the focus width is something other than **d** (default), the focus width determines the set of threads returned. If the focus width is **d**, the *TV::focus_threads* command returns thread width. Using any of the options changes the default width.

**Examples**

```
$ p1.< TV::focus_threads
```

Returns a list containing all threads in process 1.

**RELATED TOPICS**

- focus_processes Command
- focus_threads Command
- dfocus Command
- Using Groups, Processes, and Threads in the TotalView for HPC User Guide
group

Sets and gets group properties

Format
TV::group action [ object-id ] [ other-args ]

Arguments

action

The action to perform, as follows:

commands
Displays the subcommands that you can use. The CLI responds by displaying these four action subcommands. Do not use additional arguments with this subcommand.

get
Gets the values of one or more group properties. The other-args argument can include one or more property names. The CLI returns the values for these properties in a list in the same order as you entered the property names.

If you use the -all option instead of object-id, the CLI returns a list containing one (sublist) element for each group.

properties
Displays the properties that the CLI can access. Do not use additional arguments with this option.

set
Sets the values of one or more properties. The other-args argument is a sequence of property name and value pairs.

object-id
The group ID. If you use the -all option, TotalView executes this operation on all groups in the current focus.

other-args
Arguments required by the get and set subcommands.

Description
The TV::group command lets you examine and set the following group properties and states:

actionpoint_count
The number of shared action points planted in the group. This is only valid for share groups and shared action points that are associated with the share group containing the process, rather than with the process itself.

When you obtain the results of this read-only value, the number may not look correct as this number also includes “magic breakpoints”. These are breakpoints that TotalView sets behind the scene; they are not usually visible. In addition, these magic breakpoints seldom appear when you use the dactions command.

canonical_execution_name
The absolute file name of the program being debugged. If you had entered a relative name, TotalView finds this absolute name.
count
  The number of members in a group.

executable
  Like `canonical_execution_name`, this is the absolute file name of the program being debugged. It differs in
  that it contains symbolic links and the like that exist for the program.

id
  The ID of the object.

member_type
  The type of the group's members, either `process` or `thread`.

member_type_values
  A list of all possible values for the `member_type` -property. For all groups, this is a two-item list with the first be-
  ing the number of process groups and the second being the number of thread groups. In many ways, this is re-
  lated to the `type_values` property, which is a list values the `type` property may take.

members
  A list of a group's processes or threads.

type
  The group's type. Possible values are `control`, `lockstep`, `share`, `user`, and `workers`.

type_values
  A list of all possible values for the `type` property.

Examples

```
TV::group get 1 count
```

Returns the number of objects in group 1.

RELATED TOPICS

- `focus_groups Command`
- `dworker Command`
- `process Command`
- `thread Command`
- `Using Groups, Processes, and Threads` in the `TotalView for HPC User Guide`
hex2dec
Converts a hexadecimal number to decimal

**Format**
TV::hex2dec number

**Arguments**
*number*
A hexadecimal number to convert.

**Description**
The TV::hex2dec command converts a hexadecimal number to decimal. You can type 0x before this value. The CLI correctly manipulates 64-bit values, regardless of the size of a long value.

**RELATED TOPICS**
dec2hex Command
process

Sets and gets process properties

Format
TV::process action [ object-id ] [ other-args ]

Arguments

action
The action to perform, as follows:

commands
Displays the subcommands that you can use. The CLI responds by displaying these four action subcommands. Do not use other arguments with this subcommand.

get
Gets the values of one or more process properties. The other-args argument can include one or more property names. The CLI returns these property values in a list whose order is the same as the names you enter. If you use the -all option instead of object-id, the CLI returns a list containing one (sublist) element for each object.

properties
Displays the properties that the CLI can access. Do not use other arguments with this subcommand.

set
Sets the values of one or more properties. The other-args arguments contains pairs of property names and values.

object-id
An identifier for a process. For example, 1 represents process 1. If you use the -all option, the operation executes upon all objects of this class in the current focus.

other-args
Arguments required by the get and set subcommands.

Description
The TV::process command lets you examine and set process properties and states, as the following list describes:

cannonical_executable_name
The full pathname of the current executable.

clusterid
The ID of the cluster containing the process. This is a number uniquely identifying the TotalView server that owns the process. The ID for the cluster TotalView is running in is always 0 (zero).

data_size
The size of the process's data segment.
duid
The internal unique ID associated with an object.

executable
Like `canonical_execution_name`, this is the absolute file name of the program being debugged. It differs in
that it contains an symbolic links and the like that exist for the program.

heap_size
The amount of memory currently being used for data created at runtime. Stated in a different way, the heap is
an area of memory that your program uses when it needs to dynamically allocate memory. For example, calls to
the `malloc()` function allocate space on the heap while the `free()` function releases the space.

held
A Boolean value (either 1 or 0) indicating if the process is held. (1 means that the process is held.)

hia_guard_max_size
The value set for the maximum size for guard blocks that surround a memory allocation. See the Debugging Mem-
ory Problems with MemoryScape™ for information on what this size represents.

hia_guard_payload_alignment
The number of bits the guard block is aligned to.

hia_guard_pre_pattern
The numerical value of the bit pattern written into the guard block preceding an allocated memory block.

hia_guard_pre_size
The number of bits into which the guard block preceding an allocated memory block is written.

hia_guard_post_pattern
The numerical value of the bit pattern written into the guard block following an allocated memory block.

hia_guard_post_size
The number of bits into which the guard block following an allocated memory block is written.

hia_paint_pattern_width
Deprecated

hostname
A name of the process's host computer and operating system (if needed); for example, linux-x86 would be re-
turned if the program is running on a 32-bit linux system.

is_parallel
Contains a value indicating if the current process is a parallel process. If it is, its value is 1. Otherwise, its value is
0.

id
The process ID.

image_ids
A list of the IDs of all the images currently loaded into the process both statically and dynamically. The first ele-
ment of the list is the current executable.
is_parallel
Contains a value indicating if the current process is a parallel process. If it is, its value is 1. Otherwise, its value is 0.
	nodeid
The ID of the node upon which the process is running. The ID of each processor node is unique within a cluster.

parallel_attach_subset
Contains the specification for MPI ranks to be attached to when an MPI job is created or attached to. See -parallel_attach_subset subset_specification.

proc_name
The name of the process currently being executed.

rank
The rank of the currently selected process.

stack_size
The amount of memory used by the currently executing block or routines, and all the routines that have invoked it. For example, if your main routines invokes the foo() function, the stack contains two groups of information—these groups are called frames. The first frame contains the information required for the execution of your main routine and the second, which is the current frame, contains the information needed by the foo() function. If foo() invokes the bar() function, the stack contains three frames. When foo() finishes executing, the stack only contains one frame.

stack_vm_size
The logical size of the stack is the difference between the current value of the stack pointer and the address from which the stack originally grew. This value can be different from the size of the virtual memory mapping in which the stack resides. For example, the mapping can be larger than the logical size of the stack if the process previously had a deeper nest of procedure calls or made memory allocations on the stack, or it can be smaller if the stack pointer has advanced but the intermediate memory has not been touched.

The stack_vm_size value is this difference in size.

state
Current state of the process. See state_values for a list of states.

state_values
A list of all possible values for the state property: break, error, exited, running, stopped, or watch.

syspid
The system process ID.

target_architecture
The machine architecture upon which the current process is executing.

target_byte_ordering
The bit ordering of the current machine. This is either little_endian or big_endian.
**target_processor**
The kind of processor upon which the program is executing. For example, this could be **x86** or **x86-64**.

**text_size**
The amount of memory used to store your program's machine code instructions. The text segment is sometimes called the code segment.

**threadcount**
The number of threads in the process.

**threads**
A list of threads in the process.

**vm_size**
The sum of the mapping sizes in the process's address space.

**Examples**

`fg TV::process get -all id threads`
For each process in the group, creates a list with the process ID followed by the list of threads; for example:
```
{1 {1.1 1.2 1.4}} {2 {2.3 2.5}} {3 {3.1 3.7 3.9}}
```

`TV::process get 3 threads`
Gets the list of threads for process 3; for example:
```
1.1 1.2 1.4
```

`TV::process get 1 image_ids`
Returns a list of image IDs in process 1; for example:
```
1|1 1|2 1|3 1|4
```

**RELATED TOPICS**

- Using Groups, Processes, and Threads in the *TotalView for HPC User Guide*
- `focus_processes` Command
- `group` Command
- `thread` Command
**read_symbols**

Reads shared library symbols

**Format**

Reads symbols from libraries

```
TV::read_symbols -lib lib-name-list
```

Reads symbols from libraries associated with a stack frame

```
TV::read_symbols -frame [number]
```

Reads symbols for all stack frames in the backtrace

```
TV::read_symbols -stack
```

**Arguments**

- **-lib [lib-name-list]**
  
  Tells TotalView to read symbols for all libraries whose names are contained within the *lib-name-list* argument. Each name can include the asterisk (*) and question mark (?) wildcard characters.

  This command ignores the current focus; libraries for any process can be affected.

- **-frame [number]**
  
  Tells TotalView to read the symbols for the library associated with the current stack frame. If you also enter a frame number, TotalView reads the symbols for the library associated with that frame.

- **-stack**
  
  Reads the symbols for every frame in the backtrace. This is the same as right-clicking in the Stack Trace Pane and selecting the **Load All Symbols in Stack** command. If, while reading in a library, TotalView may also need to read in the symbols from additional libraries.

**Description**

The **TV::read_symbols** command reads debugging symbols from one or more libraries that TotalView has already loaded but whose symbols have not yet been read. They are not yet read because the libraries were included within either the **TV::dll_read_loader_symbols_only** or **TV::dll_read_no_symbols** lists.

For more information, see “Preloading Shared Libraries” in the **TotalView for HPC User Guide**.
respond

Provides responses to commands

**Format**

TV::respond response command

**Arguments**

*response*

The response to one or more commands. If you include more than one response, separate the responses with newline -characters.

*command*

One or more commands that the CLI executes.

**Description**

The TV::respond command executes a command. The command argument can be a single command or a list of commands. In most cases, you place this information in braces ({}). If the CLI asks questions while command is executing, you are not asked for the answer. Instead, the CLI uses the characters in the response string for the argument. If more than one question is asked and strings within the response argument have all been used, The TV::respond command starts over at the beginning of the response string. If response does not end with a newline, the TV::respond command appends one.

Do not use this command to suppress the MORE prompt in macros. Instead, use the following command:

```bash
dset LINES_PER_SCREEN 0
```

The most common values for response are *y* and *n*.

**NOTE >>** If you are using the TotalView GUI and the CLI at the same time, your CLI command might cause dialog boxes to appear. You cannot use the TV::respond command to close or interact with these dialog boxes.

**Examples**

TV::respond {y} {exit}

Exits from TotalView. This command automatically answers the “Do you really wish to exit TotalView” question that the exit command asks.

```bash
set f1 y
set f2 exit
TV::respond $f1 $f2
```

A way to exit from TotalView without seeing the “Do you really wish to exit TotalView” question. This example and the one that preceded are not really what you would do as you would use the exit -force command.
scope

Sets and gets internal scope properties

Format

```
TV::scope action [ object-id ] [ other-args ]
```

Arguments

**action**

The action to perform, as follows:

**cast**

Attempts to find or create the type named by the *other-args* argument in the given scope.

**code_unit_by_soid**

Look up loader symbols by using a regular expression to match the base name. For example:

```
TV::scope lookup $scope_id \
  loader_sym_by_regexp \"regular expression\"
```

**commands**

Displays the subcommands that you can use. The CLI responds by displaying the subcommands shown here. Do not use additional arguments with this subcommand.

**create**

Allows you to create blocks, enum_type, named_constant, typedef, upc_shared_type, and variable symbols. The type of symbol determines the properties you need to specify. In all cases, you must specify the *kind* property. If you are creating a located symbol such as a block, you need to provide a location. If you are creating a upc_shared_type, you need a target_type index.

**dump**

Dump all properties of all symbols in the scope and in the enclosed scope.

**get**

Returns properties of the symbols whose soids are specified. Specify the kinds of properties using the *other-args* argument.

If you use the `-all` option instead of *object-id*, the CLI returns a list containing one (sublist) element for each object.

**lookup**

Look up a symbol by name. Specify the kind of lookup using the *other-args* argument. The values you can enter are:

- **by_language_rules**: Use the language rules of the language of the scope to find a single name.
- **by_path**: Look up a symbol using a pathname.
- **by_properties [proptery_regexp_pair]**: TotalView recurses down the scope tree after it visits a symbol. This means TotalView will search for matching symbols in the specified scope and any nested scope. The *walk* property shows an example.
- **by_type_index**: Look up a symbol using a type index.
**in_scope**: Look up a name in the given scope and in all enclosing scopes, and in the global scope.

**lookup_keys**
Displays the kinds of lookup operations that you can perform.

**properties**
Displays the properties that the CLI can access. Do not use additional arguments with this option. The arguments displayed are those that are displayed for the scope of all types. Additional properties also exist but are not shown. (Only the ones used by all are visible.) For more information, see TV::symbol.

**walk**
Walk the scope, calling Tcl commands at particular points in the walk. The commands are named using the following options:

**by_properties [proptery_regex_pair]**: TotalView recurses down the scope tree after it visits a symbol. This means TotalView will search for matching symbols in the specified scope and any nested scope. For example:

```
TV::scope walk $scope_id by_properties \
kind typedef base_name "^__BMN_.*$"
```

- **pre_scope** *cmd*: Names the commands called before walking a scope.
- **pre_sym** *cmd*: Names the commands called before walking a symbol.
- **post_scope** *cmd*: Names the commands called after walking a scope.
- **post_symbol** *cmd*: Names the commands called after walking a symbol.
- **tcl_cmd**: Names the commands called for each symbol.

**object-id**
The ID of a scope.

**other-args**
Arguments required by the get subcommand.

**Description**
The TV::scope command lets you examine and set a scope’s properties and states.

**Examples**

```
TV::scope create $scope kind [kind] \n[required_property_regex_pair]...\n[non-required_property_regex_pair]...
```

This is the general specification for creating a symbol

```
TV::scope create 1|31 kind block location {ldam 0x12}
```

Create a block. A block should have a length. However, you can set the length later using the set property.
source_process_startup  Reads, then executes a .tvd file when a process is loaded

Format
TV::source_process_startup process_id

Arguments
process_id
  The PID of the current process.

Description
The TV::source_process_startup command loads and interprets the .tvd file associated with the current process. That is, if a file named executable.tvd exists, the CLI reads and then executes the commands in it.

RELATED TOPICS
  Initializing TotalView in the TotalView for HPC User Guide
symbol

Gets and sets symbol properties

Format

TV::symbol action [ object-id ] [ other-args ]

Arguments

action

The action to perform, as follows:

code_unit_by_soid

Returns the containing scope of a line number. For example:

TV::symbol code_unit_by_soid $start_line

commands

Displays the subcommands that you can use. The CLI responds by displaying the subcommands shown here. Do not use additional arguments with this subcommand.

dump

Dumps all properties of the symbol whose soid (symbol object ID) is named. Do not use additional arguments with this command.

get

Returns properties of the symbols whose soids are specified here. The other-args argument names the properties to be returned.

properties

Displays the properties that the CLI can access. Do not use additional arguments with this option. These properties are discussed later in this section.

read_delayed

Only global symbols are initially read; other symbols are only partially read. This command forces complete symbol processing for the compilation units that contain the named symbols.

resolve_final

Performs a sequence of resolve_next operations until the symbol is no longer undiscovered. If you apply this operation to a symbol that is not undiscovered, it returns the symbol itself.

resolve_next

Some symbols only serve to hold a reference to another symbol. For example, a typedef is a reference to the aliased type, or a const-qualified type is a reference to the non-const qualified type. These reference types are called undiscovered symbols. This operation, when performed on an undiscovered symbol, returns the symbol the type refers to. When this is performed on a symbol, it returns the symbol itself.

rebind

Changes one or more structural properties of a symbol. These operations can crash TotalView or cause it to produce inconsistent results. The properties that you can change are:

address: the new address:
**base_name**: the new base name. The symbol must be a base name.

**line_number**: the new line number. The symbol must be a line number symbol.

**loader_name**: the new loader name and a file name.

**scope**: the soid of a new scope owner.

**type_index**: the new type index, in the form <n, m, p>. The symbol must be a type.

**set**

Sets a symbol's property. Not all properties can be set. Determine which properties can be set using the **writable_properties** property. For example,

```bash
tv::symbol set $new_upc_type type_index $old_idx
```

**writable_properties**

Returns a list of writable properties. For example:

```bash
tv::symbol writable_properties $symbol_id
```

**object-id**

The ID of a symbol.

**other-args**

Arguments required by the **get** subcommand.

**Description**

The **TV::symbol** command lets you examine and set the symbol properties and states.

**Symbol Properties**

**Table 1** lists the properties associated with the symbols information that TotalView stores. Not all of this information will be useful when creating transformations. However, it is possible to come across some of these properties and this information will help you decide if you need to use it in your transformation. In general, the properties used in the transformation files that Rogue Wave Software provided will be the ones that you will use.

**Table 1: Symbol Properties**

<table>
<thead>
<tr>
<th>Symbol Kind</th>
<th>Has base_name</th>
<th>Has type_index</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate_type</td>
<td>X</td>
<td>X</td>
<td>aggregate_kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>artificial</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>external_name</td>
</tr>
<tr>
<td>array_type</td>
<td>X</td>
<td>X</td>
<td>artificial</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>data_addressing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>element_addressing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>external_name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>index_type_index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>submembers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>target_type_index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>upper_bound</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>validator</td>
</tr>
<tr>
<td>Symbol Kind</td>
<td>Has base name</td>
<td>Has type index</td>
<td>Property</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>----------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>block</td>
<td></td>
<td></td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td>kind</td>
</tr>
<tr>
<td>char_type</td>
<td></td>
<td>X</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>kind</td>
</tr>
<tr>
<td>code_type</td>
<td></td>
<td>X</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>kind</td>
</tr>
<tr>
<td>ds_undiscovered_type</td>
<td>X</td>
<td>X</td>
<td>id</td>
</tr>
<tr>
<td>enum_type</td>
<td></td>
<td>X</td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>id</td>
</tr>
<tr>
<td>file</td>
<td></td>
<td></td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>id</td>
</tr>
<tr>
<td>float_type</td>
<td></td>
<td>X</td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>id</td>
</tr>
<tr>
<td>function_type</td>
<td></td>
<td>X</td>
<td>full_pathname</td>
</tr>
<tr>
<td>image</td>
<td></td>
<td></td>
<td>id</td>
</tr>
<tr>
<td>int_type</td>
<td></td>
<td>X</td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>id</td>
</tr>
<tr>
<td>label</td>
<td></td>
<td></td>
<td>address_class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kind</td>
</tr>
<tr>
<td>Symbol Kind</td>
<td>Has base name</td>
<td>Has type index</td>
<td>Property</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------</td>
<td>----------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>linenumber</td>
<td>address_class</td>
<td>id</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td>artificial</td>
<td>kind</td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td>full_pathname</td>
<td>location</td>
<td></td>
</tr>
<tr>
<td>loader_symbol</td>
<td>address_class</td>
<td>id</td>
<td>location</td>
</tr>
<tr>
<td></td>
<td>artificial</td>
<td>kind</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td>full_pathname</td>
<td>length</td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>member</td>
<td>address_class</td>
<td>id</td>
<td>location</td>
</tr>
<tr>
<td></td>
<td>artificial</td>
<td>kind</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td>full_pathname</td>
<td>length</td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>type_index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module</td>
<td></td>
<td>id</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kind</td>
<td>scope_owner</td>
</tr>
<tr>
<td>named_constant</td>
<td></td>
<td>id</td>
<td>type_index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kind</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>namespace</td>
<td></td>
<td>id</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kind</td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td>opaque_type</td>
<td></td>
<td>id</td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kind</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td>pathname_reference_symbol</td>
<td></td>
<td>id</td>
<td>resolved_symbol_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kind</td>
<td>path_name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lookup_scope</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>logical_scope_owner</td>
<td></td>
</tr>
<tr>
<td>pointer_type</td>
<td></td>
<td>id</td>
<td>target_type_index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kind</td>
<td>validator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scope_owner</td>
</tr>
<tr>
<td>qualified_type</td>
<td></td>
<td>id</td>
<td>qualification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kind</td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>target_type_index</td>
</tr>
<tr>
<td>soid_reference_symbol</td>
<td></td>
<td>id</td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>resolved_symbol_id</td>
</tr>
</tbody>
</table>
Table 1: Symbol Properties

<table>
<thead>
<tr>
<th>Symbol Kind</th>
<th>Has base name</th>
<th>Has type index</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>stringchar_type</td>
<td>X</td>
<td>X</td>
<td>artificial, external name, full_pathname, id, kind, logical_scope_owner, scope_owner, target_type_index</td>
</tr>
<tr>
<td>subroutine</td>
<td>X</td>
<td></td>
<td>address_class, artificial, full_pathname, id, kind, length, location, logical_scope_owner, return_type_index, scope_owner, static_chain, static_chain_height</td>
</tr>
<tr>
<td>typedef</td>
<td>X</td>
<td>X</td>
<td>artificial, external name, full_pathname, id, kind, length, logical_scope_owner, scope_owner, target_type_index</td>
</tr>
<tr>
<td>variable</td>
<td>X</td>
<td></td>
<td>address_class, artificial, full_pathname, id, is_argument, kind, location, logical_scope_owner, ordinal, scope_owner, type_index</td>
</tr>
<tr>
<td>void_type</td>
<td>X</td>
<td>X</td>
<td>artificial, external name, full_pathname, id, kind, length, logical_scope_owner, scope_owner</td>
</tr>
<tr>
<td>wchar_type</td>
<td>X</td>
<td>X</td>
<td>artificial, external name, full_pathname, id, kind, logical_scope_owner, scope_owner, target_type_index</td>
</tr>
</tbody>
</table>

Figure 1 on page 211 shows how these symbols are related.
Here are definitions of the properties associated with these symbols.

**address_class**
contains the location for a variety of objects such as a `func`, `global_var`, and a `tls_global`.

**aggregate_kind**
One of the following: `struct`, `class`, or `union`.

**artificial**
A Boolean (0 or 1) value where true indicates that the compiler generated the symbol.

**compiler_kind**
The compiler or family of compiler used to create the file; for example, `gnu`, `xlc`, `intel`, and so on.
data_addressing
Contains additional operands to get from the base of an object to its data; for example, a Fortran by-desc array contains a descriptor data structure. The variable points to the descriptor. If you do an addc operation on the descriptor, you can then do an indirect operation to locate the data.

Figure 2 – Data Addressing

![Diagram of data addressing]

delayed_symbol
Indicates if a symbol has been full or partially read-in. The following constants are or'd and returned: skim, index, line, and full.

demangler
The name of demangler used by your compiler.

element_addressing
The location containing additional operands that let you go from the data's base location to an element.

enumerators
Name of the enumerator tags. For example, if you have something like enum[R,G,B], the tags would be R, G, and B.

external_name
When used in data types, it translates the object structure to the type name for the language. For example, if you have a pointer that points to an int, the external name is int *.

full_pathname
This is the # separated static path to the variable; for example, ##image#file#externalname....

id
The internal object handle for the symbol. These symbols always take the form number | number.

index_type_index
The array type's index type_index; for example, this indicates if the index is a 16-, 32-, 64-bit, and so on.

inheritance
For C++ variables, this string is as follows: [ virtual ] [ { private | protected | public } ] [ base class ]
**is_argument**
A true/false value indicating if a variable was a parameter (dummy variable) passed into the function.

**kind**
One of the symbol types listed in the first column of the previous table.

**language**
A string containing a value such as C, C++, or Fortran.

**length**
The byte size of the object. For example, this might represent the size of an array or a subroutine.

**location**
The location in memory where an object's storage begins.

**logical_scope_owner**
The current scope's owner as defined by the language's rules.

---

Figure 3 – Logical Scope Owner

---

**lookup_scope**
This is a pathname reference symbol that refers to the scope in which to look up a pathname.

**lower_bound**
The location containing the array's lower bound. This is a numeric value, not the location of the first array item.

**ordinal**
The order in which a member or variable occurred within a scope.
qualification
A qualifier to a data type such as `const` or `volatile`. These can be chained together if there is more than one qualifier.

**Figure 4 – Qualification**

```
volatile const int
```

resolved_symbol_id
The soid to lookup in a soid reference symbol.

resolved_symbol_pathname
The pathname to lookup in a Fortran reference symbol.

return_type_index
The data type of the value returned by a function.

scope_owner
The ID of the symbol's scope owner. (This is illustrated by the figure within the `logical_scope_owner` definition.)

static_chain
The location of a static link for nested subroutines.

static_chain_height
For nested subroutines, this indicates the nesting level.

stride_bound
Location of the value indicating an array's stride.

submembers
If you have an array of aggregates or pointers and you have already dived on it, this property gives you a list of `{name type}` tuples where name is the name of the member of the array (or `*` if it's an array of pointers), and type is the soid of the type that should be used to dive in all into that field.

target_type_index
The type of the following entities: `array`, `ds undiscovered_type`, `pointer`, and `typedef`.

type_index
One of the following: `member`, `variable`, or `named_constant`.

upper_bound
The location of the value indicating an array's upper bound or extent.

validator
The name of an array or pointer validator. This looks at an array descriptor or pointer to determine if it is allocated and associated.
value
For enumerators, this indicates the item’s value in hexadecimal bytes.

value_size
For enumerators, this indicates the length in bytes

Symbol Namespaces
The symbols described in the previous section all reside within namespaces. Like symbols, namespaces also have properties. Table 1 lists the properties associated with a namespace. Figure 5 on page 216 illustrates how these namespaces are related.

Table 2: Namespace Properties

<table>
<thead>
<tr>
<th>Symbol Namespaces</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>block_symname</td>
<td>base_name</td>
</tr>
<tr>
<td>c_global_symname</td>
<td>base_name loader_name</td>
</tr>
<tr>
<td></td>
<td>loader_file_path</td>
</tr>
<tr>
<td>c_local_symname</td>
<td>base_name</td>
</tr>
<tr>
<td>c_type_symname</td>
<td>base_name type_index</td>
</tr>
<tr>
<td>cplusplus_global_symname</td>
<td>base_name cplusplus_template_types</td>
</tr>
<tr>
<td></td>
<td>cplusplus_class_name cplusplus_type_name</td>
</tr>
<tr>
<td></td>
<td>cplusplus_local_name loader_file_path</td>
</tr>
<tr>
<td></td>
<td>cplusplus_overload_list loader_name</td>
</tr>
<tr>
<td>cplusplus_local_symname</td>
<td>base_name cplusplus_overload_list</td>
</tr>
<tr>
<td></td>
<td>cplusplus_class_name cplusplus_template_types</td>
</tr>
<tr>
<td></td>
<td>cplusplus_local_name cplusplus_type_name</td>
</tr>
<tr>
<td>cplusplus_type_symname</td>
<td>base_name cplusplus_template_types</td>
</tr>
<tr>
<td></td>
<td>cplusplus_class_name cplusplus_type_name</td>
</tr>
<tr>
<td></td>
<td>cplusplus_local_name type_index</td>
</tr>
<tr>
<td></td>
<td>cplusplus_overload_list</td>
</tr>
<tr>
<td>file_symname</td>
<td>base_name directory_path</td>
</tr>
<tr>
<td></td>
<td>directory_hint</td>
</tr>
<tr>
<td>fortran_global_symname</td>
<td>base_name loader_file_path</td>
</tr>
<tr>
<td></td>
<td>fortran_module_name loader_name</td>
</tr>
<tr>
<td></td>
<td>fortran_parent_function_name</td>
</tr>
<tr>
<td>fortran_local_symname</td>
<td>base_name fortran_parent_function_name</td>
</tr>
</tbody>
</table>
Table 2: Namespace Properties

<table>
<thead>
<tr>
<th>Symbol Namespaces</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>fortran_module_name</td>
<td>fortran_parent_function_name</td>
</tr>
<tr>
<td>fortran_type_symname</td>
<td>base_name</td>
</tr>
<tr>
<td>fortran_module_name</td>
<td>type_index</td>
</tr>
<tr>
<td>image_symname</td>
<td>base_name</td>
</tr>
<tr>
<td></td>
<td>member_name</td>
</tr>
<tr>
<td>directory_path</td>
<td>node_name</td>
</tr>
<tr>
<td>label_symname</td>
<td>base_name</td>
</tr>
<tr>
<td>linenumber_symname</td>
<td>linenumber</td>
</tr>
<tr>
<td>loader_symname</td>
<td>loader_file_path</td>
</tr>
<tr>
<td>module_symname</td>
<td>base_name</td>
</tr>
<tr>
<td>type_symname</td>
<td>type_index</td>
</tr>
</tbody>
</table>

Figure 5 – Namespace Architecture
Many of the following properties are used in more than one namespace. The explanations for these properties will assume a limited context as their use is similar. Some of these definitions assume that you’re are looking at the following function prototype:

```c
void c::foo<int>(int &)
```

- **base_name**: The name of the function; for example, `foo`.
- **cplus_class_name**: The C++ class name; for example, `c`.
- **cplus_local_name**: Not used.
- **cplus_overload_list**: The function’s signature; for example, `int &`.
- **cplus_template_types**: The template used to instantiate the function; for example: `<int>`.
- **cplus_type_name**: The data type of the returned value; for example, `void`.
- **directory_hint**: The directory to which you were attached when you started TotalView.
- **directory_path**: Your file’s pathname as it is named within your program.
- **fortran_module_name**: The name of your module. Typically, this looks like `module'var` or `module'subr'var`.
- **fortran_parent_function_name**: The parent of the subroutine. For example, the parent is `module` in a reference such as `module'subr`. If you have an inner subroutine, the parent is the outer subroutine.
- **linenumber**: The line number at which something occurred.
- **loader_file_path**: The file’s pathname.
- **loader_name**: The mangled name.
- **member_name**: In a library, you might have an object reference; for example, `libC.a(foo.so)`. `foo.so` is the member name.
- **node_name**: Not used.
type_index

A handle that points to the type definition. Its format is `<number,number,number>`.
thread

Gets and sets thread properties

**Format**

TV::thread *action* [ *object-id* ] [ *other-args* ]

**Arguments**

*action*

The action to perform, as follows:

*commands*

Displays the subcommands that you can use. The CLI responds by displaying these four *action* subcommands. Do not use other arguments with this option.

*get*

Gets the values of one or more thread properties. The *other-args* argument can include one or more property names. The CLI returns these values in a list, and places them in the same order as the names you enter.

If you use the `-all` option instead of *object-id*, the CLI returns a list containing one (sublist) element for each object.

*properties*

Lists an object’s properties. Do not use other arguments with this option.

*set*

Sets the values of one or more properties. The *other-args* argument contains paired property names and values.

*object-id*

A thread ID. If you use the `-all` option, the operation is carried out on all threads in the current focus.

*other-args*

Arguments required by the *get* and *set* subcommands.

**Description**

The TV::thread command lets you examine and set the following thread properties and states:

*canonical_executable_name*

The absolute file name of the program being debugged. If you had entered a relative name, TotalView find this absolute name.

*continue_sig*

The signal to pass to a thread the next time it runs. On some systems, the thread receiving the signal might not always be the one for which this property was set.

*current_ap_id*

The ID of the action point at which the current thread is stopped.
dpid
The ID of the process associated with a thread.

duid
The internal unique ID associated with the thread.

held
A Boolean value (either 1 or 0) indicating if the thread is held. (1 means that the thread is held.) (settable)

id
The ID of the thread.

manager
A Boolean value (either 1 or 0) indicating if this is a system manager thread. (1 means that it is a system manager thread.)

pc
The current PC at which the target is executing. (settable)

sp
The value of the stack pointer.

state
The current state of the target. See state_values for a list of states.

state_values
A list of values for the state property: break, error, exited, running, stopped, and watch.

stop_reason_message
The reason why the current thread is stopped; for example, Stop Signal.

systid
The system thread ID.

target_architecture
The machine architecture upon which the current thread is executing.

target_byte_ordering
The bit ordering of the current machine. This is either little_endian or big_endian.

target_processor
The kind of processor upon which the current thread is executing. For example, this could be x86 or x86-64.

Examples
f p3 TV::thread get -all id
Returns a list of thread IDs for process 3; for example:
1.1 1.2 1.4

proc set_signal {val} {
TV::thread set \
[f t TV::focus_threads] continue_sig $val
Set the starting signal for the focus thread.

```tcl
proc show_signal {} {
    foreach th [TV::focus_threads] {
        puts "Continue_sig ($th): \n             [TV::thread get $th continue_sig]";
    }
}
```

Show all starting signals

**RELATED TOPICS**

- Using Groups, Processes, and Threads in the TotalView for HPC User Guide
- `focus_threads` Command
- `group` Command
- `process` Command
type

Gets and sets type properties

Format

TV::type action [ object-id ] [ other-args ]

Arguments

action

The action to perform, as follows:

commands

Displays the subcommands that you can use. The CLI responds by displaying these four action subcommands. Do not use other arguments with this option.

get

Gets the values of one or more type properties. The other-args argument can include one or more property names. The CLI returns these values in a list, and places them in the same order as the names you enter.

If you use the -all option instead of object-id, the CLI returns a list containing one (sublist) element for each object.

properties

Lists a type's properties. Do not use other arguments with this option.

set

Sets the values of one or more type properties. The other-args argument contains paired property names and values.

object-id

An identifier for an object; for example, 1 represents process 1, and 1.1 represents thread 1 in process 1. If you use the -all option, the operation is carried out on all objects of this class in the current focus.

other-args

Arguments required by the get and set subcommands.

Description

The TV::type command lets you examine and set the following type properties and states:

enum_values

For an enumerated type, a list of (name value) pairs giving the definition of the enumeration. If you apply this to a non-enumerated type, the CLI returns an empty list.

id

The ID of the object.

image_id

The ID of the image in which this type is defined.
language
    The language of the type.

length
    The length of the type.

name
    The name of the type; for example, class foo.

prototype
    The ID for the prototype. If the object is not prototyped, the returned value is {}.

rank
    (array types only) The rank of the array.

struct_fields
    (class/struct/union types only). A list of lists that contains descriptions of all the type's fields. Each sublist contains the following fields:
    { name type_id addressing properties }
    where:
        name is the name of the field.
        type_id is simply the type_id of the field.
        addressing contains additional addressing information that points to the base of the field.
        properties contains an additional list of properties in the following format:
        “[virtual] [public|private|protected] base class”
    If no properties apply, this string is null.
    If you use get struct_fields for a type that is not a class, struct, or union, the CLI returns an empty list.

target
    For an array or pointer type, returns the ID of the array member or target of the pointer. For commands without this argument applied to one of these types, the CLI returns an empty list.

type
    Returns a string describing this type; for example, signed integer.

type_values
    Returns all possible values for the type property.

Examples

TV::type get 1|25 length target

Finds the length of a type and, assuming it is a pointer or an array type, the target type. The result might look something like:

4 1|12
The following example uses the **TV::type properties** command to obtain the list of properties. It begins by defining a procedure:

```bash
proc print_type {id} {
    foreach p [TV::type properties] {
        puts [format "%-13s %s" $p [TV::type get $id $p]]
    }
}
```

You then display information with the following command:

```bash
print_type 1|6
```

```
enum_values
id                1|6
image_id          1|1
language          f77
length            4
name              <integer>
prototype         
rank              0
struct_fields     
target            
type              Signed Integer
type_values       {Array} {Array of characters} {Enumeration}...
```
type_transformation

Format

TV::type_transformation action [ object-id ] [ other-args ]

Arguments

action

The action to perform, as follows:

commands
Displays the subcommands that you can use. The CLI responds by displaying the subcommands shown here. Do not use additional arguments with this subcommand.

create
Creates a new transformation object. The object-id argument is not used; other-args is Array, List, Map, Set, Umap, Uset or Struct, indicating the type of transformation being created. You can change a transformation's properties up to the time you install it. After being installed, you can longer change them.

get
Gets the values of one or more transformation properties. The other-args argument can include one or more property names. The CLI returns these property values in a list whose order is the same as the property names you entered.

If you use the -all option instead of object-id, the CLI returns a list containing one (sublist) element for the object.

properties
Displays the properties that the CLI can access. Do not use additional arguments with this option. These properties are discussed later in this section.

set
Sets the values of one or more properties. The other-args argument consists of pairs of property names and values. The argument pairs that you can set are listed later in this section.

object-id
The type transformation ID. This value is returned when you create a new transformation; for example, 1 represents process 1. If you use the -all option, the operation executes upon all objects of this class in the current focus.

other-args

Arguments required by get and set subcommands.

Description

The TV::type_transformation command lets you define and examine properties of a type transformation. The states and properties you can set are:
Common Properties

id
The type transformation ID returned from a create operation.

language
The language property specifies source language for the code of the aggregate type (class) to transform. This is always C++.

name
Contains a regular expression that checks to see if a symbol is eligible for type transformation. This regular expression must match the definition of the aggregate type (class) being transformed.

type_callback
The type_callback property is used in two ways.

1. When it is used within a list or vector transformation, it names the procedure that determines the type of the list or vector element. The callback procedure takes one parameter, the symbol ID of the symbol that was validated during the callback to the procedure specified by the validate_callback. The call structure for this callback is:

    type_callback id

    where id is the symbol ID of the symbol that was validated using the validate_callback procedure.

2. When it is used within a struct transformation, it names the procedure that specifies the data type to be used when displaying the struct.

type_transformation_description
A string containing a description of what is being transformed; for example, you might enter “GNU Vector”.

validate_callback
Names a procedure that is called when a data type matches the regular expression specified in the name property. The call structure for this callback is:

    validate_callback id

    where id is the symbol ID of the symbol being validated.

Your callback procedure should check the symbol’s structure to insure that it should be transformed. While not required, most users will extract symbol information such as its type and its data members while validating the datatype. The callback procedure must return a Boolean value, where true means the symbol is valid and can be transformed.

compiler
Reserved for future use.

Array Properties

addressing_callback
Names the procedure that locates the address of the start of an array. The call structure for this callback is:

    addressing_callback id
where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

This callback defines a TotalView addressing expression that computes the starting address of an array's first element.

**lower_bounds_callback**

Names the procedure that obtains a lower bound value for the array type being transformed. For C/C++ arrays, this value is always 0. The call structure for this callback is:

\begin{verbatim}
lower_bounds_callback \textit{id}
\end{verbatim}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

**upper_bounds_callback**

Names the procedure that defines an addressing expression that computes the extent (number of elements) in an array. The call structure for this callback is:

\begin{verbatim}
upper_bounds_callback \textit{id}
\end{verbatim}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

**List Properties**

**list_element_count_addressing_callback**

Names the procedure that determines the total number of elements in a list. The call structure for this callback is:

\begin{verbatim}
list_element_count_addressing_callback \textit{id}
\end{verbatim}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

This callback defines an addressing expression that specifies how to get to the member of the symbol that specifies the number of elements in the list.

If your data structure does not have this element, you still must use this callback. In this case, simply return \{nop\} as the addressing expression and the transformation will count the elements by following all the pointers. This can be very time consuming.

**list_element_data_addressing_callback**

Names the procedure that defines an addressing expression that specifies how to access the data member of a list element. The call structure for this callback is:

\begin{verbatim}
list_element_data_addressing_callback \textit{id}
\end{verbatim}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

**list_element_next_addressing_callback**

Names the procedure that defines an addressing expression that specifies how to access the next element of a list. The call structure for this callback is:

\begin{verbatim}
list_element_next_addressing_callback \textit{id}
\end{verbatim}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.
list_element_prev_addressing_callback
Names the procedure that defines an addressing expression that specifies how to access the previous element of a list. The call structure for this callback is:

```
list_element_prev_addressing_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

This property is optional. For example, you would not use it in a singly linked list.

list_end_value
Specifies if a list is terminated by NULL or the head of the list. Enter one of the following: NULL or ListHead

list_first_element_addressing_callback
Names the procedure that defines an addressing expression that specifies how to go from the head element of the list to the first element of the list. It is not always the case that the head element of the list is the first element of the list. The call structure for this callback is:

```
list_first_element_addressing_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

list_head_addressing_callback
Names the procedure that defines an addressing expression to obtain the head element of the linked list. The call structure for this callback is:

```
list_head_addressing_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

Struct Properties

struct_member_count_callback
Names the procedure that obtains the total number of members in a struct. The call structure for this callback is:

```
struct_member_count_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

struct_member_addressing_callback
Names the procedure that defines an addressing expression that specifies how to access the specified member of a struct. The call structure for this callback is:

```
struct_member_addressing_callback id index
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure and `index` specifies the zero-based position of the member within the struct.

struct_member_type_callback
Names the procedure that obtains the type id of the specified member of a struct. The call structure for this callback is:

```
struct_member_type_callback id index
```
where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure and \textit{index} specifies the zero-based position of the member within the struct.

\textbf{struct_member_name_callback}

Names the procedure that obtains the name of the specified member of a struct. The call structure for this callback is:

\texttt{struct_member_name_callback \textit{id} \textit{index}}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure and \textit{index} specifies the zero-based position of the member within the struct.

\textbf{Red/Black Tree Properties}

The implementation of map/multimap and set/multiset STL types uses red/black trees. These properties are common to all these types.

\textbf{rbtree_head_addressing_callback}

Names the procedure that defines an addressing expression to obtain the head element of the map. The call structure for this callback is:

\texttt{rbtree_head_addressing_callback \textit{id}}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

\textbf{rbtree_head_type_callback}

Names the procedure that obtains the type ID of the head of a map. The call structure for this callback is:

\texttt{rbtree_head_type_callback \textit{id}}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

\textbf{rbtree_element_left_addressing_callback}

Names the procedure that defines an addressing expression that specifies how to access the left sub-tree of the current element of a map. The call structure for this callback is:

\texttt{rbtree_element_left_addressing_callback \textit{id}}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

\textbf{rbtree_element_right_addressing_callback}

Names the procedure that defines an addressing expression that specifies how to access the right sub-tree of the current element of a map. The call structure for this callback is:

\texttt{rbtree_element_right_addressing_callback \textit{id}}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

\textbf{rbtree_element_parent_addressing_callback}

Names the procedure that defines an addressing expression that specifies how to access the parent of the current element of a map. The call structure for this callback is:

\texttt{rbtree_element_parent_addressing_callback \textit{id}}

where \textit{id} is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.
**rbtree_element_count_addressing_callback**

Names the procedure that determines the total number of elements in a map. The call structure for this callback is:

```plaintext
rbtree_element_count_addressing_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

This callback defines an addressing expression that specifies how to get to the member of the symbol that specifies the number of elements in the map.

If your data structure does not have this element, you still must use this callback. In this case, simply return `{nop}` as the addressing expression and the transformation will count the elements by following all the pointers. Unfortunately, this can be very time consuming.

**rbtree_element_count_type_callback**

Names the procedure that obtains the type ID of the member that specifies the number of elements in the map. The call structure for this callback is:

```plaintext
rbtree_element_count_type_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

If your data structure does not have a count element, this property is not required.

**rbtree_left_most_addressing_callback**

Names the procedure that defines an addressing expression to obtain the left-most element of the map. The call structure for this callback is:

```plaintext
rbtree_left_most_addressing_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

---

**Map/Multimap Properties**

For map and multimap STL types these properties are used in combination with those for red/black trees above.

**map_element_key_data_addressing_callback**

Names the procedure that defines an addressing expression that specifies how to access the key of an element of a map. The call structure for this callback is:

```plaintext
map_element_key_data_addressing_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

**map_element_key_type_callback**

Names the procedure that obtains the type ID of the key of a map. The call structure for this callback is:

```plaintext
map_element_key_type_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

**map_element_type_callback**

Names the procedure that obtains the type ID of the element in the red/black tree that contains the key/value pair. The call structure for this callback is:
map_element_type_callback id
where id is the symbol ID of the symbol that was validated using the validate_callback procedure.

map_element_value_data_addressing_callback
Names the procedure that defines an addressing expression that specifies how to access the value of an element of a map. The call structure for this callback is:

map_element_value_data_addressing_callback id
where id is the symbol ID of the symbol that was validated using the validate_callback procedure.

map_element_value_type_callback
Names the procedure that obtains the type id of the value of a map. The call structure for this callback is:

map_element_value_type_callback id
where id is the symbol ID of the symbol that was validated using the validate_callback procedure.

map_iterator_end_value
Specifies if a map is terminated by NULL or the head of the map. Enter one of the following: NULL or MapHead

Set/Multiset Properties
For set and multiset STL types these properties are used in combination with those for red/black trees above.

set_element_data_addressing_callback
Names the procedure that defines an addressing expression that specifies how to access an element of a set. The call structure for this callback is:

set_element_data_addressing_callback id
where id is the symbol ID of the symbol that was validated using the validate_callback procedure.

set_element_type_callback
Names the procedure that obtains the type id of an element in the set. The call structure for this callback is:

set_element_type_callback id
where id is the symbol ID of the symbol that was validated using the validate_callback procedure.

set_iterator_end_value
Specifies if a set is terminated by NULL or the head of the set. Enter one of the following: NULL or SetHead

Hashtable Properties
The implementations of unordered map/multimap and unordered set/multiset STL types use hash tables. These properties are common to all these types.

hashtable_head_addressing_callback
Names the procedure that defines an addressing expression to obtain the head element of the map. Depending on the implementation, this element may be the address of the bucket list or the beginning element of a forward list. The call structure for this callback is:

hashtable_head_addressing_callback id
where \( id \) is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

\textbf{hashtable\_element\_count\_addressing\_callback}

Names the procedure that determines the total number of elements in a hashtable. The call structure for this callback is:

\begin{verbatim}
hashtable_element_count_addressing_callback id
\end{verbatim}

where \( id \) is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

This callback defines an addressing expression that specifies how to get to the member of the symbol that specifies the number of elements in the map.

\textbf{hashtable\_element\_count\_type\_callback}

Names the procedure that obtains the type ID of the member that specifies the number of elements in the map. The call structure for this callback is:

\begin{verbatim}
hashtable_element_count_type_callback id
\end{verbatim}

where \( id \) is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

\textbf{hashtable\_element\_addressing\_callback}

Names the procedure that defines an addressing expression that specifies how to access the next element. The call structure for this callback is:

\begin{verbatim}
hashtable_element_addressing_callback id
\end{verbatim}

where \( id \) is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

\textbf{hashtable\_begin\_index\_addressing\_callback}

Names the procedure that determines the index of the first used bucket in a hashtable. The call structure for this callback is:

\begin{verbatim}
hashtable_begin_index_addressing_callback id
\end{verbatim}

where \( id \) is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

This callback defines an addressing expression that specifies how to get to the member of the symbol that specifies the first used bucket in the hashtable. This allows a small optimization since the transformation can skip empty buckets at the start of the bucket table. If your data does not supply this value you can use \texttt{{nop}}.

\textbf{hashtable\_begin\_index\_type\_callback}

Names the procedure that determines the type of the value that contains the index of the first used bucket in a hashtable. The call structure for this callback is:

\begin{verbatim}
hashtable_begin_index_type_callback id
\end{verbatim}

where \( id \) is the symbol ID of the symbol that was validated using the \texttt{validate_callback} procedure.

\textbf{hashtable\_bucket\_count\_addressing\_callback}

Names the procedure that determines the total number of buckets in a hash table. The call structure for this callback is:

\begin{verbatim}
hashtable_bucket_count_addressing_callback id
\end{verbatim}
where \( id \) is the symbol ID of the symbol that was validated using the `validate_callback` procedure. This callback defines an addressing expression that specifies how to get to the member of the symbol that specifies the number of buckets in a hashtable.

This property can be `{nop}` when the hash table elements can be found without scanning the bucket list, for example, when the elements are also stored in a forward list.

**hashtable_bucket_count_type_callback**

Names the procedure that obtains the type id of the member that specifies the number of buckets in a hash table. The call structure for this callback is:

```
hashtable_bucket_count_type_callback id
```

where \( id \) is the symbol ID of the symbol that was validated using the `validate_callback` procedure. If you are not scanning the bucket list for the hashed values, this property is not required.

**Unordered Map/Multimap Properties**

For unordered map and unordered multimap STL types these properties are used in combination with those for hash tables above.

**umap_element_key_data_addressing_callback**

Names the procedure that defines an addressing expression that specifies how to access the key of an element of a map. The call structure for this callback is:

```
umap_element_key_data_addressing_callback id
```

where \( id \) is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

**umap_element_key_type_callback**

Names the procedure that obtains the type id of the key of a map. The call structure for this callback is:

```
umap_element_key_type_callback id
```

where \( id \) is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

**umap_element_type_callback**

Names the procedure that obtains the type id of the element in the hashtable that contains the key/value pair. The call structure for this callback is:

```
umap_element_type_callback id
```

where \( id \) is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

**umap_element_value_data_addressing_callback**

Names the procedure that defines an addressing expression that specifies how to access the value of an element of a map. The call structure for this callback is:

```
umap_element_value_data_addressing_callback id
```

where \( id \) is the symbol ID of the symbol that was validated using the `validate_callback` procedure.
**umap_element_value_type_callback**

Names the procedure that obtains the type id of the value of a map. The call structure for this callback is:

```
umap_element_value_type_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

**Unordered Set/Multiset Properties**

For unordered set and unordered multiset STL types these properties are used in combination with those for hash tables above.

**uset_element_key_data_addressing_callback**

Names the procedure that defines an addressing expression that specifies how to access an element of a set. The call structure for this callback is:

```
uset_element_key_data_addressing_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.

**uset_element_key_type_callback**

Names the procedure that obtains the type id of an element in the set. The call structure for this callback is:

```
uset_element_key_type_callback id
```

where `id` is the symbol ID of the symbol that was validated using the `validate_callback` procedure.
Batch Debugging Using tvscript

Overview

Batch debug programs by starting TotalView using the `tvscript` command, which allows TotalView to run unattended. If you invoke `tvscript` using `cron`, you can schedule debugging for a certain time, for instance in the evening, so reports are available in the morning.

To perform complex actions, use a script file, which can contain CLI and Tcl commands. Here, for example, is how `tvscript` is invoked on a program:

```bash
tvscript \
   -create_actionpoint "method1=>display_backtrace -show_arguments" \ 
   -create_actionpoint "method2#37=>display_backtrace \       -show_locals -level 1" \ 
   -display_specifiers "noshow_pid,noshow_tid" \   -maxruntime "00:00:30" \ 
   filterapp -a 20
```

You can also execute MPI programs using `tvscript`. Here is a small example:

```bash
tvscript -mpi "Open MP" -tasks 4 \ 
   -create_actionpoint \ 
   "hello.c#14=>display_backtrace" \ 
   ~/tests/MPI_hello
```

This chapter discusses `tvscript` command-line options.
tvscript Command Syntax

The syntax for the tvscript command is:

```
tvscript [ options ] [ filename ] [-a program_args ]
```

**options**
TotalView and tvscript command-line options. You can use any options described in Chapter 7, “TotalView Command Syntax,” on page 338. tvscript command-line options are described in the next section.

**filename**
The program being debugged.

**-a program_args**
Program arguments.

The command-line options most often used with tvscript are:

- **-mpi** (The MPI environments supported are those listed in the Parallel tab of the File > New Program dialog box.)
- **-starter_args**
- **-nodes**
- **-np** or **-procs** or **-tasks**

For more information on these command-line options, see Chapter 7, “TotalView Command Syntax,” on page 338.

**Blue Gene/L and Blue Gene/P**

The syntax for using tvscript with an MPI on Blue Gene/L and Blue Gene/P systems is:

```
tvscript [ options ] -mpi BlueGene -np number-of-processes -starter_args "filename [ mpi-arguments ] [-args program_args ]" mpirun
```

**-np**
The number of processes or tasks that the starter program will create.

**-starter_args**
Required, with the arguments following enclosed in quotes; the application executable (filename) to be debugged must be the first argument.

**-mpi-arguments**
The command arguments for mpirun, such as "-cwd", "-mode", and "-partition".

**-args**
Command argument for mpirun that passes to the launched application on the compute node.
BATCH DEBUGGING USING TVSCRIPT

**Blue Gene/Q with SLURM**

The syntax for using tvscript with an MPI on Blue Gene/Q systems using SLURM is:

```bash
mpirun
```

Required; the executable at the end of the command line.

```
tvscript [ options ] -mpi BlueGeneQ-SLURM -np number-of-processes -starter_args [program_args]" srun
```

- **np**
  The number of processes or tasks that the starter program will create.

- **starter_args**
  Required, with the arguments following enclosed in quotes; the application executable (filename) to be debugged must follow the arguments for srun.

```
srun-arguments
```

The command arguments for srun.

```
filename
```

The program being debugged.

```
program_args
```

The arguments for the program being debugged.

```
srun
```

Required; the executable at the end of the command line.

**Blue Gene/Q for ANL's Cobalt Job Manager and IBM's runjob**

The syntax for using tvscript on Blue Gene/Q for ANL's Cobalt job manager and IBM's runjob: is

```
tvscript [ options ] -mpi BlueGeneQ-Cobalt -np number-of-processes -starter_args [runjob-arguments]:filename [program_args]"
```

- **np**
  The number of processes or tasks that the starter program will create.

- **starter_args**
  Required, with the arguments following enclosed in quotes; the application executable (filename) to be debugged must follow the arguments for runjob and be separated by a colon (:) .

```
runjob-arguments
```

The command arguments for runjob.

```
filename
```

The program being debugged.

```
program_args
```

The arguments for the program being debugged.
runjob
   Required; the executable at the end of the command line.

**Blue Gene/Q for the LoadLeveler job manager and IBM’s runjob**

The syntax for using `tvscript` on Blue Gene/Q for the LoadLeveler job manager and IBM’s `runjob` is:

```
tvscript [ options ] -mpi BlueGeneQ-LoadLeveler -np number-of-processes -starter_args "[runjob-arguments]" --exe filename [program_args]" runjob
```

- `runjob`
  The number of processes or tasks that the starter program will create.

- `starter_args`
  Required, with the arguments following enclosed in quotes; the application executable (`filename`) to be debugged must follow the arguments for `runjob` and `--exe`.

  `runjob-arguments`
  The command arguments for `runjob`.

- `filename`
  The program being debugged.

- `program_args`
  The arguments for the program being debugged.

**Cray Xeon Phi**

The syntax for using `tvscript` on Cray Xeon Phi Knights Corner (KNC) native nodes is:

```
tvscript [ options ] -mpi CrayKNC-aprun -np number-of-processes -starter_args "[aprun-arguments] filename [program_args]" aprun
```

- `aprun`
  The number of processes or tasks that the starter program will create.

- `starter_args`
  Required, with the arguments following enclosed in quotes; the application executable (`filename`) to be debugged must follow the arguments for `aprun`.

  `aprun-arguments`
  The command arguments for `aprun` (except the `-k` argument).

- `filename`
  The program being debugged.

- `program_args`
  The arguments for the program being debugged.
aprun
   Required; the executable at the end of the command line.

For example:

tvscript \n   -create_actionpoint "tx_basic_mpi.c#98=>display_backtrace
      -show_arguments, print myid" \n   -mpi CrayKNC-aprun -np 16 \n   -starter_args "tx_basic_mpi" \n   aprun

Cray XK7

The syntax for using tvscript on Cray XK7 is:

tvscript [ options ] -mpi CrayXK7-aprun -np number-of-processes -starter_args
   "[aprun-arguments] filename [program_args]" aprun

   -np
   The number of processes or tasks that the starter program will create.

   -starter_args
   Required, with the arguments following enclosed in quotes; the application executable (filename) to be de-
   bugged must follow the arguments for aprun.

aprun-arguments
   The command arguments for aprun.

filename
   The program being debugged.

program_args
   The arguments for the program being debugged.

aprun
   Required; the executable at the end of the command line.

For example:

tvscript \n   -create_actionpoint "tx_basic_mpi.c#98=>display_backtrace
      -show_arguments, print myid" \n   -mpi CrayXK7-aprun -np 16 \n   -starter_args "tx_basic_mpi" \n   aprun
tvscript Options

- **create_actionpoint** "source_location_expr \[ = >\text{action1}, \text{action2}\ldots\]"

  Creates an action point at a source location using an expression. (See “Action Point API” on page 246 for writing expressions.) When the action point is hit, tvscript can trigger one or more actions. Add one

- **create_watchpoint** command-line option for each action point.

  See **event_action** for information about actions.

- **event_action** "event_action_list"

  Performs an action when an event occurs. Events represent an unanticipated condition, such as **free_not_allocated** in the Memory Debugger. You can use more than one **event_action** command-line option when invoking tvscript.

  Here is how you enter an **event_action_list**:

```plaintext
  event1 = action1, event2 = action2
```

  or

```plaintext
  event1 => action1, action2, action3
```

Table 3: Supported tvscript Events

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>General event</td>
<td>any_event</td>
<td>A generated event occurred.</td>
</tr>
<tr>
<td>Memory debugging event</td>
<td>addr_not_at_start</td>
<td>Program attempted to free a block using an incorrect address.</td>
</tr>
<tr>
<td></td>
<td>alloc_not_in_heap</td>
<td>The memory allocator returned a block not in the heap; the heap may be corrupt.</td>
</tr>
<tr>
<td></td>
<td>alloc_null</td>
<td>An allocation either failed or returned NULL; this usually means that the system is out of memory.</td>
</tr>
<tr>
<td></td>
<td>alloc_returned_bad_alignment</td>
<td>The memory allocator returned a misaligned block; the heap may be corrupt.</td>
</tr>
<tr>
<td></td>
<td>any_memory_event</td>
<td>A memory event occurred.</td>
</tr>
<tr>
<td></td>
<td>bad_alignment_argument</td>
<td>Program supplied an invalid alignment argument to the heap manager.</td>
</tr>
<tr>
<td></td>
<td>double_alloc</td>
<td>The memory allocator returned a block currently being used; the heap may be corrupt.</td>
</tr>
<tr>
<td></td>
<td>double_dealloc</td>
<td>Program attempted to free an already freed block.</td>
</tr>
<tr>
<td></td>
<td>free_not_allocated</td>
<td>Program attempted to free an address that is not in the heap.</td>
</tr>
<tr>
<td></td>
<td>guard_corruption</td>
<td>Program overwrote the guard areas around a block.</td>
</tr>
</tbody>
</table>
### Table 3: Supported tvscript Events

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory debugging actions</td>
<td><strong>check_guard_blocks</strong></td>
<td>Checks all guard blocks and write violations into the log file.</td>
</tr>
<tr>
<td></td>
<td><strong>list_allocations</strong></td>
<td>Writes a list of all memory allocations into the log file.</td>
</tr>
<tr>
<td></td>
<td><strong>list_leaks</strong></td>
<td>Writes a list of all memory leaks into the log file.</td>
</tr>
<tr>
<td></td>
<td><strong>save_html_heap_status_source_view</strong></td>
<td>Generates and saves an HTML version of the Heap Status Source View Report.</td>
</tr>
<tr>
<td></td>
<td><strong>save_memory_debugging_file</strong></td>
<td>Generates and saves a memory debugging file.</td>
</tr>
<tr>
<td></td>
<td><strong>save_text_heap_status_source_view</strong></td>
<td>Generates and saves a text version of the Heap Status Source View Report.</td>
</tr>
<tr>
<td>Source code debugging event</td>
<td><strong>actionpoint</strong></td>
<td>A thread hit an action point.</td>
</tr>
<tr>
<td></td>
<td><strong>error</strong></td>
<td>An error occurred.</td>
</tr>
</tbody>
</table>

For each occurring event, define the action to perform:

<table>
<thead>
<tr>
<th>Action Type</th>
<th>Action</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory debugging actions</td>
<td><strong>check_guard_blocks</strong></td>
<td>Checks all guard blocks and write violations into the log file.</td>
</tr>
<tr>
<td></td>
<td><strong>list_allocations</strong></td>
<td>Writes a list of all memory allocations into the log file.</td>
</tr>
<tr>
<td></td>
<td><strong>list_leaks</strong></td>
<td>Writes a list of all memory leaks into the log file.</td>
</tr>
<tr>
<td></td>
<td><strong>save_html_heap_status_source_view</strong></td>
<td>Generates and saves an HTML version of the Heap Status Source View Report.</td>
</tr>
<tr>
<td></td>
<td><strong>save_memory_debugging_file</strong></td>
<td>Generates and saves a memory debugging file.</td>
</tr>
<tr>
<td></td>
<td><strong>save_text_heap_status_source_view</strong></td>
<td>Generates and saves a text version of the Heap Status Source View Report.</td>
</tr>
<tr>
<td>Action Type</td>
<td>Action</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>------------</td>
</tr>
</tbody>
</table>
| Source code debugging actions | display_backtrace [-level level-num ] [ num_levels ] [ options ] | Writes the current stack backtrace into the log file. 
- **level**: sets the level at which information starts being logged. 
- **num_levels**: restricts output to this number of levels in the call stack. 
If you do not set a level, **tvscript** displays all levels in the call stack. 
- **options** is one or more of the following: 
  - ![image](https://via.placeholder.com/150) `show_arguments` 
  - ![image](https://via.placeholder.com/150) `show_fp` 
  - ![image](https://via.placeholder.com/150) `show_fp_registers` 
  - ![image](https://via.placeholder.com/150) `show_image` 
  - ![image](https://via.placeholder.com/150) `show_locals` 
  - ![image](https://via.placeholder.com/150) `show_pc` 
  - ![image](https://via.placeholder.com/150) `show_registers` 

| | print [-slice {slice_exp}] {variable | exp} | Writes the value of a variable or an expression into the log file. If the variable is an array, the **slice** option limits the amount of data defined by slice_exp. A slice expression is a way to define the slice, such as **var[100:130]** in C and C++. (This displays all values from **var[100]** to **var[130]**.) To display every fourth value, add an additional argument; for example, **var[100:130:4]**. For additional information, see "Examining Arrays". |
-display_specifiers "display_specifiers_list"

By default, tvscript writes all of the information in the following table to the log file. You can exclude information by using one of the following specifiers:

<table>
<thead>
<tr>
<th>Type of Specifier</th>
<th>Specifier</th>
<th>Display ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>General display specifiers</td>
<td>no_show_fp</td>
<td>Does not show the frame pointer (FP)</td>
</tr>
<tr>
<td></td>
<td>no_show_image</td>
<td>Does not show the process/library in backtrace</td>
</tr>
<tr>
<td></td>
<td>no_show_pc</td>
<td>Does not show the program counter (PC)</td>
</tr>
<tr>
<td></td>
<td>no_show_pid</td>
<td>Does not show the system process ID with process information</td>
</tr>
<tr>
<td></td>
<td>no_show_rank</td>
<td>Does not show the rank of a process, which is shown only for a parallel process</td>
</tr>
<tr>
<td></td>
<td>no_show_tid</td>
<td>Does not show the thread ID with process information</td>
</tr>
<tr>
<td></td>
<td>no_show_allocator</td>
<td>Does not show the allocator for the address space</td>
</tr>
<tr>
<td></td>
<td>no_show_backtrace</td>
<td>Does not show the backtraces for memory blocks</td>
</tr>
<tr>
<td></td>
<td>no_show_backtrace_id</td>
<td>Does not show the backtrace ID for memory blocks</td>
</tr>
<tr>
<td></td>
<td>no_show_block_address</td>
<td>Does not show the memory block start and end addresses</td>
</tr>
<tr>
<td></td>
<td>no_show_flags</td>
<td>Does not show the memory block flags</td>
</tr>
<tr>
<td></td>
<td>no_show_guard_id</td>
<td>Does not show the guard ID for memory blocks</td>
</tr>
<tr>
<td></td>
<td>no_show_guard_settings</td>
<td>Does not show the guard settings for memory blocks</td>
</tr>
<tr>
<td></td>
<td>no_show_leak_stats</td>
<td>Does not show the leaked memory block statistics</td>
</tr>
<tr>
<td></td>
<td>no_show_owner</td>
<td>Does not show the owner of the allocation</td>
</tr>
<tr>
<td></td>
<td>no_show_red_zones_settings</td>
<td>Does not show the Red Zone entries for allocations (and deallocations) for the address space</td>
</tr>
</tbody>
</table>

-memory_debugging

Enables memory debugging and memory event notification. This option is required with any option that begins with -mem. These options are TotalView command line options, as they can be invoked directly by TotalView.

-mem_detect_leaks

Performs leak detection before generating memory information.

-mem_detect_use_after_free

Tests for use after memory is freed.

-mem_guard_blocks

Adds guard blocks to an allocated memory block.
-mem_hoard_freed_memory
   Holds onto freed memory rather than returning it to the heap.

-mem_hoard_low_memory_threshold nnnn
   Sets the low memory threshold amount. When memory falls below this amount, an event is fired.

-mem_paint_all
   Paints memory blocks with a bit pattern when a memory is allocated or deallocated.

-mem_paint_on_alloc
   Paints memory blocks with a bit pattern when a memory block is allocated.

-mem_paint_on_dealloc
   Paints memory blocks with a bit pattern when a memory block is deallocated.

-mem_red_zones_overruns
   Turns on testing for Red Zone overruns.

-mem_red_zones_size_ranges min:max,min:max,...
   Defines the memory allocations ranges for which Red Zones are in effect. Ranges can be specified as follows:
   x:y allocations from x to y
   :y allocations from 1 to y
   x: allocations of x and higher
   x allocation of x

-mem_red_zones_underruns
   Turns on testing for Red Zone underruns.

-maxruntime "hh:mm:ss"
   Specifies how long the script can run.

-script_file script_file
   Names a file containing tvscript API calls and Tcl callback procedures that you create.

-script_log_filename logFilename
   Overrides the name of the TVScript log file.
   WARNING: Previous log files of the same name are overwritten.

-script_summary_log_filename summaryLogFilename
   Overrides the name of the TVScript summary log file.
   WARNING: Previous summary log files with the same name are overwritten.

**tvscript Example:**

The following example is similar to that shown in “Batch Debugging Using tvscript” on page 235.

```
tvscript \
   -create_actionpoint "method1=>display_backtrace -show_arguments" \ 
   -create_actionpoint "method2#37=>display_backtrace \ 
```
This script performs the following actions:

- Creates an action point at the beginning of method1. When tvscript reaches that breakpoint, it logs a backtrace and the method's arguments.
- Creates an action point at line 37 of method2. When tvscript reaches this line, it logs a backtrace and the local variables. The backtrace information starts at level 1.
- Logs the backtrace, the current routine's arguments, and its local variables when an error event occurs.
- Excludes the process ID and thread ID from the information that tvscript logs.
- Limits tvscript execution time to 30 seconds.
- Names the program being debugged and passes a value of 20 to the application.
tvscript External Script Files

The section tvscript Command Syntax discussed the command-line options used when invoking the tvscript command. You can also place commands in a file and provide them to tvscript using the -script_file command-line option. Using a script file supports the use of Tcl to create more complex actions when events occur. The following sections describe the functions that you can use within a CLI file.

Logging Functions API

   tvscript_log msg
   Logs a message to the log file set up by tvscript.

   tvscript_slog msg
   Logs a message to the summary log file set up by tvscript.

Process Functions API

   tvscript_get_process_property process_id property
   Gets the value of a property about the process.

   The properties you can name are the same as those used with the TV::process command. See process on page 197 for more information.

Thread Functions API

   tvscript_get_thread_property thread_id property
   Gets the value of a property about the thread.

   The properties you can name are the same as those used with the TV::thread command. See thread on page 219 for more information.

Action Point API

   tvscript_add_actionpoint_handler actionpoint_id actionpoint_handler
   Registers a procedure handler to call when the action point associated with actionpoint_id is hit. This actionpoint_id is the value returned from the tvscript_create_actionpoint routine. The value of actionpoint_handler is the string naming the procedure.

   When tvscript calls an action point handler procedure, it passes one argument. This argument contains a list that you must convert into an array. The array indices are as follows:

   event—The event that occurred, which is the action point
process_id—The ID of the process that hit the action point
thread_id—The ID of the thread that hit the action point
actionpoint_id—The ID of the action point that was hit
actionpoint_source_loc_expr—The initial source location expression used to create the action point

tvscript_create_actionpoint source_loc_expr
Creates an action point using a source location expression.
This procedure returns an action point ID that you can use in a tvscript_add_actionpoint_handler procedure.

source_loc_expr
Sets a breakpoint at the line specified by source_loc_expr or an absolute address. For example:

- `[[##image##]filename#]line_number`
  Indicates all addresses at this line number.

- A function signature; this can be a partial signature.
  Indicates all addresses that are the addresses of functions matching signature. If parts of a function signature are missing, this expression can match more than one signature. For example, “`f`” matches “`f(void)`” and “`A::f(int)`”.
  You cannot specify a return type in a signature.

You can also enter a source location expression with sets of addresses using the class and virtual keywords. For example:

- **class** class_name
  Names a set containing the addresses of all member functions of class class_name.

- **virtual** class::signature
  Names the set of addresses of all virtual member functions that match signature, and that are in the classes or derived from the class.

If the expression evaluates to a function that has multiple overloaded implementations, TotalView sets a barrier on each of the overloaded functions.

Event API

tvscript_add_event_handler event event_handler
Registers a procedure handler to call when the named event occurs. The event is either error or actionpoint.
When tvscript calls an event handler procedure, it passes one argument to it. This argument contains a list that you must convert into an array.

error
When any error occurs, the array has the following indices:
event—The event, which is set to error
process_id—The ID of the process that hit the action point
thread_id—The ID of the thread that hit the action point
error_message—A message describing the error that occurred

actionpoint
When any action point is hit, the array has the following indices:

- **event**—The event, which is set to *actionpoint*
- **process_id**—The ID of the process that hit the action point
- **thread_id**—The ID of the thread that hit the action point
- **actionpoint_id**—The ID of the action point that was hit
- **actionpoint_source_loc_expr**—The initial source location expression used to create the action point

**Example tvscript Script File**

The following information is passed to **tvscript** as follows:

```
tvscript -script_file script_file
```

This script installs an error handler and an action point handler. When an error is encountered during execution, **tvscript** passes an array of information to the error handler. Similarly, when an action point is hit, it passes an array of information to the action point handler. These arrays are described in “Event API” on page 247.

```bash
# Get the process so we have some information about it
tvscript_log "PID: \n [tvscript_get_process_property 1 "syspid"]";

tvscript_log "Status: \n [tvscript_get_process_property 1 "state"]";

tvscript_log "Executable: \n [tvscript_get_process_property 1 "executable"]";

# Proc error_handler {error_data} {
proc error_handler {error_data} {
    tvscript_log "Inside error_handle: $error_data"

    # Change the incoming list into an array.
    # It contains the following indices:
    #    process_id
    #    thread_id
    #    error_message
    array set error_data_array $error_data

    # Get the process so we have some information about it
    temp = [tvscript_get_process_property 
```
$error_data_array(process_id) "syspid"

    tvscript_log "Process ID: $temp"

    temp = [tvscript_get_thread_property 
        $error_data_array(thread_id) "systid"

    tvscript_log "Thread ID: $temp"

    temp = $error_data_array(error_message)
    tvscript_log "Error Message: $temp"

}

########################################################################
# Action point handlers

proc l1_actionpoint_handler {event_data} {
    tvscript_log "Inside l1_actionpnt_handler: $event_data"
    tvscript_slog "Inside l1_actionpnt_handler: $event_data"

    # Change the incoming list into an array.
    # It contains the following indices:
    #   actionpoint_id
    #   actionpoint_source_loc_expr
    #   event
    #   process_id
    #   thread_id
    array set event_data_array $event_data

    # Get the process so we have some information about it
    temp = [tvscript_get_process_property 
        $event_data_array(process_id) "syspid"

    tvscript_log "Process ID: $temp"

    temp = [tvscript_get_thread_property 
        $event_data_array(thread_id) "systid"

    tvscript_log "Thread ID: $temp"

    temp = [tvscript_get_process_property 
        $event_data_array(process_id) "state"

    tvscript_log "Status: $temp"

    temp = [tvscript_get_process_property 
        $event_data_array(process_id) "executable"

    tvscript_log "Executable: $temp"

    temp = $event_data_array(actionpoint_source_loc_expr)
    tvscript_log "Action point Expression: $temp"
tvscript_log "Value of i:"
set output [capture "dprint i"]
tvscript_log $output
}

### Event handlers

proc generic_actionpoint_event_handler {actionpoint_data} {
    tvscript_log "Inside generic_actionpoint_event_handler: ">
    tvscript_log $actionpoint_data
    tvscript_slog "Inside generic_actionpoint_event_handler: "
    tvscript_slog $actionpoint_data

    # Change the incoming list into an array.
    # It contains the following indices:
    #   actionpoint_id
    #   actionpoint_source_loc_expr
    #   event
    #   process_id
    #   thread_id
    array set actionpnt_data_array $actionpoint_data
    temp = $actionpnt_data_array(process_id)
tvscript_log "            Process ID: $temp"

    temp = $actionpnt_data_array(thread_id)
tvscript_log "            Thread ID: $temp"

    temp = $actionpnt_data_array(actionpoint_id)
tvscript_log "       Action Point ID: $temp"

    temp = $actionpnt_data_array(actionpoint_source_loc_expr)
tvscript_log "Action Point Expression: "
}

### Add event handlers

### Add event handlers

# Setup action points and action point handlers
set actionpoint_id [tvscript_create_actionpoint "l1"]
tvscript_add_actionpoint_handler $actionpoint_id \  "l1_actionpoint_handler"

# Setup a generic actionpoint handler
tvscript_add_event_handler "actionpoint" \  "generic_actionpoint_event_handler"
# Add error handler
tvscript_add_event_handler "error" "error_handler"
Overview

This chapter contains a list of all CLI and TotalView variables, organized into sections that each correspond to a CLI namespace:

- Top-Level (::) Namespace
- TV:: Namespace
- TV::MEMDEBUG:: Namespace
- TV::GUI:: Namespace
Top-Level (::) Namespace

**ARGS(\text{dpmid})**
Contains the arguments to be passed the next time the process starts, with TotalView ID \text{dpid}.

- **Permitted Values**: A string
- **Default**: None

**ARGS_DEFAULT**
Contains the argument passed to a new process when no \text{ARGS(\text{dpmid})} variable is defined.

- **Permitted Values**: A string
- **Default**: None

**BARRIER_STOP_ALL**
Contains the value for the “stop\_when\_done” property for newly created action points. This property defines additional elements to stop when a barrier point is satisfied or a thread encounters this action point. You can also set this value using the **When barrier hit, stop** value in the **Action Points** Page of the **File > Preferences** dialog box. The values are:

- **group**: Stops all processes in a thread's control group when a thread reaches a barrier created using this default.
- **process**: Stops the process in which the thread is running when a thread reaches a barrier created using this default.
- **none**: Stops only the thread that hit a barrier created using this default.

This variable is the same as the **TV::barrier_stop_all** variable.

- **Permitted Values**: group, process, or thread
- **Default**: group

**BARRIER_STOP_WHEN_DONE**
Contains the default value used when a barrier point is satisfied. You can also set this value using the **-stop\_when\_done** command-line option or the **When barrier done, stop** value in the **Action Points** Page of the **File > Preferences** dialog box. The values are:

- **group**: When a barrier is satisfied, stops all processes in the control group.
- **process**: When a barrier is satisfied, stops the processes in the satisfaction set.
none

Stops only the threads in the satisfaction set; other threads are not affected. For process barriers, there is no difference between process and none.

In all cases, TotalView releases the satisfaction set when the barrier is satisfied.

This variable is the same as the TV::barrier_stop_when_done variable.

Permitted Values: group, process, or thread

Default: group

CGROUP(dp)<id>

Contains the control group for the process with the TotalView ID dp<id>. Setting this variable moves process dp<id> into a different control group. For example, the following command moves process 3 into the same group as process 1:

```dset CGROUP(3) $CGROUP(1)
```

Permitted Values: A number

Default: None

COMMAND_EDITING

Enables some Emacs-like commands for use when editing text in the CLI. These editing commands are always available in the CLI window of the TotalView GUI. However, they are available only in the stand-alone CLI if the terminal in which it is running supports cursor positioning and clear-to-end-of-line. The commands you can use are:

^A: Moves the cursor to the beginning of the line

^B: Moves the cursor one character backward

^D: Deletes the character to the right of cursor

^E: Moves the cursor to the end of the line

^F: Moves the cursor one character forward

^K: Deletes all text to the end of line

^N: Retrieves the next entered command (only works after ^P)

^P: Retrieves the previously entered command

^R or ^L: Redraws the line

^U: Deletes all text from the cursor to the beginning of the line

Rubout or Backspace: Deletes the character to the left of the cursor

Permitted Values: true or false

Default: false
**EXECUTABLE_PATH**

Contains a colon-separated list of the directories searched for source and executable files.

*Permitted Values:* Any directory or directory path. To include the current setting, use `$EXECUTABLE_PATH`.

*Default:* . (dot)

**EXECUTABLE_SEARCH_MAPPINGS**

Contains pairs of regular expressions and replacement and replacement strings—these replacements are called mappings—separated by colons. TotalView applies these mappings to the search paths before it looks for source, object, and program files.

The syntax for mapping strings is:

```
+regular_exp+:+replacement+:+regular_exp+:+replacement+
```

This example shows two pairs, each delimited by a colon (":"). Each element within a pair is delimited by any character except a colon. The first character entered is the delimiter. This example uses a "+" as a delimiter. (Traditionally, forward slashes are used as delimiters but are not used here, as a forward slash is also used to separate components of a pathname. For example, `/home/my_dir` contains forward slashes.)

Be aware that special characters must follow standard Tcl rules and conventions, for example:

```
dset EXECUTABLE_SEARCH_MAPPINGS  {+[^/nfs/compiled/u2/(.*)$]+ = +/nfs/host/u2/\1+ }
```

This expression applies a mapping so that a directory named `/nfs/compiled/u2/project/src1` in the expanded search path becomes `/nfs/host/u2/project/src1`.

*Default:* {}

**EXECUTABLE_SEARCH_PATH**

Contains a list of paths, separated by a colon, to search for executables. For information, see “Setting Search Paths Using TotalView for HPC Variables” in the TotalView for HPC in-product help.

*Permitted Values:* Any directory or directory path.

*Default:* `${EXECUTABLE_PATH}:${PATH}:.`

**GROUP( gid )**

Contains a list of the TotalView IDs for all members in group *gid*.

The first element indicates the type of group:

- **control**
  - The group of all processes in a program

- **lockstep**
  - A group of threads that share the same PC

- **process**
  - A user-created process group
share
   The group of processes in one program that share the same executable image

thread
   A user-created thread group

workers
   The group of worker threads in a program

Elements that follow are either pids (for process groups) or pid.tid pairs (for thread groups).

The gid is a simple number for most groups. In contrast, a lockstep group’s ID number is of the form pid.tid. Thus, GROUP(2.3) contains the lockstep group for thread 3 in process 2. Note, however, that the CLI does not display lockstep groups when you use dset with no arguments because they are hidden variables.

The GROUP(id) variable is read-only.

   Permitted Values: A Tcl array of lists indexed by the group ID. Each entry contains the members of one group.
   Default: None

GROUPS
Contains a list of all TotalView groups IDs. Lockstep groups are not contained in this list. This is a read-only value and cannot be set.

   Permitted Values: A Tcl list of IDs.

LINES_PER_SCREEN
Defines the number of lines shown before the CLI stops printing information and displays its more prompt. The following values have special meaning:

   0
   No more processing occurs, and the printing does not stop when the screen fills with data.

   NONE
   A synonym for 0

   AUTO
   The CLI uses the tty settings to determine the number of lines to display. This may not work in all cases. For example, Emacs sets the tty value to 0. If AUTO works improperly, you need to explicitly set a value.

   Permitted Values: A positive integer, or the AUTO or NONE strings
   Default: Auto

MAX_LEVELS
Defines the maximum number of levels that the dwhere command displays.

   Permitted Values: A positive integer
   Default: 512
**MAX_LIST**
Defines the number of lines that the dlist command displays.

- **Permitted Values:** A positive integer
- **Default:** 20

**OBJECT_SEARCH_MAPPINGS**
Contains pairs of regular expressions and replacement and replacement strings (called mappings) separated by colons. TotalView applies these mappings to the search paths when searching for source, object, and program files. For more information, see EXECUTABLE_SEARCH_MAPPINGS.

- **Default:** \{

**OBJECT_SEARCH_PATH**
Contains a list of paths separated by a colon to search for your program's object files. For information, see “Search Path Variables That You Can Set” in the TotalView for HPC in-product help.

- **Permitted Values:** Any directory or directory path.
- **Default:** ${COMPILATION_DIRECTORY}: ${EXECUTABLE_PATH}: ${EXECUTABLE_DIRECTORY}: $links${EXECUTABLE_DIRECTORY}): ..:${TOTALVIEW_SRC}

**PROCESS(dpid)**
Contains a list of information associated with a dpid. This is a read-only value and cannot be set.

- **Permitted Values:** An integer
- **Default:** None

**PROMPT**
Defines the CLI prompt. Any information within brackets ([ ]) is assumed to be a Tcl command, so therefore evaluated before the prompt string is created.

- **Permitted Values:** Any string. To access the value of PTSET, place the variable within brackets; that is, [dset PTSET].
- **Default:** {[dfocus]> }

**PTSET**
Contains the current focus. This is a read-only value and cannot be set.

- **Permitted Values:** A string
- **Default:** d1.<

**SGROUP(pid)**
Contains the group ID of the share group for process pid. The share group is determined by the control group for the process and the executable associated with this process. You cannot directly modify this group.

- **Permitted Values:** A number
SHARE_ACTION_POINT

Indicates the scope for newly created action points. In the CLI, this is the dbarrier, dbreak, and dwatch commands. If this boolean value is true, newly created action point are shared across the group; if false, a newly created action point is active only in the process in which it is set.

As an alternative to setting this variable, you can select the Plant in share group check box in the Action Points Page in the File > Preferences dialog box. To override this value in the GUI, use the Plant in share group check-box in the Action Point > Properties dialog box.

Permitted Values: true or false

Default: true

SHARED_LIBRARY_SEARCH_MAPPINGS

Contains pairs of regular expressions and replacement strings (mappings), separated by colons. TotalView applies these mappings to the search paths before it looks for shared library files.

Default: {}

SHARED_LIBRARY_SEARCH_PATH

Contains a list of paths, each separated by a colon, to search for your program's shared library files.

Permitted Values: Any directory or directory path.

Default: ${EXECUTABLE_PATH}:

SOURCE_SEARCH_MAPPINGS

Contains pairs of regular expressions and replacement strings (mappings) separated by colons. TotalView applies these mappings to the search paths before it looks for source, object, and program files. For more information, see EXECUTABLE_SEARCH_MAPPINGS.

Default: {}

SOURCE_SEARCH_PATH

Contains a list of paths, separated by a colon, to search for your program's source files. For information, see “Search Path Variables That You Can Set” in the TotalView for HPC in-product help.

Permitted Values: Any directory or directory path.

Default: ${COMPILATION_DIRECTORY}: ${EXECUTABLE_PATH}: ${EXECUTABLE_DIRECTORY}: ${links(${EXECUTABLE_DIRECTORY})}: .:${TOTALVIEW_SRC}

STOP_ALL

Indicates a default property for newly created action points, defining additional elements to stop when this action point is encountered.

Permitted Values: Any directory or directory path.

Default: None

Indicates the scope for newly created action points. In the CLI, this is the dbarrier, dbreak, and dwatch commands. If this boolean value is true, newly created action point are shared across the group; if false, a newly created action point is active only in the process in which it is set.

As an alternative to setting this variable, you can select the Plant in share group check box in the Action Points Page in the File > Preferences dialog box. To override this value in the GUI, use the Plant in share group check-box in the Action Point > Properties dialog box.

Permitted Values: true or false

Default: true

SHARED_LIBRARY_SEARCH_MAPPINGS

Contains pairs of regular expressions and replacement strings (mappings), separated by colons. TotalView applies these mappings to the search paths before it looks for shared library files.

Default: {}

SHARED_LIBRARY_SEARCH_PATH

Contains a list of paths, each separated by a colon, to search for your program's shared library files.

Permitted Values: Any directory or directory path.

Default: ${EXECUTABLE_PATH}:

SOURCE_SEARCH_MAPPINGS

Contains pairs of regular expressions and replacement strings (mappings) separated by colons. TotalView applies these mappings to the search paths before it looks for source, object, and program files. For more information, see EXECUTABLE_SEARCH_MAPPINGS.

Default: {}

SOURCE_SEARCH_PATH

Contains a list of paths, separated by a colon, to search for your program's source files. For information, see “Search Path Variables That You Can Set” in the TotalView for HPC in-product help.

Permitted Values: Any directory or directory path.

Default: ${COMPILATION_DIRECTORY}: ${EXECUTABLE_PATH}: ${EXECUTABLE_DIRECTORY}: ${links(${EXECUTABLE_DIRECTORY})}: .:${TOTALVIEW_SRC}

STOP_ALL

Indicates a default property for newly created action points, defining additional elements to stop when this action point is encountered.

Permitted Values: Any directory or directory path.

Default: None

Indicates the scope for newly created action points. In the CLI, this is the dbarrier, dbreak, and dwatch commands. If this boolean value is true, newly created action point are shared across the group; if false, a newly created action point is active only in the process in which it is set.

As an alternative to setting this variable, you can select the Plant in share group check box in the Action Points Page in the File > Preferences dialog box. To override this value in the GUI, use the Plant in share group check-box in the Action Point > Properties dialog box.

Permitted Values: true or false

Default: true

SHARED_LIBRARY_SEARCH_MAPPINGS

Contains pairs of regular expressions and replacement strings (mappings), separated by colons. TotalView applies these mappings to the search paths before it looks for shared library files.

Default: {}

SHARED_LIBRARY_SEARCH_PATH

Contains a list of paths, each separated by a colon, to search for your program's shared library files.

Permitted Values: Any directory or directory path.

Default: ${EXECUTABLE_PATH}:

SOURCE_SEARCH_MAPPINGS

Contains pairs of regular expressions and replacement strings (mappings) separated by colons. TotalView applies these mappings to the search paths before it looks for source, object, and program files. For more information, see EXECUTABLE_SEARCH_MAPPINGS.

Default: {}

SOURCE_SEARCH_PATH

Contains a list of paths, separated by a colon, to search for your program's source files. For information, see “Search Path Variables That You Can Set” in the TotalView for HPC in-product help.

Permitted Values: Any directory or directory path.

Default: ${COMPILATION_DIRECTORY}: ${EXECUTABLE_PATH}: ${EXECUTABLE_DIRECTORY}: ${links(${EXECUTABLE_DIRECTORY})}: .:${TOTALVIEW_SRC}

STOP_ALL

Indicates a default property for newly created action points, defining additional elements to stop when this action point is encountered.
process
   Stops the entire process when the action point is hit

thread
   Stops only the thread that hit the action point. Note that none is a synonym for thread

Permitted Values: group, process, or thread

Default: process

TAB_WIDTH
Indicates the number of spaces used to simulate a tab character when the CLI displays information.

Permitted Values: A positive number. A value of -1 indicates that the CLI does not simulate tab expansion.

Default: 8

THREADS(pid)
Contains a list of all threads in the process pid, in the form {pid.1 pid.2 ...}. This is a read-only variable and cannot be set.

Permitted Values: A Tcl list

Default: None

TOTALVIEW_ROOT_PATH
Names the directory containing the TotalView executable. This is a read-only variable and cannot be set. This variable is exported as TVROOT, and can be used in launch strings.

Permitted Values: The location of the TotalView installation directory

TOTALVIEW_TCLLIB_PATH
Contains a list of the directories in which the CLI searches for TCL library components.

Permitted Values: Any valid directory or directory path. To include the current setting, use $TOTALVIEW_TCLLIB_PATH.

Default: The directory containing the CLI's Tcl libraries

TOTALVIEW_VERSION
Contains the version number and the type of computer architecture upon which TotalView is executing. This is a read-only variable and cannot be set.

Permitted Values: A string containing the platform and version number

Default: Platform-specific
**VERBOSE**

Sets the error message information displayed by the CLI:

- **info**
  - Prints errors, warnings, and informational messages. Informational messages include data on dynamic libraries and symbols.

- **warning**
  - Prints only errors and warnings.

- **error**
  - Prints only error messages.

- **silent**
  - Does not print error, warning, and informational messages. This also shuts off printing results from CLI commands. This should be used only when the CLI is run in batch mode.

**Permitted Values:** info, warning, error, and silent

**Default:** info

**WGROUP(pid)**

The group ID of the thread group of worker threads associated with the process `pid`. This variable is read-only.

**Permitted Values:** A number

**Default:** None

**WGROUP(pid.tid)**

Contains one of the following:

- The group ID of the workers group in which thread `pid.tid` is a member
- 0 (zero), which indicates that thread `pid.tid` is not a worker thread

Storing a nonzero value in this variable marks a thread as a worker. In this case, the returned value is the ID of the workers group associated with the control group, regardless of the actual nonzero value assigned to it.

**Permitted Values:** A number representing the `pid.tid`

**Default:** None
**TV:: Namespace**

**TV::aix_use_fast_ccw**
This variable is defined only on AIX, and is a synonym for the platform-independent variable `TV::use_fast_wp`, providing TotalView script backward compatibility. See `TV::use_fast_wp` for more information.

**TV::aix_use_fast_trap**
This variable is defined only on AIX, and is a synonym for the platform-independent variable `TV::use_fast_trap`, for TotalView script backward compatibility. See `TV::use_fast_trap` for more information.

**TV::ask_on_cell_spu_image_load**
If true, TotalView might ask whether to stop the process when a Cell SPU image is loaded. If false, TotalView does not stop execution when a Cell SPU image is loaded.

  *Permitted Values:* true or false
  *Default:* true

**TV::ask_on_dlopen**
If true, TotalView asks about stopping processes that use the `dlopen` or `load` (AIX only) system calls dynamically load a new shared library.

If false, TotalView does not ask about stopping a process that dynamically loads a shared library.

  *Permitted Values:* true or false
  *Default:* true

**TV::auto_array_cast_bounds**
Indicates the number of array elements to display when the `TV::auto_array_cast_enabled` variable is true. This is the variable set by the Bounds field of the Pointer Dive Page in the File > Preferences dialog box.

  *Permitted Values:* An array specification
  *Default:* [10]

**TV::auto_array_cast_enabled**
When true, TotalView automatically dereferences a pointer into an array. The number of array elements is indicated in the `TV::auto_array_cast_bounds` variable. This is the variable set by the Cast to array with bounds checkbox of the Pointer Dive Page in the File > Preferences dialog box.

  *Permitted Values:* true or false
  *Default:* false
**TV::auto_deref_in_all_c**

Defines if and how to dereference C and C++ pointers when performing a View > Dive in All operation, as follows:

- **yes_dont_push**
  - While automatic dereferencing will occur, does not allow use of the Undive command to see the undereferenced value when performing a Dive in All operation.

- **yes**
  - Allows use of the Undive control to see undereferenced values.

- **no**
  - Does not automatically dereference values when performing a Dive in All operation.

This is the variable set when you select the Dive in All element in the Pointer Dive Page of the File > Preferences dialog box.

*Permitted Values:* no, yes, or yes_dont_push

*Default:* no

**TV::auto_deref_in_all_fortran**

Tells TotalView if and how it should dereference Fortran pointers when you perform a Dive in All operation, as follows:

- **yes_dont_push**
  - While automatic dereferencing will occur, does not allow use of the Undive command to see the undereferenced value when performing a Dive in All operation.

- **yes**
  - Allows use of the Undive control to see undereference values.

- **no**
  - Does not automatically dereference values when performing a Dive in All operation.

This is the variable set when you select the Dive in All element in the Pointer Dive Page of the File > Preferences dialog box.

*Permitted Values:* no, yes, or yes_dont_push

*Default:* no

**TV::auto_deref_initial_c**

Tells TotalView if and how it should dereference C pointers when they are displayed, as follows:

- **yes_dont_push**
  - While automatic dereferencing will occur, does not allow use of the Undive command to see the undereferenced value.

- **yes**
  - Allows use of the Undive control to see undeferenced values.
no

Does not automatically dereference values.

This is the variable set when you select the initially element in the Pointer Dive Page of the File > Preferences dialog box.

Permitted Values: no, yes, or yes_dont_push
Default: no

TV::auto_deref_initial_fortran

Defines if and how to dereference Fortran pointers when they are displayed, as follows:

yes_dont_push

While automatic dereferencing will occur, does not allow use of the Undive command to see the underreferenced value.

yes

Allows use of the Undive control to see undeferenced values.

no

Does not automatically dereference values.

This is the variable set when you select the initially element in the Pointer Dive Page of the File > Preferences dialog box.

Permitted Values: no, yes, or yes_dont_push
Default: no

TV::auto_deref_nested_c

Tells TotalView if and how it should dereference C pointers when you dive on structure elements:

yes_dont_push

While automatic dereferencing will occur, you can't use the Undive command to see the undereferenced value.

yes

You will be able to use the Undive control to see undeferenced values.

no

Do not automatically dereference values.

This is the variable set when you select the from an aggregate element in the Pointer Dive Page of the File > Preferences dialog box.

Permitted Values: no, yes, or yes_dont_push
Default: yes_dont_push

TV::auto_deref_nested_fortran

Defines if and how to dereference Fortran pointers when they are displayed:
yes_dont_push

While automatic dereferencing will occur, does not allow use of the Undive command to see the undereferenced value.

yes

Allows use of the Undive control to see undeferenced values.

no

Does not automatically dereference values.

This is the variable set when you select the from an aggregate element in the Pointer Dive Page of the File > Preferences dialog box.

Permitted Values: no, yes, or yes_dont_push
Default: yes_dont_push

TV::auto_load_breakpoints

If true, TotalView automatically loads action points from the file named filename.TVD.v3breakpoints where filename is the name of the file being debugged. If false, breakpoints are not automatically loaded. If you set this to false, you can still load breakpoints using the Action Point > Load All or the dactions -load command.

Permitted Values: true or false
Default: true

TV::auto_read_symbols_at_stop

If false, TotalView does not automatically read symbols if execution stops when the program counter is in a library whose symbols were not read. If true, TotalView reads in loader and debugging symbols. You would set it to false if you have prevented symbol reading using either the TV::dll_read_loader_symbols_only or TV::dll_read_no_symbols variables (or the preference within the GUI) and reading these symbols is both unnecessary and would affect performance.

Permitted Values: true or false
Default: true

TV::auto_save_breakpoints

If true, TotalView automatically writes information about breakpoints to a file named filename.TVD.v3breakpoints, where filename is the name of the file being debugged. Information about watchpoints is not saved.

TotalView writes this information when you exit from TotalView. If you set this variable to false, you can explicitly save this information by using the Action Point > Save All or the dactions -save command.

Permitted Values: true or false
Default: false
**TV::barrier_stop_all**

Contains the value of the “stop_all” property for newly created action points. This property defines additional elements to stop when a thread encounters this action point. You can also set this value using the `-stop_all` command-line option or the *When barrier hit, stop* value in the *Action Points* page of the *File > Preferences* dialog box. The values that you can use are as follows:

- **group**
  - Stops all processes in a thread’s control group when a thread reaches a barrier created using this as a default.

- **process**
  - Stops the process in which the thread is running when a thread reaches a barrier created using this default.

- **thread**
  - Stops only the thread that hit a barrier created using this default.

This variable is the same as the **BARRIER_STOP_ALL** variable.

- **Permitted Values**: group, process, or thread
- **Default**: group

**TV::barrier_stop_when_done**

Contains the value for the “stop_when_done” property for newly created action points. This property defines additional elements to stop when a barrier point is satisfied. You can also set this value using the `-stop_when_done` command-line option or the *When barrier done, stop* value in the *Action Points* page of the *File > Preferences* dialog box. The values you can use are:

- **group**
  - When a barrier is satisfied, stops all processes in the control group.

- **process**
  - When a barrier is satisfied, stops the processes in the satisfaction set.

- **thread**
  - Stops only the threads in the satisfaction set; other threads are not affected. For process barriers, there is no difference between process and none.

In all cases, TotalView releases the satisfaction set when the barrier is satisfied.

This variable is the same as the **BARRIER_STOP_WHEN_DONE** variable.

- **Permitted Values**: group, process, or thread
- **Default**: group

**TV::bluegene_io_interface**

If the Bluegene front-end cannot resolve the network name, you must initialize this variable (or set it as a command-line option). By default, TotalView assumes that it can resolve the address as follows:
front_endHostname-iod

For example, if the front-end hostname is fred, TotalView assumes that the servers are connecting to fred-io.

- **Permitted Values:** A string
- **Default:** none

**TV::bluegene_server_launch_string**

 Defines the launch string used when launching tvdsvr processes on I/O nodes.

- **Permitted Values:** A string
- **Default:** -callback %L -set_pw %P -verbosity %V %F

**TV::bluegene_launch_timeout**

 Specifies the number of seconds to wait to hear back from the TotalView Debugger Server (tvdsvr) after its launch.

- **Permitted Values:** An integer from 1 to 3600 (1 hour)
- **Default:** 240

**TV::bulk_launch_base_timeout**

 Defines the base timeout period used to execute a bulk launch.

- **Permitted Values:** A number from 1 to 3600 (1 hour)
- **Default:** 20

**TV::bulk_launch_enabled**

 If true, uses bulk launch features when automatically launching the TotalView Debugger Server (tvdsvr) for remote processes.

- **Permitted Values:** true or false
- **Default:** false

**TV::bulk_launch_incr_timeout**

 Defines the incremental timeout period to wait for a process to launch when automatically launching the TotalView Debugger Server (tvdsvr) using the bulk server feature.

- **Permitted Values:** A number from 1 to 3600 (1 hour)
- **Default:** 10

**TV::bulk_launch_tmpfile1_header_line**

 Defines the header line used in the first temporary file for a bulk server launch operation.

- **Permitted Values:** A string
- **Default:** None
**TV::bulk_launch_tmpfile1_host_lines**
Defines the host line used in the first temporary file when performing a bulk server launch operation.

**Permitted Values:** A string

**Default:** %R

**TV::bulk_launch_tmpfile1_trailer_line**
Defines the trailer line used in the first temporary file when performing a bulk server launch operation.

**Permitted Values:** A string

**Default:** None

**TV::bulk_launch_tmpfile2_header_line**
Defines the header line used in the second temporary file when performing a bulk server launch operation.

**Permitted Values:** A string

**Default:** None

**TV::bulk_launch_tmpfile2_host_lines**
Defines the host line used in the second temporary file when performing a bulk server launch operation.

**Permitted Values:** A string

**Default:** {tvdsvr -working_directory %D -callback %L -set_pw %P -verbosity %V}

**TV::bulk_launch_tmpfile2_trailer_line**
Defines the trailer line used in the second temporary file when performing a bulk server launch operation.

**Permitted Values:** A string

**Default:** None

**TV::c_type_strings**
If true, uses C type string extensions to display character arrays; when false, uses string type extensions.

**Permitted Values:** true or false

**Default:** true

**TV::cell_spu_image_ignore_regexp**
If set to a non-empty string, and TV::ask_on_cell_spu_image_load is true, TotalView matches the SPU image's name with the regular expression. For a match, TotalView does not ask to stop the process but allows the process to continue running after loading the SPU image.

If the image name does not match this regular expression or the regular expression contained within TV::cell_spu_images_stop_regexp, TotalView asks if it should stop the process, unless you've answered the stop to set breakpoint question by pressing No (or the equivalent from within the CLI).

**Permitted Values:** A regular expression
Default: {}  

**TV::cell_spurs_jm_name**

A string that names the file containing the symbols for the “jm” SPURS job policy module. When TotalView detects an embedded SPURS kernel image being loaded into an SPU context, it extracts the GUIDs of the policy modules from the kernel, and searches for either the default SPU ELF image file, which is `spurs_jm.elf` or the file named by this variable.

**Permitted Values:** An ELF file name  
**Default:** `spurs_jm.elf`

**TV::cell_spurs_kernel_dll_regexp**

Defines a regular expression that matches the image path component name of the SPURS kernel SPU ELF image embedded in the `libspurs.so` DLL.

When TotalView sees a new image loaded into an SPU thread by `libspe` or `libspe2`, it checks if the image path component name matches this variable. If so, TotalView handles the SPURS kernel library in a different way. You may need to change this regular expression to match the name of your SPURS kernel if it is embedded in a shared library other than `libspurs.so` or if the name of the SPURS kernel is different than `spurs_kernel.elf`.

**Permitted Values:** A regular expression  
**Default:** `{/libspurs\.so\(spurs_kernel\.elf@[0-9]+\)$}

**TV::cell_spurs_ss_name**

A string that names the file containing the symbols for the “ss” SPURS system service policy module. When TotalView detects an embedded SPURS kernel image being loaded into an SPU context, it extracts the GUIDs of the policy modules from the kernel, and searches for either the default SPU ELF image file, which is `spurs_tss.elf` or the file named by this variable.

**Permitted Values:** An ELF file name  
**Default:** `spurs_ss.elf`
**TV::cell_spurs_tm_name**

A string that names the file containing the symbols for the “tm” SPURS task policy module. When TotalView detects an embedded SPURS kernel image being loaded into an SPU context, it extracts the GUIDs of the policy modules from the kernel, and searches for either the default SPU ELF image file, which is `spurs_tm.elf` or the file named by this variable.

- **Permitted Values**: An ELF file name
- **Default**: `spurs_tm.elf`

**TV::checksum_libraries**

- **Permitted Values**:  
- **Default**: `auto`

**TV::comline_patch_area_base**

Allocates the patch space dynamically at the given address. See “Allocating Patch Space for Compiled Expressions” in the TotalView for HPC Users Guide.

- **Permitted Values**: A hexadecimal value indicating space accessible to TotalView
- **Default**: `0xffffffffffffffff`

**TV::comline_patch_area_length**

Sets the length of the dynamically allocated patch space to the specified length. See “Allocating Patch Space for Compiled Expressions” in the TotalView for HPC Users Guide.

- **Permitted Values**: A positive number
- **Default**: `0`

**TV::command_editing**

Enables some Emacs-like commands for use while editing text in the CLI. These editing commands are always available in the CLI window of TotalView UI. However, they are available only within the stand-alone CLI if the terminal in which it is running supports cursor positioning and clear-to-end-of-line. The commands that you can use are:

- **^A**: Moves the cursor to the beginning of the line.
- **^B**: Moves the cursor one character backward.
- **^D**: Deletes the character to the right of cursor.
- **^E**: Moves the cursor to the end of the line.
- **^F**: Moves the cursor one character forward.
- **^K**: Deletes all text to the end of line.
^N: Retrieves the next entered command (only works after ^P).

^P: Retrieves the previously entered command.

^R or ^L: Redraws the line.

^U: Deletes all text from the cursor to the beginning of the line.

Rubout or Backspace: Deletes the character to the left of the cursor.

Permitted Values: true or false
Default: false

**TV::compile_expressions**

When true, TotalView enables compiled expressions. If false, TotalView interprets your expression.

On an IBM AIX system, you can use the -aix_use_fast_trap command line option to speed up the performance of compiled expressions. Check the TotalView for HPC Release Notes to determine if your version of the operating system supports this feature.

Permitted Values: true or false
Default: false

**TV::compiler_vars**

(SGI only) When true, TotalView shows variables created by your Fortran compiler as well as the variables in your program. When false (which is the default), TotalView does not show the variables created by your compiler.

SGI 7.2 Fortran compilers write debugging information that describes variables the compiler created to assist in some operations. For example, it could create a variable used to pass the length of character*(*) variables. You might want to set this variable to true if you are looking for a corrupted runtime descriptor.

You can override the value set to this variable in a startup file with these command-line options:

- **-compiler_vars**: sets this variable to true
- **-no_compiler_vars**: sets this variable to false

Permitted Values: true or false
Default: false

**TV::control_c_quick_shutdown**

When true, TotalView kills attached processes and exits. When false, TotalView can sometimes better manage the way it kills parallel jobs when it works with management systems. This has been tested only with SLURM and may not work with other systems.

If you set the **TV::ignore_control_c** variable to true, TotalView ignores this variable.

Permitted Values: true or false
Default: true

---

[Image of RogueWave logo]

TotalView Variables / TV:: Namespace  270
**TV::copyright_string**
A read-only string containing the copyright information displayed when you start the CLI and TotalView.

**TV::cppview**
If true, the C++View facility allows the formatting of program data in a more useful or meaningful form than the concrete representation visible by default when you inspect data in a running program. For more information on using C++View, see “C++View” on page 318.

- **Permitted Values:** true or false
- **Default:** true

**TV::cuda_debugger**
Indicates whether cuda debugging is currently enabled. This is a read-only variable.

- **Permitted Values:** true or false
- **Default:** true

**TV::current_cplus_demangler**
Setting this variable overrides the C++ demangler used by default. Note that this value is ignored unless you also set the value of the TV::force_default_cplus_demangler variable. The following values are supported:

- **gnu:** GNU C++ on Linux Alpha
- **gnu_dot:** GNU C++ Linux x86
- **gnu_v3:** GNU C++ Linux x86
- **kai:** KAI C++
- **kai3_n:** KAI C++ version 3.n
- **kai_4_0:** KAI C++
- **spro:** SunPro C++ 4.0 or 5.2
- **spro5:** SunPro C++ 5.0 or later
- **sun:** Sun CFRONT C++
- **xlc:** IBM XLC/VAC++ compilers
  - **Permitted Values:** A string naming the compiler
  - **Default:** Derived from your platform and information within your program

**TV::current_fortran_demangler**
Setting this variable overrides the Fortran demangler used by default. Note that this value is ignored unless you also set the value of the TV::force_default_f9x_demangler variable. The following values are supported:
• xlf90: IBM Fortran
• fujitsu_f9x: Fujitsu Fortran 9x
• intel: Intel Fortran 9x
• sunpro_f9x_4: Sun ProFortran 4
• sunpro_f9x_5: Sun ProFortran 5

Permitted Values: A string naming the compiler
Default: Derived from your platform and information within your program

**TV::data_format_double**

Defines the format to use when displaying double-precision values. This is one of a series of variables that define how to display data. The format of each is similar:

\{presentation format-1 format-2 format 3\}

*presentation*

Selects which format to use when displaying information. Note that you can display floating point information using **dec**, **hex**, and **oct** formats. You can display integers using **auto**, **dec**, and **sci** formats.

**auto**

Equivalent to the C language's `printf()` function's `%g` specifier. You can use this with integer and floating-point numbers. This format is either **hexdec** or **dechex**, depending upon the programming language being used.

**dec**

Equivalent to the `printf()` function's `%d` specifier. You can use this with integer and floating-point numbers.

**dechex**

Displays information using the **dec** and **hex** formats. You can use this with integers.

**hex**

Equivalent to the `printf()` function's `%x` specifier. You can use this with integer and floating-point numbers.

**hexdec**

Displays information using the **hex** and **dec** formats. You can use this with integer numbers.

**oct**

Equivalent to the `printf()` function's `%o` specifier. You can use this with integer and floating-point numbers.

**sci**

Equivalent to the `printf()` function's `%e` specifier. You can use this with floating-point numbers.
**format**

For integers, `format-1` defines the decimal format, `format-2` defines the hexadecimal format, and `format-3` defines the octal format.

For floating point numbers, `format-1` defines the fixed point display format, `format-2` defines the scientific format, and `format-3` defines the auto (`printf()`'s `%g`) format.

The format string is a combination of the following specifiers:

- `%` A signal indicating the beginning of a format.
- `width` A positive integer. This is the same width specifier used in the `printf()` function.
- `. (period)` A punctuation mark separating the width from the precision.
- `precision` A positive integer. This is the same precision specifier used in the `printf()` function.
- `# (pound)` Displays a 0x prefix for hexadecimal and 0 for octal formats. This isn't used within floating-point formats.
- `0 (zero)` Pads a value with zeros. This is ignored if the number is left-justified. If you omit this character, TotalView pads the value with spaces.
- `- (hyphen)` Left-justifies the value within the field's width.

**Permitted Values:** A value in the described format

**Default:** `{auto %-1.15 %-1.15 %-20.2}`

**TV::data_format_ext**

Defines the format to use when displaying extended floating point values such as long doubles. For a description of the contents of this variable, see `TV::data_format_double`.

**Permitted Values:** A value in the described format

**Default:** `{auto %-1.15 %-1.15 %-1.15}`

**TV::data_format_int8**

Defines the format to use when displaying 8-bit integer values. For a description of the contents of this variable, see `TV::data_format_double`.

**Permitted Values:** A value in the described format

**Default:** `{auto %1.1 %#4.2 %#4.3}`
TV::data_format_int16
Defines the format to use when displaying 16-bit integer values. For a description of the contents of this variable, see TV::data_format_double.

- **Permitted Values:** A value in the described format
- **Default:** {auto %1.1 %#6.4 %#7.6}

TV::data_format_int32
Defines the format to use when displaying 32-bit integer values. For a description of the contents of this variable, see TV::data_format_double.

- **Permitted Values:** A value in the described format
- **Default:** {auto %1.1 %#10.8 %#12.11}

TV::data_format_int64
Defines the format to use when displaying 64-bit integer values. For a description of the contents of this variable, see TV::data_format_double.

- **Permitted Values:** A value in the described format
- **Default:** {auto %1.1 %#18.16 %#23.22}

TV::data_format_int128
Defines the format to use when displaying 128-bit integer values. For a description of the contents of this variable, see TV::data_format_double.

- **Permitted Values:** A value of the described format.
- **Default:** {auto %1.1 %#34.32 %#44.43}

TV::data_format_long_stringlen
Defines the number of characters allowed in a long string.

- **Permitted Values:** A positive integer number
- **Default:** 8000

TV::data_format_single
Defines the format to use when displaying single precision, floating-point values. For a description of the contents of this variable, see TV::data_format_double.

- **Permitted Values:** A value in the described format
- **Default:** {auto %-1.6 %-1.6 %-1.6}

TV::data_format_stringlen
Defines the maximum number of characters displayed for a string.

- **Permitted Values:** A positive integer number
Default: 100

**TV::dbfork**

When `true`, TotalView catches the `fork()`, `vfork()`, and `execve()` system calls if your executable is linked with the `dbfork` library. See “Linking with the dbfork Library” on page 368.

*Permitted Values:* `true` or `false`  
*Default:* `true`

**TV::default_launch_command**

Names the compiled-in launch command appropriate for the platform.

*Permitted Values:* A string indicating the default compiled-in launch command value.  
*Default:* Sun SPARC: `rsh` All other platforms: `ssh -x`

**TV::default_parallel_attach_subset**

Names the default subset specification listing MPI ranks to attach to when an MPI job is created or attached to.

*Permitted Values:* A string indicating the default subset specification.  
*Default:* Initialized to the value specified with the `-default_parallel_attach_subset` command line option.

**TV::default_stderr_append**

When `true`, TotalView appends the target program’s `stderr` information to the file set in the GUI, by the `-stderr` command-line option, or in the `TV::default_stderr_filename` variable. If no pathname is set, the value of this variable is ignored. If the file does not exist, TotalView creates it.

*Permitted Values:* `true` or `false`  
*Default:* `false`

**TV::default_stderr_filename**

Names the file to which to write the target program’s `stderr` information. If the file exists, TotalView overwrites it. If the file does not exist, TotalView creates it.

*Permitted Values:* A string indicating a pathname  
*Default:* None

**TV::default_stderr_is_stdout**

When `true`, TotalView writes the target program’s `stderr` information to the same location as `stdout`.

*Permitted Values:* `true` or `false`  
*Default:* `false`

**TV::default_stdin_filename**

Names the file from which the target program reads `stdin` information.
**Permitted Values:** A string indicating a pathname  
**Default:** None

**TV::default_stdout_append**
When true, TotalView appends the target program’s stdout information to the file set in the GUI, by the -stdout command-line option, or in the TV::default_stdout_filename variable. If no pathname is set, the value of this variable is ignored. If the file does not exist, TotalView creates it.

**Permitted Values:** true or false  
**Default:** false

**TV::default_stdout_filename**
Names the file to which to write the target program’s stdout information. If the file exists, TotalView overwrites it. If the file does not exist, TotalView creates it.

**Permitted Values:** A string indicating a pathname  
**Default:** None

**TV::display_assembler_symbolically**
When true, TotalView displays assembler locations as label+offset. When false, these locations are displayed as hexadecimal addresses.

**Permitted Values:** true or false  
**Default:** false

**TV::dll_ignore_prefix**
Defines a list of library files that will not result in a query to stop the process when loaded. This list contains a colon-separated list of prefixes. Also, TotalView will not ask if you would like to stop a process if:

- You also set the TV::ask_on_dlopen variable to true.
- The suffix of the library being loaded does not match a suffix contained in the TV::dll_stop_suffix variable.
- One or more of the prefixes in this list match the name of the library being loaded.

**Permitted Values:** A list of path names, each item of which is separated from another by a colon  
**Default:** /lib:/usr/lib:/usr/lpp:/usr/ccs/lib:/usr/dt/lib:/tmp/

**TV::dll_read_all_symbols**
Always reads loader and debugging symbols of libraries named within this variable.

This variable is set to a colon-separated list of library names. A name can contain the * (asterisk) and ? (question mark) wildcard characters, which have their usual meaning:

- *: zero or more characters.
•  ?: a single character.

Because this is the default behavior, include only library names here that would be excluded because they are selected by a wildcard match within the TV::dll_read_loader_symbols_only and TV::dll_read_no_symbols variables.

**Permitted Values:** One or more library names separated by colons  
**Default:** None

### TV::dll_read_loader_symbols_only

When TotalView loads libraries named in this variable, it reads only loader symbols. Because TotalView checks and processes the names in TV::dll_read_all_symbols list before it processes this list, it ignores names that are in that list and in this one.

This variable is set to a colon-separated list of strings. Any string can contain the * (asterisk) and ? (question mark) wildcard characters, which have their usual meaning:

-  *: zero or more characters.
-  ?: a single character.

If you do not need to debug most of your shared libraries, set this variable to * and then put the names of any libraries you wish to debug on the TV::dll_read_all_symbols list.

**Permitted Values:** One or more library names separated by colons  
**Default:** None

### TV::dll_read_no_symbols

When TotalView loads libraries named in this variable, it does not read in either loader or debugging symbols. Because TotalView checks and processes the names in the TV::dll_read_loader_symbols_only lists before it processes this list, it ignores names that are in those lists and in this one.

This variable is set to a colon-separated list of strings. Any string can contain the * (asterisk) and ? (question mark) wildcard characters having their usual meaning:

-  *, which means zero or more characters
-  ?, which means a single character.

Because information about subroutines, variables, and file names are not known for these libraries, stack backtraces may be truncated. However, if your program uses large shared libraries and it’s time consuming to read even their loader symbols, you may want to put those libraries on this list.

**Permitted Values:** One or more library names separated by colons  
**Default:** None
**TV::dll_stop_suffix**

Contains a colon-separated list of suffixes that stop the current process when it loads a library file with this suffix.

You must confirm that you want to stop the process:

- If **TV::ask_on_dlopen** variable is set to **true**
- If one or more of the suffixes in this list match the name of the library being loaded.

**Permitted Values:** A Tcl list of suffixes  
**Default:** None

**TV::dlopen_always_recalculate**

When **false**, enables **dlopen** event filtering (see “**dlopen Options for Scalability**” on page 382).

**TV::dlopen_always_recalculate** is **true** by default, meaning that breakpoint specifications are reevaluated on every **dlopen** call. This is referred to as Slow Mode.

A value of **false** enables **dlopen** event filtering, deferring the reevaluation of breakpoint specifications until after the **dlopen** event and thus reducing the number of events per process that TotalView evaluates. This is useful in improving performance when a process loads large numbers of libraries. Depending on the setting of **TV::dlopen_recalculate_on_match**, performance can be improved with the Medium or Fast modes of **dlopen** event filtering.

**Permitted Values:** **true** or **false**  
**Default:** **true**

**TV::dlopen_recalculate_on_match**

Contains a colon-separated list of simple glob patterns (a glob list) containing library names. If **TV::dlopen_always_recalculate** is set to **true**, the value of this variable is ignored.

**glob patterns** specify sets of filenames with wildcard characters. A simple glob pattern is a string, optionally ending with an asterisk character (*).

If **TV::dlopen_always_recalculate** is **false** and a **dlopen** event occurs, the name of the library associated with the event is matched against the list of glob patterns. If the glob-list is empty (default) or the name of the dlopened library does not match any patterns in the glob-list, then breakpoint reevaluation is deferred until the process stops for some other reason (e.g., the process hits a breakpoint, the user stops the process, the process encounters a signal, etc.). If the library name matches a pattern, the breakpoints are reevaluated immediately. A glob-list that contains the empty string results in Fast mode, since all the dlopened libraries will have their breakpoint reevaluation deferred. Medium mode is when select libraries are to have their breakpoints reevaluated immediately.

The matching rules are:

- If the simple glob pattern does not end in an asterisk, then the tail of the loaded library name must match the string. For example, the string "libfoo.so" matches library name "/dir/path/libfoo.so", but does not match "/dir/path/libfoo.so.1.0".
• If the simple glob pattern ends in an asterisk, then the asterisk is removed from the string, and the remaining portion of the string matches any substring found in the library name. For example, the string "libfoo.so*" matches "/dir/path/libfoo.so" or "/dir/path/libfoo.so.1.0", and the string "/path/*" matches "/dir/path/libfoo.so" or "/dir/path/libbar.so".

For a more complete explanation of `dlopen` event filtering, including use-case examples, please refer to “dlopen Options for Scalability” on page 382.

Permitted Values: String
Default: "", the empty string

**TV::dlopen_read_libraries_in_parallel**

When **false**, (the default), TotalView handles `dlopen` events in the target application serially. (Note that for parallel applications, handling `dlopen` events serially can degrade debugger performance.)

When **true**, TotalView attempts to handle `dlopen` events in parallel.

On non-MRNet platforms, or if MRNet is not enabled, then the value of this variable is ignored. For more information, see “Handling dlopen Events in Parallel” in the TotalView for HPC User Guide.

Permitted Values: true or false
Default: false

**TV::dump_core**

When **true**, a core file is created when an internal TotalView error occurs. This is used only when debugging TotalView problems. You can override this variable's value by using the following command-line options:

- `-dump_core` sets this variable to true
- `-no_dumpcore` sets this variable to false

Permitted Values: true or false
Default: false

**TV::dwhere_qualification_level**

Controls the amount of information displayed when you use the `dwhere` command. Here are three examples:

`dset TV::dwhere_qualification_level +overload_list`
`dset TV::dwhere_qualification_level -class_name`
`dset TV::dwhere_qualification_level -parent_function`

You could combine these arguments into one command. For example:

`dset TV::dwhere_qualification_level +overload_list \ -class_name -parent_function`

In these examples "+" means that the information should be displayed and "-" means the information should not be displayed.

The arguments to this command are:

• all
- `class_name`
- `file_directory`
- `hint`
- `image_directory`
- `loader_directory`
- `member`
- `module`
- `node`
- `overload_list`
- `parent_function`
- `template_args`
- `type_name`

The **all** argument is often used as follows:

\[
\text{dset TV::dwhere_qualification_level all-parent_function}
\]

This states that all elements are displayed except for a parent function. For more information on these arguments, see “symbol” on page 206.

**Permitted Values:** One or more of the arguments listed above.

**Default:** `class_name+template_args+module+parent_function+member+node`

**TV::dynamic**

When **true**, TotalView loads symbols from shared libraries. This variable is available on all platforms supported by Rogue Wave Software. (This may not be true for platforms ported by others. For example, this feature is not available for Hitachi computers.) Setting this value to **false** can cause the **dbfork** library to fail because TotalView might not find the **fork()**, **vfork()**, and **execve()** system calls.

**Permitted Values:** true or false

**Default:** true

**TV::editor_launch_string**

Defines the editor launch string command. The launch string substitution characters you can use are:

- **%E:** The editor
- **%F:** The display font
- **%N:** The line number
%S: The source file

Permitted Values: Any string value—as this is a Tcl variable, you'll need to enclose the string within {} (braces) if the string contains spaces

Default: \{xterm -e %E +%N %S\}

TV::env

Names a variable that is already contained within your program’s environment. This is a read-only variable and is set by using the -env command-line option. For more information, see -env variable=value on page 343.

To set this variable from within TotalView, use the File > New Program or Process > Startup dialog boxes.

Permitted Values: None. The variable is read-only.

Default: None

TV::follow_clone

When a value greater than 0, allows TotalView to pickup threads created using the clone() system call. The supported values are:

0: TotalView does not follow clone() calls. This is most often used if problems occur.

1: TotalView follows clone() calls until the first pthread_create() call is made. This value is then set to 0.

2: TotalView follows clone() calls whenever they occur. Calls to clone() and pthread_create() can be interleaved. This may affect performance if the program has many threads.

3: (default) Like 2, TotalView follows clone() calls whenever they occur. However, TotalView uses a feature available on newer Linux systems to reduce the overhead.

NOTE >> Linux threads are not affected by this variable. This variable should be left set at 3 unless you have reason to believe it is malfunctioning on your system.

Permitted Values: 0, 1, 2, or 3

Default: 3

TV::force_default_cplus_demangler

When true, TotalView uses the demangler set in the TV::current_cplus_demangler variable. Set this variable only if TotalView uses the wrong demangler which may occur if you are using an unsupported compiler, an unsupported language preprocessor, or if your vendor has made changes to your compiler.

Permitted Values: true or false

Default: false
**TV::force_default_f9x_demangler**

When `true`, TotalView uses the demangler set in the `TV::current_fortran_demangler` variable. Set this variable only if TotalView uses the wrong demangler which may occur if you are using an unsupported compiler, an unsupported language preprocessor, or if your vendor has made changes to your compiler.

*Permitted Values:* `true` or `false`  
*Default:* `false`

**TV::global_typenames**

When `true`, TotalView assumes that type names are globally unique within a program and that all type definitions with the same name are identical. This must be true for standard-conforming C++ compilers.

If you set this option to `true`, TotalView attempts to replace an opaque type (`struct foo *p;`) declared in one module with an identically named defined type (`struct foo { ... }`) in a different module.

If TotalView has read the symbols for the module containing the non-opaque type definition, it automatically displays the variable by using the non-opaque type definition when displaying variables declared with the opaque type.

If `false`, TotalView does *not* assume that type names are globally unique within a program. Use this variable only if your code has different definitions of the same named type, since TotalView can pick the wrong definition when it substitutes for an opaque type in this case.

*Permitted Values:* `true` or `false`  
*Default:* `true`

**TV::gnu_debuglink**

When `true`, TotalView checks for a `.gnu_debuglink` section within your process. If it is found, it looks for the file named in this section. If `false`, TotalView ignores the contents of this section. This means that a gnu_debuglink file will not be loaded. For more information, see “Using gnu_debuglink Files” on page 366.

*Permitted Values:* `true` or `false`  
*Default:* `true`

**TV::gnu_debuglink_checksum**

When `true`, TotalView compares the checksum of the gnu_debuglink file against the checksum contained within the `.gnu_debuglink` section. TotalView will only load the information from the gnu_debuglink file when the checksums match. For more information, see “Using gnu_debuglink Files” on page 366.

*Permitted Values:* `true` or `false`  
*Default:* `true`

**TV::gnu_debuglink_global_directory**

Names the directory to store gnu_debuglink files. For more information, see “Using gnu_debuglink Files” on page 366.
Permitted Values: A pathname within your file system. While this path can be relative, it is usually a full pathname.
Default: /usr/lib/debug

TV::gnu_debuglink_global_search_path
Defines the search path to use when searching for .gnu_debuglink files. You can use two substituting variables when assigning values:

- %D: The directory containing the .gnu_debuglink file.
- %G: The contents of the TV::gnu_debuglink_global_directory variable.
- %/: The target directory delimiter; for example “/”.

For more information, see “Using gnu_debuglink Files” on page 366.
Permitted Values: A string containing directory paths.
Default: %D:%D.debug:%G%/%D

TV::hia_local_dir
This variable affects only those cases where TotalView preloads the agent. It names the directory in which TotalView will look for the hia for a local job. The default is the value of TV::hia_local_installation_dir. Change this variable if you want TotalView to look for the agent in a different directory.

TV::hia_local_installation_dir
A read-only variable that names the directory where the hia distributed with the executing instance of TotalView is found.

TV::hia_remote_dir
This variable affects only those cases where TotalView preloads the agent. It names the directory on a remote host where TotalView will look for the hia that is to be used by the remote job. If the variable is not set, the server uses its default, which is the same as the default value of the server’s TV::hia_local_dir but is interpreted in the remote file system.

TV::hpf
Deprecated.

TV::hpf_node
Deprecated.

TV::host_platform
A read-only value that returns the architecture upon which TotalView is running.
TV::ignore_control_c
When true, TotalView ignores Ctrl+C. This prevents you from inadvertently terminating the TotalView process. You would set this option to true when your program catches the Ctrl+C (SIGINT) signal. You may want to set File > Signals so that TotalView resends the SIGINT signal, instead of just stopping the program.

Permitted Values: true or false
Default: false

TV::image_load_callbacks
Contains a Tcl list of procedure names. TotalView invokes the procedures named in this list whenever it loads a new program. This could occur when:

- A user invokes a command such as dload.
- TotalView resolves dynamic library dependencies.
- User code uses dlopen() to load a new image.

TotalView invokes the functions in order, beginning at the first function in this list.

Permitted Values: A Tcl list of procedure names
Default: {::TV::S2S::handle_image_load}

TV::in_setup
Contains a true value if called while TotalView is being initialized. Your procedures would read the value of this variable so that code can be conditionally executed based on whether TotalView is being initialized. In most cases, this is used for code that should be invoked only while TotalView is being initialized. This is a read-only variable.

Permitted Values: true or false
Default: false

TV::ipv6_support
When true, ipv6 support is enabled. If false, ipv6 support is disabled.

Permitted Values: true or false
Default: false

TV::jnibridge
Internal use only.

TV::kcc_classes
When true, TotalView converts structure definitions created by the KCC compiler into classes that show base classes and virtual base classes in the same way as other C++ compilers. When false, TotalView does not perform this conversion. In this case, TotalView displays virtual bases as pointers rather than as the data.
TotalView converts structure definitions by matching the names given to structure members. This means that TotalView may not convert definitions correctly if your structure component names look like KCC processed classes. However, TotalView never converts these definitions unless it believes that the code was compiled with KCC. (It does this when it sees one of the tag strings that KCC outputs, or when you use the KCC name demangler.) Because all recognized structure component names start with “__” and the C standard forbids this use, your code should not contain names with this prefix.

Under some circumstances, TotalView may not be able to convert the original type names because type definition are not available. For example, it may not be able to convert “struct __SO_foo” to “struct foo”. In this case, TotalView shows the “__SO_foo” type. This is just a cosmetic problem. (The “__SO__” prefix denotes a type definition for the nonvirtual components of a class with virtual bases).

Since KCC output does not contain information on the accessibility of base classes (private, protected, or public), TotalView cannot provide this information.

Permitted Values: true or false
Default: true

**TV::kernel_launch_string**

This is not currently used.

**TV::kill_callbacks**

Names a Tcl function to run before TotalView kills a process. The contents of this variable is a list of pairs. For example:

```tcl
dset TV::kill_callbacks {
    {^srun$  TV::destroy_srun}
}
```

The first element in the pair is a regular expression, and the second is the name of a Tcl function. If the process's name matches the regular expression, TotalView runs the Tcl procedure, giving it the DPID of the process as its argument. This procedure can do anything that needs to be done for orderly process termination.

If your Tcl procedure returns false, TotalView kills your process as you would expect. If the procedure returns true, TotalView takes no further action to terminate the process.

Any slave processes are killed before the master process is killed. If there is a kill_callback for the master process, it is called after the slave processes are killed. If there are kill_callbacks for the slave processes, they will be called before the slave is killed.

Permitted Values: List of one or more list of pairs
Default: {}

**TV::library_cache_directory**

Specifies the directory to write library cache data.

Permitted Values: A string indicating a path
Default: $USERNAME/.totalview/lib_cache

**TV::launch_command**

Specifies the launch command.

**Permitted Values:** A string indicating the launch command

**Default:** The value of **TVDSVRLAUNCHCMD** if set, otherwise the value of **default_launch_command**. Note: changing the value of **TVDSVRLAUNCHCMD** in the environment after starting TotalView does not affect this variable or how %C is expanded.

**TV::local_interface**

Sets the interface name that the server uses when it makes a callback. For example, on an IBM PS2 machine, you would set this to css0. However, you can use any legal inet interface name. (You can obtain a list of the interfaces if you use the `netstat -i` command.)

**Permitted Values:** A string

**Default:** {}

**TV::local_server**

(Sun only) This variable tells TotalView which local server it should launch. By default, TotalView finds the local server in the same place as the remote server. On Sun platforms, TotalView can launch a 32- and 64-bit version.

**Permitted Values:** A file or path name to the local server

**Default:** tvdsvr

**TV::local_server_launch_string**

(Sun only) If TotalView will not be using the server contained in the same working directory as the TotalView executable, the contents of this string indicate the shell command that TotalView uses to launch this alternate server.

**Permitted Values:** A string enclosed with {} (braces) if it has embedded spaces

**Default:** {%M -working_directory %D -local %U -set_pw %P -verbosity %V}

**TV::message_queue**

When **true**, TotalView displays MPI message queues when you are debugging an MPI program. When **false**, these queues are not displayed. Disable these queues only if something is overwriting the message queues, thereby confusing TotalView.

**Permitted Values:** **true** or **false**

**Default:** **true**

**TV::mrnet_port_base**

The start of the port range that MRNet attempts to use for listening sockets on Cray systems. This string is passed to MRNet instead of using the **MRNET_PORT_BASE** environment variable. This value is only used when TotalView uses MRNet on Cray systems.

**Permitted Values:** a port number
TV::native_platform
A read-only state variable that identifies the native (host) platform on which the TotalView client (GUI or CLI) is running. This variable’s value is the same as the value of TV::platform.

   **Permitted Values:** a string indicating a platform

   **Default:** platform-specific

TV::nptl_threads
When set to auto, TotalView determines which threads package your program is using. A value of true identifies use of NPTL threads, while false means that the program is not using this package.

   **Permitted Values:** true, false, or auto

   **Default:** auto

TV::open_cli_window_callback
Contains the string that the CLI executes after you open the CLI by selecting the **Tools > Command Line** command. It is ignored when you open the CLI from the command line.

This variable is most commonly used to set the terminal characteristics of the (pseudo) tty that the CLI is using, since these are inherited from the tty on which TotalView was started. Therefore, if you start TotalView from a shell running inside an Emacs buffer, the CLI uses the raw terminal modes that Emacs is using. You can change your terminal mode by adding the following command to your .tvdrd file:

   dset TV::open_cli_window_callback "stty sane"

   **Permitted Values:** A string representing a Tcl or CLI command

   **Default:** Null

TV::parallel
When true, enables TotalView support for parallel program runtime libraries such as MPI, PE, and UPC. You might set this to false if you need to debug a parallel program as if it were a single-process program.

   **Permitted Values:** true or false

   **Default:** true

TV::parallel_attach
automatically attaches to processes. Your choices are:

   - **yes**: Attach to all started processes.
   - **no**: Do not attach to any started processes.
   - **ask**: Display a dialog box listing the processes to which TotalView can attach, and let the user decide to which ones TotalView should attach.

   **Permitted Values:** yes, no, or ask
**TV::parallel_stop**

Tells TotalView if it should automatically run processes when your program launches them. Your choices are:

- **yes**: Stop the processes before they begin executing.
- **no**: Do not interfere with the processes; that is, let them run.
- **ask**: Display a question box asking if it should stop before executing.

**Permitted Values:** yes, no, or ask

**Default:** ask

**TV::platform**

Indicates the platform on which you are running TotalView. This is a read-only variable.

**Permitted Values:** A string indicating a platform, such as **alpha** or **sun5**

**Default:** Platform-specific

**TV::process_load_callbacks**

Names the procedures that TotalView runs after it loads or attaches to a program and just before it runs the program. TotalView executes these procedures after it invokes the procedures in the **TV::image_load_callbacks** list.

The procedures in this list are called at most once per process load or attach, even though your executable may use many shared libraries. After attaching to the processes in a parallel job, the callback procedures listed in **TV::process_load_callbacks** are invoked on one representative process in each share group, and only when the share group is first created. If the parallel job is restarted, the callback procedures are not invoked because the share groups are not recreated. All processes in a parallel job are attached before calling the procedures. The calls to the procedures are queued and executed at a later time, and are not guaranteed to be during the lifetime of the processes.

**Permitted Values:** A list of Tcl procedures

**Default:** **TV::source_process_startup**. The default procedure looks for a file with the same name as the newly loaded process's executable image that has a **.tvd** suffix appended to it. If it exists, TotalView executes the commands contained within it. This function is passed an argument that is the ID for the newly created process.

**TV::recurse_subroutines:**

Determines whether a data window displaying the subroutines associated with a source file initially displays just the subroutine names, or also the data values in the subroutine scopes. This situation most commonly occurs in the Program Browser.

- **true**: Displays both the subroutine names and the data in their scope.
- **false**: Displays only the subroutine names.
For complex applications, determining the state of the data values in the scope of all subroutines can significantly slow down TotalView. If set to `false` so only the subroutine names appear, data values for a particular subroutine can still be viewed by explicitly diving into the subroutine.

- **Permitted Values:** true or false
- **Default:** true

**TV::replay_history_mode**

Controls how ReplayEngine handles the history buffer when it is full, as follows:

- 1: Discards the oldest history and continue.
- 2: Stops the process.

- **Permitted Values:** 1 or 2
- **Default:** 1

**TV::replay_history_size**

Specifies the size of ReplayEngine's buffer for recorded history, in either bytes, kilobytes (K) or megabytes (M). To specify kilobytes or megabytes, append a K or M to the number, as follows: 10000K or 1024M

- **Permitted Values:** An integer or an integer followed by K or M
- **Default:** 0 (Limited only by available memory)

**TV::restart_threshold**

When killing a multi-threaded or multiprocess program, specifies the number of threads or processes that must be running before a prompt launches confirming that you wish to kill the program. By default, this prompt appears if there is more than one thread or process running.

- **Permitted Values:** a positive integer
- **Default:** 1

**TV::save_global_dialog_defaults**

Obsolete.

**TV::save_search_path**

Obsolete.

**TV::save_window_pipe_or_filename**

Names the file to which TotalView writes or pipes the contents of the current window or pane when you select the File > Save Pane command.

- **Permitted Values:** A string naming a file or pipe
- **Default:** None, until something is saved. Afterward, the saved string is the default.
**TV::search_case_sensitive**

When **true**, text searches are case-sensitive, succeeding only for an exact match for the entry in the **Edit > Find** dialog box. For example, searching *Foo* won't find *foo* if this variable is set to **true**. It will be found if this variable is set to **false**.

*Permitted Values: true or false*

*Default: false*

**TV::server_launch_enabled**

When **true**, TotalView uses its single-process server launch procedure when launching remote **tvdsvr** processes. When **false**, **tvdsvr** is not automatically launched.

*Permitted Values: true or false*

*Default: true*

**TV::server_launch_timeout**

Specifies the number of seconds to wait for a response from the TotalView Debugger Server (**tvdsvr**) that it has launched.

*Permitted Values: An integer from 1 to 3600 (1 hour)*

*Default: 30*

**TV::server_response_wait_timeout**

Specifies how long to wait for a response from the TotalView Debugger Server (**tvdsvr**). Using a higher value may help avoid server timeouts if you are debugging across multiple nodes that are heavily loaded.

*Permitted Values: An integer from 1 to 3600 (1 hour)*

*Default: 30*

**TV::share_action_point**

Indicates the scope in which TotalView places newly created action points. In the CLI, this is the dbarrier, dbreak, and dwatch commands. If **true**, newly created action points are shared across the group. If **false**, a newly created action point is active only in the process in which it is set.

As an alternative to setting this variable, you can select the **Plant in share group** check box in the **Action Points** Page in the **File > Preferences** dialog box. You can override this value in the GUI by selecting the **Plant in share group** checkbox in the **Action Point > Properties** dialog box.

*Permitted Values: true or false*

*Default: true*

**TV::signal_handling_mode**

A list that modifies the way in which TotalView handles signals. This list consists of a list of **signal_action** descriptions, separated by spaces:

*signal_action[signal_action] ...*
A signal action description consists of an action, an equal sign (=), and a list of signals:

\[
\text{action}=\text{signal\_list}
\]

An action can be one of the following: Error, Stop, Resend, or Discard.

A signal list is a list of one or more signal specifiers, separated by commas:

\[
\text{signal}\_\text{specifier},\text{signal}\_\text{specifier} \ldots
\]

A signal specifier can be a signal name (such as SIGSEGV), a signal number (such as 11), or a star (*), which specifies all signals. We recommend using the signal name rather than the number because number assignments vary across UNIX versions.

The following rules apply when you are specifying an action list:

- If you specify an action for a signal in an action list, TotalView changes the default action for that signal.
- If you do not specify a signal in the action list, TotalView does not change its default action for the signal.
- If you specify a signal that does not exist for the platform, TotalView ignores it.
- If you specify an action for a signal twice, TotalView uses the last action specified. In other words, TotalView applies the actions from left to right.

If you need to revert the settings for signal handling to built-in defaults, use the Defaults button in the File > Signals dialog box.

For example, to set the default action for the SIGTERM signal to Resend, you specify the following action list:

\[
\{\text{Resend}=\text{SIGTERM}\}
\]

As another example, to set the action for SIGSEGV and SIGBUS to Error, the action for SIGHUP and SIGTERM to Resend, and all remaining signals to Stop, you specify the following action list:

\[
\{\text{Stop}=*,\text{Error}=\text{SIGSEGV, SIGBUS}\text{ Resend}=\text{SIGHUP, SIGTERM}\}
\]

This action list shows how TotalView applies the actions from left to right.

1. Sets the action for all signals to Stop.
2. Changes the action for SIGSEGV and SIGBUS from Stop to Error.
3. Changes the action for SIGHUP and SIGTERM from Stop to Resend.

Permitted Values: A list of signals, as was just described

Default: This differs from platform to platform; type `dset TV::signal\_handling\_mode` to see what a platform's default values are
**TV::source_pane_tab_width**
Sets the width of the tab character that is displayed in the Process Window's Source Pane. You may want to set this value to the same value as you use in your text editor.

- **Permitted Values:** An integer
- **Default:** 8

**TV::spell_correction**
When you use the View > Lookup Function or View > Lookup Variable commands in the Process Window or edit a type string in a Variable Window, TotalView checks the spelling of your entries. By default (verbose), TotalView displays a dialog box before it corrects spelling. You can set this resource to brief to run the spelling corrector silently. (TotalView makes the spelling correction without displaying it in a dialog box first.) You can also set this resource to none to disable the spelling corrector.

- **Permitted Values:** verbose, brief, or none
- **Default:** verbose

**TV::stack_trace_qualification_level**
Controls the amount of information displayed in stack traces. For more information, see TV::dwhere_qualification_level.

- **Permitted Values:** One or more of the following arguments: all, class_name, file_directory, hint, image_directory, loader_directory, member, module, node, overload_list, parent_function, template_args, type_name.
- **Default:** class_name+template_args+module+ parent_function+member+node

**TV::stop_all**
Indicates a default property for newly created action points. This property tells TotalView what else it should stop when it encounters this action point. The values you can set are:

- **group**
  Stops the entire control group when the action point is hit.
- **process**
  Stops the entire process when the action point is hit.
- **thread**
  Only stops the thread that hit the action point. Note that none is a synonym for thread.

- **Permitted Values:** group, process, or thread
- **Default:** group

**TV::stop_relatives_on_proc_error**
When true, TotalView stops the control group when an error signal is raised. This is the variable used by the Stop control group on error signal option in the Options Page of the File > Preferences dialog box.

- **Permitted Values:** true or false
Default: true

TV::suffixes

Use a space separated list of items to identify the contents of a file. Each item on this list has the form: suffix:lang[:include]. You can set more than suffix for an item. If you want to remove an item from the default list, set its value to unknown.

Permitted Values: A list identifying how suffixes are used


TV::target_platform

A read-only variable that displays a list of the platforms on which you can debug from the native (host) platform, usually in the format os-cpu. For example, from a native platform of Linux-x86, the list is “linux-power linux-x86_64 linux-x86 catamount-x86_64 catamount-x86.” The platform names may be listed differently than in TV::platform and TV::native_platform. For example, for AIX, TV::target_platform is “aix-power” but TV::platform and TV::native_platform are “rs6000.”

Permitted Values: A list of platform names

Default: Platform-dependent

TV::ttf

When true, TotalView uses registered type transformations to change the appearance of data types that have been registered using the TV::type_transformation command.

Permitted Values: true or false

Default: true

TV::ttf_max_length

When transforming STL structures, TotalView must chase through pointers to obtain values. This number indicates how many of these pointers it should follow.

Permitted Values: an integer number

Default: 10000

TV::use_fast_trap

Controls TotalView's use of the target operating system's support of the fast trap mechanism for compiled conditional breakpoints, also known as EVAL points. You cannot interactively use this variable. Instead, you must set it within a TotalView startup file; for example, set its value with a .tvdrc file.

Your operating system may not be configured correctly to support this option. See the TotalView for HPC Release Notes on our web site for more information.

Permitted Values: true or false
**TV::use_fast_wp**

Controls TotalView's use of the target operating system's support of the fast trap mechanism for compiled conditional watchpoints, also known as CDWP points. You cannot interactively use this variable. Instead, you must set it within a TotalView startup file; for example, set its value with a `.tvdrc` file.

Your operating system may not be configured correctly to support this option. See the TotalView for HPC Release Notes on our web site for more information.

*Permitted Values:* true or false

*Default:* false

**TV::use_interface**

This variable is a synonym for `TV::local_interface`.

**TV::user_threads**

When `true`, it enables TotalView support for handling user-level (M:N) thread packages on systems that support two-level (kernel and user) thread scheduling.

*Permitted Values:* true or false

*Default:* true

**TV::version**

Indicates the current TotalView version. This is a read-only variable.

*Permitted Values:* A string

*Default:* Varies from release to release

**TV::visualizer_launch_enabled**

When `true`, TotalView automatically launches the Visualizer when you first visualize something. If you set this variable to `false`, TotalView disables visualization. This is most often used to stop evaluation points containing a `$visualize` directive from invoking the Visualizer.

*Permitted Values:* true or false

*Default:* true

**TV::visualizer_launch_string**

Specifies the command string that TotalView uses when it launches a visualizer. Because the text is actually used as a shell command, you can use a shell redirection command to write visualization datasets to a file (for example, “cat > your_file”).

*Permitted Values:* A string

*Default:* %B/visualize
**TV::visualizer_max_rank**

Specifies the default value used in the Maximum permissible rank field in the Launch Strings Page of the File > Preferences dialog box. This field sets the maximum rank of the array that TotalView will export to a visualizer. The Visualizer cannot visualize arrays of rank greater than 2. If you are using another visualizer or just dumping binary data, you can set this value to a larger number.

- **Permitted Values:** An integer
- **Default:** 2

**TV::warn_step_throw**

If this is set to true and your program throws an exception during a single-step operation, TotalView asks if you wish to stop the step operation. The process will be left stopped at the C++ run-time library's “throw” routine. If this is set to false, TotalView will not catch C++ exception throws during single-step operations. Setting it to false may mean that TotalView will lose control of the process, and you may not be able to control the program.

- **Permitted Values:** true or false
- **Default:** true

**TV::wrap_on_search**

When true, TotalView will continue searching from either the beginning (if Down is also selected in the Edit > Find dialog box) or the end (if Up is also selected) if it doesn't find what you're looking for. For example, you search for foo and select the Down button. If TotalView doesn't find it in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you set this option.

- **Permitted Values:** true or false
- **Default:** true

**TV::xplat_remcmd**

A command that needs to be executed before executing a process on a remote host, e.g., runauth. This string is passed to MRNet instead of using the XPLAT_REMCMD environment variable. This value is only used when TotalView uses MRNet.

- **Permitted Values:** a command
- **Default:** {}

**TV::xplat_rsh**

An rsh command that is passed to MRNet instead of using the XPLAT_RSH environment variable. This command is used to launch remote processes. If this variable isn't explicitly set and the XPLAT_RSH environment variable is empty, TotalView uses the value of TV::launch_command. This value is used only when TotalView uses MRNet.

- **Permitted Values:** a remote launch command
- **Default:** {}
**TV::xplat_rsh_args**
A list of arguments that need to be given to the remote launch command. This string is passed to MRNet instead of using the XPLAT_RSH_ARGS environment variable. This value is only used when TotalView uses MRNet.

*Permitted Values:* a space-separated list of remote launch arguments  
*Default:* \{

**TV::xterm_name**
The name of the program that TotalView should use when spawning the CLI. In most cases, you will set this using the -xterm_name command-line option.

*Permitted Values:* a string  
*Default:* xterm
TV::MEMDEBUG:: Namespace

TV::MEMDEBUG::default_snippet_extent
Defines the number of code lines above and below point of allocation that the Memory Debugger saves when it is adding code snippets to saved output.

You can also set this value using a Memory Debugger preference.

Permitted Values: A positive integer number
Default: 5

TV::MEMDEBUG::do_not_apply_hia_defaults
If set to true, tells the Memory Debugger that it should use settings it finds in a default .hiarc file. Otherwise, the Memory Debuggers sets all options to off.

You can also set this value using a Memory Debugger preference.

Permitted Values: true or false
Default: false

TV::MEMDEBUG::hia_allow_ibm_poe
Tells the Memory Debugger if you can enable memory debugging on poe. As the default value is false, set this variable if you want memory debugging to be on by default. This variable is hardly ever used.

Permitted Values: true or false
Default: false

TV::MEMDEBUG::ignore_snippets
When true, the Memory Debugger ignores code snippets that it saved and instead locates the information from your program's files.

You can also set this value using Memory Debugger preference.

Permitted Values: true or false
Default: false

TV::MEMDEBUG::leak_check_interior_pointers
When true, the Memory Debugger considers a block as being referenced if a pointer is pointing anywhere within the block instead of just at the block's starting location. In most programs, the code should be keeping track of the block's boundary. However, if your C++ program is using multiple inheritance, you may be pointing into the middle of the block without knowing it.

Permitted Values: true or false
Default: true
TV::MEMDEBUG::leak_detection_alignment
Specifies the alignment and stride TotalView uses as it steps through memory looking for pointers during leak detection. If 0 (the default value), then TotalView defaults to using the size of a pointer, which varies according to platform and programming model. In normal circumstances you should not need to adjust the alignment.

Permitted Values: A non-negative integer number
Default: 0

TV::MEMDEBUG::leak_max_cache
Sets the size of the Memory Debugger's cache. We urge you not to change this value unless your program is exceptionally large or are asked to make the change by someone on the TotalView support team.

Permitted Values: A positive integer number
Default: 4194304

TV::MEMDEBUG::leak_max_chunk
Tells the Memory Debugger how much memory it should obtain when it obtains memory from your operating system. You shouldn't change this value unless asked to by someone on the TotalView support team.

Permitted Values: A positive integer number
Default: 4194304

TV::MEMDEBUG::shared_data_filters
Names a filter definition file that is not located in the default directory. (The default directory is the lib subdirectory within the TotalView installation directory.) The contents of this variable are read when TotalView begins executing. Consequently, TotalView ignores any changes you make during the debugging session. The following example names the directory in which the filter file resides. This example assumes that filter has the default name, which is tv_filters.tvd.

dset TV::MEMDEBUG::shared_data_filters {/home/projects/filters/}

Use brackets so that Tcl doesn't interpret the “/” as a mathematical operator. If you wish to use a specific file, just use its name in this command. For example:

dset TV::MEMDEBUG::shared_data_filters \\ {/home/projects/filters/filter.tvd}

The file must have a .tvd extension.

Permitted Values: A string naming the path to the filter directory.
Default: none
NOTE >> The variables in this section have meaning (and in some cases, a value) only when you are using the TotalView GUI.

**TV::GUI::chase_mouse**

When this variable is set to **true**, TotalView displays dialog boxes at the location of the mouse cursor. If this is set to **false**, TotalView displays them centered in the upper third of the screen.

*Permitted Values:* true or false

*Default:* true

**TV::GUI::display_bytes_kb_mb**

When **true**, the Memory Debugger displays memory block sizes in megabytes. If set to **false**, it displays memory blocks sizes in kilobytes.

*Permitted Values:* true or false

*Default:* true

**TV::GUI::display_font_dpi**

Indicates the video monitor DPI (dots per inch) at which fonts are displayed.

*Permitted Values:* An integer

*Default:* 75

**TV::GUI::enabled**

When **true**, you invoked the CLI from the GUI or a startup script. Otherwise, this read-only value is **false**.

*Permitted Values:* true or false

*Default:* true if you are running the GUI even though you are seeing this in a CLI window; false if you are only running the CLI

**TV::GUI::fixed_font**

Indicates the specific font TotalView uses when displaying program information such as source code in the Process Window or data in the Variable Window. This variable contains the value set when you select a Code and Data Font entry in the Fonts Page of the File > Preferences dialog box.

This is a read-only variable.

*Permitted Values:* A string naming a fixed font residing on your system

*Default:* While this is platform specific, here is a representative value:-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1
**TV::GUI::fixed_font_family**

Indicates the specific font TotalView uses when displaying program information such as source code in the Process Window or data in the Variable Window. This variable contains the value set when you select a Code and Data Font entry of the Fonts Page of the File > Preferences dialog box.

*Permitted Values:* A string representing an installed font family  
*Default:* fixed

**TV::GUI::fixed_font_size**

Indicates the point size at which TotalView displays fixed font text. This is only useful if you have set a fixed font family because if you set a fixed font, the value entered contains the point size.

Font sizes are indicated using printer points.  
*Permitted Values:* An integer  
*Default:* 12

**TV::GUI::font**

Indicates the specific font used when TotalView writes information as the text in dialog boxes and in menu bars. This variable contains the information set when you select a Select by full name entry in the Fonts Page of the File > Preferences dialog box.

*Permitted Values:* A string naming a fixed font residing on your system. While this is platform specific, here is a representative value: `-adobe-helvetica-medium-r-normal--12-120-75-75-p-67-iso8859-1`  
*Default:* helvetica

**TV::GUI::force_window_positions**

Setting this variable to true tells TotalView that it should use the version 4 window layout algorithm. This algorithm tells the window manager where to set the window. It also cascades windows from a base location for each window type. If this is not set, which is the default, newer window managers such as kwm or Enlightenment can use their smart placement modes.

Dialog boxes still chase the pointer as needed and are unaffected by this setting.  
*Permitted Values:* true or false  
*Default:* false

**TV::GUI::frame_offset_x**

Not implemented.

**TV::GUI::frame_offset_y**

Not implemented.
**TV::GUI::geometry_call_tree**

Specifies the position at which TotalView displays the **Tools > Call Tree** Window. This position is set using a list containing four values: the window's \(x\) and \(y\) coordinates. These are followed by two more values specifying the window's width and height.

If you set any of these values to 0 (zero), TotalView uses its default value. This means, however, you cannot place a window at \(x, y\) coordinates of 0, 0. Instead, you'll need to place the window at 1, 1.

If you specify negative \(x\) and \(y\) coordinates, TotalView aligns the window to the opposite edge of the screen.

**Permitted Values:** A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height

**Default:** \(\{0 \ 0 \ 0 \ 0\}\)

**TV::GUI::geometry_cli**

Specifies the position at which TotalView displays the **Tools > CLI** Window.

See **TV::GUI::geometry_call_tree** for information on setting this list.

**Permitted Values:** A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height

**Default:** \(\{0 \ 0 \ 0\}\)

**TV::GUI::geometry_expressions**

Specifies the position at which TotalView displays the **Tools > Expression List** Window.

See **TV::GUI::geometry_call_tree** for information on setting this list.

**Permitted Values:** A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height

**Default:** \(\{0 \ 0 \ 0\}\)

**TV::GUI::geometry_globals**

Specifies the position at which TotalView displays the **Tools > Program Browser** Window.

See **TV::GUI::geometry_call_tree** for information on setting this list.

**Permitted Values:** A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height

**Default:** \(\{0 \ 0 \ 0\}\)

**TV::GUI::geometry_help**

Specifies the position at which TotalView displays the **Help** Window.

See **TV::GUI::geometry_call_tree** for information on setting this list.

**Permitted Values:** A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height
**TV::GUI::geometry_memory_stats**

Specifies the position at which TotalView displays the **Tools > Memory Statistics** Window.

See **TV::GUI::geometry_call_tree** for information on setting this list.

- **Permitted Values**: A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height
- **Default**: \{0 0 0 0\}

**TV::GUI::geometry_message_queue**

Specifies the position at which TotalView displays the **Tools > Message Queue** Window.

See **TV::GUI::geometry_call_tree** for information on setting this list.

- **Permitted Values**: A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height
- **Default**: \{0 0 0\}

**TV::GUI::geometry_message_queue_graph**

Specifies the position at which TotalView displays the **Tools > Message Queue Graph** Window.

See **TV::GUI::geometry_call_tree** for information on setting this list.

- **Permitted Values**: A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height
- **Default**: \{0 0 0\}

**TV::GUI::geometry_modules**

Specifies the position at which TotalView displays the **Tools > Fortran Modules** Window.

See **TV::GUI::geometry_call_tree** for information on setting this list.

- **Permitted Values**: A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height
- **Default**: \{0 0 0\}

**TV::GUI::geometry_process**

Specifies the position at which TotalView displays the Process Window.

See **TV::GUI::geometry_call_tree** for information on setting this list.

- **Permitted Values**: A list containing four integers indicating the window's \(x\) and \(y\) coordinates and the window's width and height
- **Default**: \{0 0 0\}
**TV::GUI::geometry_ptset**
No longer used.

**TV::GUI::geometry_root**
Specifies the position at which TotalView displays the Root Window.

See TV::GUI::geometry_call_tree for information on setting this list.

- **Permitted Values:** A list containing four integers indicating the window's `x` and `y` coordinates and the window's width and height
- **Default:** `{0 0 0 0}`

**TV::GUI::geometry_thread_objects**
Specifies the position at which TotalView displays the Tools > Thread Objects Window.

See TV::GUI::geometry_call_tree for information on setting this list.

- **Permitted Values:** A list containing four integers indicating the window's `x` and `y` coordinates and the window's width and height
- **Default:** `{0 0 0 0}`

**TV::GUI::geometry_variable**
Specifies the position at which TotalView displays the Variable Window.

See TV::GUI::geometry_call_tree for information on setting this list.

- **Permitted Values:** A list containing four integers indicating the window's `x` and `y` coordinates and the window's width and height
- **Default:** `{0 0 0 0}`

**TV::GUI::geometry_variable_stats**
Specifies the position at which TotalView displays the Tools > Statistics Window.

See TV::GUI::geometry_call_tree for information on setting this list.

- **Permitted Values:** A list containing four integers indicating the window's `x` and `y` coordinates and the window's width and height
- **Default:** `{0 0 0 0}`

**TV::GUI::hand_cursor_enabled**
Specifies whether the cursor should change to a hand cursor when hovering over an element you can dive into in the source pane of the process window.

- **Permitted Values:** `true` or `false`
- **Default:** `true`
**TV::GUI::heap_summary_refresh**
Not user settable.

**TV::GUI::inverse_video**
Not implemented.

**TV::GUI::keep_expressions**
Deprecated.

**TV::GUI::keep_search_dialog**
When true, TotalView doesn’t remove the Edit > Find dialog box after you select that dialog box’s Find button. If you select this option, you will need to select the Close button to dismiss the Edit > Find box.

*Permitted Values: true or false
Default: true*

**TV::GUI::old_root_window**
When true, TotalView replaces the Root Window with the Root Window used in versions prior to TotalView for HPC 8.15. You can override this value using the following command-line options:

- -oldroot sets this variable to true
- -newroot sets this variable to false

*NOTE >> Using the previous-version Root Window may affect performance of applications containing thousands of threads/processes.*

*Permitted Values: true or false
Default: false*

**TV::GUI::pop_at_breakpoint**
When true, TotalView sets the Open (or raise) process window at breakpoint check box to be selected by default. If this variable is set to false, it sets that check box to be deselected by default.

*Permitted Values: true or false
Default: false*

**TV::GUI::pop_on_error**
When true, TotalView sets the Open process window on error signal check box in the File > Preferences's Option Page to be selected by default. If you set this to false, TotalView sets that check box to be deselected by default.

*Permitted Values: true or false*
**TV::GUI::process_grid_wanted**

When `true`, TotalView enables the Processes/Ranks Tab in the Process Window. Enabling this tab can significantly affect performance, particularly for large, massively parallel applications.

**Permitted Values:** `true` or `false`

**Default:** `false`

**TV::GUI::show_startup_parameters**

Setting this value to `true` tells TotalView to display the Process > Startup dialog box when you use a program name as an argument to the TotalView command.

**Permitted Values:** `true` or `false`

**Default:** `true`

**TV::GUI::show_sys_thread_id**

Setting this value to `true` tells TotalView to display the current thread's system thread ID within the TotalView GUI.

**Permitted Values:** `true` or `false`

**Default:** `true`

**TV::GUI::single_click_dive_enabled**

When set, you can perform dive operations using the middle mouse button. Diving using a left-double-click still works. If you are editing a field, clicking the middle mouse performs a paste operation.

**Permitted Values:** `true` or `false`

**Default:** `true`

**TV::GUI::toolbar_style**

This value set defines toolbar display.

**Permitted Values:** `icons_above_text`, `icons_besides_text`, `icons`, or `text`

**Default:** `icons_above_text`

**TV::GUI::tooltips_enabled**

When `true`, variable tooltips are displayed in the Process Window Source Pane.

**Permitted Values:** `true` or `false`

**Default:** `true`

**TV::GUI::ui_font**

Indicates the specific font used when TotalView writes information as the text in dialog boxes and in menu bars. This variable contains the information set when you select a **Select by full name** entry in the **Fonts** Page of the **File > Preferences** dialog box.
Permitted Values: While this is platform specific, here is a representative value: \texttt{-adobe-helvetica-medium-\r-normal--12-120-75-75-p-67-iso8859-1}

Default: helvetica

**TV::GUI::ui_font_family**

Indicates the family of fonts that TotalView uses when displaying such information as the text in dialog boxes and menu bars. This variable contains the information set when you select a Family in the Fonts Page of the File > Preferences dialog box.

Permitted Values: A string

Default: helvetica

**TV::GUI::ui_font_size**

Indicates the point size at which TotalView writes the font used for displaying such information as the text in dialog boxes and menu bars. This variable contains the information set when you select a User Interface Size in the Fonts Page of the File > Preferences dialog box.

Permitted Values: An integer

Default: 12

**TV::GUI::using_color**

Not implemented.

**TV::GUI::using_text_color**

Not implemented.

**TV::GUI::using_title_color**

Not implemented.

**TV::GUI::version**

This number indicates which version of the TotalView GUI is being displayed. This is a read-only variable.

Permitted Values: A number
Creating Type Transformations

Overview

The Type Transformation Facility (TTF) lets you define the way TotalView displays aggregate data. Aggregate data is simply a collection of data elements from within one class or structure. These elements can also be other aggregated elements. In most cases, you will create transformations that model data that your program stores in an array- or list-like way. You can also transform arrays of structures.

This chapter describes the TTF. It presents information on how you create your own. Creating transformations can be quite complicated. This chapter looks at transformations for which TotalView can automatically create an addressing expression.

The chapter also describes C++View (CV), a facility that allows you to format program data in a more useful or meaningful form than the concrete representation that you see in TotalView when you inspect data in a running program.

Topics in this chapter are:

- “Why Type Transformations” on page 308
- “Creating Structure and Class Transformations” on page 310
- “C++View” on page 318
Why Type Transformations

Modern programming languages allow you to use abstractions such as structures, class, and STL data types such as lists, maps, multimaps, sets, multiset, and vectors to model the data that your program uses. For example, the STL (Standard Template Library) allows you to create vectors of the data contained within a class. These abstractions simplify the way in which you think of and manipulate program's data. These abstractions can also complicate the way in which you debug your program because it may be nearly impossible or very inconvenient to examine your program's data. For example, Figure 6 shows a vector transformation.

Figure 6 – A Vector Transformation

The upper left window shows untransformed information. In this example, TotalView displays the complete structure of this GNU C++ STL structure. This means that you are seeing the data exactly as your compiler created it.
The logical model that is the reason for using an STL vector is buried within this information. Neither TotalView nor your compiler has this information. This is where type transformations come in. They give TotalView knowledge of how the data is structured and how it can access data elements. The bottom Variable Window shows how TotalView reorganizes this information.

**NOTE >>** By default, TotalView transforms STL strings, vectors, lists, maps, multimaps, sets, and multisets. The unordered STL types, unordered_map, unordered_multimap, unordered_set and unordered_multiset, are transformed for recent g++ compilers. If you do not want TotalView to transform your information, select the Options Tab within the File > Preferences Dialog Box and remove the check mark from View simplified STL containers (and user-defined transformations).
Creating Structure and Class Transformations

The procedure for transforming a structure or a class requires that you create a mapping between the elements of the structure or class and the way in which you want this information to appear.

This section contains the following topics:

- “Transforming Structures” on page 310
- “build_struct_transform Function” on page 312
- “Type Transformation Expressions” on page 312
- “Using Type Transformations” on page 316

Transforming Structures

The following small program contains a structure and the statements necessary to initialize it:

```c
#include <stdio.h>

int main () {
    struct stuff {
        int month;
        int day;
        int year;
        char * pName;
        char * pStreet;
        char CityState[30];
    };

    struct stuff info;
    char my_name[]   = "John Smith";
    char my_street[] = "24 Prime Parkway, Suite 106";
    char my_CityState[] = "Natick, MA 01760";

    info.month = 6;
    info.day   = 20;
    info.year  = 2004;
    info.pName = my_name;
    info.pStreet = my_street;
    strcpy(info.CityState, my_CityState);

    printf("The year is %d\n", info.year);
}
```

Suppose that you do not want to see the `month` and `day` components. You can do this by creating a transformation that names just the elements you want to include:
::TV::TTF::RTF::build_struct_transform {
    name   (^struct stuff$)
    members {
        { year   { year      } }
        { pName   { * pName   } }
        { pStreet { * pStreet } }
    }
}

You can apply this transformation to your data in the following ways:

- After opening the program, use the **Tools > Command Line** command to open a CLI Window. Next, type this function call.
- If you write the function call into a file, use the Tcl **source** command. If the name of the file is **stuff.tvd**, enter the following command into a CLI Window:
  
  source stuff.tvd

- You can place the transformation source file into the same directory as the executable, giving it the same root name as the executable. If the executable file has the name **stuff**, TotalView will automatically execute all commands within a file named **stuff.tvd** when it loads your executable.

After TotalView processes your transformation, it displays the Variable Window when you dive on the **info** structure:

**Figure 7 – Transforming a Structure**

![Variable Window](image)

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>int</td>
<td>0x0000007d4 (2004)</td>
</tr>
<tr>
<td>pName</td>
<td>$string</td>
<td>&quot;John Smith&quot;</td>
</tr>
<tr>
<td>pStreet</td>
<td>$string</td>
<td>&quot;24 Prime Parkway, Suite&quot;</td>
</tr>
</tbody>
</table>
build_struct_transform Function

The `build_struct_transform` routine used in the example in the previous section is a Tcl helper function that builds the callbacks and addressing expressions that TotalView needs when it transforms data. It has two required arguments: `name` and `members`.

name Argument

The `name` argument contains a regular expression that identifies the structure or class. In this example, `struct` is part of the identifier's name. It does not mean that you are creating a structure. In contrast, if `stuff` is class, you would type:

```tcl
name {^class stuff$}
```

If you use a wildcard such as asterisk (*) or question mark (?), TotalView can match more than one thing. In some cases, this is what you want. If it isn't, you need to be more precise in your wildcard.

members Argument

The `members` argument names the elements that TotalView will include in the information it will display. This argument contains one or more lists. The example in the previous section contained three lists: `year`, `pName`, and `pStreet`. Here again is the `pName` list:

```tcl
{ pName   { * pName  } }
```

The first element in the list is the display name. In most cases, this is the name that exists in the structure or class. However, you can use another name. For example, since the transformation dereferences the pointer, you might want to change its name to `Name`:

```tcl
{ Name    { * pName   } }
```

The sublist within the list defines a type transformation expression. These expressions are discussed in the next section.

Type Transformation Expressions

The list that defines a member has a name component and sublist within the list. This sublist defines a `type transformation expression`. This expression tells TotalView what it needs to know to locate the member. The example in the previous section used two of the six possible expressions. The following list describes these expressions:

```tcl
{member}
```

No transformation occurs. The structure or class member that TotalView displays is the same as it displays if you hadn't used a transformation. This is most often used for simple data types such as ints and floats.

```tcl
{* expr}
```

Dereferences a pointer. If the data element is a pointer to an element, this expression tells TotalView to dereference the pointer and display the dereferenced information.
Creation Type Transformations / Creating Structure and Class Transformations

{expr . expr}
Names a subelement of a structure. This is used in the same way as the dot operator that exists in C and C++. You must type a space before and after the dot operator.

{expr + offset}
Use the data whose location is an offset away from expr. This behaves just like pointer arithmetic in C and C++. The result is calculated based on the size of the type that expr points to:

\[ \text{result} = \text{expr} + \text{sizeof(*expr)} \times \text{offset} \]

{expr -> expr}
Names a subelement in a structure accessed using a pointer. This is used in the same way as the -> operator in C and C++. You must type a space before and after the -> operator.

{datatype cast expr}
Casts a data type. For example:

{double cast national_debt}

{N upcast expr}
Converts the current class type into one of its base classes. For example:

{base_class upcast expr}

You can nest expressions within expressions. For example, here is the list for adding an int member that is defined as int **pfoo:

{foo { * { * pfoo}}

Example
The example in this section changes the structure elements of the example in the previous section so that they are now class members. In addition, this example contains a class that is derived from a second class:

```c
#include <stdio.h>
#include <string.h>

class xbase
{
    public:
        char * pName;
        char * pStreet;
        char CityState[30];
};

class x1 : public xbase
{
    public:
        int month;
        int day;
        int year;
        void * v;
```
void *q;

class x2
{
    public:
    int q1;
    int q2;
};

int main () {
    class x1 info;
    char my_name[] = "John Smith";
    char my_street[] = "24 Prime Parkway, Suite 106";
    char my_CityState[] = "Natick, MA 01760";

    info.month = 6;
    info.day = 20;
    info.year = 2004;
    info.pName = my_name;
    info.pStreet = my_street;
    info.v = (void *) my_name;
    strcpy(info.CityState, my_CityState);

    class x2 x;
    x.q1 = 100;
    x.q2 = 200;
    info.q = (void *) &x;

    printf("The year is %d\n", info.year);
}

Figure 8 shows the Variables Windows that TotalView displays for the info class and the x struct.
The following transformation remaps this information:

```cpp
::TV::TTF::RTF::build_struct_transform {
  name   {^(class|struct) x1$}
  members {
    { pmonth   { month } }
    { pName    { xbase upcast { * pName } } }
    { pStreet  { xbase upcast { * pStreet } } }
    { pVoid1   { "$string *" cast v     } }
    { pVoid2   { * { "class x2 *" cast q } } }
  }
}
```

After you remap the information, TotalView displays the `x1` class.
The members of this transformation are as follows:

- **pmonth**: The *month* member is added to the transformed structure without making any changes to the way TotalView displays its data. This member, however, changes the display name of the data element. That is, the name that TotalView uses to display a member within the remapped structure does not have to be the same as it is in the actual structure.

- **pName**: The *pName* member is added. The transformation contains two operations. The first dereferences the pointer. In addition, as *x1* is derived from *xbase*, you need to upcast the variable when you want to include it.

Notice that one expression is nested within another.

- **pStreet**: The *pStreet* member is added. The operations that are performed are the same as for *pName*.

- **pVoid1**: The *v* member is added. Because the application's definition of the data is *void* *, casting tells TotalView how it should interpret the information. In this example, the data is being cast into a pointer to a string.

- **pVoid2**: The *q* member is added. The transformation contains two operations. The first casts *q* into a pointer to the *x2* class. The second dereferences the pointer.

### Using Type Transformations

When TotalView begins executing, it loads its built-in transformations. To locate the directory in which these files are stored, use the following CLI command:
Type transformations are always loaded. By default, they are turned on. From the GUI, you can control whether transformations are turned on or off by going to the Options Page of the File > Preferences Dialog Box and changing the View simplified STL containers (and user-defined transformations) item. For example, the following turns on type transformations:

dset TV::ttf true
C++View

C++View (CV) is a facility that allows you to format program data in a more useful or meaningful form than the concrete representation that you see in TotalView when you inspect data in a running program. To use C++View, you must write a function for each type whose format you would like to control.

This section contains the following topics:

- “Writing a Data Display Function” on page 319
- “Templates” on page 321
- “Precedence - Searching for TV_ttf_display_type” on page 322
- “TV_ttf_add_row” on page 322
- “Return values from TV_ttf_display_type” on page 323
- “Elision” on page 324
- “Other Constraints” on page 325
- “Safety” on page 325
- “Memory Management” on page 326
- “Multithreading” on page 326
- “Tips and Tricks” on page 327
- “Core Files” on page 327
- “Using C++View with ReplayEngine” on page 327
- “C” on page 329
- “Fortran” on page 330
- “Compiling and linking tv_data_display.c” on page 334
- “C++View Example Files” on page 335
- “Limitations” on page 336
- “Licensing” on page 336
Writing a Data Display Function

The frame of reference in describing this is C++.

In order for C++View to work correctly, the code you write and TotalView must cooperate. There are two key issues here. The first is registering your function so that TotalView can find it when it needs to format data for display. This is straightforward: all you need to do is to define your function to have the right name and prototype. When TotalView needs to format the data of type T, it will look for a function with this signature:

```c
int TV_ttf_display_type ( const T * );
```

The `const` is deliberate to remind you that changes should not be made to the object being formatted for display. Many real-world applications are not entirely `const`-correct, and in cases where you must cast away the `const`, extreme caution is advised.

You will need to define a `TV_ttf_display_type` function for each type you want to format. A `TV_ttf_display_type` function may be at global scope, or it may be a class (static) method. It cannot be a member function.

The second issue concerns how the `TV_ttf_display_type` function which you will write communicates with TotalView. The API you will need to use is given in the header file `tv_data_display.h` included with your TotalView distribution in the `<totalview-installation>/src` directory.

Your `TV_ttf_display_type` will use the provided function `TV_ttf_add_row` to tell TotalView what information should be displayed. Its prototype is:

```c
int TV_ttf_add_row ( const char *field_name,
                     const char *type_name,
                     const char *address );
```

The `field_name` parameter is the descriptive name of the data field being computed. It will be shown by TotalView in a form similar to that of the name of a structure's field. The `type_name` parameter is the type of the data to be displayed. It must be the name of a legal type name in the program, or one of TotalView's types.

As a convenience, the header file provides these symbols for you:

**TV_ttf_type_ascii_string**

This tells TotalView to format a character array as a string (i.e., left to right) instead of an array (top to bottom).

**TV_ttf_type_int**

This is an alias for TotalView integer type `$int$`.

The third parameter, `address`, is the address in your program's address space of the object to be displayed.

`TV_ttf_add_row` should be called only as a result of TotalView invoking your `TV_ttf_display_type` function. It may be called by a `TV_ttf_display_type` called by TotalView, or by one of the descendant callees of that `TV_ttf_display_type`.  

![RogueWave logo](logo.png)
Example

Here are the definitions of a couple of classes:

```cpp
class A {
    int   i;
    char *s;
};

class B {
    A      a;
    double d;
};
```

We can define the display callback functions as follows:

```cpp
int TV_ttf_display_type ( const A *a )
{
    /* NOTE: error checking of value returned from TV_ttf_add_row omitted */
    (void) TV_ttf_add_row ( "i", TV_ttf_type_int, &(a->i) );
    (void) TV_ttf_add_row ( "s", TV_ttf_type_ascii_string, a->s );

    /* indicate success to TotalView */
    TV_ttf_format_ok;
}

int TV_ttf_display_type ( const B *b )
{
    /* NOTE: error checking of value returned from TV_ttf_add_row omitted */
    (void) TV_ttf_add_row ( "a", "A", &(b->a) );
    (void) TV_ttf_add_row ( "d", "double", &(b->d) );

    /* indicate success to TotalView */
    return TV_ttf_format_ok;
}
```

For brevity and clarity, we have omitted all error checking of the value returned from `TV_ttf_add_row`. We will discuss the possible values that a `TV_ttf_display_type` may return later.

For now, we just return a simple success.

We could have made one or both of the display callbacks a class method:

```cpp
class A {
    int   i;
    char *s;

public:
    static int TV_ttf_display_type ( const A *a );
};
```
int A::TV_ttf_display_type ( const A *a )
{
    /* as before */
}

and similarly for class B.

Templates

C++View can also be used with template classes. Consider this container class:

```cpp
template <class T> class BoundsCheckedArray {
private:
    int size;
    T   *array;
public:
    typedef T value_type;
    T ( int s ) { ... }
    ...}
```

Writing a collection of overloaded display functions for each instantiated `BoundsCheckedArray` can rapidly become an overwhelming maintenance burden. Instead, consider whether you can write a template function.

One potential difficulty is getting the name of the type parameter to pass to `TV_ttf_add_row`. Here we follow the convention used by the container classes in the standard library which typedefs the template type parameter to the standard name `value_type`.

We can construct our template function like this:

```cpp
template <class T>
int TV_ttf_display_type ( const BoundsCheckedArray<T> *a )
{
    char type [ 4096 ];
    snprintf ( type, sizeof ( type ), "value_type[%d]", a->get_size () );
    TV_ttf_add_row ( "array_values", type, a->get_array () );
    return TV_ttf_format_ok;
}
```

What we've done here is constructed the type of a fixed-sized array of the type named by the template type parameter. (In some cases you may need to use the compiler's demangler to get the name of the type. See also “Tips and Tricks” on page 327.)

This one definition can be used for any instance of the template class. In some cases, however, you may want a specialized implementation of the display function. As an illustration, consider this:
int TV_ttf_display_type ( const BoundsCheckedArray<char> *s )
{
    TV_ttf_add_row ( "string", TV_ttf_type_ascii_string, s->get_array () );
    return TV_ttf_format_ok;
}

Here we want to tell TotalView to display the array horizontally as a string instead of vertically as an array. For this reason, we want to pass TV_ttf_type_ascii_string to TV_ttf_add_row as the name of the type instead of the name constructed by the implementation of the general template display function. We therefore define a special version of the display function to handle BoundsCheckedArray<char>.

One remaining issue relating to templates is arranging for the various template display function instances to be instantiated. It is unlikely that display functions will be called directly by your program. (Indeed, we mentioned earlier that TV_ttf_add_row should not be called other than as a result of a call initiated by TotalView.) Consequently, the template functions may well not be generated automatically. You can either arrange for functions to be referenced, such as by calling them in a controlled manner, or by explicit template instantiation:

template int TV_ttf_display_type ( const BoundsCheckedArray<int> * );
template int TV_ttf_display_type ( const BoundsCheckedArray<double> * );

Precedence - Searching for TV_ttf_display_type

Only one call to a TV_ttf_display_type will be attempted per object to be displayed, even if multiple candidates are defined. For a type T, TotalView will look for the function in this order:

1. A class-qualified class (static) function returning int and taking a single const T * as its only argument.
2. A function at file scope, returning int and taking a single const T * as its only argument.
3. A global function, returning int and taking a single const T * as its only argument.
4. A TCL transformation

Namespace qualifications are not directly considered.

TV_ttf_add_row

TV_ttf_add_row will return one of the following values defined in the enum TV_ttf_error_codes given in the file tv_data_display.h, located in the <totalview-installation>/include directory in your distribution of TotalView.
The values returned by `TV_ttf_add_row` are:

**TV_ttf_ec_ok**
Indicates that the operation succeeded.

**TV_ttf_ec_not_active**
Indicates that `TV_ttf_add_row` was called when the type formatting facility is not active. This is most likely to occur if `TV_ttf_add_row` is called other than as a result of a call to a `TV_ttf_display_type` initiated by TotalView.

**TV_ttf_ec_invalid_characters**
Indicates that either the field name or the type name contained illegal characters, such as `newline` or `tab`.

**TV_ttf_ec_buffer_exhausted**
Indicates that the internal buffer used by `TV_ttf_add_row` to marshal your formatted data for onward transmission to TotalView is full. See “Tips and Tricks” on page 327 for suggestions for reducing the number of calls to `TV_ttf_add_row`.

**Return values from TV_ttf_display_type**
The set of values your `TV_ttf_display_function` may return to TotalView is defined in the enum `TV_ttf_format_result` given in the file `tv_data_display.h` included with your distribution of TotalView. These values are:

**TV_ttf_format_ok**
Your function should return this value if it has successfully formatted the data and successfully registered its output using `TV_ttf_add_row`.

**TV_ttf_format_ok_elide**
As `TV_ttf_format_ok` but indicates that the output may be subject to type elision (see below).

**TV_ttf_format_failed**
Return this if your function was unable to format the data. When displaying the data, TotalView will indicate that an error occurred.
**TV_ttf_format_raw**

Use this to have your function tell TotalView to display the raw data as it would normally do, that is, as if there were no **TV_ttf_display_type** present for that type.

**TV_ttf_format_never**

As **TV_ttf_format_raw**. In addition, this value tells TotalView never to call the display function again.

**Elision**

Elision is a feature that allows you to simplify how your data are presented. Consider the **BoundsCheckedArray<char>** class and the specialized **TV_ttf_display_type** function we defined earlier:

```cpp
int TV_ttf_display_type ( const BoundsCheckedArray<char> *s )
{
    (void) TV_ttf_add_row ( "string", TV_ttf_type_ascii_string, \
                              s->get_array () );

    return TV_ttf_format_ok;
}
```

We used **TV_ttf_type_ascii_string** so that the array of characters is presented horizontally as a string, rather than vertically as an array. If our program declares a variable **BoundsCheckedArray<char> var1**, we will see output like this in the CLI:

```
d1<> dprint var1
  var1 = {
    string = "Hello World!"
  }
d1<> 
```

Note, however, that the variable **var1** is still presented as an aggregate or class. Conceptually this is unnecessary, and in this arrangement an extra dive may be necessary to examine the data. Additionally, more screen space is needed than is necessary.

You can use elision to promote the member of a class out one level. With elision, we will get output that looks like this:

```
d1<> dprint var1
  string = "Hello World!"
d1<> 
```

TotalView will engage elision if your **TV_ttf_display_type** function returns **TV_ttf_format_ok_elide** (in place of **TV_ttf_format_ok**). In addition, for elision to occur, the object being presented must have only one field.
Other Constraints

An aggregate type cannot contain itself. (An attempt to do so would result in an infinite sized aggregate.) When generating a field of an aggregate $T$ using $\text{TV\_ttf\_add\_row}$, the named type may not be $T$, or anything which directly or indirectly contains a $T$ as a member. If you do need to do something like that, use a pointer or reference.

As an illustration, consider this:

```cpp
class A { ... };  
class B { A a; ... };  

int TV_ttf_display_type ( const A *a )  
{  
  (void) TV_ttf_add_row ( ... );  
  return TV_ttf_format_ok;  
}  
int TV_ttf_display_type ( const B *b )  
{  
  (void) TV_ttf_add_row ( ... );  
  (void) TV_ttf_add_row ( "a", "A", &b->a );  
  return TV_ttf_format_ok;  
}
```

Note the following:

- $\text{TV\_ttf\_display\_type ( const A *a )}$ may not add an object of type $A$ (direct inclusion) nor one of type $B$ (indirect inclusion).
- When viewing an object of type $B$, TotalView will invoke $\text{TV\_ttf\_add\_row ( const B * )}$, and then $\text{TV\_ttf\_add\_row ( const *A )}$.

Safety

When you stop your program to inspect data, objects might not be in a fully consistent state. This may happen in a number of circumstances, such as:

- Stopping in the middle of a constructor or destructor.
- Displaying an object in scope, but before its constructor has been called.
- Viewing a dangling pointer to an object, that is, a pointer to an object in memory that has been released by the program. This may be stack memory, but also heap memory. (If the target is running with memory debugging enabled, then TotalView does check that the object to be displayed does not lie in a deallocated region. If it does, then it does not call your $\text{TV\_ttf\_display\_type}$, and will display the data in their raw form You should not, however, rely on this check.)
In the absence of C++View, this is not a problem, as displaying the data is just a matter of reading memory. However, with C++View, displaying data now involves executing functions in the target code. Your functions should be careful to check that the object to be displayed is in a consistent state. If you can't establish that with certainty, then it should not attempt to format the data, and instead it should return TV_ttf_format_failed.

Otherwise, your target program may crash when you attempt to display an object at an inappropriate time. As with any function call made from TotalView (expression list, evaluation window, etc.), TotalView recovers from this in a limited manner by posting an error message and restoring the stack to its original state. However, the target code may be left in an inconsistent or corrupted state, and further progress may not be possible or useful.

You may not place a breakpoint in a TV_ttf_display_type function. If you do, the callback will be aborted similarly, and TotalView will display an error.

**Memory Management**

You must make sure that the formatted data you want displayed by TotalView (the data whose address you supply as the third parameter to TV_ttf_add_row) remains allocated after the call to your TV_ttf_display_type function returns. In practice this means that you shouldn't allocate these data on the stack. Your TV_ttf_display_type function may be called at anytime, including when your target program may be in the memory manager. For this reason it is inadvisable to allocate or deallocate dynamic memory in your TV_ttf_display_type functions. If the formatted data are manufactured, that is, generated by TV_ttf_display_type rather than already existing, then the memory for those data should be allocated during the target's normal course of execution.

You may find it convenient to have your program format data as part of its normal operations. That way there are no side-effects to worry about when TotalView calls your TV_ttf_display_type callback function.

The field_name and type_name string parameters to TV_ttf_add_row do not need to remain allocated after the call to TV_ttf_add_row.

**Multithreading**

Accessing shared data in multithreaded environments will usually need some sort of access control mechanism to protect its consistency and correctness. Your TV_ttf_display_type functions must be coded carefully if they need to access data that are usually protected by a lock or mutex. Attempting to take the lock or mutex may result in deadlock if the mutex is already locked.

Usually the threads in the program will have been stopped when TotalView calls the TV_ttf_display_type function. If the mutex is locked before TotalView calls TV_ttf_display_type, then an attempt by TV_ttf_display_type to lock the mutex will result in deadlock.
If you are designing a `TV_ttf_display_type` that needs to access data usually protected by a lock or mutex, consider whether you are able to determine whether the data are in a consistent state without having to take the lock. It might be enough to be able to determine whether the mutex is locked. If the data cannot be accessed safely, have the `TV_ttf_display_type` return `TV_ttf_format_failed` or `TV_ttf_format_raw` according to what fits best with your requirements.

**Tips and Tricks**

Consider constructing the type name on-the-fly. This can save time and memory. As an example, consider the `TV_ttf_display_type` for `BoundsCheckedArray<T>` we discussed earlier:

```cpp
template <class T>
int TV_ttf_display_type ( const BoundsCheckedArray<T> *a )
{
    char type [ 4096 ];

    snprintf ( type, sizeof ( type ), "value_type[%d]", a->get_size () );

    (void) TV_ttf_add_row ( "array_values", type, a->get_array () );
    return TV_ttf_format_ok;
}
```

Note how we constructed an array type. The alternative would be to iterate `a->get_size()` times calling `TV_ttf_add_row()`. Depending on the number of elements, this could exhaust the API's buffer. In addition, there is a time penalty since TotalView will need to handle each line added by `TV_ttf_add_row` separately.

Constructing the array type as we did not only eliminates these disadvantages, it also provides other advantages. For example, as TotalView now knows that what is being presented is really an array, all the normal operations on arrays such as sorting, filtering, etc. are available.

**Core Files**

Because C++View needs to call a function in your program, C++View does not work with core files.

**Using C++View with ReplayEngine**

In general, C++View can be used with ReplayEngine just as with normal TotalView debugging. However, there are some differences you should be aware of. In both record mode and replay mode, TotalView switches your process into ReplayEngine's `volatile` mode before calling your `TV_ttf_display_type` function. When the call finishes, TotalView switches the process out of volatile mode. On entering volatile mode, ReplayEngine saves the state of the process, and on exiting volatile mode, ReplayEngine restores the saved status.
In most cases, executing `TV_ttf_display_type` in volatile mode behaves as you would expect. However, because ReplayEngine restores the earlier process state when it leaves volatile mode, any changes to process memory, such as writing to a variable, made while in volatile mode are lost.

This fact has implications for your program if your `TV_ttf_display_type` function modifies global or static data upon which either the function or the program relies. If `TV_ttf_display_type` does not change any global state, you will see no change in behavior when you engage ReplayEngine. However, if you generate synthetic values, such as the average, maximum or minimum values in an array, you cannot compute these in your `TV_ttf_display_type` function as the results will be lost when the function call terminates. Instead, consider generating them as a by-product of the program's normal execution as described in the section on Memory Management.

For more information on ReplayEngine, see *Getting Started with ReplayEngine*.

The behavior of C++View transformations under ReplayEngine is similar to TotalView for HPC's behavior when evaluating expressions with ReplayEngine enabled. For a discussion of this, see “Expression Evaluation with ReplayEngine” in the TotalView for HPC User Guide.

The following code demonstrates how engaging ReplayEngine might affect calls to `TV_ttf_display_type`. This example is shipped with the ReplayEngine example files as `cppview_example_5.cc`.

```c
/* Example program demonstrating TotalView's C++View with ReplayEngine. */
/* Run with (in both record and replay modes) and without ReplayEngine. */
/* Note how c in main is displayed in the various cases. */

#include <stdio.h>
#include "tv_data_display.h"

static int counter;

class C {
  public:
    int  value;

    C () : value ( 0 ) {}
  / * C */

  int TV_ttf_display_type(const C *c)
  {  
    int ret_val = TV_ttf_format_ok;
    int err;

    // if Replay is engaged, this write to the global is lost because
    // the ttf function is evaluated in volatile mode
    counter++;
```
// error checking omitted for brevity
(void) TV_ttf_add_row ( "value", "int", &(c->value) );

// show how many times we've been called. Will always be zero
// with Replay engaged because the update is lost when the
// call to TV_ttf_display_type returns.
(void) TV_ttf_add_row ( "number_of_times_called", "int", &counter );

return ret_val ;
} /* TV_ttf_display_type */

int main(int argc, char *argv[])
{
  C c;

  c.value = 1;

  c.value++;  // should be 1 **before** this line is executed
  c.value++;  // should be 2 **before** this line is executed

  /* c.value should be 3 */

  return 0;
} /* main */

Compile and link the program with tv_data_display.c (see Compiling and linking tv_data_display.c). Follow this procedure:

1. Start the program under TotalView and enter the function main.
2. Dive on the local variable c, and note how the synthetic member number_of_times_called changes as you step through the program.
3. Restart, but this time with ReplayEngine engaged.
4. Notice the changes to the value member as you move forwards and backwards, and that the synthetic member number_of_times_called remains 0 because the increment in TV_ttf_display_type is lost when the function returns.

C

Although primarily intended for C++, C++View may be usable with C. C does not allow overloading so there may be at most one TV_ttf_display_type function with external linkage present. If you are interested in formatting only one type, then this restriction will not be constraining.
You may be able to work around this problem by defining separate `TV_ttf_display_type` functions as before, but placing each in a different file, and defining them to be static. Since the visibility of each definition is limited to the translation unit in which it appears, multiple functions can coexist.

This work-around, however, depends on the nature of the debug information emitted by the compiler. Some compilers do not place static functions in an indexable section in the debug information, or may try to optimize them out. If TotalView cannot find the function, it will not be called. TotalView cannot traverse the entire resolved symbol table to find these functions, as it would incur significant performance problems.

**Fortran**

Fortran variables don't readily lend themselves to transformation by C++View, but in some cases, such as when using a common block with Cray pointer variables, it is possible to set up a corresponding C structure and then use that type to push the transformation.

**Example**

Consider this test case using Cray pointers in a common block, including three parts:

- The Fortran code
- A common block defined in an include file
- The C code containing the C++View code

**The Fortran Code**

Here, the Fortran code sets up a common block with a few variables and then assigns them some values.

```fortran
program pointerp
  call stuff
  end

subroutine stuff

  include 'foop.cmn'

  foo = 42
  ix = 11
  iy = 12
  iz = 13

  call doit(ix)

  call readit
```

return
end

subroutine doit(ix_x)
include 'foop.cmn'
ipxp = malloc(8*ix_x*foo)
ipyp = malloc(8*iy*iz)
xp = 3
yp = 5
return
end

subroutine readit
include 'foop.cmn'
xp = 4
return
end

The include File

The Fortran include file foop.cmn sets up a common block foo1 that corresponds to the C structure extern foo1_, both in bold below.

The include file, foop.cmn:

integer :: foo, ix, iy, iz
real(kind=8) :: xp, yp
pointer (ipxp, xp(foo,ix))
pointer (ipyp, yp(iy,iz))
common /foo1/ ix, iy, iz, foo, ipxp, ipyp

The C Code

The C code fortranTV.c defines structure extern foo1_, aligned to the Fortran common block foo1. Then, in the TV_ttf_display_type routine for the struct foo, the calls to TV_ttf_add_row follow the layout of the data in the common block, allowing us to view the data as we want to see it.

The C code, fortranTV.c:

#include <stdio.h>
#include "tv_data_display.h"
#ifdef __cplusplus
extern "C" {
#endif

extern struct foo { int x ; } foo1_;

#ifdef __cplusplus
}
#endif

// Routine data display declaration
int TV_ttf_display_type(const struct foo *parameter) {

    // Assign 'data' to the start of the common block
    int *data = (int *)parameter;

    // Pick up the Cray pointer
    double **ptr = (double **) &data[4];
    char typeName[64];

    TV_ttf_add_row("ix", "int", &data[0]);
    TV_ttf_add_row("iy", "int", &data[1]);
    TV_ttf_add_row("iz", "int", &data[2]);
    TV_ttf_add_row("foo", "int", &data[3]);

    sprintf(typeName, "double[%d]", data[0]*data[3]);
    TV_ttf_add_row("ipxp", typeName, ptr[0]);

    sprintf(typeName, "double[%d]", data[1]*data[2]);
    TV_ttf_add_row("ipyp", typeName, ptr[1]);

    return TV_ttf_format_ok;
}

## Compiling and Linking

First compile the TotalView `tv_data_display.c` routine, as described in “Compiling and linking tv_data_display.c” on page 334.

Build the program and the C program to add in the C++View transform:

    ifort -g -c pointerp.f

    ifort -g -c fortranTV.c -I$TVINCLUDE

Finally, link the program:

    ifort -g -o crayptr pointerp.o tv_data_display.o fortranTV.o
Debugging

When you debug, set a breakpoint on the return statement on line 20, in subroutine `dooit`. Run to the breakpoint and then dive on the common block `foo1`.

Figure 10 – Using C++ View with Fortran, diving on the Fortran pointer data

To see the data transformed more clearly, expand the type information (downward arrow with the + sign) and change the language to C or C++.

Figure 11 – Using C++ View with Fortran, changing language to C++
Then change the type from $void to $foo, Figure 12.

Note that, while the original display of the common block shows the Cray pointers as integers (because a Cray pointer is actually an integer that holds only a memory address), the final, transformed display shows the data referenced by the pointers, or the arrays of doubles.

**Figure 12 – Using C++ View with Fortran, transform the type**

---

**Compiling and linking `tv_data_display.c`**

Your distribution includes the file `tv_data_display.c` in the `<totalview-installation>/src` directory. This file contains the implementation of the interface between your `TV_ttf_display_type` functions and TotalView. This is distributed as source. You will need to compile this file and link it with your application.

You should take care to ensure that there is only one instance of `tv_data_display.c` present in your running application. One way in which multiple instances could creep in is if you link separate copies of the `tv_data_display.c` into independent shared libraries that your program uses. To avoid this type of problem, we strongly suggest that you build `tv_data_display.c` into its own separate shared library that can be shared by all the libraries your application uses. For example:

```
setenv TVSOURCE /usr/local/toolworks/totalview.8.9.0-2/src
setenv TVINCLUDE /usr/local/toolworks/totalview.8.9.0-2/include
gcc -g -Wall -fPIC -c $TVSOURCE/tv_data_display.c -I$TVINCLUDE gcc -g 
```
Some compilers or linkers will perform a type of garbage collection step and eliminate code or data that your application does not use. This affects C++View in two ways:

1. Your `TV_ttf_display_type` functions are unlikely to be called by your program.
2. Leading on from this, some of the entities in `tv_data_display.c` may not be reachable from your program.

As a result, the compiler or linker may identify your `TV_ttf_display_type` or `tv_data_display.c` as candidates for garbage collection and elimination. You can try to work around this problem by trying to create references to the `TV_ttf_display_type` functions.

Better still, we suggest identifying the flags for your compiler or linker that disable garbage collection. On AIX, for example, the linker flag `-bkeepfile:<filename>` tells the linker not to perform garbage collection in the file named `<filename>`.

**C++View Example Files**

Your TotalView distribution includes an examples directory, `<totalview-installation>/examples`, which includes the following C++View example files:

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cppview_example_1</code></td>
<td>A simple example showing two <code>TV_ttf_display_type</code> functions, one a function at global scope, the other a class function. It also demonstrates elision.</td>
</tr>
<tr>
<td><code>cppview_example_2</code></td>
<td>A simple example using templates, showing how the type named in the template can be passed to <code>TV_ttf_add_row</code>.</td>
</tr>
<tr>
<td><code>cppview_example_3</code></td>
<td>A more complex example using templates, showing how a <code>TV_ttf_display_type</code> function can be either generic or specialized for a particular instantiation of a template class. It also demonstrates elision.</td>
</tr>
<tr>
<td><code>cppview_example_4</code></td>
<td>A more complex example showing the use of STL container classes, elision, and the different values that <code>TV_ttf_display_type</code> can return.</td>
</tr>
<tr>
<td><code>cppview_example_5</code></td>
<td>This example adds a synthetic member to a class, and can be used to explore how C++View behaves under ReplayEngine.</td>
</tr>
</tbody>
</table>

**NOTE >>** Some compilers, such as some versions of gcc, do not emit debug information for typedefs in class scopes, and therefore TotalView cannot find the type underlying `value_type` so C++View may not work with those compilers.
Limitations

With the exception of Sun, compilers that emit STABS debug information do not handle C++ namespaces. This affects TotalView in general and C++View in particular, in that references to entities in namespaces are not always resolved.

Licensing

The C++View API library is distributed as two files. The first is tv_data_display.c, an ANSI C file that contains the implementation of the API used by your TV_ttf_display_type functions. The other is tv_data_display.h, which is a matching header file.

These files are licensed so as to permit unlimited embedding and redistribution.
Running TotalView

This section of the *TotalView for HPC Reference Guide* contains information about command-line options you use when starting TotalView and the TotalView Debugger Server.

**Chapter 7, “TotalView Command Syntax,”** on page 338

TotalView contains a great number of command-line options. Many of these options allow you to override default behavior or a behavior that you've set in a preference or a startup file.

**Chapter 8, “TotalView Debugger Server Command Syntax,”** on page 352

This chapter describes how you modify the behavior of the `tvdsvr`. These options are most often used if a problem occurs in launching the server or if you have some very specialized need. In most cases, you can ignore the information in this chapter.
Overview

This chapter describes the syntax of the `totalview` command. Topics in this chapter are:

- Command-Line Syntax
- Command-Line Options
Command-Line Syntax

Format

```
totalview [ options ] [ executable [ core-file | recording-file ] ] [-a [ args ] ]
```

or

```
totalview [ options ] -args executable [ args ]
```

Arguments

```
options
  TotalView options.

executable
  Specifies the path name of the executable being debugged. This can be an absolute or relative path name. The executable must be compiled with debugging symbols turned on, normally the `-g` compiler option. Any multiprocess programs that call `fork()`, `vfork()`, or `execve()` should be linked with the `dbfork` library.

core-file
  Specifies the name of a core file. Use this argument in addition to `executable` when you want to examine a core file with TotalView.

recording-file
  Specifies the name of a saved replay recording session file. Use this argument in addition to `executable` when you want to replay the recording session with TotalView.

args
  Default target program arguments.
```

Description

TotalView is a source-level debugger with a motif-based graphic user interface and features for debugging distributed programs, multiprocess programs, and multithreaded programs. TotalView is available on a number of different platforms.

If you specify mutually exclusive options on the same command line (for example, `-dynamic` and `-no_dynamic`), the last option listed is used.
Command-Line Options

-a args
Pass all subsequent arguments (specified by args) to the program specified by filename. This option must be the last one on the command line.

-aix_use_fast_ccw
Defined only on AIX, a synonym for the platform-independent -use_fast_wp, for TotalView script backward compatibility. See -use_fast_wp for more information. You must set this option on the command line; you cannot set it interactively using the CLI.

-aix_use_fast_trap
Defined only on AIX, a synonym for the platform-independent -use_fast_trap, for TotalView script backward compatibility. See -use_fast_trap for more information. You must set this option on the command line; you cannot set it interactively using the CLI.

-args filename [args]
Specifies filename as the executable to debug, with args as optional arguments to pass to your program. This option must be listed last on the command line. You can also use --args instead of -args, for compatibility with other debuggers.

-background color
Sets the general background color to color.

-bg color
Same as -background.

Default: light blue

-bluegene_q_user_threads
Enables handling of user-level (M:N) thread packages on BlueGene/Q systems.

-no_bluegene_q_user_threads
(Default) Disables handling of user-level (M:N) thread packages, improving startup performance at high scale. There is usually a 1:1 correspondence between user-level threads and kernel-level threads on BlueGene/Q systems.

-compiler_vars
(Alpha, HP, and SGI only.) Shows variables created by the Fortran compiler, as well as those in the user’s program.

Some Fortran compilers (HP f90/f77, HP f90, SGI 7.2 compilers) output debugging information that describes variables the compiler itself has invented for purposes such as passing the length of character*(*) variables. By default, TotalView suppresses the display of these compiler-generated variables.

However, you can specify the -compiler_vars option to display these variables. This is useful when you are looking for a corruption of a run-time descriptor or are writing a -compiler.
-no_compiler_vars
  (Default) Tells TotalView that it should not show variables created by the Fortran compiler.

-control_c_quick_shutdown-ccq
  (Default) Tells TotalView to kills attached processes and exits.

-no_control_c_quick_shutdown-nccq
  Invokes code that sometimes allows TotalView to better manage the way it kills parallel jobs when it works with
  management systems. This has only been tested with SLURM. It may not work with other systems.

-cuda
  (Default) Enables CUDA debugging with TotalView.

  -no_cuda
    Disables CUDA debugging. Any CUDA kernels launched on a GPU device are not seen by the debugger, so the debugger can only debug the host code. -nocuda is the identical command.

-dbfork
  (Default) Catches the fork(), vfork(), and execve() system calls if your executable is linked with the dbfork library.

  -no_dbfork
    Tells TotalView that it should not catch fork(), vfork(), and execve() system calls even if your executable is linked with the dbfork library.

-debug_file console_outputfile
  Redirects TotalView console output to a file named console_outputfile.
  
  If consoleoutputfile is the string UNIQUE, the filename tv_dump.hostname.pid is used. If console_outputfile contains
  the string '$$' (note the escaping single quotes), hostname.pid is substituted. UNIQUE and 'SS' are useful
  for separating the console output when running multiple tvdsvr processes.
  
  All TotalView console output is written to stderr.

-default_parallel_attach_subset subset_specification
  Specifies a set of MPI ranks to be attached to when an MPI job is created or attached to. The subset_specification
  is a space-separated list, the elements of which can be in one of these forms:

  rank: that rank only

  rank1-ranks2: all ranks between rank1 and rank2 inclusive

  rank1-rank2:stride: every stride\text{th} rank between rank1 and rank2

  A rank must be either a positive decimal integer or max (the last rank in the MPI job).

  A subset_specification that is the empty string ("") is equivalent to 0-max.

  The default_parallel_attach_subset is used to initialize the -parallel_attach_subset property of an MPI
  starter process, which can be get or set in the CLI using:

  TV::process get dpid parallel_attach_subset

  TV::process set dpid parallel_attach_subset -subset_specification
The CLI `dattach` and `dload -parallel_attach_subset -subset_specification` overrides the default `parallel_attach_subset` and sets the `parallel_attach_subset` property of the process being attached or loaded.

```-demangler=compiler```

Overrides the demangler and mangler TotalView uses by default. The following indicate override options.

```-demangler=compaq: HP cc on Linux (alpha)```

```-demangler=gnu: GNU C++ on Linux Alpha```

```-demangler=gnu_dot: GNU C++ on Linux x86```

```-demangler=gnu_v3: GNU C++ Linux x86```

```-demangler=kai: KAI C++```

```-demangler=kai3_n: KAI C++ version 3.n```

```-demangler=kai_4_0: KAI C++```

```-demangler=spro: SunPro C++ 4.0 or 4.2```

```-demangler=spro5: SunPro C++ 5.0 or later```

```-demangler=sun: Sun CFRONT C++```

```-demangler=xlc: IBM XLC/VAC++ compilers```

```-display displayname```

Set the name of the X Windows display to `displayname`. For example, `-display vinnie:0.0` will display TotalView on the machine named "vinnie."

**Default:** The value of your DISPLAY environment variable.

```-dll_ignore_prefix list```

The colon-separated argument to this option tells TotalView that it should ignore files having this prefix when making a decision to ask about stopping the process when it `dlopen` a dynamic library. If the DLL being opened has any of the entries on this list as a prefix, the question is not asked.

```-dll_stop_suffix list```

The colon-separated argument to this option tells TotalView that if the library being opened has any of the entries on this list as a suffix, it should ask if it should open the library.

```-dlopen_always_recalculate```

(DefaultValue). Reevaluates breakpoint specifications on every `dlopen` call.

```-no_dlopen_always_recalculate```

Enables `dlopen` event filtering, deferring the reevaluation of breakpoint specifications until after the `dlopen` event. The point at which the breakpoint specifications are reevaluated depends on the value of the `TV::dlopen_recalculate_on_match` variable (see `-dlopen_recalculate_on_match glob-list`).

This setting impacts scalability in HPC computing environments. For details, see “Filtering dlopen Events” on page 382.
-dlopen_recalculate_on_match glob-list

Default: "" (the empty string)

This option's argument is a colon-separated list of simple glob patterns used to compare and match the dlopened library. A simple glob pattern is a string, optionally ending with asterisk character ('*'). For information on the semantics of glob pattern matching, see TV::dlopen_recalculate_on_match.

Used with -no_dlopen_always_recalculate, when a dlopen event occurs, the name of the dlopened library is matched against the list of glob patterns; if the glob-list is empty (the default) or the name of the dlopened library does not match the glob-list, then breakpoint reevaluation is deferred until the process stops for some other reason (e.g., the process hits a breakpoint, the user stops the process, the process encounters a signal, etc.). If the library name matches a pattern, the breakpoints are reevaluated immediately.

-dlopen_read_libraries_in_parallel

Enables dlopen events to be handled in parallel, reducing client/server communication overhead by using MRNet to fetch the library information.

-no_dlopen_read_libraries_in_parallel

(Default). Disables handling dlopened events in parallel.

This setting impacts scalability in HPC computing environments. For details, see “Handling dlopen Events in Parallel” on page 384.

-dump_core

Allows TotalView to dump a core file of itself when an internal error occurs. This is used to help Rogue Wave Software debug problems.

-e commands

Tells TotalView to immediately execute the CLI commands named within this argument. All information you enter here is sent directly to the CLI's Tcl interpreter. For example, the following writes a string to stdout:

cli -e 'puts hello'

You can have more than one -e option on a command line.

-ent

Tells TotalView that it should only use an Enterprise license.

-no_ent

Tells TotalView that it should not use an Enterprise license. You may combine this with -no_team or --noteamplus.

-env variable=value

Tells TotalView to add an environment variable to the environment variables passed to your program by the shell. If the variable already exists, it effectively replaces the previous value. You need to use this command for each variable being added; that is, you cannot add more than one variable with an env command.

-foreground color

Sets the general foreground color (that is, the text color) to color.
-fg color
    Same as -foreground.

Default: black

-f9x_demangler=compiler
    Overrides the Fortran demangler and mangler TotalView uses by default. The following indicate override options.

    -demangler=spro_f9x_4: SunPro Fortran, 4.0 or later
    -demangler=xlf: IBM Fortran

-global_types
    (Default) Lets TotalView assume that type names are globally unique within a program and that all type definitions with the same name are identical. The C++ standard asserts that this must be true for standard-conforming code.

    If this option is set, TotalView will attempt to replace an opaque type (struct foo *p;) declared in one module, with an identically named defined type in a different module.

    If TotalView has read the symbols for the module containing the non-opaque type definition, then when displaying variables declared with the opaque type, TotalView will automatically display the variable by using the non-opaque type definition.

    -no_global_types
        Specifies that TotalView cannot assume that type names are globally unique in a program. You should specify this option if your code has multiple different definitions of the same named type, since otherwise TotalView can use the wrong definition for an opaque type.

-gnu_debuglink
    Tells TotalView that if a program or library has a .gnu_debug_link section, it should look for a gnu_debug_link file. If found, TotalView reads the debugging information from this file.

    -no_gnu_debuglink
        Do not load information from a gnu_debug_link file even if the file has a .gnu_debug_link section.

-gnu_debuglink_checksum
    Tells TotalView that it should validate the gnu_debug_link file's checksum against the checksum contained in the process's .gnu_debuglink section.

    -no_gnu_debuglink_checksum
        Do not compare checksums. Only do this if you are absolutely certain that the debug file matches.

-ipv6_support
    Directs TotalView to support IPv6 addresses.

    -no_ipv6_support
        (Default) Do not support IPv6 addresses.
-kcc_classes
  (Default) Converts structure definitions output by the KCC compiler into classes that show base classes and virtual base classes in the same way as other C++ compilers. See the description of the TV::kcc_classes variable for a description of the conversions that TotalView performs.

-no_kcc_classes
  Specifies that TotalView will not convert structure definitions output by the KCC compiler into classes. Virtual bases will show up as pointers, rather than as data.

-ib
  (Default) Loads action points automatically from the filename.TVD.v3breakpoints file, providing the file exists.

-nlb
  Tells TotalView that it should not automatically load action points from an action points file.

-load_session session_name
  Loads into TotalView the session named in session_name. If the preference “Show Startup Parameters when TotalView starts” is set, this option launches the Session Manager’s Program Session screen where you can edit the session’s properties and then launch the session; otherwise, the option immediately loads the session into TotalView, launching the Root and Process windows. Session names with spaces must be enclosed in quotes, for example, "my debug session". Sessions that attach to an existing process cannot be loaded using this option; rather, use the -pid option instead.

-local_interface string
  Sets the interface name that the server uses when it makes a callback. For example, on an IBM PS2 machine, you would set this to css0. However, you can use any legal inet interface name. (You can obtain a list of the interfaces if you use the netstat -i command.)

-memory_debugging
  Enables memory debugging. By adding the following suboptions, you enable that particular feature using its the feature’s default configuration. In most cases, you will want to use one or more of the following sub-options.

  -mem_detect_use_after_free
    Tests for memory use after memory is freed.

  -mem_detect_use_after_free
    Tests for memory use after memory is freed.

  -mem_guard_blocks
    Surrounds allocated memory blocks with guard blocks.

  -mem_hoard_freed_memory
    Tells the Memory Debugger to hoard memory blocks instead of releasing them when a free() routine is called.

  -mem_hoard_low_memory_threshold nnnn
    Sets the low memory threshold amount. When memory falls below this amount an event will be fired.

  -mem_notify_events
    Turns on memory event notification.
-no_mem_notify_events turns event notification off.

-mem_paint_all
Paint both allocated and deallocated blocks with a bit pattern.

-mem_paint_on_alloc
Paint memory blocks with a bit pattern when they are allocated.

-mem_paint_on_dealloc
Paint memory blocks with a bit pattern when they are freed.

-mem_red_zones_overruns
Turn on testing for Red Zones overruns.

-mem_red_zones_size_ranges min:max, min:max,...
Defines the memory allocations ranges for which Red Zones are in effect. Ranges can be specified as follows: x:y allocations from x to y.
: y allocations from 1 to y
x: allocations of x and higher
x allocation of x

-mem_red_zones_underruns
Turn on testing for Red Zones underruns.

-message_queue
(Default) Enables the display of MPI message queues when debugging an MPI program.

-mqd
Same as -message_queue.

-mqd
Same as -message_queue.

-no_message_queue
Disables the display of MPI message queues when you are debugging an MPI program. This might be useful if something is overwriting the message queues and causing TotalView to become confused.

-no_mqd
Same as -no_message_queue.

-mpi starter
Names the MPI that your program requires. The list of starter names that you enter are those that appear in the Parallel system pull down list contained within the New Program's Parallel tab. If the starter name has more than one word (for example, Open MPI), enclose the name in quotes. For example:

-mpi "Open MPI"

-newUI
Launches the NextGen TotalView for HPC UI rather than TotalView's traditional interface.

-nodes
Specifies the number of nodes upon which the MPI job will run.
**-no_startup_scripts**

Tells TotalView not to reference any initialization files during startup. Note that this negates all settings in all initialization files. Aliases are `-nostartupscripts` and `-nss`.

**-nohand_cursor**

By default, the cursor in the source pane of the process window turns into a hand cursor when hovering over an element you can dive on (a red box is also drawn around the applicable code). Specify this option to override this behavior and retain the usual arrow cursor.

**-np**

Specifies how many tasks that TotalView should launch for the job. This argument usually follows a `-mpi` command-line option.

**-nptl_threads**

Tells TotalView that your application is using NPTL threads. You only need use this option if default cannot determine that you are using this threads package.

**-no_nptl_threads**

Tells TotalView that you are not using the NPTL threads package. Use this option if TotalView thinks your application is using it and it isn’t.

**-oldroot**

Displays the Root Window used in versions prior to TotalView 8.15.0. Using `--oldroot` or `--newroot` overrides the `TV::GUI::old_root_window` value.

- `newroot` (Default) Displays the new Root Window. This is useful when `TV::GUI::old_root_window` is set to `true` in the `.tvdrc` file and you wish to use the new Root Window.

**-parallel**

(Default) Enables handling of parallel program run-time libraries such as MPI, PE, and UPC.

- `no_parallel` Disables handling of parallel program run-time libraries such as MPI, PE, and UPC. This is useful for debugging parallel programs as if they were single-process programs.

**-parallel_attach option**

Sets the action that TotalView takes when starting a parallel program. Possible options are:

- `yes` (default) Attaches to all processes in a parallel program, unless the process being launched or attached to has a non-empty `parallel_attach_subset` property. In this case, only the subset of processes specified in the `parallel_attach_subset` are attached.
- `no`: Attaches to no processes in a parallel program.
- `ask` Asks which processes to attach to by posting the subset attach dialog box if the debugger GUI is open.

This option works in concert with the `parallel_attach_subset` property (see `-default_parallel_attach_subset`) of an MPI starter process, which specifies a set of MPI tasks to attach to when the debugger launches or attaches to an MPI job.
Modifying this setting does not affect the `parallel_attach_subset` property itself.

`-patch_area_base address`
Allocates the patch space dynamically at `address`. See “Allocating Patch Space for Compiled Expressions” in the TotalView for HPC User Guide.

`-patch_area_length length`
Sets the length of the dynamically allocated patch space to this `length`. See “Allocating Patch Space for Compiled Expressions” in the TotalView for HPC User Guide.

`-pid pid filename`
Attaches to process `pid` for executable `filename` when TotalView starts executing.

`-procs`
Specifies how many tasks that TotalView should launch for the job. This argument usually follows a `-mpi` command-line option.

`-processgrid`
Displays the Processes/Ranks Tab in the Process Window. Note that enabling this tab can significantly affect performance, particularly for large, massively parallel applications.

`-noprocessgrid (default)`
Does not display the Processes/Ranks Tab in the Process Window. This command qualifier is helpful when you wish to disable the Processes/Ranks Tab for a debug session and you have `TV::GUI::process_grid_wanted` set to `true` in your `.tvdrc` file.

`-remote hostname[:portnumber]`
Debugs an executable that is not running on the same machine as TotalView. For `hostname`, you can specify a TCP/IP host name (such as `vinnie`) or a TCP/IP address (such as `128.89.0.16`). Optionally, you can specify a TCP/IP port number for `portnumber`, such as `:4174`. When you specify a port number, you disable the autolaunch feature. For more information on the autolaunch feature, see “Setting Single Process Server Launch” in the TotalView for HPC User Guide.

`-r hostname[:portnumber]`
Same as `-remote`.

`-replay`
Enables the ReplayEngine when TotalView begins. This command-line option is ignored if you do not have a license for ReplayEngine.

`-s pathname`
Specifies the path name of a startup file that will be loaded and executed. This path name can be either an absolute or relative name.

You can add more than one `-s` option on a command line.
-serial device[options]

Debugs an executable that is not running on the same machine as TotalView. For device, specify the device name of a serial line, such as /dev/com1. Currently, the only option you are allowed to specify is the baud rate, which defaults to 38400. For more information on debugging over a serial line, see "Debugging Over a Serial Line" in the TotalView for HPC Users Guide.

-search_path pathlist

Specify a colon-separated list of directories that TotalView will search when it looks for source files. For example:

totalview -search_path proj/bin:proj/util

-signal_handling_mode "action_list"

Modifies the way in which TotalView handles signals. You must enclose the action_list string in quotation marks to protect it from the shell.

An action_list consists of a list of signal_action descriptions separated by spaces:

    signal_action[ signal_action] ...

A signal action description consists of an action, an equal sign (=), and a list of signals:

    action=signal_list

An action can be one of the following: Error, Stop, Resend, or Discard. For more information on the meaning of each action, see Chapter 3 of the TotalView for HPC User Guide.

A signal_specifier can be a signal name (such as SIGSEGV), a signal number (such as 11), or a star (*), which specifies all signals. We recommend that you use the signal name rather than the number because number assignments vary across UNIX sessions.

The following rules apply when you are specifying an action_list:
(1) If you specify an action for a signal in an action_list, TotalView changes the default action for that signal.
(2) If you do not specify a signal in the action_list, TotalView does not change its default action for the signal.
(3) If you specify a signal that does not exist for the platform, TotalView ignores it.
(4) If you specify an action for a signal more than once, TotalView uses the last action specified.

If you need to revert the settings for signal handling to built-in defaults, use the Defaults button in the File > Signals dialog box.

For example, here's how to set the default action for the SIGTERM signal to resend:

    "Resend=SIGTERM"

Here's how to set the action for SIGSEGV and SIGBUS to error, the action for SIGHUP to resend, and all remaining signals to stop:

    "Stop=* Error=SIGSEGV,SIGBUS Resend=SIGHUP"

-shm "action_list"

Same as -signal_handling_mode.
-starter_args "arguments"
  Tells TotalView to pass arguments to the starter program. You can omit the quotation marks if arguments is just one string without any embedded spaces.

-stderr pathname
  Names the file to which TotalView writes the target program's stderr information while executing within TotalView. If the file exists, TotalView overwrites it. If the file does not exist, TotalView creates it.

-stderr_append
  Tells TotalView to append the target program's stderr information to the file named in the -stderr command, specified in the GUI, or in the TotalView TV::default_stderr_filename variable. If the file does not exist, TotalView creates it.

-stderr_is_stdout
  Tells TotalView to redirect the target program's stderr to stdout.

-stdin pathname
  Names the file from which the target program reads information while executing within TotalView.

-stdout pathname
  Names the file to which TotalView writes the target program's stdout information while executing within TotalView. If the file exists, TotalView overwrites it. If the file does not exist, TotalView creates it.

-stdout_append
  Tells TotalView to append the target program's stdout information to the file named in the -stdout command, specified in the GUI, or in the TotalView TV::default_stdout_filename variable. If the file does not exist, TotalView creates it.

-tasks
  Specifies how many tasks that TotalView should launch for the job. This argument usually follows a -mpi command-line option.

-team
  Tells TotalView that it should only use a Team license.

  -no_team
    Tells TotalView that it should not use an Enterprise license. You may combine this with -no_ent or -noeamplusr.

  -teamplus
    Tells TotalView that it should only use a Team Plus license.

  -no_teamplus
    Tells TotalView that it should not use a Team PLus license. You may combine this with -no_ent or -noteam.

-tvhome pathname
  The directory from which TotalView reads preferences and other related information and the directory to which it writes this information.
-use_fast_trap

Controls TotalView’s use of the target operating system’s support of the fast trap mechanism for compiled conditional breakpoints, also known as EVAL points. As of TotalView 8.7, when this was introduced, only AIX supported the fast trap mechanism for breakpoints, but we anticipate other operating systems adding support. You must set this option on the command line; you cannot set it interactively using the CLI.

Your operating system may not be configured correctly to support this option. See the TotalView for HPC Release Notes on our web site for more information.

-use_fast_wp

Controls TotalView’s use of the target operating system’s support of the fast trap mechanism for compiled conditional watchpoints, also known as CDWP points. As of TotalView 8.7, when this was introduced, only AIX supported the fast trap mechanism for watchpoints, but we anticipate other operating systems adding support. You must set this option on the command line; you cannot set it interactively using the CLI.

Your operating system may not be configured correctly to support this option. See the TotalView for HPC Release Notes on our web site for more information.

-user_threads

(Default) Enables handling of user-level (M:N) thread packages on systems where two-level (kernel and user) thread scheduling is supported. (Note: This option does not apply to -BlueGene/Q systems; instead, see -bluegene_q_user_threads.)

-no_user_threads

Disables handling of user-level (M:N) thread packages. This option may be useful in situations where you need to debug kernel-level threads, but in most cases, this option is of little use on systems where two-level thread scheduling is used.

-verbosity level

Sets the verbosity level of TotalView messages to level, which may be one of silent, error, warning, or info.

Default: info

-xterm_name pathname

Sets the name of the program used when TotalView needs to create a the CLI. If you do not use this command or have not set the TV::xterm_name variable, TotalView will attempt to create an xterm window.
Overview

This chapter summarizes the syntax of the TotalView Debugger Server command, \texttt{tvdsvr}, which is used for remote debugging. Remote debugging occurs when you explicitly call for it or when you are using disciplines like MPI that startup processes on remote servers. For more information on remote debugging, refer to \textit{Setting Up Remote Debugging Sessions} in the TotalView for HPC Users Guide.

Topics in this chapter are:

- The \texttt{tvdsvr} Command and Its Options
- Replacement Characters
The tvdsvr Command and Its Options

tvdsvr { -server | -callback hostname:port | -serial device} [other options]

Description

tvdsvr allows TotalView to control and debug a program on a remote machine. To accomplish this, the tvdsvr program must run on the remote machine, and it must have access to the executables being debugged. These executables must have the same absolute path name as the executable that TotalView is debugging, or the PATH environment variable for tvdsvr must include the directories containing the executables.

You must specify a -server, -callback, or -serial option with the tvdsvr command. By default, TotalView automatically launches tvdsvr using the -callback option, and the server establishes a connection with TotalView. (Automatically launching the server is called autolaunching.)

If you prefer not to automatically launch the server, you can start tvdsvr manually and specify the -server option. Be sure to note the password that tvdsvr prints out with the message:

pw = hexnumhigh:hexnumlow

TotalView will prompt you for hexnumhigh:hexnumlow later. By default, tvdsvr automatically generates a password that it uses when establishing connections. If desired, you can set your own password by using the -set_pw option.

To connect to the tvdsvr from TotalView, you use the File > New Program Dialog Box and must specify the host name and TCP/IP port number, hostname:portnumber on which tvdsvr is running. Then, TotalView prompts you for the password for tvdsvr.

Options

The following options name the port numbers and passwords that TotalView uses to connect with tvdsvr.

-callback hostname:port

(Autolaunch feature only) Immediately establishes a connection with a TotalView process running on hostname and listening on port, where hostname is either a host name or TCP/IP address. If tvdsvr cannot connect with TotalView, it exits.

If you use the -port, -search_port, or -server options with this option, tvdsvr ignores them.

-callback_host hostname

Names the host upon which the callback is made. The hostname argument indicates the machine upon which TotalView is running. This option is most often used with a bulk launch.
-callback_port port-list
Names the ports on the host machines that are used for callbacks. The port-list argument contains a comma-separated list of the host names and TCP/IP port numbers (hostname:port,hostname:port...) on which TotalView is listening for connections from tvdsrv. This option is most often used with a bulk launch.

For more information, see “Setting Up Remote Debugging Sessions” in the TotalView for HPC Users Guide.

-debug_file console_outputfile
Redirects TotalView Debugger Server console output to a file named console_outputfile.

If console_outputfile is the string UNIQUE, the filename tv_dump.hostname.pid is used. If console_outputfile contains the string "$$" (note the escaping single quotes), hostname.pid is substituted. UNIQUE and "$$" are useful for separating the console output when running multiple tvdsrv processes.

Default: All console output is written to stderr.

-nodes_allowed num
Explicitly tells tvdsrv how many nodes the server supports and how many licenses it needs. This is only used for the Cray XT3.

-port number
Sets the TCP/IP port number on which tvdsrv should communicate with TotalView. If this port is busy, tvdsrv does not select an alternate port number (that is, it won't communicate with anything) unless you also specify -search_port.

Default: 4142

-search_port
Searches for an available TCP/IP port number, beginning with the default port (4142) or the port set with the -port option and continuing until one is found. When the port number is set, tvdsrv displays the chosen port number with the following message:

    port = number

Be sure that you remember this port number, since you will need it when you are connecting to this server from TotalView.

-serial device[options]
Waits for a serial line connection from TotalView. For device, specifies the device name of a serial line, such as /dev/com1. The only option you can specify is the baud rate, which defaults to 38400. For more information on debugging over a serial line, see "Debugging Over a Serial Line" in the TotalView for HPC Users Guide.

-server
Listens for and accepts network connections on port 4142 (default).

Using -server can be a security problem. Consequently, you must explicitly enable this feature by placing an empty file named tvdsrv.conf in your /etc directory. This file must be owned by user ID 0 (root). When tvdsrv encounters this option, it checks if this file exists. This file's contents are ignored.
You can use a different port by using one of the following options: `-search_port` or `-port`. To stop `tvdsvr` from listening and accepting network connections, you must terminate it by pressing Ctrl+C in the terminal window from which it was started or by using the `kill` command.

**-set_pw** `hexnumhigh:hexnumlow`

Sets the password to the 64-bit number specified by the `hexnumhigh` and `hexnumlow` 32-bit numbers. When a connection is established between `tvdsvr` and TotalView, the 64-bit password passed by TotalView must match this password set with this option. `tvdsvr` displays the selected number in the following message:

```
pw = hexnumhigh:hexnumlow
```

We recommend using this option to avoid connections by other users.

If necessary, you can disable password checking by specifying the “-set_pw 0:0” option with the `tvdsvr` command. Disabling password checking is dangerous; it allows anyone to connect to your server and start programs, including shell commands, using your UID. Therefore, we do not recommend disabling password checking.

**-set_pws** `password-list`

Sets 64-bit passwords. TotalView must supply these passwords when `tvdsvr` establishes the connection with it. The argument to this command is a comma-separated list of passwords that TotalView automatically generates. This option is most often used with a bulk launch.

For more information, see “Setting Up Remote Debugging Sessions” in the TotalView for HPC Users Guide.

**-verbosity** `level`

Sets the verbosity level of TotalView Debugger Server-generated messages to `level`, which may be one of `silent`, `error`, `warning`, or `info`.

Default: `info`

**-working_directory** `directory`

Makes `directory` the directory to which TotalView connects.

Note that the command assumes that the host machine and the target machine mount identical file systems. That is, the path name of the directory to which TotalView is connected must be identical on both the host and target machines.

After performing this operation, the TotalView Debugger Server is started.
Replacement Characters

When placing a `tvdsvr` command in a Server Launch or Bulk Launch string (see the File > Preferences command within the online Help for more information), you will need to use special replacement characters. When your program needs to launch a remote process, TotalView replaces these command characters with what they represent. Here are the replacement characters:

- **%A**
  Expands to the ALPS Application ID (`apid`), which is a unique identifier for an application started using ALPS `aprun` on Cray XT, XE, and XK. The token is used to construct server path references copied onto the compute nodes' ramdisk under the `/var/spool/alps/apid` directory by the ALPS Tool Helper library.

- **%B**
  Expands to the bin directory where `tvdsvr` is installed.

- **%C**
  Is replaced by the name of the server launch command being used. On most platforms, this is `ssh -x`. On Sun SPARC, this command is `rsh`. If the `TVDSVRLAUNCHCMD` environment variable exists, TotalView will use its value instead of its platform-specific value.

- **%D**
  Is replaced by the absolute path name of the directory to which TotalView will be connected.

- **%F**
  Contains the “tracer configuration flags” that need to be sent to `tvdsvr` processes. These are system-specific startup options that the `tvdsvr` process needs.

- **%H**
  Expands to the host name of the machine upon which TotalView is running. (This replacement character is most often used in bulk server launch commands. However, it can be used in a regular server launch and within a `tvdsvr` command contained within a temporary file.)

- **%I**
  Expands to the `pid` of the MPI starter process. For example, it can contain `mpirun`, `aprun`, etc. It can also be the process to which you manually attach. If no `pid` is available, `%I` expands to 0.

- **%J**
  Expands to the job ID. For MPICH or poe jobs, is the contents of the `totalview_jobid` variable contained either in the starter or first process. If that variable does not exist, it is set to zero (“0”). If it is not appropriate for the kind of job being launched, its value is -1.

- **%K**
  Expands to the `tvdsvr` platform suffix string in situations where a different server must be used. On Blue Gene/L and Blue Gene/P, the `%K` expansion includes `_bg1`, on Blue Gene/Q it includes `_bgq`, and on Cray XT3 Catamount (RedStorm) it includes `_rs`. 

RogueWave

TotalView Debugger Server Command Syntax / Replacement Characters 356
When MRNet is being used as the debugger infrastructure, _mrnet is appended to the normal %K expansion. On Cray XT with MRNet enabled the %K token is expanded to _mrnet, while on Blue Gene/L or Blue Gene/P with MRNet enabled the %K token is expanded to _bgl_mrnet. This convention allows MRNet-specific debugger servers to be launched only when MRNet is being used as the debugger infrastructure.

%L
If TotalView is launching one process, this is replaced by the host name and TCP/IP port number (hostname:port) on which TotalView is listening for connections from tvdsvr.
If a bulk launch is being performed, TotalView replaces this with a comma-separated list of the host names and TCP/IP port numbers (hostname:port,hostname:port...) on which TotalView is listening for connections from tvdsvr.
For more information, see Setting Up Remote Debugging Sessions in the TotalView for HPC Users Guide.

%M
(Sun) Expands to the command name used for a local server launch.

%N
Is replaced by the number of servers that TotalView will launch. This is only used in a bulk server launch command.

%P
If TotalView is launching one process, this is replaced by the password that it automatically generated.
If a bulk launch is being performed, TotalView replaces this with a comma-separated list of 64-bit passwords.

%R
Is replaced by the host name of the remote machine specified in the File > New Program command. When performing a bulk launch, this is replaced by a comma-separated list of the names of the hosts upon which TotalView will launch tvdsvr processes.

%S
If TotalView is launching one process, it replaces this symbol with the port number on the machine upon which TotalView is running.
If a bulk server launch is being performed, TotalView replaces this with a comma-separated list of port numbers.

%t1 and %t2
Is replaced by files that TotalView creates containing information it generates. This is only available in a bulk launch.
These temporary files have the following structure:
(1) An optional header line containing initialization commands required by your system.
(2) One line for each host being connected to, containing host-specific information.
(3) An optional trailer line containing information needed by your system to terminate the temporary file.
The File > Preferences Bulk Server Page allows you to define templates for the contents of temporary files. These files may use these replacement characters. The %N, %t1, and %t2 replacement characters can only be used within header and trailer lines of temporary files. All other characters can be used in header or trailer lines.
or within a host line defining the command that initiates a single-process server launch. In header or trailer lines, they behave as defined for a bulk launch within the host line. Otherwise, they behave as defined for a single-server launch.

%U
(Sun) Expands to the local socket ID.

%V
Is replaced by the current TotalView verbosity setting.

%Z
Expands to the job ID. For MPICH or poe jobs, is the contents of the totalview_jobid variable contained either in the starter or first process. If that variable does not exist, it is set to zero ("0"). If it is not appropriate for the kind of job being launched, its value is -1.
PART III

Platforms and Operating Systems

The three chapters in this part of the Reference Guide describe information that is unique to the computers, operating systems, and environments in which TotalView runs.

Chapter 9, “Platforms and Compilers,” on page 360
Here you will find general information on the compilers and runtime environments that TotalView supports. This chapter also contains commands for starting TotalView and information on linking with the dbfork library.

Chapter 10, “Operating Systems,” on page 371
While how you use TotalView is the same on all operating systems, there are some things you will need to know that are differ from platform to platform.

Chapter 11, “Architectures,” on page 388
When debugging assembly-level functions, you will need to know how TotalView refers to your machines registers.
Chapter 9

Platforms and Compilers

Overview

This chapter describes the compilers and parallel runtime environments used on platforms supported by TotalView. You must refer to the TotalView for HPC Platforms and Systems Requirement Guide for information on the specific compiler and runtime environments that TotalView supports.

For information on supported operating systems, please refer to Chapter 10, “Operating Systems,” on page 371.

Topics in this chapter are:

• Compiling with Debugging Symbols
• Using gnu_debuglink Files
• Linking with the dbfork Library
Compiling with Debugging Symbols

You need to compile programs with the `-g` option and possibly other compiler options so that debugging symbols are included. This section shows the specific compiler commands to use for each compiler that TotalView supports.

NOTE >> Please refer to the release notes in your TotalView distribution for the latest information about supported versions of the compilers and parallel runtime environments listed here.

Apple Running Mac OS X

On Mac OS, in all cases use the standard compiler invocation, just being sure to include the `-g` option.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absoft Fortran 77</td>
<td><code>f77 -g program</code></td>
</tr>
<tr>
<td></td>
<td><code>.f f77 -g program.for</code></td>
</tr>
<tr>
<td>Absoft Fortran 90</td>
<td><code>f90 -g program.f90</code></td>
</tr>
<tr>
<td>GCC C</td>
<td><code>gcc -g program.c</code></td>
</tr>
<tr>
<td>GCC C++</td>
<td><code>g++ -g program.cxx</code></td>
</tr>
<tr>
<td>GCC Fortran</td>
<td><code>g77 -g program.f</code></td>
</tr>
<tr>
<td>IBM xlc C</td>
<td><code>xlc -g program.c</code></td>
</tr>
<tr>
<td>IBM xlc C++</td>
<td><code>xlc -g program.cxx</code></td>
</tr>
<tr>
<td>IBM xlf Fortran 77</td>
<td><code>xlf -g program.f</code></td>
</tr>
<tr>
<td>IBM xlf90 Fortran 90</td>
<td><code>xlf90 -g program.f90</code></td>
</tr>
</tbody>
</table>

On Mac OS X, you can create 64-bit applications using GCC 4 by adding the `-m64` command-line option.

IBM AIX on RS/6000 Systems

The following table lists the procedures to compile programs on IBM RS/6000 systems running AIX.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC C</td>
<td><code>gcc -g program.c</code></td>
</tr>
<tr>
<td>GCC C++</td>
<td><code>g++ -g program.cxx</code></td>
</tr>
<tr>
<td>IBM xlc C</td>
<td><code>xlc -g program.c</code></td>
</tr>
</tbody>
</table>
You can set up to seven variables when debugging threaded applications. Here's how you might set six of these variables within a C shell:

```
setenv AIXTHREAD_MNRATIO"1:1"
setenv AIXTHREAD_SLPRATIO"1:1"
setenv AIXTHREAD_SCOPE"S"
setenv AIXTHREAD_COND_DEBUG"ON"
setenv AIXTHREAD_MUTEX_DEBUG"ON"
setenv AIXTHREAD_RWLOCK_DEBUG"ON"
```

The first three variables must be set. Depending upon what you need to examine, you will also need to set one or more of the “DEBUG” variables.

The seventh variable, AIXTHREAD_DEBUG, should not be set. If it is, you should unset it before running TotalView.

**NOTE >>** Setting these variables can slow down your application’s performance. None of them should be set when you are running non-debugging versions of your program.

When compiling with KCC, you must specify the *-qnofullpath* option; KCC is a preprocessor that passes its output to the IBM xlC C compiler. It will discard #line directives necessary for source-level debugging if you do not use the *-qfullpath* option. We also recommend that you use the +K0 option and not the -g option.

When compiling with *guidef77*, the *-WG,-cmpo=i* option may not be required on all versions because *-g* can imply these options.

When compiling Fortran programs with the C preprocessor, pass the *-d* option to the compiler driver. For example:

```
xlf -d - program.F
```

If you will be moving any program compiled with any of the IBM xl compilers from its creation directory, or you do not want to set the search directory path during debugging, use the *-qfullpath* compiler option. For example:

```
xlf -qfullpath -g -c program.f
```
# IBM Blue Gene

The following table lists the procedures to compile programs on IBM Blue Gene computers.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Visual Age C</td>
<td>xlc -g program.c</td>
</tr>
<tr>
<td>IBM Visual Age C++</td>
<td>xlC -g program.cxx</td>
</tr>
<tr>
<td>IBM Visual Age FORTRAN 77</td>
<td>xlf -g program.f</td>
</tr>
<tr>
<td>IBM Visual Age Fortran 90</td>
<td>xlf90 -g program.f90</td>
</tr>
</tbody>
</table>

# IBM Power Linux

The following table lists the procedures to compile programs on the IBM Power Linux computer.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absoft Fortran 77</td>
<td>f77 -g program.f77</td>
</tr>
<tr>
<td>Absoft Fortran 90</td>
<td>f90 -g program.f90</td>
</tr>
<tr>
<td>GCC C</td>
<td>gcc -g program.c</td>
</tr>
<tr>
<td>GCC C++</td>
<td>g++ -g program.cxx</td>
</tr>
<tr>
<td>IBM Visual Age C</td>
<td>xlc -g program.c</td>
</tr>
<tr>
<td>IBM Visual Age C++</td>
<td>xlC -g program.cxx</td>
</tr>
<tr>
<td>IBM Visual Age FORTRAN 77</td>
<td>xlf -g program.f</td>
</tr>
<tr>
<td>IBM Visual Age Fortran 90</td>
<td>xlf90 -g program.cc</td>
</tr>
</tbody>
</table>

# Linux Running on an x86 Platform

The following table lists the procedures to compile programs on Linux x86 platforms.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absoft Fortran 77</td>
<td>f77 -g program.f77</td>
</tr>
<tr>
<td>Absoft Fortran 90</td>
<td>f90 -g program.f90</td>
</tr>
<tr>
<td>Absoft Fortran 95</td>
<td>f95 -g program.f95</td>
</tr>
<tr>
<td>GCC C</td>
<td>gcc -g program.c</td>
</tr>
</tbody>
</table>
### Linux Running on an x86-64 Platform

The following table lists the procedures to compile programs on Linux x86-64 platforms.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absoft Fortran 77</td>
<td>f77 -g program.f</td>
</tr>
<tr>
<td>Absoft Fortran 90</td>
<td>f90 -g program.f90</td>
</tr>
<tr>
<td>Absoft Fortran 95</td>
<td>f95 -g program.f95</td>
</tr>
<tr>
<td>GCC C</td>
<td>gcc -g program.c</td>
</tr>
<tr>
<td>GCC C++</td>
<td>g++ -g program.cxx</td>
</tr>
<tr>
<td>G77</td>
<td>g77 -g program.f</td>
</tr>
<tr>
<td>Intel C++ Compiler</td>
<td>icc -g program.cxx</td>
</tr>
<tr>
<td>Intel Fortran Compiler</td>
<td>ifc -g program.f</td>
</tr>
<tr>
<td>Lahey/Fujitsu Fortran</td>
<td>lf95 -g program.f</td>
</tr>
<tr>
<td>PGI Fortran 77</td>
<td>pgf77 -g program.f</td>
</tr>
<tr>
<td>PGI Fortran 90</td>
<td>pgf90 -g program.f</td>
</tr>
<tr>
<td>Pathscale EKO C</td>
<td>pathcc -g program.f</td>
</tr>
<tr>
<td>Pathscale EKO C++</td>
<td>pathCC -g program.f</td>
</tr>
<tr>
<td>Lahey/Fujitsu Fortran</td>
<td>lf95 -g program.f</td>
</tr>
<tr>
<td>PGI C++</td>
<td>pcCC -g program.f</td>
</tr>
<tr>
<td>PGI Fortran 77</td>
<td>pgf77 -g program.f</td>
</tr>
<tr>
<td>PGI Fortran 90</td>
<td>pgf90 -g program.f</td>
</tr>
</tbody>
</table>
Linux Running on an Itanium Platform

The following table lists the procedures to compile programs running on the Intel Itanium platform.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC C</td>
<td>gcc -g program.c</td>
</tr>
<tr>
<td>GCC C++</td>
<td>g++ -g program.cxx</td>
</tr>
<tr>
<td>G77</td>
<td>g77 -g program.f</td>
</tr>
<tr>
<td>Intel C++ Compiler</td>
<td>icc -g program.cxx</td>
</tr>
<tr>
<td>Intel Fortran Compiler</td>
<td>ifc -g program.f</td>
</tr>
</tbody>
</table>

Sun Solaris

The following table lists the procedures to compile programs on SunOS 5 SPARC.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apogee C</td>
<td>apcc -g program.c</td>
</tr>
<tr>
<td>Apogee C++</td>
<td>apcc -g program.cxx</td>
</tr>
<tr>
<td>GCC C</td>
<td>gcc -g program.c</td>
</tr>
<tr>
<td>GCC C++</td>
<td>g++ -g program.cxx</td>
</tr>
<tr>
<td>Sun One Studio C</td>
<td>cc -g program.c</td>
</tr>
<tr>
<td>Sun One Studio C++</td>
<td>CC -g program.cxx</td>
</tr>
<tr>
<td>Sun One Studio Fortran 77</td>
<td>f77 -g program.f</td>
</tr>
<tr>
<td>Sun One Studio Fortran 90</td>
<td>f90 -g program.f0</td>
</tr>
</tbody>
</table>
Using gnu_debuglink Files

Some versions of Linux allow you to place debugging information in a separate file. These files, which can have any name, are called gnu_debuglink files. Because this information is stripped from the program's file, it almost always greatly reduces the size of your program. In most cases, you would create gnu_debuglink files for system libraries or other programs for which it is inappropriate to ship versions have debugging information.

After you create an unstripped executable or shared library, you can prepare the gnu_debuglink file as follows:

1. Create a .debug copy of the file. This second file will only contain debugging symbol table information. That is, it differs from the original in that it does not contain code or data.

Create this file on Linux systems that support the -add-gnu-debuglink and -only-keep-debug command-line options. If objcopy --help mentions -add-gnu-debuglink, you should be able to create this file. See man objcopy for more details.

2. Create a stripped copy of the image file, and add a .gnu_debuglink section to the stripped file that contains the name of the .debug file and the checksum of the .debug file.

2. Distribute the stripped image and .debug files separately. The idea is that the stripped image file will normally take up less space on the disk, and if you want the debug information, you can also install the corresponding .debug file.

The following example creates the gnu_debuglink file for a program named hello. It also strips the debugging information from hello:

```
objcopy --only-keep-debug hello hello.gnu_debuglink.debug
objcopy --strip-all hello hello.gnu_debuglink
objcopy --add-gnu-debuglink=hello.gnu_debuglink.debug \
    hello.gnu_debuglink
```

Total View Command-Line Options and CLI State Variables

The following command line options and CLI variables control how TotalView handles gnu_debuglink files.

- **-gnu_debuglink** and **-no_gnu_debuglink, TV::gnu_debuglink**
  Controls Total View processing of the .gnu_debuglink section in executables and shared libraries; the default value is true. Setting the variable to false or using the no_ command-line option prefix saves time when you do not want to process the debug-only files or when you need to avoid other problems associated with the debug-only files.

- **-[no_]gnu_debuglink_checksum** and **TV::gnu_debuglink_checksum**
Tells TotalView if it should validate the checksum of the debug-only files against the checksum stored in the .gnu_debuglink section of the executable or shared library; the default is true. Setting the variable to false or using the no_ command-line option prefix can save time associated with computing the checksum of large files. Do this only if you are absolutely certain that the debug file matches.

- `.gnu_debuglink_global_directory` and `TV::gnu_debuglink_global_directory`
  Specifies the global debug directory; the default value is `/usr/lib/debug`.

### Searching for the gnu_debug_link File

If the `TV::gnu_debuglink` variable is true and if the process contains a `.gnu_debug_link` section, TotalView searches for the `gnu_debug_link` file as follows:

1. In the directory containing the program.
2. In the `.debug` subdirectory of the directory containing the program.
3. In a directory named in the `TV::gnu_debuglink_global_directory` variable.

For example, assume that the program's pathname is `/A/B/hello_world` and the debug filename stored in the `.gnu_debuglink` section of this program is `hello_world.debug`. If the `TV::gnu_debuglink_global_directory` variable is set to `/usr/lib/debug`, TotalView searches for the following files:

1. `/A/B/hello_world.debug`
2. `/A/B/.debug/hello_world.debug`
3. `/usr/lib/debug/A/B/hello_world.debug`
Linking with the dbfork Library

If your program uses the `fork()` and `execve()` system calls, and you want to debug the child processes, you need to link programs with the dbfork library.

NOTE >> While you must link programs that use `fork()` and `execve()` with the TotalView dbfork library so that TotalView can automatically attach to them when your program creates them, programs that you attach to need not be linked with this library.

dbfork on IBM AIX on RS/6000 Systems

Add either the `-ldbfork` or `-ldbfork_64` argument to the command that you use to link your programs. If you are compiling 32-bit code, use the following arguments:

- `/usr/totalview/lib/libdbfork.a -L/usr/totalview/rs6000/lib/libdbfork.a -bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a`
- `-L/usr/totalview/lib -ldbfork -bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a`

For example:
```
cc -o program program.c \
    -L/usr/totalview/rs6000/lib/ -ldbfork \ 
    -bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
```

If you are compiling 64-bit code, use the following arguments:

- `/usr/totalview/lib/libdbfork_64.a -L/usr/totalview/rs6000/lib/libdbfork.a`
- `-L/usr/totalview/lib -ldbfork_64 -bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a`

For example:
```
cc -o program program.c \
    -L/usr/totalview/rs6000/lib -ldbfork \ 
    -bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
```

When you use `gcc` or `g++`, use the `-Wl,-bkeepfile` option instead of using the `-bkeepfile` option, which will pass the same option to the binder. For example:
```
gcc -o program program.c \
    -L/usr/totalview/rs6000/lib -ldbfork -Wl, \ 
    -bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
```

Linking C++ Programs with dbfork

You cannot use the `-bkeepfile` binder option with the IBM xlC C++ compiler. The compiler passes all binder options to an additional pass called `munch`, which will not handle the `-bkeepfile` option.
To work around this problem, we have provided the C++ header file `libdbfork.h`. You must include this file somewhere in your C++ program. This forces the components of the `dbfork` library to be kept in your executable. The file `libdbfork.h` is included only with the RS/6000 version of TotalView. This means that if you are creating a program that will run on more than one platform, you should place the `include` within an `#ifdef` statement's range. For example:

```c
#ifdef _AIX
#include "/usr/totalview/include/libdbfork.h"
#endif
int main (int argc, char *argv[])
{
}
```

In this case, you would not use the `-bkeepfile` option and would instead link your program using one of the following options:

- `/usr/totalview/include/libdbfork.a`
- `-L/usr/totalview/include -ldbfork`

### Linux or Mac OS X

Add one of the following arguments or command-line options to the command that you use to link your programs:

- `/usr/totalview/platform/lib/libdbfork.a`
- `-L/usr/totalview/platform/lib -ldbfork` or `-L/usr/totalview/platform/lib -ldbfork_64`

where `platform` is one of the following: `darwin-power`, `linux-x86`, `linux-x86-64`, or `linux-ia64`.

In general, 32-bit programs use `libdbfork.a` and 64-bit programs use `libdbfork_64.a`. Of course, if your architecture doesn't support 32-bit programs, the option won't work.

For example:

```bash
cc -o program program.c \
    -L/usr/totalview/linux-x86/lib -ldbfork
```

However, `linux-ia64` uses `libdbfork` for 64-bit programs.

### SunOS 5 SPARC

Add one of the following command line arguments or options to the command that you use to link your programs:

- `/opt/totalview/sun5/lib/libdbfork.a`
- `-L/opt/totalview/sun5/lib -ldbfork`

However, `linux-ia64` uses `libdbfork` for 64-bit programs.
For example:

```bash
cc -o program program.c \
   -L/opt/totalview/sun5/lib -ldbfork
```

As an alternative, you can set the `LD_LIBRARY_PATH` environment variable and omit the `-L` option on the command line:

```bash
setenv LD_LIBRARY_PATH /opt/totalview/sun5/lib
```
This chapter describes the operating system features that can be used with TotalView. This chapter includes the following topics:

- Supported Operating Systems
- Troubleshooting Mac OS X Installations
- Mounting the /proc File System (SunOS 5 only)
- Swap Space
- Shared Libraries
- Debugging Your Program’s Dynamically Loaded Libraries
- Remapping Keys (Sun Keyboards only)
- Expression System
Supported Operating Systems

Here is an overview of operating systems and some of the environments supported by TotalView at the time when this book was printed. As this book isn't printed nearly as often as vendors update compilers and operating systems, the compiler and operating system versions mentioned here may be obsolete. For a definitive list, see the most recent platform guide on our website. You can download this document at http://www.roguewave.com/products-services/totalview, and selecting Supported Platforms to download the most recent platform guide.

- Apple Macintosh OS X 10.7, 10.8, and 10.9.
- Cray XT, XE, and XK running the Cray Linux Environment (CLE), version 2.2 or later.
- IBM Blue Gene systems running Linux on the front end nodes.
- IBM RS/6000 and SP systems running AIX versions 5.3L, 6.1, and 7.1.
- Sun x86_64, Solaris 10
- Sun SPARC Solaris 10.
- Linux: see the Platforms Guide.
Troubleshooting Mac OS X Installations

Problem Description

At TotalView startup, the OS checks whether the Mach system call -task_for_pid() is working properly. If the call returns an error, no debugging is possible, and TotalView outputs an error message that begins “The Mach system call -task_for_pid() is not working properly.” TotalView cannot distinguish the circumstances that can lead to this error, which are varied and depend on the version of OS X. The following sections describe a series of steps to troubleshoot this problem.

For Mac OS X Versions 10.8 (Mountain Lion) or Later

Requirements:

- The TotalView executables must be codesigned.
- The Mac host’s system security policy must have Developer mode enabled.
- TotalView users must be members of the _developer group, and must run TotalView within the context of a login session that has been validated by a password challenge at the OS X console.

For Mac OS X Versions 10.11 (Capitan) or Later

In the Mac OS X El Capitan release, Apple has added a new layer named System Integrity Protection (SIP) to its security model. SIP’s protections are not limited to protecting the system from file system changes. There are also system calls that are now restricted in their functionality, which can affect developing and debugging on Mac OS X. For runtime protection the following restrictions exist:

- task_for_pid() fails with EPERM if called incorrectly, which may cause TotalView to crash
- dyld environment variables are ignore
- DTrace probes are unavailable

However, SIP does not block inspection by the developer of their own applications while they are being developed. TotalView tools will continue to allow applications to be inspected and debugged during the development process.

For more information about SIP, please see Apple's developer documentation.
Remotely Debugging without Console Access

If you want to debug a remote machine to which you do not have console access, you can try the following procedure:

1. Install Xquartz and TotalView on the remote machine, and then perform the following steps on that remote machine:
2. Allow X11 forwarding in the `sshd_config` file (disabled by default).
3. Make sure every user who might need to debug is in the `_developer` group.
5. Type: `sudo security authorizationdb write system.privilege.taskport allow`.

The last step above is necessary to allow the launching of TotalView when not on the console.
Mounting the `/proc` File System

To debug programs on SunOS 5 with TotalView, you need to mount the `/proc` file system.

If you receive one of the following errors from TotalView, the `/proc` file system might not be mounted:

- `job_t::launch, creating process: process not found`
- `Error launching process while trying to read -dynamic symbols`
- `Creating Process... Process not found Clearing Thrown Flag Operation Attempted on an unbound process object`

To determine whether the `/proc` file system is mounted, enter the appropriate command from the following table.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>SunOS 5</td>
<td>`% /sbin/mount</td>
</tr>
</tbody>
</table>

If you receive one of these messages from the `mount` command, the `/proc` file system is mounted.

### Mounting /proc with SunOS 5

To make sure that the `/proc` file system is mounted each time your system boots, add the appropriate line from the following table to the appropriate file.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Name of File</th>
<th>Line to add</th>
</tr>
</thead>
<tbody>
<tr>
<td>SunOS 5</td>
<td><code>/etc/vfstab</code></td>
<td><code>/proc - /proc proc - no -</code></td>
</tr>
</tbody>
</table>

Then, to mount the `/proc` file system, enter the following command:

```
/sbin/mount /proc
```
**Swap Space**

Debugging large programs can exhaust the swap space on your machine. If you run out of swap space, TotalView exits with a fatal error, such as:

- **Fatal Error: Out of space trying to allocate**

This error indicates that TotalView failed to allocate dynamic memory. It can occur anytime during a debugging session. It can also indicate that the data size limit in the C shell is too small. You can use the C shell's `limit` command to increase the data size limit. For example:

```bash
limit datasize unlimited
```

- **job_t::launch, creating process: Operation failed**

This error indicates that the `fork()` or `execve()` system call failed while TotalView was creating a process to debug. It can happen when TotalView tries to create a process.

**Swap Space on IBM AIX**

To find out how much swap space has been allocated and is currently being used, use the `/usr/sbin/pstat -s` command:

To find out how much swap space is in use while you are running TotalView:

1. Start TotalView with a large executable:
   ```bash
totalview executable
   ```
   Press Ctrl+Z to suspend TotalView.

1. Use the following command to see how much swap space TotalView is using:
   ```bash
   ps u
   ```
   For example, in this case the value in the SZ column is 5476 KB:

   ```bash
   USER    PID %CPU %MEM    SZ  RSS    TTY  ...  
   smith 15080  0.0 6.0 5476 547  pts/1 ...
   ```

   To add swap space, use the AIX system management tool, `smit`. Use the following path through the `smit` menus:
   ```bash
   System Storage Management > Logical Volume Manager > Paging Space
   ```

**Swap Space on Linux**

To find out how much swap space has been allocated and is currently being used, use either the `swapon` or `top` commands on Linux:
You can use the `mkswap(8)` command to create swap space. The `swapon(8)` command tells Linux that it should use this space.

**Swap Space on SunOS 5**

To find out how much swap space has been allocated and is currently being used, use the `swap -s` command:

To find out how much swap space is in use while you are running TotalView:

1. Start TotalView with a large executable:
   
   ```
   totalview executable
   ```

   Press Ctrl+Z to suspend TotalView.

   1. Use the following command to see how much swap space TotalView is using:

   ```
   /bin/ps -l
   ```

   To add swap space, use the `mkfile(1M)` and `swap(1M)` commands. You must be `root` to use these commands. For more information, refer to the online manual pages for these commands.
Shared Libraries

TotalView supports dynamically linked executables, that is, executables that are linked with shared libraries.

When you start TotalView with a dynamically linked executable, TotalView loads an additional set of symbols for the shared libraries, as indicated in the shell from which you started TotalView. To accomplish this, TotalView:

1. Runs a sample process and discards it.
2. Reads information from the process.
3. Reads the symbol table for each library.

When you create a process without starting it, and the process does not include shared libraries, the PC points to the entry point of the process, usually the `start` routine. If the process does include shared libraries, TotalView takes the following actions:

   • Runs the dynamic loader (SunOS 5: `ld.so`, Linux: `/lib/ld-linux.so.2`).
   • Sets the PC to point to the location after the invocation of the dynamic loader but before the invocation of C++ static constructors or the `main()` routine.

When you attach to a process that uses shared libraries, TotalView takes the following actions:

   • If you attached to the process after the dynamic loader ran, then TotalView loads the dynamic symbols for the shared library.
   • If you attached to the process before it runs the dynamic loader, TotalView allows the process to run the dynamic loader to completion. Then, TotalView loads the dynamic symbols for the shared library.

If desired, you can suppress the recording and use of dynamic symbols for shared libraries by starting TotalView with the `-no_dynamic` option. Refer to Chapter 7, “TotalView Command Syntax,” on page 338 for details on this TotalView startup option.

If a shared library has changed since you started a TotalView session, you can use the Group > Rescan Library command to reload library symbol tables. Be aware that only some systems such as AIX permit you to reload library information.

Changing Linkage Table Entries and LD_BIND_NOW

If you are executing a dynamically linked program, calls from the executable into a shared library are made using the Procedure Linkage Table (PLT). Each function in the dynamic library that is called by the main program has an entry in this table. Normally, the dynamic linker fills the PLT entries with code that calls the dynamic linker. This means that the first time that your code calls a function in a dynamic library, the runtime environment calls the dynamic linker. The linker will then modify the entry so that next time this function is called, it will not be involved.
This is not the behavior you want or expect when debugging a program because TotalView will do one of the following:

- Place you within the dynamic linker (which you don't want to see).
- Step over the function.

And, because the entry is altered, everything appears to work fine the next time you step into this function.

You can correct this problem by setting the `LD_BIND_NOW` environment variable. For example:

```
setenv LD_BIND_NOW 1
```

This tells the dynamic linker that it should alter the PLT when the program starts executing rather than doing it when the program calls the function.
Debugging Your Program’s Dynamically Loaded Libraries

TotalView automatically reads the symbols of shared libraries that are dynamically loaded into your program at runtime. These libraries are those loaded using `dlopen` (or, on IBM AIX, `load` and `loadbind`).

TotalView automatically detects these calls, and then loads the symbol table from the newly loaded libraries and plants any enabled saved breakpoints for these libraries. TotalView then decides whether to ask you about stopping the process to plant breakpoints. You will set these characteristics by using the Dynamic Libraries page in the File > Preferences Dialog Box.

Figure 13 – File > Preferences Dialog Box: Dynamic Libraries Page

TotalView decides according to the following rules:

1. If either the Load symbols from dynamic libraries or Ask to stop when loading dynamic libraries preference is set to false, TotalView does not ask you about stopping.

2. If one or more of the strings in the When the file suffix matches preference list is a suffix of the full library name (including the path), TotalView asks you about stopping.

3. If one or more of the strings in the When the file path prefix does not match list is a prefix of the full library name (including the path), TotalView does not ask you about stopping.
4. If the newly loaded libraries have any saved breakpoints, TotalView does not ask you about stopping.

5. If none of the rules above apply, TotalView asks you about stopping.

If TotalView does not ask you about stopping the process, the process is continued.

If TotalView decides to ask you about stopping, it displays a dialog box, asking if it should stop the process so you can set breakpoints. To stop the process, answer **Yes**.

**Figure 14 – Stop Process Question Dialog Box**

![Stop Process Question Dialog Box](image)

To allow the process to continue executing, answer **No**. Stopping the process allows you to insert breakpoints in the newly loaded shared library.

Do either or both of the following to tell TotalView if it should ask:

- If you can set the `-ask_on_dlopen` command-line option to **true**, or you can set the `-no_ask_on_dlopen` option to false.
- Unset the **Load symbols from dynamic libraries** preference.

The following table lists paths where you are not asked if TotalView should stop the process:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM AIX</td>
<td><code>/lib/ /usr/lib/ /usr/lpp/ /usr/ccs/lib/ /usr/dt/lib/ /tmp/</code></td>
</tr>
<tr>
<td>SUN Solaris 2.x</td>
<td><code>/lib//usr/lib/ /usr/ccs/lib/</code></td>
</tr>
<tr>
<td>Linux x86</td>
<td><code>/lib/usr/lib</code></td>
</tr>
<tr>
<td>Linux Alpha</td>
<td><code>/lib/usr/lib</code></td>
</tr>
</tbody>
</table>

The values you enter in the TotalView preference should be space-separated lists of the prefixes and suffixes to be used.

After starting TotalView, you can change these lists by using the **When the file suffix matches** and **And the file path prefix does not match** preferences.
dlopen Options for Scalability

When a target process calls `dlopen()`, a `dlopen` event is generated in TotalView. Because handling `dlopen` events impacts startup time for dynamically linked executables, TotalView provides ways to configure `dlopen` for better performance and scalability in HPC computing environments:

- Filtering `dlopen` events to avoid stopping a process for each event.
- Handling `dlopen` events in parallel to reduce client/server communication overhead with MRNet enabled.

Filtering dlopen Events

Two state variables and their related command line options enable you to filter `dlopen` events to plant breakpoints in the `dlopened` libraries only when the process stops for some other reason.

`dlopen` event filtering is controlled by the settings on two state variables, TV::`dlopen_always_recalculate` and TV::`dlopen_recalculate_on_match`, and their related command line options `dlopen_always_recalculate` and `dlopen_recalculate_on_match`.

Three possible `dlopen` filtering modes are made possible by these variables: Slow, Medium and Fast.

In Fast mode, the process never stops for a `dlopen` event, not even "null" `dlopen` events. Using this option can result in significant performance gains, but may be impractical for some applications. In Medium mode, some libraries can be specified to always reevaluate their breakpoints, rather than all or none.

- Slow Mode: Reloads libraries on every `dlopen` event

  Option:

  `dlopen_always_recalculate=true`

  Reloads libraries on every `dlopen` event, retaining TotalView's traditional breakpoint reevaluation semantics. This mode is compatible with CUDA and is a good choice when your session has pending breakpoints. However, this mode does not perform or scale as well as the other modes, because it requires the TotalView client to handle every (non-null) `dlopen` event for every process.

  If performance is not the primary concern, or the application or runtime environment does not perform many `dlopen` events, then this may be a good choice.

  In this mode, when the target stops with a `dlopen` event, the server reports the event to the client, where the library list is reloaded and checked to see if any additional breakpoint locations need to be planted in the newly loaded libraries.

- Medium Mode: Reports only libraries that match defined patterns on a `dlopen` event

  Options:

  `dlopen_always_recalculate=false`
  `dlopen_recalculate_on_match="glob-list"`
A **glob-list** is a colon-separated list of simple glob patterns used to compare and match the `dlopened` library. A simple glob pattern is a string, optionally ending with asterisk character (`*`). For example:

```plaintext
dlopen_recalculate_on_match=="libcuda.so*:libmylib1*:libmylib2.so"
```

This mode strikes a balance between performance and enabling breakpoints to be planted in `dlopened` libraries.

In **Medium** mode, the target process stops on every `dlopen` event (just as in **Slow** mode), but the event is not reported to the client unless one of the newly loaded libraries matches the provided pattern.

This setting requires:

- Adding the names of any `dlopened` libraries to the `TV::dlopen_recalculate_on_match` list if you want breakpoints planted in the library when the library is loaded.
- Adding `"libcuda.so*"` to the match list if you are debugging CUDA; otherwise TotalView will miss CUDA kernel launch events.

**Fast Mode**: Does not stop for `dlopen` events

**Options**:

```plaintext
dlopen_always_recalculate==false
dlopen_recalculate_on_match=="
```

This mode provides the best performance, disallowing planting breakpoints in `dlopened` libraries when the library is loaded. Breakpoints are planted in the `dlopened` libraries only when the process stops for some other reason; however, be aware with this option that an application may have executed past the point at which you want to start debugging inside the `dlopened` library.

Because the debugger does not plant the `dlopen` breakpoint in the process, the process never stops for a `dlopen` event, not even "null" `dlopen` events. While this mode may be impractical for some applications, the performance gains are significant.

Table 4 summarizes the pros and cons of each mode.

### Table 4: `dlopen` Event Filtering Modes

<table>
<thead>
<tr>
<th>Mode/Speed</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow</td>
<td>dlopen_always_recalculate==true</td>
</tr>
<tr>
<td></td>
<td>Pros:</td>
</tr>
<tr>
<td></td>
<td>• Retains TotalView's traditional breakpoint reevaluation semantics.</td>
</tr>
<tr>
<td></td>
<td>• Works best with pending breakpoints.</td>
</tr>
<tr>
<td></td>
<td>• Compatible with CUDA.</td>
</tr>
<tr>
<td></td>
<td>Cons:</td>
</tr>
<tr>
<td></td>
<td>• Does not perform or scale as well as the other modes because the TotalView client handles every (non-null) <code>dlopen</code> event for every process.</td>
</tr>
<tr>
<td>Medium</td>
<td>dlopen_always_recalculate==false</td>
</tr>
<tr>
<td></td>
<td>dlopen_recalculate_on_match==&quot;glob-list&quot;</td>
</tr>
</tbody>
</table>
Handling dlopen Events in Parallel

TotalView's default behavior is to handle dlopened libraries serially, creating multiple, single-cast client-server communications. This can degrade performance, depending on the number of libraries a process dlopen, and the number of processes in the job.

To handle dlopened libraries in parallel, use the TV::dlopen_read_libraries_in_parallel and its related command line option -dlopen_read_libraries_in_parallel.

This sets the state variable to true. Placing this dset command in the tvsrc file ensures that all instances of TotalView launch with this option:

dset TV::dlopen_read_libraries_in_parallel true

To set this option on an individual instance of TotalView, use the command line option when you start TotalView:

totalview -dlopen_read_libraries_in_parallel

<table>
<thead>
<tr>
<th>Mode/Speed</th>
<th>Option</th>
<th>Pros:</th>
<th>Cons:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast</td>
<td></td>
<td>• Performs better by filtering out dlopen events.</td>
<td>• Process stops at the dlopen breakpoint, even for &quot;null&quot; dlopen events.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Allows the TotalView client to process multiple dlopen events at a time.</td>
<td>• An application may execute past the point at which you want to start debugging inside the dlopened library.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Compatible with CUDA.</td>
<td>• Requires adding to the match list any libraries that should have breakpoints planted when the library is loaded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Requires adding to the match list libcuda.so* for CUDA support.</td>
</tr>
</tbody>
</table>

Table 4: dlopen Event Filtering Modes
NOTE >> Enabling this option does not guarantee that dlopen performance will improve on all systems in all scenarios. Be sure to test the impact of this setting on your system and debugging environments.

Remember that MRNet must also be enabled for this to work.

Known Limitations

Dynamic library support has the following known limitations:

• TotalView does not deal correctly with parallel programs that call \texttt{dlopen} on different libraries in different processes. TotalView requires that the processes have a uniform address space, including all shared libraries.

• TotalView does not yet fully support unloading libraries (using \texttt{dlclose}) and then reloading them at a different address using \texttt{dlopen}.
Remapping Keys

On the SunOS 5 keyboard, you may need to remap the page-up and page-down keys to the prior and next key-sym so that you can scroll TotalView windows with the page-up and page-down keys. To do so, add the following lines to your X Window System startup file:

```
# Remap F29/F35 to PgUp/PgDn
xmodmap -e 'keysym F29 = Prior'
xmodmap -e 'keysym F35 = Next'
```
Expression System

Depending on the target platform, TotalView supports:

- An interpreted expression system only
- Both an interpreted and a compiled expression system

Unless stated otherwise below, TotalView supports interpreted expressions only.

Expression System on IBM AIX-Power and Blue Gene/Q

On IBM AIX and Blue Gene/Q, TotalView supports compiled and interpreted expressions. TotalView also supports assembly language in expressions.

Some program functions called from the TotalView expression system on the Power architecture cannot have floating-point arguments that are passed by value. However, in functions with a variable number of arguments, floating-point arguments can be in the varying part of the argument list. For example, you can include floating-point arguments with calls to `printf`:

```c
    double d = 3.14159;
    printf("d = %f\n", d);
```

On Blue Gene/Q, currently TotalView supports only statically allocated patch spaces linked into the base executable. The executable may be statically or dynamically linked. The static patch space must not reside in a shared library. See Allocating Static Patch Space in the TotalView for HPC User Guide.
Overview

This chapter describes the architectures TotalView supports, including:

- “AMD and Intel x86-64” on page 389
- “Power Architectures” on page 394
- “Intel IA-64” on page 401
- “Intel x86” on page 407 (Intel 80386, 80486 and Pentium processors)
- “Sun SPARC” on page 412
AMD and Intel x86-64

This section describes AMD's 64-bit processors and the Intel EM64T processors, including:

• “x86-64 General Registers” on page 389
• “x86-64 Floating-Point Registers” on page 390
• “x86-64 FPCR Register” on page 391
• “x86-64 FPSR Register” on page 392
• “x86-64 MXCSR Register” on page 393

The x86-64 can be programmed in either 32- or 64-bit mode. TotalView supports both. In 32-bit mode, the processor is identical to an x86, and the stack frame is identical to the x86. The information within this section describes 64-bit mode.

The AMD x86-64 processor supports the IEEE floating-point format.

x86-64 General Registers

TotalView displays the x86-64 general registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each general register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAX</td>
<td>General registers</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$rax</td>
</tr>
<tr>
<td>RDX</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$rdx</td>
</tr>
<tr>
<td>RCX</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$rcx</td>
</tr>
<tr>
<td>RBX</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$rbx</td>
</tr>
<tr>
<td>RSI</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$rsi</td>
</tr>
<tr>
<td>RDI</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$rdi</td>
</tr>
<tr>
<td>RBP</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$rbp</td>
</tr>
<tr>
<td>RSP</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$rsp</td>
</tr>
<tr>
<td>R8-R15</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$r8-$r15</td>
</tr>
<tr>
<td>RA</td>
<td>Selector registers</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$ra</td>
</tr>
<tr>
<td>SS</td>
<td></td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$ss</td>
</tr>
<tr>
<td>DS</td>
<td></td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$ds</td>
</tr>
</tbody>
</table>
### x86-64 Floating-Point Registers

TotalView displays the x86-64 floating-point registers in the Stack Frame Pane of the Process Window. The next table describes how TotalView treats each floating-point register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST0</td>
<td>ST(0)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st0</td>
</tr>
<tr>
<td>ST1</td>
<td>ST(1)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st1</td>
</tr>
<tr>
<td>ST2</td>
<td>ST(2)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st2</td>
</tr>
<tr>
<td>ST3</td>
<td>ST(3)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st3</td>
</tr>
<tr>
<td>ST4</td>
<td>ST(4)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st4</td>
</tr>
<tr>
<td>ST5</td>
<td>ST(5)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st5</td>
</tr>
<tr>
<td>ST6</td>
<td>ST(6)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st6</td>
</tr>
<tr>
<td>ST7</td>
<td>ST(7)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st7</td>
</tr>
<tr>
<td>FPCR</td>
<td>Floating-point control register</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$fpcr</td>
</tr>
<tr>
<td>FPSR</td>
<td>Floating-point status register</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpsr</td>
</tr>
<tr>
<td>FPTAG</td>
<td>Tag word</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fptag</td>
</tr>
<tr>
<td>FPOP</td>
<td>Floating-point operation</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpop</td>
</tr>
<tr>
<td>FPI</td>
<td>Instruction address</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpi</td>
</tr>
<tr>
<td>FPD</td>
<td>Data address</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpd</td>
</tr>
</tbody>
</table>
Architectures / AMD and Intel x86-64

NOTE >> The x86-64 has 16 128-bit registers that are used by SSE and SSE2 instructions. TotalView displays these as 32 64-bit registers. These registers can be used in the following ways: 16 bytes, 8 words, 2 longs, 4 floating point, 2 double, or a single 128-bit value. TotalView shows each of these hardware registers as two $long registers. To change the type, dive and then edit the type in the data window to be an array of the type you wish. For example, cast it to “$char[16]”, “$float[4]”, and so on.

x86-64 FPCR Register

For your convenience, TotalView interprets the bit settings of the FPCR and FPSR registers.

You can edit the value of the FPCR and set it to any of the bit settings outlined in the next table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC=RN</td>
<td>0x0000</td>
<td>To nearest rounding mode</td>
</tr>
<tr>
<td>RC=R-</td>
<td>0x2000</td>
<td>Toward negative infinity rounding mode</td>
</tr>
<tr>
<td>RC=R+</td>
<td>0x4000</td>
<td>Toward positive infinity rounding mode</td>
</tr>
<tr>
<td>RC=RZ</td>
<td>0x6000</td>
<td>Toward zero rounding mode</td>
</tr>
</tbody>
</table>
Using the x86-64 FPCR Register

You can change the value of the FPCR within TotalView to customize the exception handling for your program.

For example, if your program inadvertently divides by zero, you can edit the bit setting of the FPCR register in the Stack Frame Pane. In this case, you would change the bit setting for the FPCR to include 0x0004 so that TotalView traps the “divide-by-zero” bit. The string displayed next to the FPCR register should now include \textit{EM=(ZM)}. Now, when your program divides by zero, it receives a \textit{SIGFPE} signal, which you can catch with TotalView. See “\textit{Handling Signals}” in Chapter 5 of the \textit{TotalView for HPC Users Guide} for information on handling signals. If you did not set the bit for trapping divide by zero, the processor would ignore the error and set the \textit{EF=(ZE)} bit in the FPSR.

x86-64 FPSR Register

The bit settings of the x86-64 FPSR register are outlined in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP</td>
<td>0x3800</td>
<td>Register $&lt;i&gt;$ is top of FPU stack</td>
</tr>
<tr>
<td>B</td>
<td>0x8000</td>
<td>FPU busy</td>
</tr>
<tr>
<td>C0</td>
<td>0x0100</td>
<td>Condition bit 0</td>
</tr>
<tr>
<td>C1</td>
<td>0x0200</td>
<td>Condition bit 1</td>
</tr>
<tr>
<td>C2</td>
<td>0x0400</td>
<td>Condition bit 2</td>
</tr>
<tr>
<td>C3</td>
<td>0x4000</td>
<td>Condition bit 3</td>
</tr>
<tr>
<td>ES</td>
<td>0x0080</td>
<td>Exception summary status</td>
</tr>
<tr>
<td>SF</td>
<td>0x0040</td>
<td>Stack fault</td>
</tr>
<tr>
<td>EF=PE</td>
<td>0x0020</td>
<td>Precision exception</td>
</tr>
<tr>
<td>EF=UE</td>
<td>0x0010</td>
<td>Underflow exception</td>
</tr>
</tbody>
</table>
This register contains control and status information for the SSE registers. Some of the bits in this register are editable. You cannot dive in these values.

The bit settings of the x86-64 MXCSR register are outlined in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FZ</td>
<td>0x8000</td>
<td>Flush to zero</td>
</tr>
<tr>
<td>RC=RN</td>
<td>0x0000</td>
<td>To nearest rounding mode</td>
</tr>
<tr>
<td>RC=R-</td>
<td>0x2000</td>
<td>Toward negative infinity rounding mode</td>
</tr>
<tr>
<td>RC=R+</td>
<td>0x4000</td>
<td>Toward positive infinity rounding mode</td>
</tr>
<tr>
<td>RC=RZ</td>
<td>0x6000</td>
<td>Toward zero rounding mode</td>
</tr>
<tr>
<td>EM=PM</td>
<td>0x1000</td>
<td>Precision mask</td>
</tr>
<tr>
<td>EM=UM</td>
<td>0x0800</td>
<td>Underflow mask</td>
</tr>
<tr>
<td>EM=OM</td>
<td>0x0400</td>
<td>Overflow mask</td>
</tr>
<tr>
<td>EM=ZM</td>
<td>0x0200</td>
<td>Divide-by-zero mask</td>
</tr>
<tr>
<td>EM=DM</td>
<td>0x0100</td>
<td>Denormal mask</td>
</tr>
<tr>
<td>EM=IM</td>
<td>0x0080</td>
<td>Invalid operation mask</td>
</tr>
<tr>
<td>DAZ</td>
<td>0x0040</td>
<td>Denormals are zeros</td>
</tr>
<tr>
<td>EF=PE</td>
<td>0x0020</td>
<td>Precision flag</td>
</tr>
<tr>
<td>EF=UE</td>
<td>0x0010</td>
<td>Underflow flag</td>
</tr>
<tr>
<td>EF=OE</td>
<td>0x0008</td>
<td>Overflow flag</td>
</tr>
<tr>
<td>EF=ZE</td>
<td>0x0004</td>
<td>Divide-by-zero flag</td>
</tr>
<tr>
<td>EF=DE</td>
<td>0x0002</td>
<td>Denormal flag</td>
</tr>
<tr>
<td>EF=IE</td>
<td>0x0001</td>
<td>Invalid operation flag</td>
</tr>
</tbody>
</table>
Power Architectures

This section contains the following information:

- Power General Registers
- Blue Gene Power Registers
- Power MSR Register
- Power Floating-Point Registers
- Blue Gene/Q QPX Floating-Point Registers
- Power FPSCR Register
- Using the Power FPSCR Register

NOTE >> The Power architecture supports the IEEE floating-point format.

Power General Registers

TotalView displays Power general registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each general register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>General register 0</td>
<td>$int/$long</td>
<td>yes</td>
<td>yes</td>
<td>$r0</td>
</tr>
<tr>
<td>SP</td>
<td>Stack pointer</td>
<td>$int/$long</td>
<td>yes</td>
<td>yes</td>
<td>$sp</td>
</tr>
<tr>
<td>RTOC</td>
<td>TOC pointer</td>
<td>$int/$long</td>
<td>yes</td>
<td>yes</td>
<td>$rtoc</td>
</tr>
<tr>
<td>R3 - R31</td>
<td>General registers 3 - 31</td>
<td>$int/$long</td>
<td>yes</td>
<td>yes</td>
<td>$r3 - $r31</td>
</tr>
<tr>
<td>INUM</td>
<td></td>
<td>$int/$long</td>
<td>yes</td>
<td>no</td>
<td>$inum</td>
</tr>
<tr>
<td>PC</td>
<td>Program counter</td>
<td>$code[]</td>
<td>no</td>
<td>yes</td>
<td>$pc</td>
</tr>
<tr>
<td>SRR1</td>
<td>Machine status save/restore register</td>
<td>$int/$long</td>
<td>yes</td>
<td>no</td>
<td>$srr1</td>
</tr>
<tr>
<td>LR</td>
<td>Link register</td>
<td>$code[]</td>
<td>yes</td>
<td>no</td>
<td>$lr</td>
</tr>
<tr>
<td>CTR</td>
<td>Counter register</td>
<td>$int/$long</td>
<td>yes</td>
<td>no</td>
<td>$ctr</td>
</tr>
<tr>
<td>CR</td>
<td>Condition register (see below)</td>
<td>$int/$long</td>
<td>yes</td>
<td>no</td>
<td>$cr</td>
</tr>
</tbody>
</table>

NOTE >> The Power architecture supports the IEEE floating-point format.
CR Register

TotalView writes information for each of the eight condition sets, appending a >, <, or = symbol. For example, if the summary overflow (0x1) bit is set, TotalView might display the following:

0x22424444 (574768196) (0=,1=,2>,3=,4>,5>,6>,7>)

XER Register

Depending upon what was set, TotalView can display up to five kinds of information, as follows:

- **STD:**0x%02x
  - The string terminator character (bits 25-31)

- **SL:**%d
  - The string length field (bits 16-23)

- **SO**
  - Displayed if the summary overflow bit is set (bit 0)

- **OV**
  - Displayed if the overflow bit is set (bit 1)

- **CA**
  - Displayed if the carry bit is set (bit 2)

For example:
Blue Gene Power Registers

TotalView displays Blue Gene Power registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each Blue Gene register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>General register 0</td>
<td>$int/$long</td>
<td>yes</td>
<td>yes</td>
<td>$r0</td>
</tr>
<tr>
<td>SP</td>
<td>Stack pointer</td>
<td>$int/$long</td>
<td>yes</td>
<td>yes</td>
<td>$sp</td>
</tr>
<tr>
<td>RTOC</td>
<td>TOC pointer</td>
<td>$int/$long</td>
<td>yes</td>
<td>yes</td>
<td>$rtoc</td>
</tr>
<tr>
<td>R3 - R31</td>
<td>General registers 3 - 31</td>
<td>$int/$long</td>
<td>yes</td>
<td>yes</td>
<td>$r3 - $r31</td>
</tr>
<tr>
<td>LR</td>
<td>Link register</td>
<td>$code[]</td>
<td>yes</td>
<td>no</td>
<td>$lr</td>
</tr>
<tr>
<td>CR</td>
<td>Condition register (see below)</td>
<td>$int/$long</td>
<td>yes</td>
<td>no</td>
<td>$cr</td>
</tr>
<tr>
<td>XER</td>
<td>Integer exception register (see below)</td>
<td>$int/$long</td>
<td>yes</td>
<td>no</td>
<td>$xer</td>
</tr>
<tr>
<td>CTR</td>
<td>Counter register</td>
<td>$int/$long</td>
<td>yes</td>
<td>no</td>
<td>$ctr</td>
</tr>
<tr>
<td>IAR</td>
<td>Program counter</td>
<td>$code[]</td>
<td>no</td>
<td>yes</td>
<td>$pc</td>
</tr>
<tr>
<td>MSR</td>
<td>Machine state register</td>
<td>$int/$long</td>
<td>yes</td>
<td>no</td>
<td>$msr</td>
</tr>
<tr>
<td>DEAR</td>
<td>Data address register</td>
<td>$int/$long</td>
<td>yes</td>
<td>no</td>
<td>$dar</td>
</tr>
<tr>
<td>ESR</td>
<td>Exception status register</td>
<td>$int/$long</td>
<td>no</td>
<td>no</td>
<td>$esr</td>
</tr>
</tbody>
</table>
Blue Gene/Q QPX Floating-Point Registers

TotalView displays the Blue Gene/Q Quad Processing eXtension to the Power ISA (QPX) registers in the Stack Frame Pane of the Process Window. The architecture provides for 32 256-bit registers that can be used as four doubles, eight floats, four 64-bit integers, or eight 32-bit integers. The next table describes how TotalView treats each floating-point register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0 - F31</td>
<td>QPX floating-point registers 0 - 31</td>
<td>$qpx_reg</td>
<td>yes</td>
<td>yes</td>
<td>$q0 - $q31</td>
</tr>
<tr>
<td>FPSCR</td>
<td>Floating-point status register</td>
<td>$long</td>
<td>yes</td>
<td>no</td>
<td>$fpscr</td>
</tr>
</tbody>
</table>

The data type $qpx_reg is a TotalView predefined type that is defined as follows:

```c
union $qpx_reg {
    $double q4_double[4];
    $float q8_float[8];
    $int64 q4_int64[4];
    $int32 q8_int32[8];
};
```

The traditional Book 1 Power PC floating point instructions that operate on the FPR register set operate on slot 0 of the corresponding QPX register. Therefore, the Stack Frame Pane of the Process Window shows the double contained in slot 0 (the q4_double[0] field of the $qpx_reg data type) for each QPX register. Dive on a QPX register to open a Data Pane displaying the full contents of the register.

Power MSR Register

For your convenience, TotalView interprets the bit settings of the Power MSR register. You can edit the value of the MSR and set it to any of the bit settings outlined in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8000000000000000</td>
<td>SF</td>
<td>Sixty-four bit mode</td>
</tr>
<tr>
<td>0x000000000000040000</td>
<td>POW</td>
<td>Power management enable</td>
</tr>
<tr>
<td>0x000000000000020000</td>
<td>TGPR</td>
<td>Temporary GPR mapping</td>
</tr>
<tr>
<td>0x000000000000010000</td>
<td>ILE</td>
<td>Exception little-endian mode</td>
</tr>
<tr>
<td>0x000000000000080000</td>
<td>EE</td>
<td>External interrupt enable</td>
</tr>
<tr>
<td>0x000000000000040000</td>
<td>PR</td>
<td>Privilege level</td>
</tr>
<tr>
<td>0x000000000000020000</td>
<td>FP</td>
<td>Floating-point available</td>
</tr>
<tr>
<td>0x000000000000010000</td>
<td>ME</td>
<td>Machine check enable</td>
</tr>
</tbody>
</table>
Power Floating-Point Registers

TotalView displays the Power floating-point registers in the Stack Frame Pane of the Process Window. The next table describes how TotalView treats each floating-point register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0 - F31</td>
<td>Floating-point registers 0 - 31</td>
<td>double</td>
<td>yes</td>
<td>yes</td>
<td>$f0 - $f31</td>
</tr>
<tr>
<td>FPSCR</td>
<td>Floating-point status register</td>
<td>int</td>
<td>yes</td>
<td>no</td>
<td>$fpscr</td>
</tr>
<tr>
<td>FPSCR2</td>
<td>Floating-point status register 2</td>
<td>int</td>
<td>yes</td>
<td>no</td>
<td>$fpscr2</td>
</tr>
</tbody>
</table>

Power FPSCR Register

For your convenience, TotalView interprets the bit settings of the Power FPSCR register. You can edit the value of the FPSCR and set it to any of the bit settings outlined in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000000</td>
<td>FX</td>
<td>Floating-point exception summary</td>
</tr>
<tr>
<td>0x40000000</td>
<td>FEX</td>
<td>Floating-point enabled exception summary</td>
</tr>
<tr>
<td>0x20000000</td>
<td>VX</td>
<td>Floating-point invalid operation exception summary</td>
</tr>
<tr>
<td>0x10000000</td>
<td>OX</td>
<td>Floating-point overflow exception</td>
</tr>
<tr>
<td>0x08000000</td>
<td>UX</td>
<td>Floating-point underflow exception</td>
</tr>
<tr>
<td>Value</td>
<td>Bit Setting</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>0x04000000</td>
<td>ZX</td>
<td>Floating-point zero divide exception</td>
</tr>
<tr>
<td>0x02000000</td>
<td>XX</td>
<td>Floating-point inexact exception</td>
</tr>
<tr>
<td>0x01000000</td>
<td>VXSNAN</td>
<td>Floating-point invalid operation exception for SNaN</td>
</tr>
<tr>
<td>0x00800000</td>
<td>VXISI</td>
<td>Floating-point invalid operation exception: $\div$, or infinity/0</td>
</tr>
<tr>
<td>0x00400000</td>
<td>VXIDI</td>
<td>Floating-point invalid operation exception: $/ \div$, or infinity/0 divided by infinity</td>
</tr>
<tr>
<td>0x00200000</td>
<td>VXZDZ</td>
<td>Floating-point invalid operation exception: 0 / 0</td>
</tr>
<tr>
<td>0x00100000</td>
<td>VXIMZ</td>
<td>Floating-point invalid operation exception: $\cdot \times$, or infinity*infinity</td>
</tr>
<tr>
<td>0x00080000</td>
<td>VXVC</td>
<td>Floating-point invalid operation exception: invalid compare</td>
</tr>
<tr>
<td>0x00040000</td>
<td>FR</td>
<td>Floating-point fraction rounded</td>
</tr>
<tr>
<td>0x00020000</td>
<td>FI</td>
<td>Floating-point fraction inexact</td>
</tr>
<tr>
<td>0x00010000</td>
<td>FPRF=(C)</td>
<td>Floating-point result class descriptor</td>
</tr>
<tr>
<td>0x00008000</td>
<td>FPRF=(L)</td>
<td>Floating-point less than or negative</td>
</tr>
<tr>
<td>0x00004000</td>
<td>FPRF=(G)</td>
<td>Floating-point greater than or positive</td>
</tr>
<tr>
<td>0x00002000</td>
<td>FPRF=(E)</td>
<td>Floating-point equal or zero</td>
</tr>
<tr>
<td>0x00001000</td>
<td>FPRF=(U)</td>
<td>Floating-point unordered or NaN</td>
</tr>
<tr>
<td>0x00011000</td>
<td>FPRF=(QNAN)</td>
<td>Quiet NaN; alias for FPRF=(C+U)</td>
</tr>
<tr>
<td>0x00009000</td>
<td>FPRF=(-INF)</td>
<td>-Infinity; alias for FPRF=(L+U)</td>
</tr>
<tr>
<td>0x00008000</td>
<td>FPRF=(-NORM)</td>
<td>-Normalized number; alias for FPRF=(L)</td>
</tr>
<tr>
<td>0x00018000</td>
<td>FPRF=(-DENORM)</td>
<td>-Denormalized number; alias for FPRF=(C+L)</td>
</tr>
<tr>
<td>0x00012000</td>
<td>FPRF=(-ZERO)</td>
<td>-Zero; alias for FPRF=(C+E)</td>
</tr>
<tr>
<td>0x00020000</td>
<td>FPRF=(+ZERO)</td>
<td>+Zero; alias for FPRF=(E)</td>
</tr>
<tr>
<td>0x00014000</td>
<td>FPRF=(+DENORM)</td>
<td>+Denormalized number; alias for FPRF=(C+G)</td>
</tr>
<tr>
<td>0x00004000</td>
<td>FPRF=(+NORM)</td>
<td>+Normalized number; alias for FPRF=(G)</td>
</tr>
<tr>
<td>0x00005000</td>
<td>FPRF=(+INF)</td>
<td>+Infinity; alias for FPRF=(G+U)</td>
</tr>
<tr>
<td>0x00000400</td>
<td>VXSOFT</td>
<td>Floating-point invalid operation exception: software request</td>
</tr>
<tr>
<td>0x00000200</td>
<td>VXSQRT</td>
<td>Floating-point invalid operation exception: square root</td>
</tr>
<tr>
<td>0x00000100</td>
<td>VXCVI</td>
<td>Floating-point invalid operation exception: invalid integer convert</td>
</tr>
<tr>
<td>0x00000080</td>
<td>VE</td>
<td>Floating-point invalid operation exception enable</td>
</tr>
</tbody>
</table>
Using the Power FPSCR Register

On AIX, if you compile your program to catch floating-point exceptions (IBM compiler `-qflttrap` option), you can change the value of the FPSCR within TotalView to customize the exception handling for your program.

For example, if your program inadvertently divides by zero, you can edit the bit setting of the FPSCR register in the Stack Frame Pane. In this case, you would change the bit setting for the FPSCR to include `0x10` so that TotalView traps the "divide by zero" exception. The string displayed next to the FPSR register should now include `ZE`. Now, when your program divides by zero, it receives a `SIGTRAP` signal, which will be caught by TotalView. See “Handling Signals” in Chapter 5 of the `TotalView for HPC Users Guide` for more information. If you did not set the bit for trapping divide by zero or you did not compile to catch floating-point exceptions, your program would not stop and the processor would set the `ZX` bit.

### Value | Bit Setting | Meaning
--- | --- | ---
0x0000040 | OE | Floating-point overflow exception enable
0x0000020 | UE | Floating-point underflow exception enable
0x0000010 | ZE | Floating-point zero divide exception enable
0x0000008 | XE | Floating-point inexact exception enable
0x0000004 | NI | Floating-point non-IEEE mode enable
0x0000000 | RN=NEAR | Round to nearest
0x0000001 | RN=ZERO | Round toward zero
0x0000002 | RN=PINF | Round toward +infinity
0x0000003 | RN=NINF | Round toward -infinity
Intel IA-64

This section contains the following information:

- Intel IA-64 General Registers
- “IA-64 Processor Status Register Fields (PSR)” on page 402
- “Current Frame Marker Register Fields (CFM)” on page 403
- “Register Stack Configuration Register Fields (RSC)” on page 404
- “Previous Function State Register Fields (PFS)” on page 404
- “Floating Point Registers” on page 405
- “Floating Point Status Register Fields” on page 405

The Cray XT3 front end runs on this chip.

Intel IA-64 General Registers

TotalView displays the IA-64 general registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each general register, and the actions you can take with each.

NOTE >> The descriptions in this section are taken (almost verbatim) from the “Intel Itanium Architecture Software Developer's Manual. Volume 1: Application Architecture”. This was revision 2.0, printed in December 2001.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>in expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>register 0</td>
<td>$long</td>
<td>N</td>
<td>Y</td>
<td>$r0</td>
</tr>
<tr>
<td>r1</td>
<td>global pointer</td>
<td>$long</td>
<td>N</td>
<td>Y</td>
<td>$r1</td>
</tr>
<tr>
<td>r2-r31</td>
<td>static general registers</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$r2-$r31</td>
</tr>
<tr>
<td>r31-r127</td>
<td>stacked general registers (all may not be valid)</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$r32-$r127</td>
</tr>
<tr>
<td>b0-b7</td>
<td>branch registers</td>
<td>$code[]</td>
<td>Y</td>
<td>Y</td>
<td>$b0-$b7</td>
</tr>
<tr>
<td>ip</td>
<td>instruction pointer</td>
<td>$code[]</td>
<td>N</td>
<td>Y</td>
<td>$ip</td>
</tr>
<tr>
<td>cfm</td>
<td>current frame marker</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$cfm</td>
</tr>
<tr>
<td>psr</td>
<td>processor status register</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$psr</td>
</tr>
<tr>
<td>rsc</td>
<td>register stack configuration register (AR 16)</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$rsc</td>
</tr>
</tbody>
</table>
NOTE >> All general registers r32-r127 may not be valid in a given stack frame.

IA-64 Processor Status Register Fields (PSR)

These fields control memory access alignment, byte-ordering, and user-configured performance monitors. It also records the modification state of floating-point registers.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Meaning</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>in expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>be</td>
<td>big-endian enable</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$bsp</td>
</tr>
<tr>
<td>2</td>
<td>up</td>
<td>user performance monitor enable</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$bspstore</td>
</tr>
<tr>
<td>3</td>
<td>ac</td>
<td>alignment check</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$rnat</td>
</tr>
<tr>
<td>4</td>
<td>mfl</td>
<td>lower (f2-f31) floating point registers written</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$ccv</td>
</tr>
<tr>
<td>5</td>
<td>mfh</td>
<td>upper (f32-f127) floating point registers written</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$unat</td>
</tr>
<tr>
<td>13</td>
<td>ic</td>
<td>interruption collection</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$fpsr</td>
</tr>
<tr>
<td>14</td>
<td>i</td>
<td>interrupt bit</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$pfs</td>
</tr>
<tr>
<td>15</td>
<td>pk</td>
<td>protection key enable</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$lc</td>
</tr>
<tr>
<td>17</td>
<td>dt</td>
<td>data address translation</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$ec</td>
</tr>
<tr>
<td>18</td>
<td>dfl</td>
<td>disabled lower floating point register set</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$pr</td>
</tr>
<tr>
<td>19</td>
<td>dfh</td>
<td>disabled upper floating point register set</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$nat</td>
</tr>
</tbody>
</table>
Current Frame Marker Register Fields (CFM)

Each general register stack frame is associated with a frame marker. The frame maker describes the state of the general register stack. The Current Frame Marker (CFM) holds the state of the current stack frame.

### Bit Range | Field | Meaning
--- | --- | ---
6:0 | sof | Size of frame
13:7 | sol | Size of locals portion of stack frame
17:14 | sor | Size of rotating portion of stack frame (number of rotating registers is sor*8)
24:18 | rrb.gr | Register rename base for general registers

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>sp</td>
<td>secure performance monitors</td>
</tr>
<tr>
<td>21</td>
<td>pp</td>
<td>privileged performance monitor enable</td>
</tr>
<tr>
<td>22</td>
<td>di</td>
<td>disable instruction set transition</td>
</tr>
<tr>
<td>23</td>
<td>si</td>
<td>secure interval timer</td>
</tr>
<tr>
<td>24</td>
<td>db</td>
<td>debug breakpoint fault</td>
</tr>
<tr>
<td>25</td>
<td>lp</td>
<td>lower privilege transfer trap</td>
</tr>
<tr>
<td>26</td>
<td>tb</td>
<td>taken branch trap</td>
</tr>
<tr>
<td>27</td>
<td>rt</td>
<td>register stack translation</td>
</tr>
<tr>
<td>33:32</td>
<td>cpl</td>
<td>current privilege level</td>
</tr>
<tr>
<td>34</td>
<td>is</td>
<td>instruction set</td>
</tr>
<tr>
<td>35</td>
<td>mc</td>
<td>machine check abort mask</td>
</tr>
<tr>
<td>36</td>
<td>it</td>
<td>instruction address translation</td>
</tr>
<tr>
<td>37</td>
<td>id</td>
<td>instruction debug fault disable</td>
</tr>
<tr>
<td>38</td>
<td>da</td>
<td>disable data access and dirty-bit faults</td>
</tr>
<tr>
<td>39</td>
<td>dd</td>
<td>data debug fault disable</td>
</tr>
<tr>
<td>40</td>
<td>ss</td>
<td>single step enable</td>
</tr>
<tr>
<td>42:41</td>
<td>ri</td>
<td>restart instruction</td>
</tr>
<tr>
<td>43</td>
<td>ed</td>
<td>exception deferral</td>
</tr>
<tr>
<td>44</td>
<td>bn</td>
<td>register bank</td>
</tr>
<tr>
<td>45</td>
<td>ia</td>
<td>disable instruction access-bit faults</td>
</tr>
</tbody>
</table>
Register Stack Configuration Register Fields (RSC)

The Register Stack Configuration (RSC) Register is a 64-bit register used to control the operation of the Register Stack Engine (RSE).

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:25</td>
<td>rrb.fr</td>
<td>Register rename base for float registers</td>
</tr>
<tr>
<td>37:32</td>
<td>rrb.pr</td>
<td>Register rename base for predicate registers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0 00</td>
<td>mode</td>
<td>enforced lazy</td>
</tr>
<tr>
<td>1:0 01</td>
<td></td>
<td>load intensive</td>
</tr>
<tr>
<td>1:0 10</td>
<td></td>
<td>store intensive</td>
</tr>
<tr>
<td>1:0 11</td>
<td></td>
<td>eager</td>
</tr>
<tr>
<td>3:2</td>
<td>pl</td>
<td>RSE privilege level</td>
</tr>
<tr>
<td>4</td>
<td>be</td>
<td>RSE endian mode (0=little endian, 1=big endian)</td>
</tr>
<tr>
<td>29:16</td>
<td>loadrs</td>
<td>RSE load distance to tear point</td>
</tr>
</tbody>
</table>

Previous Function State Register Fields (PFS)

The Previous Function State register (PFS) contains multiple fields: Previous Frame Marker (pfm), Previous Epilog Count (pec), and Previous Privilege Level (ppl). These values are copied automatically on a call from the CFM register, Epilog Count Register (EC), and PSR.cpl (Current Privilege Level in the Processor Status Register) to accelerate procedure calling.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>37:0</td>
<td>pfm</td>
<td>previous frame marker</td>
</tr>
<tr>
<td>57:52</td>
<td>pec</td>
<td>previous epilog count</td>
</tr>
<tr>
<td>63:62</td>
<td>ppl</td>
<td>previous privilege level</td>
</tr>
</tbody>
</table>
Floating Point Registers

The IA-64 contains 128 floating-point registers. The first two are read only.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>f0</td>
<td>float register 0</td>
<td>$long</td>
<td>N</td>
<td>Y</td>
<td>$f0</td>
</tr>
<tr>
<td>f1</td>
<td>float register 1</td>
<td>$long</td>
<td>N</td>
<td>Y</td>
<td>$f1</td>
</tr>
<tr>
<td>f2-f31</td>
<td>lower float registers</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$f2-$f31</td>
</tr>
<tr>
<td>f32-f127</td>
<td>upper float registers</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$f32-$f127</td>
</tr>
</tbody>
</table>

Floating Point Status Register Fields

The Floating-Point Status Register (FPSR) contains the dynamic control and status information for floating-point operations. There is one main set of control and status information and three alternate sets.

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>traps.vd</td>
<td>0</td>
<td>Invalid Operation Floating-Point Exception fault disabled</td>
</tr>
<tr>
<td>traps.dd</td>
<td>1</td>
<td>Denormal/Unnormal Operating Floating-Point Exception fault disabled</td>
</tr>
<tr>
<td>traps.zd</td>
<td>2</td>
<td>Zero Divide Floating-Point Exception trap disabled</td>
</tr>
<tr>
<td>traps.od</td>
<td>3</td>
<td>Overflow Floating-Point Exception trap disabled</td>
</tr>
<tr>
<td>traps.ud</td>
<td>4</td>
<td>Underflow Floating-Point Exception trap disabled</td>
</tr>
<tr>
<td>traps.id</td>
<td>5</td>
<td>Inexact Floating-Point Exception trap disabled</td>
</tr>
<tr>
<td>sfo</td>
<td>18:6</td>
<td>main status field</td>
</tr>
<tr>
<td>sf1</td>
<td>31:19</td>
<td>alternate status field 1</td>
</tr>
<tr>
<td>sf2</td>
<td>44:32</td>
<td>alternate status field 2</td>
</tr>
<tr>
<td>sf3</td>
<td>57:45</td>
<td>alternate status field 3</td>
</tr>
</tbody>
</table>

Here is a description of the FPSR status field descriptions.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ftz</td>
<td>flush-to-zero mode</td>
</tr>
<tr>
<td>1</td>
<td>wre</td>
<td>widest range exponent</td>
</tr>
<tr>
<td>3:2</td>
<td>pc</td>
<td>precision control</td>
</tr>
<tr>
<td>5:4</td>
<td>rc</td>
<td>rounding control</td>
</tr>
<tr>
<td>Bits</td>
<td>Field</td>
<td>meaning</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>6</td>
<td>td</td>
<td>traps disabled</td>
</tr>
<tr>
<td>7</td>
<td>v</td>
<td>invalid operation</td>
</tr>
<tr>
<td>8</td>
<td>d</td>
<td>denormal/unnormal operand</td>
</tr>
<tr>
<td>9</td>
<td>z</td>
<td>zero divide</td>
</tr>
<tr>
<td>10</td>
<td>o</td>
<td>overflow</td>
</tr>
<tr>
<td>11</td>
<td>u</td>
<td>underflow</td>
</tr>
<tr>
<td>12</td>
<td>i</td>
<td>inexact</td>
</tr>
</tbody>
</table>
Intel x86

This section contains the following information:

- “Intel x86 General Registers” on page 407
- “Intel x86 Floating-Point Registers” on page 408
- “Intel x86 FPCR Register” on page 409
- “Intel x86 FPSR Register” on page 410
- “Intel x86 MXCSR Register” on page 410

NOTE >> The Intel x86 processor supports the IEEE floating-point format.

## Intel x86 General Registers

TotalView displays the Intel x86 general registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each general register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>General registers</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$eax</td>
</tr>
<tr>
<td>ECX</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDX</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBX</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBP</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESP</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESI</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td>Selector registers</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$cs</td>
</tr>
<tr>
<td>SS</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$ss</td>
</tr>
<tr>
<td>DS</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$ds</td>
</tr>
<tr>
<td>ES</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$es</td>
</tr>
<tr>
<td>FS</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$fs</td>
</tr>
<tr>
<td>GS</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$gs</td>
</tr>
<tr>
<td>EFLAGS</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$eflags</td>
</tr>
</tbody>
</table>
### Intel x86 Floating-Point Registers

TotalView displays the x86 floating-point registers in the Stack Frame Pane of the Process Window. The next table describes how TotalView treats each floating-point register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIP</td>
<td>Instruction pointer</td>
<td>$code[]</td>
<td>no</td>
<td>yes</td>
<td>$eip</td>
</tr>
<tr>
<td>FAULT</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$fault</td>
</tr>
<tr>
<td>TEMP</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$temp</td>
</tr>
<tr>
<td>INUM</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$inum</td>
</tr>
<tr>
<td>ECODE</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td></td>
<td>$ecode</td>
</tr>
<tr>
<td>ST0</td>
<td>ST(0)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st0</td>
</tr>
<tr>
<td>ST1</td>
<td>ST(1)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st1</td>
</tr>
<tr>
<td>ST2</td>
<td>ST(2)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st2</td>
</tr>
<tr>
<td>ST3</td>
<td>ST(3)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st3</td>
</tr>
<tr>
<td>ST4</td>
<td>ST(4)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st4</td>
</tr>
<tr>
<td>ST5</td>
<td>ST(5)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st5</td>
</tr>
<tr>
<td>ST6</td>
<td>ST(6)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st6</td>
</tr>
<tr>
<td>ST7</td>
<td>ST(7)</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$st7</td>
</tr>
<tr>
<td>FPCR</td>
<td>Floating-point control register</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$fpcr</td>
</tr>
<tr>
<td>FPSR</td>
<td>Floating-point status register</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpsr</td>
</tr>
<tr>
<td>FPTAG</td>
<td>Tag word</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fptag</td>
</tr>
<tr>
<td>FPIOFF</td>
<td>Instruction offset</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpioff</td>
</tr>
<tr>
<td>FPISEL</td>
<td>Instruction selector</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpiisel</td>
</tr>
<tr>
<td>FPDOFF</td>
<td>Data offset</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpdoff</td>
</tr>
<tr>
<td>FPDSSEL</td>
<td>Data selector</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpdsel</td>
</tr>
<tr>
<td>MXCSR</td>
<td>SSE status and control</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$mxcsr</td>
</tr>
<tr>
<td>MXCSR_R_MASK</td>
<td>MXCSR mask</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$mxcsr_mask</td>
</tr>
</tbody>
</table>
NOTE >> The Pentium III and 4 have 8 128-bit registers that are used by SSE and SSE2 instructions. TotalView displays these as 16 64-bit registers. These registers can be used in the following ways: 16 bytes, 8 words, 2 long longs, 4 floating point, 2 double, or a single 128-bit value. TotalView shows each of these hardware registers as two $long long registers. To change the type, dive and then edit the type in the data window to be an array of the type you wish. For example, cast it to “$char[16]”, “$float[4]”, and so on.

### Intel x86 FPCR Register

For your convenience, TotalView interprets the bit settings of the FPCR and FPSR registers. You can edit the value of the FPCR and set it to any of the bit settings outlined in the next table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC=RN</td>
<td>0x0000</td>
<td>To nearest rounding mode</td>
</tr>
<tr>
<td>RC=R-</td>
<td>0x2000</td>
<td>Toward negative infinity rounding mode</td>
</tr>
<tr>
<td>RC=R+</td>
<td>0x4000</td>
<td>Toward positive infinity rounding mode</td>
</tr>
<tr>
<td>RC=RZ</td>
<td>0x6000</td>
<td>Toward zero rounding mode</td>
</tr>
<tr>
<td>PC=SGL</td>
<td>0x0000</td>
<td>Single-precision rounding</td>
</tr>
<tr>
<td>PC=DBL</td>
<td>0x0080</td>
<td>Double-precision rounding</td>
</tr>
<tr>
<td>PC=EXT</td>
<td>0x00c0</td>
<td>Extended-precision rounding</td>
</tr>
<tr>
<td>EM=PM</td>
<td>0x0020</td>
<td>Precision exception enable</td>
</tr>
<tr>
<td>EM=UM</td>
<td>0x0010</td>
<td>Underflow exception enable</td>
</tr>
<tr>
<td>EM=OM</td>
<td>0x0008</td>
<td>Overflow exception enable</td>
</tr>
<tr>
<td>EM=ZM</td>
<td>0x0004</td>
<td>Zero-divide exception enable</td>
</tr>
<tr>
<td>EM=DM</td>
<td>0x0002</td>
<td>Denormalized operand exception enable</td>
</tr>
<tr>
<td>EM=IM</td>
<td>0x0001</td>
<td>Invalid operation exception enable</td>
</tr>
</tbody>
</table>
Using the Intel x86 FPCR Register

You can change the value of the FPCR within TotalView to customize the exception handling for your program.

For example, if your program inadvertently divides by zero, you can edit the bit setting of the FPCR register in the Stack Frame Pane. In this case, you would change the bit setting for the FPCR to include 0x0004 so that TotalView traps the “divide-by-zero” bit. The string displayed next to the FPCR register should now include EM=(ZM). Now, when your program divides by zero, it receives a SIGFPE signal, which you can catch with TotalView. See “Handling Signals” in Chapter 5 of the TotalView for HPC Users Guide for information on handling signals. If you did not set the bit for trapping divide by zero, the processor would ignore the error and set the EF=(ZE) bit in the FPSR.

Intel x86 FPSR Register

The bit settings of the Intel x86 FPSR register are outlined in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP=&lt;&lt;i&gt;</td>
<td>0x3800</td>
<td>Register &lt;i&gt; is top of FPU stack</td>
</tr>
<tr>
<td>B</td>
<td>0x8000</td>
<td>FPU busy</td>
</tr>
<tr>
<td>C0</td>
<td>0x0100</td>
<td>Condition bit 0</td>
</tr>
<tr>
<td>C1</td>
<td>0x0200</td>
<td>Condition bit 1</td>
</tr>
<tr>
<td>C2</td>
<td>0x0400</td>
<td>Condition bit 2</td>
</tr>
<tr>
<td>C3</td>
<td>0x4000</td>
<td>Condition bit 3</td>
</tr>
<tr>
<td>ES</td>
<td>0x0080</td>
<td>Exception summary status</td>
</tr>
<tr>
<td>SF</td>
<td>0x0040</td>
<td>Stack fault</td>
</tr>
<tr>
<td>EF=PE</td>
<td>0x0020</td>
<td>Precision exception</td>
</tr>
<tr>
<td>EF=UE</td>
<td>0x0010</td>
<td>Underflow exception</td>
</tr>
<tr>
<td>EF=OE</td>
<td>0x0008</td>
<td>Overflow exception</td>
</tr>
<tr>
<td>EF=ZE</td>
<td>0x0004</td>
<td>Zero divide exception</td>
</tr>
<tr>
<td>EF=DE</td>
<td>0x0002</td>
<td>Denormalized operand exception</td>
</tr>
<tr>
<td>EF=IE</td>
<td>0x0001</td>
<td>Invalid operation exception</td>
</tr>
</tbody>
</table>

Intel x86 MXCSR Register

This register contains control and status information for the SSE registers. Some of the bits in this register are editable. You cannot dive in these values.
The bit settings of the Intel x86 MXCSR register are outlined in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FZ</td>
<td>0x8000</td>
<td>Flush to zero</td>
</tr>
<tr>
<td>RC=RN</td>
<td>0x0000</td>
<td>To nearest rounding mode</td>
</tr>
<tr>
<td>RC=R-</td>
<td>0x2000</td>
<td>Toward negative infinity rounding mode</td>
</tr>
<tr>
<td>RC=R+</td>
<td>0x4000</td>
<td>Toward positive infinity rounding mode</td>
</tr>
<tr>
<td>RC=RZ</td>
<td>0x6000</td>
<td>Toward zero rounding mode</td>
</tr>
<tr>
<td>EM=PM</td>
<td>0x1000</td>
<td>Precision mask</td>
</tr>
<tr>
<td>EM=UM</td>
<td>0x0800</td>
<td>Underflow mask</td>
</tr>
<tr>
<td>EM=OM</td>
<td>0x0400</td>
<td>Overflow mask</td>
</tr>
<tr>
<td>EM=ZM</td>
<td>0x0200</td>
<td>Divide-by-zero mask</td>
</tr>
<tr>
<td>EM=DM</td>
<td>0x0100</td>
<td>Denormal mask</td>
</tr>
<tr>
<td>EM=IM</td>
<td>0x0080</td>
<td>Invalid operation mask</td>
</tr>
<tr>
<td>DAZ</td>
<td>0x0040</td>
<td>Denormals are zeros</td>
</tr>
<tr>
<td>EF=PE</td>
<td>0x0020</td>
<td>Precision flag</td>
</tr>
<tr>
<td>EF=UE</td>
<td>0x0010</td>
<td>Underflow flag</td>
</tr>
<tr>
<td>EF=OE</td>
<td>0x0008</td>
<td>Overflow flag</td>
</tr>
<tr>
<td>EF=ZE</td>
<td>0x0004</td>
<td>Divide-by-zero flag</td>
</tr>
<tr>
<td>EF=DE</td>
<td>0x0002</td>
<td>Denormal flag</td>
</tr>
<tr>
<td>EF=IE</td>
<td>0x0001</td>
<td>Invalid operation flag</td>
</tr>
</tbody>
</table>
Sun SPARC

This section has the following information:

- SPARC General Registers
- SPARC PSR Register
- SPARC Floating-Point Registers
- SPARC FPSR Register
- Using the SPARC FPSR Register

**NOTE >>** The SPARC processor supports the IEEE floating-point format.

### SPARC General Registers

TotalView displays the SPARC general registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each general register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>G0</td>
<td>Global zero register</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$g0</td>
</tr>
<tr>
<td>G1 - G7</td>
<td>Global registers</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$g1 - $g7</td>
</tr>
<tr>
<td>O0 - O5</td>
<td>Outgoing parameter registers</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$o0 - $o5</td>
</tr>
<tr>
<td>SP</td>
<td>Stack pointer</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$sp</td>
</tr>
<tr>
<td>O7</td>
<td>Temporary register</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$o7</td>
</tr>
<tr>
<td>L0 - L7</td>
<td>Local registers</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$l0 - $l7</td>
</tr>
<tr>
<td>I0 - I5</td>
<td>Incoming parameter registers</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$i0 - $i5</td>
</tr>
<tr>
<td>FP</td>
<td>Frame pointer</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$fp</td>
</tr>
<tr>
<td>I7</td>
<td>Return address</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$i7</td>
</tr>
<tr>
<td>PSR</td>
<td>Processor status register</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$psr</td>
</tr>
<tr>
<td>Y</td>
<td>Y register</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$y</td>
</tr>
<tr>
<td>WIM</td>
<td>WIM register</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>TBR</td>
<td>TBR register</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>
SPARC PSR Register

For your convenience, TotalView interprets the bit settings of the SPARC PSR register. You can edit the value of the PSR and set some of the bits outlined in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ET</td>
<td>0x00000020</td>
<td>Traps enabled</td>
</tr>
<tr>
<td>PS</td>
<td>0x00000040</td>
<td>Previous supervisor</td>
</tr>
<tr>
<td>S</td>
<td>0x00000080</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>EF</td>
<td>0x00001000</td>
<td>Floating-point unit enabled</td>
</tr>
<tr>
<td>EC</td>
<td>0x00002000</td>
<td>Coprocessor enabled</td>
</tr>
<tr>
<td>C</td>
<td>0x00100000</td>
<td>Carry condition code</td>
</tr>
<tr>
<td>V</td>
<td>0x00200000</td>
<td>Overflow condition code</td>
</tr>
<tr>
<td>Z</td>
<td>0x00400000</td>
<td>Zero condition code</td>
</tr>
<tr>
<td>N</td>
<td>0x00800000</td>
<td>Negative condition code</td>
</tr>
</tbody>
</table>

SPARC Floating-Point Registers

TotalView displays the SPARC floating-point registers in the Stack Frame Pane of the Process Window. The next table describes how TotalView treats each floating-point register, and the actions you can take with each register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Data Type</th>
<th>Edit</th>
<th>Dive</th>
<th>Specify in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0, F1, F0_F1</td>
<td>Floating-point registers (f registers), used singly</td>
<td>$float</td>
<td>no</td>
<td>yes</td>
<td>$f0, $f1, $f0_f1</td>
</tr>
<tr>
<td>F2 - F31</td>
<td>Floating-point registers (f registers), used singly</td>
<td>$float</td>
<td>yes</td>
<td>yes</td>
<td>$f2 - $f31</td>
</tr>
<tr>
<td>F0, F1, F0_F1</td>
<td>Floating-point registers (f registers), used as pairs</td>
<td>$double</td>
<td>no</td>
<td>yes</td>
<td>$f0, $f1, $f0_f1</td>
</tr>
<tr>
<td>F0/F1 - F30/ F31</td>
<td>Floating-point registers (f registers), used as pairs</td>
<td>$double</td>
<td>yes</td>
<td>yes</td>
<td>$2 - $f30_f31</td>
</tr>
</tbody>
</table>
TotalView allows you to use these registers singly or in pairs, depending on how they are used by your program. For example, if you use F1 by itself, its type is $float$, but if you use the F0/F1 pair, its type is $double$.

## SPARC FPSR Register

For your convenience, TotalView interprets the bit settings of the SPARC FPSR register. You can edit the value of the FPSR and set it to any of the bit settings outlined in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEXC=NX</td>
<td>0x00000001</td>
<td>Current inexact exception</td>
</tr>
<tr>
<td>CEXC=DZ</td>
<td>0x00000002</td>
<td>Current divide by zero exception</td>
</tr>
<tr>
<td>CEXC=UF</td>
<td>0x00000004</td>
<td>Current underflow exception</td>
</tr>
<tr>
<td>CEXC=OF</td>
<td>0x00000008</td>
<td>Current overflow exception</td>
</tr>
<tr>
<td>CEXC=NV</td>
<td>0x00000010</td>
<td>Current invalid exception</td>
</tr>
<tr>
<td>AEXC=NX</td>
<td>0x00000020</td>
<td>Accrued inexact exception</td>
</tr>
<tr>
<td>AEXC=DZ</td>
<td>0x00000040</td>
<td>Accrued divide by zero exception</td>
</tr>
<tr>
<td>AEXC=UF</td>
<td>0x00000080</td>
<td>Accrued underflow exception</td>
</tr>
<tr>
<td>AEXC=OF</td>
<td>0x00000100</td>
<td>Accrued overflow exception</td>
</tr>
<tr>
<td>AEXC=NV</td>
<td>0x00000200</td>
<td>Accrued invalid exception</td>
</tr>
<tr>
<td>EQ</td>
<td>0x00000000</td>
<td>Floating-point condition =</td>
</tr>
<tr>
<td>LT</td>
<td>0x00000400</td>
<td>Floating-point condition &lt;</td>
</tr>
<tr>
<td>GT</td>
<td>0x00000800</td>
<td>Floating-point condition &gt;</td>
</tr>
<tr>
<td>UN</td>
<td>0x00000c00</td>
<td>Floating-point condition unordered</td>
</tr>
<tr>
<td>QNE</td>
<td>0x00002000</td>
<td>Queue not empty</td>
</tr>
<tr>
<td>NONE</td>
<td>0x00000000</td>
<td>Floating-point trap type None</td>
</tr>
<tr>
<td>IEEE</td>
<td>0x00004000</td>
<td>Floating-point trap type IEEE Exception</td>
</tr>
<tr>
<td>UFIN</td>
<td>0x00008000</td>
<td>Floating-point trap type Unfinished FPop</td>
</tr>
<tr>
<td>UIMP</td>
<td>0x0000c000</td>
<td>Floating-point trap type Unimplemented FPop</td>
</tr>
<tr>
<td>SEQE</td>
<td>0x00010000</td>
<td>Floating-point trap type Sequence Error</td>
</tr>
</tbody>
</table>
Using the SPARC FPSR Register

The SPARC processor does not catch floating-point errors by default. You can change the value of the FPSR within TotalView to customize the exception handling for your program.

For example, if your program inadvertently divides by zero, you can edit the bit setting of the FPSR register in the Stack Frame Pane. In this case, you would change the bit setting for the FPSR to include **0x01000000** so that TotalView traps the “divide by zero” bit. The string displayed next to the FPSR register should now include **TEM=(DZ)**. Now, when your program divides by zero, it receives a **SIGFPE** signal, which you can catch with TotalView. See “**Handling Signals**” in Chapter 5 of the *TotalView for HPC Users Guide* for more information. If you did not set the bit for trapping divide by zero, the processor would ignore the error and set the **AEXC=(DZ)** bit.

---

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS</td>
<td>0x00400000</td>
<td>Nonstandard floating-point FAST mode</td>
</tr>
<tr>
<td>TEM=NX</td>
<td>0x00800000</td>
<td>Trap enable mask - Inexact Trap Mask</td>
</tr>
<tr>
<td>TEM=DZ</td>
<td>0x01000000</td>
<td>Trap enable mask - Divide by Zero Trap Mask</td>
</tr>
<tr>
<td>TEM=UF</td>
<td>0x02000000</td>
<td>Trap enable mask - Underflow Trap Mask</td>
</tr>
<tr>
<td>TEM=OF</td>
<td>0x04000000</td>
<td>Trap enable mask - Overflow Trap Mask</td>
</tr>
<tr>
<td>TEM=NV</td>
<td>0x08000000</td>
<td>Trap enable mask - Invalid Operation Trap Mask</td>
</tr>
<tr>
<td>EXT</td>
<td>0x00000000</td>
<td>Extended rounding precision - Extended precision</td>
</tr>
<tr>
<td>SGL</td>
<td>0x10000000</td>
<td>Extended rounding precision - Single precision</td>
</tr>
<tr>
<td>DBL</td>
<td>0x20000000</td>
<td>Extended rounding precision - Double precision</td>
</tr>
<tr>
<td>NEAR</td>
<td>0x00000000</td>
<td>Rounding direction - Round to nearest (tie-even)</td>
</tr>
<tr>
<td>ZERO</td>
<td>0x40000000</td>
<td>Rounding direction - Round to 0</td>
</tr>
<tr>
<td>PINF</td>
<td>0x80000000</td>
<td>Rounding direction - Round to +Infinity</td>
</tr>
<tr>
<td>NINF</td>
<td>0xc0000000</td>
<td>Rounding direction - Round to -Infinity</td>
</tr>
</tbody>
</table>
Appendices

This section contains supplementary information, currently a single appendix, Appendix A, “MPI Startup,” on page 417 that provides information on creating startup files for various Message Passing Interface (MPI) implementations.
Appendix A

MPI Startup

Overview

Here you will find information that will allow you to create startup profiles for environments that Rogue Wave Software doesn't define. Any customizations made to your MPI environment will be available for later selection in the Sessions Manager where they will appear in the File > Debug New Parallel Program dialog's Parallel System Name list.

Rogue Wave Software products know about different Message Passing Interface (MPI) implementations. Because so many implementations are standard, our products usually do the right thing. Unfortunately, subtle differences in your environment or an implementation can cause difficulties that prevent our products from automatically starting your program. In these cases, you must declare what needs to be done.
Customizing Your Parallel Configuration

The File > Debug New Parallel Program dialog box (TotalView for HPC) or the Add parallel program screen (MemoryScape) let you select a parallel configuration. If the default configurations that Rogue Wave Software provides do not meet your needs, you can either overwrite these configurations or create new ones.

The default definitions for parallel configurations reside in the parallel_support.tvd file, located in your totalview/lib installation directory. For TotalView — and MemoryScape when used with TotalView — you can use the variable TV::parallel_configs to customize parallel configurations. For standalone MemoryScape, you need to instead add any new configurations directly to the parallel_support.tvd file. Both these methods are discussed here.

TotalView

If you are using TotalView, set the TV::parallel_configs variable, either local to your TotalView installation or globally:

- Globally, in your system's .tvdrc file. If you set this variable here, everyone using this TotalView version will see the definition.
- Locally, in your .totalview/tvdrc file. You will be the only person to see this definition when you start TotalView.

You can also directly edit the parallel_support.tvd file, located in the totalview/lib installation directory area, but reinstalling TotalView overwrites this file so this is not recommended.

For TotalView, if you are using a locally-installed MPI implementation, you should add it to your PATH variable. By default, both TotalView and MemoryScape use the information in PATH to find the parallel launcher (for example, mpirun, mpiexec, poe, srun, prun, dmpirun, and so on). Generally, if you can run your parallel job from a command line, TotalView can also run it.

If you have multiple installed MPI systems — for example, multiple versions of MPICH installed on a common file server — only one can be in your path. In this case, specify an absolute path to launch it, which means you will need to customize the TV::parallel_configs list variable or the parallel_support.tvd file contained within your installation directory so that it does not rely on your PATH variable.

The easiest way to create your own startup configuration for TotalView is to copy a similar configuration from the TV::private::parallel_configs_base variable (found in the parallel_support.tvd file, located in your installation directory at totalview/lib) to the TV::parallel_configs variable, and then edit it. Save the TV::parallel_configs variable in the tvdrc file located in the .totalview subdirectory in your home directory. For standalone MemoryScape, please see Standalone MemoryScape.
When you add configurations, they are simply added to a list. This means that if TotalView supplies a definition named `foo` and you create a definition also named `foo`, both exist and your product chooses the first one in the list. Because both are displayed, be careful to give each new definition a unique name.

**Standalone MemoryScape**

For the standalone MemoryScape product, to customize the way an MPI program starts up, edit the `parallel-support.tvd` file, located in the `totalview/lib` installation directory area, using its existing syntax and definitions as a model for any new MPI implementations you add.

Note that this file is *overwritten* when you install a new TotalView or MemoryScape release. Be sure to make a backup copy of any customizations you make to this file. See Customizing Your Parallel Configuration for information on how to edit this file.

Make a backup copy of customizations you add to the `parallel_support.tvd` file, since the file is overwritten if you reinstall TotalView or MemoryScape.
Example Parallel Configuration Definitions

This section provides three examples of customized parallel configurations. See Customizing Your Parallel Configuration on page 418 for information on where to place these definitions.

NOTE >> Any customizations made to your MPI environment will be available for later selection in the Sessions Manager where they will appear in the File > Debug New Parallel Program dialog's Parallel System Name list.

Here are three examples:

```dset TV::parallel_configs {
  #Argonne MPICH
  name: MPICH;
  description: Argonne MPICH;
  starter: mpirun -tv -ksq %s %p %a;
  style: setup_script;
  tasks_option: -np;
  nodes_option: -nodes;
  env_style: force;
  pretest: mpichversion;

  #Argonne MPICH2
  name: MPICH2;
  description: Argonne MPICH2;
  starter: $mpiexec -tvsu %s %p %a;
  style: manager_process;
  tasks_option: -n;
  env_option: -env;
  env_style: assign_space_repeat;
  comm_world: 0x44000000;
  pretest: mpich2version

  # AIX POE
  name: poe - AIX;
  description: IBM PE - AIX;
  tasks_option: -procs;
  tasks_env: MP_PROCS;
  nodes_option: -nodes;
  starter: /bin/poe %p %a %s;
  style: bootstrap;
  env: NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat;
        /usr/lib/nls/msg/%L/%N.cat;
  service_tids: 2 3 4;
  comm_world: 0;
  pretest: test -x /bin/poe
```
msq_lib: /usr/lpp/ppe.poe/lib/%m

All lines (except for comments) end with a semi-colon (;). Add spaces freely to improve the readability of these definitions as TotalView and MemoryScape ignore them.

Notice that the MPICH2 definition contains the $mpiexec variable. This variable is defined elsewhere in the parallel_support.tvd file as follows:

   set mpiexec mpiexec;

There is no limit to how many definitions you can place within the parallel_support.tvd file or within a variable. The definitions you create will appear in the Parallel system pulldown list in the File > New dialog box (TotalView) or the Add parallel program screen (MemoryScape) and can be used as an argument to the -mpi option of the CLI's dload command.

Note that for MemoryScape, you do not set this variable because the tvdrc file is not read. Rather, directly edit the parallel_support.tvd file.

The fields that you can set are as follows:

   comm_world
      Only use this option when style is set to bootstrap. This variable is the definition of MPI_COMM_WORLD in C and C++. MPI_COMM_WORLD is usually a #define or enum to a special number or a pointer value. If you do not include this field, TotalView and MemoryScape cannot acquire the rank for each MPI process.

   description
      (optional) A string describing what the configuration is used for. There is no length limit.

   env
      (optional) Defines environment variables that are placed in the starter program's environment. (Depending on how the starter works, these variables may not make their way into the actual ranked processes.) If you are defining more than one environment variable, define each in its own env clause.

      The format to use is:
      variable_name=value

   env_option
      (optional) Names the command-line option that exports environment variables to the tasks started by the launcher program. Use this option along with the env_style field.

   env_style
      (optional) Contains a list of environment variables that are passed to tasks.

      assign: The argument to be inserted to the command-line option named in env_option is a comma-separated list of environment variable name=value pairs; that is,

      NAME1=VALUE1,NAME2=VALUE2,NAME3=VALUE3

      This option is ignored if you do not use an env_option clause.
assign_space_repeat: The argument after env_option is a space-separated name/value pair that is assigned to an environment variable. The command within env_option is repeated for each environment variable; that is, suppose you enter:

- env NAME1 VALUE1 - env NAME2 VALUE2
- env NAME3 VALUE3

This mode is primarily used for the mpiexec.py MPICH2 starter program.

excenv

One of the following three strings:

export: The argument to be inserted after the command named in env_option. This is a comma-separated list of environment variable names; that is,

NAME1,NAME2,NAME3

This option is ignored if you do not use the env_option clause.

force: Environment variables are forced into the ranked processes using a shell script. TotalView or MemoryScape will generate a script that launches the target program. The script also tells the starter to run that script. This clause requires that your home directory be visible on all remote nodes. In most cases, you will use this option when you need to dynamically link memory debugging into the target. While this option does not work with all MPI implementations, it is the most reliable method for MPICH1.

none: No argument is inserted after env_option.

msq_lib

(optional) Names the dynamically loaded library that TotalView and MemoryScape use when it needs to locate message queue information. You can name this file using either a relative or full pathname.

name

A short name describing the configuration. This name shows up in such places as the File > New dialog box and in the Process > Startup Parameter's Parallel tab in TotalView and the Add parallel program screen in MemoryScape. TotalView and MemoryScape remember which configuration you use when starting a program so that they can automatically reapply the configuration when you restart the program.

Because the configuration is associated with a program's name, renaming or moving the program destroys this association.

nodes_option

Names the command-line option (usually -nodes) that sets the number of node upon which your program runs. This statement does not define the value that is the argument to this command-line option.

Only omit this statement if your system doesn't allow you to control the number of nodes from the command line. If you set this value to zero ("0"), this statement is omitted.

pretest

(optional) Names a shell command that is run before the parallel job is launched. This command must run quickly, produce a timely response, and have no side-effects. This is a test, not a setup hook.
TotalView or MemoryScape may kill the test if it takes too long. It may call it more than once to be sure if every-thing is OK. If the shell command exit is not as expected, TotalView or MemoryScape complains and asks for per-mission before continuing.

**pretest_exit**

The expected error code of the pretest command. The default is zero.

**service_tids**

(optional) The list of thread IDs that TotalView and MemoryScape marks as service threads. When using To-talView, you can use the View > Display Managers command to tell TotalView to display them.

A service thread differs from a system manager thread in that it is created by the parallel runtime and are not created by your program. POE for example, often creates three service threads.

**starter**

Defines a template that TotalView and MemoryScape use to create the command line that starts your program. In most cases, this template describes the relative position of the arguments. However, you can also use it to add extra parameters, commands, or environment variables. Here are the three substation parameters:

- **%a**: Replaced with the command-line arguments passed to rank processes.
- **%p**: Replaced with the absolute pathname of the target program.
- **%s**: Replaced with additional startup arguments. These are parameters to the starter process, not the rank pro cesses.

For example:

```
starter: mpirun -tv -all-local %s %p %a;
```

When the user selects a value for the option indicated by the **nodes_option** and **tasks_options**, the argument and the value are placed within the **%s** parameter. If you enter a value of 0 for either of these, MemoryScape and TotalView omit the parameter. In MemoryScape, 0 is the default.

**style**

MPI programs are launched in two ways: either by a manager process or by a script. Use this option to name the method, as follows:

- **manager_process**: The parallel system uses a binary manager process to oversee process creation and pro cess lifetime. Our products attach to this process and communicate with it using its debug interface. For exam ple, IBM's poe uses this style.

  
  ```
  style: manager_process;
  ```

- **setup_script**: The parallel system uses a script—which is often mpirun—to set up the arguments, environ- ment, and temporary files. However, the script does not run as part of the parallel job. This script must under stand the -tv command-line option and the TOTALVIEW environment variable.

- **bootstrap**: The parallel system attempts to launch an uninstruemented MPI by interposing TotalView or Mem oryScape inside the parallel launch sequence in place of the target program. This does not work for MPICH and SGI MPT.
tasks_env
The name of an environment variable whose value is the expected number of parallel tasks. This is consulted when the user does not explicitly specify a task count.

tasks_option
(sometimes required) Lets you define the option (usually -np or -procs) that controls the total number of tasks or processes.

Only omit this statement if your system doesn't allow you to control the number of tasks from the command line. If you set this to 0, this statement is omitted.
<table>
<thead>
<tr>
<th>Symbol/Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.debug gnu_debuglink</td>
<td>Copy subdirectory</td>
</tr>
<tr>
<td>.totalview/lib_cache</td>
<td>Subdirectory</td>
</tr>
<tr>
<td>.tvd files</td>
<td>TVD files</td>
</tr>
<tr>
<td>@ symbol for action point</td>
<td>Identifiers</td>
</tr>
<tr>
<td>* expr</td>
<td>Identifiers</td>
</tr>
<tr>
<td>/proc file system</td>
<td>Identifiers</td>
</tr>
<tr>
<td># scoping separator character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%A server launch replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%B server launch replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%C server launch replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%D path name replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%H hostname replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%L host and port replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%N line number replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%P password replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%S source file replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%t1 file replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%t2 file replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>%V verbosity setting replacement character</td>
<td>Identifiers</td>
</tr>
<tr>
<td>= symbol for PC of current buried stack frame</td>
<td>Identifiers</td>
</tr>
<tr>
<td>&gt; symbol for PC</td>
<td>Identifiers</td>
</tr>
<tr>
<td>$mpiexec variable</td>
<td>Identifiers</td>
</tr>
<tr>
<td>$newval variable in</td>
<td>Identifiers</td>
</tr>
<tr>
<td>$oldval variable in</td>
<td>Identifiers</td>
</tr>
<tr>
<td>$stop function</td>
<td>Identifiers</td>
</tr>
<tr>
<td>activating type transformations</td>
<td>Identifiers</td>
</tr>
<tr>
<td>Add parallel program screen</td>
<td>Identifiers</td>
</tr>
<tr>
<td>adding group members</td>
<td>Identifiers</td>
</tr>
<tr>
<td>adding groups</td>
<td>Identifiers</td>
</tr>
<tr>
<td>address</td>
<td>Identifiers</td>
</tr>
<tr>
<td>address property</td>
<td>Identifiers</td>
</tr>
<tr>
<td>addressing_callback</td>
<td>Identifiers</td>
</tr>
<tr>
<td>advancing by steps</td>
<td>Identifiers</td>
</tr>
<tr>
<td>aggregate data</td>
<td>Identifiers</td>
</tr>
<tr>
<td>AIX compiling on</td>
<td>Identifiers</td>
</tr>
<tr>
<td>linking C++ to dbfork library</td>
<td>Identifiers</td>
</tr>
<tr>
<td>linking to dbfork library</td>
<td>Identifiers</td>
</tr>
<tr>
<td>swap space</td>
<td>Identifiers</td>
</tr>
<tr>
<td>aix_use_fast_trap command-line option</td>
<td>Identifiers</td>
</tr>
<tr>
<td>aix_use_fast_trap variable</td>
<td>Identifiers</td>
</tr>
<tr>
<td>alias command</td>
<td>Identifiers</td>
</tr>
<tr>
<td>aliases default</td>
<td>Identifiers</td>
</tr>
<tr>
<td>removing</td>
<td>Identifiers</td>
</tr>
<tr>
<td>append, see dlappend command</td>
<td>Identifiers</td>
</tr>
<tr>
<td>appending to CLI variable lists</td>
<td>Identifiers</td>
</tr>
<tr>
<td>architectures</td>
<td>Identifiers</td>
</tr>
<tr>
<td>Intel IA-64</td>
<td>Identifiers</td>
</tr>
<tr>
<td>Intel-x86</td>
<td>Identifiers</td>
</tr>
<tr>
<td>PowerPC</td>
<td>Identifiers</td>
</tr>
<tr>
<td>SPARC</td>
<td>Identifiers</td>
</tr>
<tr>
<td>arenas</td>
<td>Identifiers</td>
</tr>
<tr>
<td>ARGSS variable</td>
<td>Identifiers</td>
</tr>
<tr>
<td>ARGSS_DEFAULT variable</td>
<td>Identifiers</td>
</tr>
<tr>
<td>ARGSS(dpomid) variable</td>
<td>Identifiers</td>
</tr>
<tr>
<td>arguments command line</td>
<td>Identifiers</td>
</tr>
<tr>
<td>default</td>
<td>Identifiers</td>
</tr>
<tr>
<td>for totalview command</td>
<td>Identifiers</td>
</tr>
<tr>
<td>for tvdsvr command</td>
<td>Identifiers</td>
</tr>
</tbody>
</table>
arrays 261
automatic dereferencing 261
gathering statistical data 118
number of elements displayed 261
arriving at barrier 38
as, see dassign command
ask on dlopen option 380
ask_on_dlopen option 381
ask_on_dlopen variable 261
assemble, displaying symbolically 276
assembler instructions, stepping 147
assign, see dassign command
assigning string values 32
assigning values 32
asynchronous execution 76
at, see dattach command
attach, see dattach command
attaching to parallel processes 34
attaching to processes 34
attaching to ranks 34
attaching, using PIDs 35
auto_array_cast_bounds variable 261
auto_array_cast_enabled variable 261
auto_deref_in_all_c variable 262
auto_deref_in_all_fortran variable 262
auto_deref_initial_c variable 262
auto_deref_initial_fortran variable 263
auto_deref_nested_c variable 263
auto_deref_nested_fortran variable 263
auto_load_breakpoints variable 264
auto_read_symbols_at_stop variable 264
auto_save_breakpoints variable 264
automatic dereferencing 262
automatic dereferencing of arrays 261
automatically attaching to processes 287
B
b, see dbreak command
ba, see dbarrier command
background command-line option 340
back-tick analogy 25
barrier breakpoint 38
barrier is satisfied 254, 265
BARRIER_STOP_ALL variable 37, 39, 253
barrier_stop_all variable 265
BARRIER_STOP_WHEN_DONE variable 37, 39, 253
barrier_stop_when_done variable 265
barrier, see dbarrier command
barriers 37, 38
arriving 38
creating 38
what else is stopped 37
base_name 207
Batch debugging, tvscript 235
baud rate, specifying 354
baw, see dbarrier command
bg, see dbarrier command
bg, see dbarrier command
temporary 153
triggering 43
bt, see dbreak command
build_struct_transform
defined 312
example 311
lists 312
members argument 312
name argument 312
bulk launch 357
bulk_launch_base_timeout variable 266
bulk_launch_enabled variable 266
bulk_launch_incr_timeout variable 266
bulk_launch_tmphfile1_header_line variable 266
bulk_launch_tmphfile1_host_lines variable 267
bulk_launch_tmphfile1_trailer_line variable 267
bulk_launch_tmphfile2_header_line variable 266
bulk_launch_tmphfile2_host_lines variable 267
bulk_launch_tmphfile2_trailer_line variable 267
buried stack frame 100
by_language_rules 203
by_path 203
by_type_index 203
C
C language
escape characters 32
using with C++View 329
C shell 376
c_type_strings 267
c_type_strings variable 267
C++  
- demangler 271, 342  
  - including libdbfork.h 369  
STL instantiation 308  
C++ View  
- and multithread safety 326  
- and programming design 325  
- and template classes 321  
  - compiling and linking 334  
  - described 318  
- example display function 320  
- example of use with ReplayEngine 328  
- use of with ReplayEngine 327  
- using with C 329  
cache, flushing 46  
cache, Memory Debugger 298  
cache, see dcache command  
call stack 156  
  - displaying 167  
  - see also stack frame 16, 166  
call tree saved position 301  
callback command-line option 353  
callback list 285  
callback_host 353  
callback_ports 354  
callbacks 284  
  - after loading a program 288  
  - when opening the CLI 287  
capture command 25, 171  
case sensitive searching 290  
cast subcommand 203  
Cast to array with bounds checkbox 261  
casting variables 121  
ccq command-line option 341  
CGROUP variable 254  
changing CLI variables 135, 136  
changing dynamic context 156  
changing focus 79  
changing value of program variable 32, 51, 96, 115, 130, 150, 153  
chase_mouse variable 299  
checking interior pointers variable 297  
class transformations 310  
classes, transforming 312  
CLI  
  - activated from GUI flag 299  
  - sourcing files 311  
  - startup file 311  
CLI variables  
  - aix_use_fast_trap 261, 293, 294  
  - ARGS 253  
  - ARGS_DEFAULT 253  
  - ask_on_dlopen 261  
  - auto_array_cast_bounds 26  
  - auto_array_cast_enabled 26  
  - auto_deref_in_all_c 262  
  - auto_deref_in_all_fortran 26  
  - auto_deref_initial_fortran 2  
  - auto_deref_initial_c 262  
  - auto_deref_nested_c 263  
  - auto_deref_nested_fortran 263  
  - auto_load_breakpoints 264  
  - auto_read_symbols_at_stop 264  
  - auto_save_breakpoints 264  
  - BARRIER_STOP_ALL 37, 39, 253  
  - barrier_stop_all 265  
  - BARRIER_STOP_WHEN_DONE 37, 253  
  - barrier_stop_when_done 26  
  - blue_gene_launch_string 26  
  - bluegene_io_interface 265  
  - bluegene_server_launch_time out 266  
  - bulk_launch_base_timeout 266  
  - bulk_launch_enabled 266  
  - bulk_launch_incr_timeout 2  
bulk_launch_tmpfile1_header_line 266  
bulk_launch_tmpfile1_host_lines 267  
bulk_launch_tmpfile1_trailer_line 267  
bulk_launch_tmpfile2_header_line 267  
bulk_launch_tmpfile2_host_lines 267  
bulk_launch_tmpfile2_trailer_line 267  
c_type_strings 267  
CGROUP 254  
  - changing 135, 136  
  - chase_mouse 299  
  - comline_patch_area_base 269  
  - comline_path_area_length 269  
  - COMMAND_EDITING 254  
  - command_editing 269  
  - compile_expressions 270  
  - compiler_vars 270  
  - control_c_quick_shutdown 270  
  - copyright_string 271  
  - current_cplusplus_demangler 271  
  - current_fortran_demangler 271  
  - data_format_double 272  
  - data_format_ext 273  
  - data_format_int16 274  
  - data_format_int32 274  
  - data_format_int64 274  
  - data_format_int8 274  
  - data_format_single 274  
  - data_format_singlen 274  
  - dbfork 275  
  - default_value_for 136  
  - default_snippet_extent 297  
  - default_stderr_append 275  
  - default_stderr_filename 275  
  - default_stderr_is_stdout 275  
  - default_stdin_filename 275  
  - default_stdout_append 276  
  - default_stdout_filename 276  
  - deleting 135, 136
display_assembler_symbolically 276
display_bytes_kb_mb 299
display_font_dpi 299
dll_ignore_prefix 276
dll_read_all_symbols 276
dll_read_loader_symbols_only 277
dll_read_no_symbols 277
dll_stop_suffix 278
dump_core 279
dwhere_qualification_level 279
dynamic 280
editor_launch_string enabled 299
env 281
eroCcodes 118
EXECUTABLE_PATH 35, 101, 255
EXECUTABLE_SEARCH_PATH 255
H
fixed_font 299
fixed_font_family 300
fixed_font_size 300
follow_clone 281
font 300
force_default_cplusplus_demangle 281
force_default_f9x_demangle 282
force_window_position 300
frame_offset_x 300
frame_offset_y 300
geometry_call_tree 301
geometry_cli 301
geometry_globals 301
geometry_help 301
geometry_memory_stats 30
2
global_typenames 282
gnu_debuglink 282, 366
gnu_debuglink_checksum_flag 366
gnu_debuglink_global_directory 282, 367
gnu_debuglink_search_path 283
GROUP 255
GROUPS 103, 256
hia_allow_ibm_poe 297
ignore_control_c 284
ignore_snippets 297
image_load_callbacks 284
in_setup 284
kcc_classes 284
keep_expressions 304
keep_search_dialog 304
kernel_launch_string 285
kill_callbacks 285
leak_check_interior_pointer_s 297, 298
leak_max_cache 298
leak_max_chunk 298
library_cache_directory 285, 286
LINES_PER_SCREEN 256
local_interface 286
local_server 286
local_server_launch_string 286
MAX_LIST 100, 257
message_queue 286
nptl_threads 287
OBJECT_SEARCH_MAPPING S 255, 257
OBJECT_SEARCH_PATH 257
open_cli_callback 287
parallel 287
parallel_attach 287
parallel_configs 418
parallel_stop 288
platform 288
pop_at_breakpoint 304
pop_on_error 304
process_load_callbacks 288
PROMPT 257
PTSET 257
restart_threshold 289
save_window_pipe_or_filename 289
search_case_sensitive 290
server_launch_enabled 290
server_launch_timeout 290
server_response_wait_time out 290
SGROUP 257
SHARE_ACTION_POINT 258
share_action_point 290
shared_data_filters 298
show_startup_parameters 305
show_sys_thread_id 305
signal_handling_mode 290
single_click_thread_id 290
source_pane_tab_width 292
SOURCE_SEARCH_MAPPING S 258
SOURCE_SEARCH_PATH 25
8
spell_correction 292
stack_trace_qualification_level 292
STOP_ALL 42, 258
stop_all 292
stop relatives_on_proc_error 292
suffix 293
TAB_WIDTH 101, 259
THREADS 259
toolbar_style 305
tooltips_enabled 305
TOTAL_VERSION 259
TOTALVIEW_ROOT_PATH 25
9
TOTALVIEW_TCLLIB_PATH 2
59
ttf 293
ui_font 305
ui_font_family 306
ui_font_size 306
user_threads 294
using_color 306
using_text_color 306
using_title_color 306
VERBOSE 260
version 294, 306
viewing 135, 136
visualizer_launch_enabled 2
94
visualizer_launch_string 294
visualizer_max_rank 295
warn_step_throw 295
WGROUP 260
wrap_on_search 295, 296
closed loop, see closed loop
closes shared libraries 185
clusterid property 197
co, see dcont command
code snippets 297
code, displaying 100
color
foreground 343
comand-line options
-ccq 341
comline_patch_area_base variable 269
comline_path_area_length variable 269
command arguments 253
command focus 79
command input, blocking 158
command line arguments 133
command output 25
command prompt 257
command summary 7
command verb
  actionpoint command 185
COMMAND_EDITING variable 254
command_editing variable 269
command, Tools > Dynamic Libraries 66
command-line options
-aix_use_fast_trap 340
-background 340
-bg 340
-ccq 341
-compiler_vars 340
  control_c_quick_shutdown 341
-cuda 341
-dbfork 341
-debug_file 341
-demangler 342
-display 342
-dll_ignore_prefix 342
-dll_stop_suffix 342
-dump_core 343
-e 343
-ent 343
-env 343
-f9x_demangler 344
-fg 344
-foreground 343
-global_types 344
-gnu_debuglink 344
  gnu_debuglink_checksum 344
-ipv6_support 344
-kcc_classes 345
-lb 345
-message_queue 345, 346
-mqd 346
-nccq 341
-nlb 345
-no_compiler_vars 341
  no_control_c_quick_shutdown 341
-no_cuda 341
-no_dbfork 341
-no_ent 343
-no_global_types 344
-no.gnu_debuglink 344
  no.gnu_debuglink_checksum 344
-no_ipv6_support 344
-no_kcc_classes 345
-no.message_queue 346
-no_mqd 346
-no.nptl_threads 347
-no.parallel 347
-no_startup_scripts 347
-no_team 350
-no.teamplus 350
-no.user_threads 340, 351
-nptl_threads 347
-parallel 347
-patch_area_base 348
-patch_area_length 348
-pid 348
-r 348
-remote 348
-s 348
-search_path 349
-serial 349
-shm 349
-signal_handling_mode 349
-stderr 350
-stderr_append 350
-stderr_is_stdout 350
-stdin 350
-stdout 350
-stdout_append 350
-team 350
-timeplus 350
-tvhome 350
-user_threads 340, 351
-verbosity 351
-xterm_name 351
commands
  responding to 202
totalview 338
tvdsvr, syntax and use 352
user-defined 23
commands verb
  actionpoint command 181
  expr command 189
  group command 194
  process command 197
  thread command 219
  type command 222
compile_expressions variable 270
compiler property 226
compiler_vars command-line option 340
compiler_vars variable 270
compilers, KCC 284
compiling
  debugging symbols 361
  for C++View 334
  -g compiler command-line option 361
  on Bluegene 363
  on IBM Power Linux 363
  on Itanium 365
  on SunOS 365
  on x86 363
  on x86-64 364
  options 361
compressed list of processes, defined. See plst.
Compressed List Syntax, defined 140
conditional watchpoints 159
connection directory 356
console output for tvdsrv 354
console output redirection 341
cont, see dcont command
continue sig property 219
continuing execution 84
control group variable 254
control group, stopping 292
control list element 255
control_c_quick_shutdown command-line option 341
core
  dumping for TotalView 343
  when needing to debug TotalView itself 279
core files, loading 35
count property 195
Cray pointers, example using Fortran with C++View 330
create subcommand 225
creating barrier breakpoints 38
creating commands 23
creating groups 86
creating new process objects 104
creating threads 84
creating type transformations 307
Ctrl+C, ignoring 284
Ctrl+D to exit CLI 170, 172
CUDA
  CLI command detail 55
dcuda > block command 55
dcuda CLI command 55
  support for GPU threads 22, 55
current frame marker register
    Intel IA-64 403
current list location 69
current_cplus_demangler variable 271
current_fortran_demangler variable 271
D
  d, see ddown command
dactions command 28
dassign command 32
data format
    presentation styles 272
    transforming
      with C++View 318
data representation
    simplifying using elision 324
data size 106
data size limit in C shell 376
data_format_double
    variable 272
data_format_ext variable 273
data_format_int16 variable 274
data_format_int32 variable 274
data_format_int64 variable 274
data_format_int8 variable 273
data_format_single variable 274
data_format_stringlen variable 274
datatype cast expr 313
data type incompatibilities 32
dattach command 34
dbfork command-line option 341
dbfork library 368
  linking with syntax 341
dbfork variable 275
dbreak command 42
  setting expression in 42
dcache command 46
dcalltree command 47
dcont command 53
dcont command 53
dcuda command 55
delete command 60
ddetach command 62
ddischarge command 64
ddl_read_all_symbols variable 276
ddlopen command 66
ddown command 69
de, see ddelete command
deactivating action points 64
deadlocks at barriers 39
debug gnu_debuglink copy 366
debug_file command-line option 341, 354
debugger server 352
debugging remote systems 46
debugging session, ending 170
dec2hex command 184
default aliases 23
default arguments 133, 253
  modifying 133
default focus 79
  default preferences, setting 137
  default value of variables, restoring 152
default_snippet_extent 297
default_sterr_append variable 275
default_sterr_filename variable 275
default_sterr_is_stdout variable 275
default_stdin_filename variable 276
default_stdout_filename variable 276
default_stdout_append variable 276
default_stdout_filename variable 276
deferred reading, shared library symbols 201
defining MPI startup implementations 417
defining the current focus 257
delete verb, expr command 189
delete, see ddelete command
deleting action points 60, 181, 194, 197, 222
deleting cache 46
deleting CLI variables 135, 136
deleting groups 86, 87
deleting variables 152
demangler 271
  C++ 271
  forcing use 281, 282
  Fortran 271
  overriding 342, 344
demangler command-line option 342
denable command 71
dereferencing 262, 263, 264
  C pointers
    automatically 262
  C structure pointers
    automatically 263
dereferencing values
  automatically 262
det, see ddetach command
detach, see ddetach command
detaching from processes 62
deflush command 73
dfocus command 76, 118
dg command 79
dga command 82
dgo command 84
dgroups command 86
  -add 87
  -delete 87
dhalt command 91
dheap command 92
dheap command, see also heap
ddebugging
dhold command 96
di, see ddisable command
directory search paths 255
disable, see ddisable command
disabled action points list (ambiguous context) 28
disabling action points 27, 28, 64
display call stack 167
display command-line option 342
display_assembler_symbolically variable 276
display_bytes_kb_mb variable 299
display_font_dpi variable 299
displaying
  code 100
  current execution location 167
  error message
    information 260
  help information 171
  information on a name 162
    lines 257
    values 118
displaying expressions 118
displaying memory 73
displaying memory values 73
diving, single click 305
dkill command 98
dlappend command 99
dlist command 100, 257
dlist, number of lines displayed 257
dll command 185
  DLL Do Query on Load list 380
  DLL Don't Query on Load list 380
dll_ignore_prefix command-line option 342
dll_ignore_prefix variable 276
dll_read_loader_symbols_only 277
  dll_read_no_symbols variable 277
dll_stop_suffix command-line option 342
  dll_stop_suffix variable 278, 279
dload command 103
dlopen 67, 380
  ask when loading 261
dmstat command 106
dnext command 109
dnexti command 112
done property 189
double-precision data format 272
dout command 115
down, see ddown command
dpid 254
  dpid property 220
  dprint command 118
  dptsets command 124
drerun - drun differences 128
drerun command 127
drestart command 130
drun - drerun differences 128
drun command 98, 132
  poe issues 128, 133
  reissuing 133
dsession 135
dset command 135, 136
dstatus command 139
dstep command 143
  iterating over focus 143
dstepi command 147
duhtp, see duhold command
duid property 198, 220
dump subcommand 203, 206
dump_core command-line option 343
dump_core variable 279
dunhold command 150
dunset command 152
duntil command 153
  group operations 154
dup command 156
dwait command 158
dwatch command 159
dwhat command 162
dwhere command 166
  levels 256
dwhere_quality_level variable 279
  dwhere_qualification_level variable 279
dworker command 169
Dynamic Libraries page 380
dynamic library support limitations 385
dynamic linker 379
dynamic variable 280
dynamically linked program 378
dynamically loaded libraries 380
E
e command-line option 343
editor_launch_string
  variable 280
eliminating tab processing 101
elision, to simplify data representation 324
Emacs-like commands 269
en, see denable command
enable, see denable command
enabled action points list (ambiguous context) 28
enabled property 182
enabled variable 299
enabling action points 27, 28, 71
ending debugging session 170
ent command-line option 343
enum_values property 222
env command-line option 343
env variable 281
error message information 260
error state 260
errorCodes command 118, 187
errorCodes variable 187
errors, raising 187
expressions, compiling 270
expressions, type transformation 312
extensions for file names 293
expr -> expr 313
expr command 118, 189
expression property 182, 189
expression system
AIX 387
expression values, printing 118
expressions in breakpoint 42
expressions, compiling 270
expressions, type transformation 312
F
f, see dfocus command
f9x_demangler command-line option 344
fast_trap, setting 261, 293, 294
fatal errors 376
fg command-line option 344
File > Preferences command 317
file name extensions 293
files
initialization 311
libdbfork.h 369
filter definition file 298
filters, sharing memory filters 298
fixed_font variable 299
fixed_font_family variable 300
fixed_font_size variable 300
floating point data format
double-precision 272
extended floating point 273
single-precision 274
floating point status register
Intel IA-64 405
flush, see dflush command
flushing cache 46
flushing suspended evaluations 76
focus
see also dfocus command
default 79
defining 257
temporarily changing 79
focus_groups command 191
focus_processes command 192
focus_threads command 193
focus_threads property 189
follow_clone variable 281
font variable 300
fonts 299
fixed 299, 300
ui 300, 305
ui font family 306
ui font size 306
force_default_cplus_demangler variable 281
force_default_f9x_demangler variable 282
force_window_position variable 300
foreground command-line option 343
fork()
about 368
calling 341
catching 275
formatting program data using C++View 318
Fortran
demangler 271
frame_offset_x variable 300
frame_offset_y variable 300
functions
evaluating 118
setting breakpoints at 43
G
g, see dgo command
GDP SPU Runtime System (SPU). See spurs.
genral registers
Intel IA-64 401
gometry_call_tree variable 301
gometry_cli position 301
gometry_cli variable 301
gometry_globals variable 301
gometry_help variable 301
gometry_memory_stats variable 302
gometry_message_queue variable 302
gometry_message_queue_
graph variable 302
geometry_modules variable 302
geometry_process variable 302
geometry_ptset variable 303
geometry_root variable 303
geometry_thread_objects variable 303
geometry_variable variable 303
geometry_variable_stats variable 303
get subcommand 203
get verb
   actionpoint command 181, 185
   expr command 189
   group command 194
   process command 197
   thread command 219
   type command 222
Global Arrays 82
setting language for display 82
global_typenames variable 282
global_types, command-line option 344
GNU C++ STL instantiation 308
gnu_debuglink command-line option 344, 366, 367
gnu_debuglink files 366
gnu_debuglink variable 282, 366
gnu_debuglink_checksum command-line option 344, 366
gnu_debuglink_checksum variable 282
gnu_debuglink_checksum_flag variable 366
gnu_debuglink_global_directory variable 282
gnu_debuglink_global_directory command-line option 367
gnu_debuglink_global_directory variable 367
gnu_debuglink_search_path variable 283
go, see dgo command
goal breakpoint 144
GPU threads. See CUDA.
gr, see dgroups command
group command 194
group ID 260
group members, stopping flag 258
group of interest 144
GROUP variable 255
group width stepping behavior 143
groups
   accessing properties 194
   adding 86
   adding members 87
   creating 86
   deleting 86, 87
   intersecting 86
   listing 86
   naming 87
   placing processes in 35
   removing members 86
   returning list of 191
   setting properties 194
GROUPS variable 103, 256
groups, see dgroups command
hostname
   expansion 356
   for tvdsrv 353
   property 198
   replacement 357
HP Tru64 UNIX
   /proc file system 375
hp, see dhold command
ht, see dhold command
htp, see dhold command
I
I/O redirection 132
id property 182, 190, 195, 198, 220, 222, 226
ignore_control_c variable 284
ignore_snippets false 297
ignoring libraries by prefix 342
image browser window position 301
image file, stripped copy 366
Image information 106
image_id property 222
image_ids property 198
image_load_callbacks list 288
image_load_callbacks variable 284
in_setup variable 284
IndextermTEST 339
inet interface name 286, 345
infinite loop, see loop, infinite
info state 260
information on a name 162
initialization file 179, 311
initially_suspended_process property 190
input, blocking 158
inserting working threads 169
instructions, stepping 147
integer (64-bit) data format 274
integer data format
   16-bit 274
   32-bit 274
   8-bit 273
Intel IA-64
   architecture 401
current frame marker register 403
floating point status register 405
general registers 401
Intel IA-64 floating point registers 405
previous function state register 404
processor status register 402
register stack configuration register 404
Intel-x86 architecture 389, 407
floating-point registers 390, 408
FPCR register 391, 409
using 392, 410
FPSR register 392, 393, 410
general registers 389, 407
interface name for server 286, 345
interior pointers, checking 297
intersecting groups 86
ipv6_support command-line option 344
ipv6_support option 344
IRIX /proc file system 375

J
job_t::launch 375

K
k, see dkill command
-kcc_classes command-line option 345
kcc_classes command-line option 345
kcc_classes variable 284
keep_expressions variable 304
keep_search_dialog variable 304
kernel_launch_string variable 285
keys, remapping 386
keysym 386
kill_callbacks variable 285
kill, see dkill command
killing attached processes 270
killing processes 98

L
l, see dlist command
language property 182, 223, 226
languages C, using with C++View 329
lappend, see dlappend command
launch string
for editor 280
for server (Sun only) 286
for Visualizer 294
Launch Strings page 295
launching
local server 286
processes 132
tvdsrv 352
Visualizer 294
lb command-line option 345
LD_BIND_NOW environment variable 378
LD_LIBRARY_PATH 370
-ldbfork linker option 369
-ldbfork option 368, 369
-ldbfork_64 option 368, 369
leak_check_interior_pointers 29
7, 298
leak_check_interior_pointers variable 297, 298
leak_max_cache 298
leak_max_chunk 298
length property 182, 223
levels for dwhere 256
levels, moving down 69
libdbfork_64.a 369
libdbfork.a 368, 369
libdbfork.h file 369
libraries dbfork 341
ignoring by prefix 342
loading by suffix 278
loading symbols from shared libraries 280
not loading based on prefix 276
shared 378
library cache data 285
library cache, flushing 46
library_cache_directory variable 285, 286
line property 182
line_number 207
LINES_PER_SCREEN variable 256
linking
for C++View 334
linking to dbfork library 368
AIX 368
C++ and dbfork 368
SunOS 5 369
Linux swap space 376
list location 69
list_element_count_addressing_callback 227
list_element_data_addressing_callback 227
list_element_next_addressing_callback 227
list_element_prev_addressing_callback 228
list_end_value property 228
list_first_element_addressing_callback 228
list_head_addressing_callback 2
28
list, see dlist command
listing groups 86
using a regular expression 88
listing lines 257
lo, see dlload command
load and loadbind 380
load, see dlload command
loader_name 207
loading
action point information 29
action points 28, 345
libraries 276, 277
programs 104
shared libraries 185
symbols from shared libraries 280
tvd files 205
loading sessions
dsession 135
local_interface variable 286
local_server variable 286
local_server_launch_string variable 286
lockstep list element 255
logical model 309
lookup subcommand 203
lookup_keys subcommand 204
loop, infinite, see infinite loop
lower_bounds_callback 227
M
machine instructions, stepping 147
manager property 220
manager threads, running 143
managing shared libraries 185
mangler, overriding 342, 344
mappings, search path 258
MAX_LIST variable 100, 257
Maximum permissible rank field 295
mem_detect_leaks memory sub-option 243
mem_guard_blocks memory sub-option 243
mem_hoard_freed_memory memory sub-option 244
mem_paint_all memory sub-option 244
mem_paint_on_alloc memory sub-option 244
mem_paint_on dealloc memory sub-option 244
member_type property 195
member_type_values property 195
members argument, build struct transform 312
members property 195
memory
data size 106
heap 106
stack 106
text size 106
memory statistics window position 302
Memory Tracker, see dheap command
memory use 106
message queue graph window position 302
message_queue window position 302
message verbosity variable 260
message_queue command-line option 345, 346
message_queue variable 286
mkswap command 377
modules window position 302
more processing 119
more prompt 171, 256
mounting /proc file system 375
MPI message queues 286
MPI startup 417
mqd command-line option 346
multiprocess programs, attaching to processes 35
multithread safety and C++View 326
N
N upcast expr 313
n, see dnex command
name argument, build struct transform 312
name property 223, 226
name, information about 162
namespaces 136
TV:: 136
TV::GUI:: 136
using wildcards 136
Namespaces, Symbol 215
naming the host 353
nccq command-line option 341
nested subroutines, stepping out of 115
newval variable in watchpoints 159
next, see dnex command
nexti, see dnexi command
ni, see dnexi command
nil, see dnexi command
niw, see dnexi command
nl, see dnex command
nlb command-line option 345
no_ask_on_dlopen command-line option 381
no_compiler_vars command-line option 341
no_control_c_quick_shutdown command-line option 341
no_dbfork command-line option 341
no_dynamic command-line option 378
no_ent command-line option 343
no_global_types command-line option 344
no_gnu_debuglink command-line option 344
no_gnu_debuglink_checksum command-line option 344
no_ipv6_support command-line option 344
no_kcc_classes command-line option 345
no_message_queue command-line option 346
no_mqd command-line option 346
no_nptl_threads command-line option 347
no_parallel command-line option 347
no_startup_scripts command line option 347
no_team command-line option 350
no_templplus command-line option 350
no_user_threads command-line option 340, 351
nodeid property 199
nodes_allowed command-line option 354
nptl_threads command-line option 347
nptl_threads variable 287
NVIDIA. See CUDA
nw, see dnext command

O
OBJECT_SEARCH_MAPPINGS variable 255, 257
OBJECT_SEARCH_PATH variable 257
oldval variable in watchpoints 159
Open (or raise) process window at breakpoint checkbox 304
Open process window on error signal check box 304
open_cli_window_callback variable 287
opening shared libraries 66
option 354
options
tvdsvr
-callback 353
-seral 353
-server 353
-set_pw 353
-user_threads 351
ou, see dout command
oul, see dout command
out, see dout command
ouw, see dout command

P
p, see dprint command
p/t expressions 124
P/T set information 124
panes, width 292
parallel backtrace data, displaying 47
parallel command-line option 347
parallel jobs
displaying state of data using dcalltree command 47
parallel processes attaching to 34
displaying process and thread state using dcalltree command 47
parallel program runtime library support 287
parallel runtime libraries 287
parallel variable 287
parallel_attach variable 287
parallel_configs variable 418
parallel_stop variable 288
parallel_support.tvd file 418
passwords 355
checking 355
generated by tvdsvr 353
patch space 269
patch_area_base command-line option 348
patch_area_length command-line option 348
PATH environment variable for tvdsvr 353
pc property 220
picking up threads 281
pid command-line option 348
Plant in share group checkbox 258, 290
platform variable 288
plist, defined 140
pop_at_breakpoint variable 304
pop_on_error variable 304
popping Process Window on error variable 304
port 4142 354
port command-line option 354
port number 354
for tvdsvr 353
replacement 357
searching 354
ports on host 353
post_scope 204
post_symbol 204
PowerPC
architecture 394
Blue Gene registers 396
floating-point registers 398
FPSCR register 398
using the 400
FPSCR register, using 400
general registers 394
MSR register 397
pre_scope 204
pre_sym 204
preferences, setting defaults for 137
previous function state register
Intel IA-64 404
print, see dprint command
printing expression values 118
printing registers 121
printing slices 119
printing variable values 118
proc file system problems 375
Procedure Linkage Table (PLT) 378
Process > Startup command 84
process barrier breakpoint, see barrier breakpoint
process command 197
process counts, see ptlist
process groups, see groups
process information, saving 51
process list element 255
process objects, creating new 104
process statistics 106
process width stepping behavior 144
process window position 302
process_load_callbacks variable 288
process, attaching to existing CLI dattach command 34
continuing or halting execution on attach 34
process/thread sets, changing 79
processes
attaching to 34, 103
automatically acquiring 275
automatically attaching to 287
current status 139
destroyed when exiting CLI 170, 172
detaching from 62
holding 96
killing 98
properties 197
releasing 150
releasing control 62
restarting 127, 132
returning list of 192
starting 127, 132
terminating 98
processor status register
processor status register Intel IA-64 402
program control groups, placing processes in 35
program stepping 143
program variable, changing value 32, 51, 96, 115, 130, 150, 153
programs, loading 104
PROMPT variable 257
prompting when screen is full 119
properties
address 181
clusterid 197
continuation_sig 219
count 195
done 189
dpid 220
duid 198, 220
enabled 182
enum_values 222
executable 198
expression 182, 189
focus_threads 189
heap_size 198
held 198, 220
hostname 198
id 182, 190, 195, 198, 220, 222
image_id 222
image_ids 198
initially_suspended_process 190
language 182, 223
length 182, 223
line 182
manager 220
member_type 195
member_type_values 195
members 195
name 223
nodeid 199
pc 220
prototype 223
rank 223
result 190
satisfaction_group 182
share 182
sp 220
stack_size 199
stack_vm_size 199
state 199, 220
state_values 199, 220
status 190
stop_when_done 182
stop_when_hit 183
struct_fields 223
symbol 207
syspid 199
systid 220
text_size 200
threadcount 200
threads 200
type 195
type_values 183, 195
vm_size 200
properties verb
actionpoint command 181, 185
expr command 189
group command 194
process command 197
thread command 219
type command 222
prototype property 223
ptlist
and dcalltree 48
and dstatus 139
and dwhere group-by property 166
defined 140
defined in 142
PTSET variable 257
ptsets, see dptsets
Q
quonopath command-line option 362
qualifying symbol names 100
quit command 172
quotation marks 32
R
r command-line option 348
r, see drun command
raising errors 187
rank property 223
ranks, attaching to 34
raw memory display 73
read_delayed subcommand 206
read_symbols command 201
reading action points file 28
reading symbols 201, 264, 276, 277
rebind subcommand 206
reenabling action points 71
register stack configuration register
Intel IA-64 404
registers
Blue Gene
PowerPC 396
BlueGene
PowerPC 396
floating-point
Intel-x86 390, 408
PowerPC 398
SPARC 413
general
Intel-x86 389, 407
PowerPC 394
SPARC 412
Intel-x86 FPCR 391, 409
using the 392, 410
Intel-x86 FPSR 392, 393, 410
Power FPCR 398
Power MSR 397
PowerPC FPSR 398
using 400
PowerPC FPCR, using 400
PowerPC MSR 397
printing 121
SPARC FPSR 414
SPARC FPCR, using 415
SPARC PSA 413
registers, using in evaluations 43
regular expressions within name argument 312
release 254
releasing control 62
releasing processes and threads 37, 150
remapping keys 386
remote command-line option 348
remote debugging, tvdsvr command syntax 352
remote systems, debugging 46
removing
aliases 179
group member 86
variables 152
worker threads 169
replacement characters 356
replacing tabs with spaces 259
replay, see dhistory command
rerun, see drerun command
resolve_final subcommand 206
resolve_next subcommand 206
respond 202
restart_threshold variable 289
restart, see drestart command
restarting processes 127, 132
restoring variables to default values 152
result property 190
resuming execution 53, 84, 98
returning error information 187
root path 259
of TotalView 259
Root Window position 303
routines, stepping out of 115
rr, see drerun command
RS/6000, compiling on 361
RTLD_GLOBAL 66
RTLD_LAZY 66, 67
RTLD_LOCAL 66
RTLD_NOW 66, 67
run, see drun command
running to an address 153
S
s command-line option 348
s, see dstep command
satisfaction set 38, 253, 265
satisfaction_group property 182
save_window_pipe_or_filename variable 289
saved position
Call Tree Window 301
CLI Window 301
Help Window 301
Image Browser Window 301
Memory Statistics Window 302
Message Queue Graph Window 302
Message Queue window 302
Modules Window 302
Process Window 302
Root Window 303
Thread Objects Window 303
Variable Window 303
saving action point
information 29
saving action points 28
saving process information 51
scope 207
scope command 203
scope of action point 258
screen size 256
search dialog, remaining displayed 304
search path 255
mappings 258
setting 255, 257, 258
search_case_sensitive variable 290
search_path command-line option 349
search_port command-line option 354
searching
case sensitive 290
wrapping 295
serial command-line option 349, 353, 354
serial line connection 354
server command-line option 353, 354
server launch command 356
server_launch_enabled variable 290
server_launch_timeout variable 290
server_response_wait_timeout variable 290
servers, number of 357
sessions
dsession 135
set verb
actionpoint command 181
group command 194
process command 197
thread command 219
type command 222
set_pw command-line option 353, 355
set_pws command-line option 355
set, see dset command
setting default preferences 137
setting lines between more prompts 256
setting terminal properties 178
setting variables 136
SGROUP variable 257
share groups, share group variable 257
share list element 256
share property 182
SHARE_ACTION_POINT variable 258
share_action_point variable 290
share_in_group flag 258
shared libraries 378
closing 185
deferred reading 201
information about 185
loading symbols from 280
managing 185
manually loading 66
reading deferred symbols 201
reading symbols 264, 276, 277
shared_data_filters 298
shm command-line option 349
show_startup_parameters 305
show_sys_thread_id variable 305
showing current status 139
showing Fortran compiler variables 270
si, see dstepi command
SIGINT 284
signal_handling_mode command-line option 349
signal_handling_mode variable 290
signals, handling in TotalView 349
sil, see dstepi command
SILENT state 260
single process server launch 290
single_click_dive_enabled variable 305
siw, see dstepi command
sl, see dstep command
slices, printing 119
SLURM, control_c_quick_shutdown variable 270, 341
source code, displaying 100
source_pane_tab_width variable 292
source_process_startup 288
source_process_startup command 205
SOURCE_SEARCH_MAPPINGS variable 258
SOURCE_SEARCH_PATH variable 258
sourcing tvd files 205
sp property 220
spaces simulating tabs 259
SPARC architecture 412
floating-point registers 413
FPSR register 414
using 415
general registers 412
PSR register 413
spell_correction variable 292
spurs
support for 22
st, see dstatus command
stack frame 100
moving down through 69
see also call stack 166
stack frame, see also call stack 16
stack memory 106
stack movements 156
stack_size property 199
stack_trace_qualification_level variable 292
stack_vm size 107
stack_vm_size property 199
stack, unwinding 76
starting a process 127, 132
startup command 84
startup file 311
start-up file, tvdinit.tvd 23
startup options
-no_startup_scripts 347
state property 199, 220
state_values property 199, 220
statistical array data, gathering 118
status of P/T sets 124
status property 190
status, see dstatus command
stderr 275
stderr command-line option 350
stderr redirection 132
stderr_append command-line option 350
stderr_is_stdout command-line option 350
stderr 275
stdin command-line option 350
stdin redirection 132
stdin 275
stdout command-line option 350
stdout redirection 132
stdout_append command-line option 350
step, see dstep command
stepi, see dstepi command
stepping
group width behavior 143
machine instructions 112, 147
process width behavior 144
see also dnext command, dnexti command, dstep command, and dstepi command 109
thread width behavior 144
warning when exception thrown 295
STL instantiation 308
stop_group breakpoint 43
stop_all property 265
STOP_ALL variable 42, 253, 258
stop_all variable 292
stop_group flag 258
stop_relatives_on_proc_error variable 292
stop_when_done command-line option 253
stop_when_done property 182, 265
stop_when_hit property 183
stopped process, responding to resume commands 38
stopping execution 91
stopping the control group 292
string length format 274
strings, assigning values to 32
stripped copy 366
struct_fields property 223
structure definitions in KCC 284
structure transformations 310
structures, transforming 312
stty command 178
suffixes variable 293
SunOS 5
/proc file system 375
key remapping 386
linking to dbfork library 369
swap space 377
sw, see dstep command
swap command 377
swap space 376
AIX 376
Linux 376
SunOS 377
swapon command 377
symbol command 206
symbol name qualification 100
symbols
namespaces 215
properties 207
reading 276, 277
symbols, interpreting 32
system variables, see CLI variables
systid property 220
T
tab processing 101
TAB_WIDTH variable 101, 259
tabs, replacing with spaces 259
target processes 91
terminating 98
target property 223
TCL library component search path 259
team command-line option 350
teamplus command-line option 350
templates
using with C++View 321
temporarily changing focus 79
terminating properties, setting 178
terminating debugging session 170
terminating processes 98
text size 106
text_size property 200
thread barrier breakpoint, see barrier breakpoint
thread command 219
thread counts, see ptlist
thread groups, see groups
thread list element 256
thread objects window position 303
thread of interest 144, 153
thread state display via ptlist 140
thread width stepping behavior 144
threadcount property 200
threads
barriers 39
creating 84
current status 139
destroyed when exiting CLI 170, 172
getting properties 219
holding 38, 96
list variable 259
picking up 281
property 200
releasing 150
returning list of 193
setting properties 219
THREADS variable 259
timeplus command-line option 350
toolbar_style variable 305
Tools > Command Line command 287
Tools > Dynamic Libraries command 66
Tools > Evaluate window 76
tooltips_enabled variable 305
totalview command 338
options 339
synopsis 338
syntax and use 338
TotalView Debugger Server 34
TotalView executable 259
TotalView GUI version 306
TotalView version 294
totalview_jobid variable 358
TOTALVIEW_ROOT_PATH variable 259
TOTALVIEW_TCLLIB_PATH command 317
TOTALVIEW_TCLLIB_PATH variable 259
TOTALVIEW_VERSION variable 259
TotalView.breakpoints file 29
totalview/lib_cache subdirectory 46
transformations
of types using C++View 318
using type 316
why type 308
transforming classes 312
transforming structures 310, 312
triggering breakpoints 43
troubleshooting 5
ttf variable 293
tv_data_display.h, and API for using C++View 319
TV_ttf_display_type function with C++View 328
TV_ttf_display_type function, writing for C++View 319
TV:: namespace 136
TV::actionpoint command 181
TV::dll command 67, 185
TV::dll_read_loader_symbols_only variable 201
TV::dll_read_no_symbols variable 201
TV::errorCodes command 187
TV::expr command 118, 189
TV::focus_groups command 191
TV::focus_processes command 192
TV::focus_threads command 193
TV::group command 194
TV::GUI:: namespace 136
TV::hex2dec command 196
TV::process command 197
TV::read_symbols command 201
TV::respond command 202
TV::scope command 203
TV::source_process_startup command 205
TV::symbol command 206
TV::thread command 219
TV::ttf variable 317
TV::type command 222
TV::type_transformation command 225
tvd files 205
TVD.breakpoints file 264
tvinit.tvd start-up file 23, 179
tvdsrv command 352, 353, 356
description 353
options 353
password 353
PATH environment variable 353
synopsis 353
tvdsrv.conf 354
TVDSVRLAUNCHCMD environment variable 356
tvhome command-line option 350
tvscript 235
action point API 246
command syntax 236
create_actionpoint command-line option 240
display_specifiers command-line option 243
event actions 241
Event API 247
event_action command-line option 240
event_action event types 240
example 244
example script file 248
external script files 246
logging functions API 246
maxruntime memory sub-option 244
memory debugging command-line options 243
memory debugging command-line sub-options 243
MPI programs 235
options 240
process functions API 246
script_file command-line option 244
script_log_filename command-line option 244
script_summary_log_filename command-line option 244
ui_font_size variable 306
ui_font variable 305
ui_font_family variable 306
ui_font_variable 306
un, see duntil command
unalias command 179
unconditional watchpoints 159
undefined symbols 66
unhold, see dunhold command
unl, see duntil command
unset, see dunset command
until, see duntil command
unw, see duntil command
unwinding the stack 76
up, see dup command
upper_bounds_callback 227
user_threads command-line option 340, 351
user_threads variable 294
user-defined commands 23
user-level (M:N) thread packages 294
using quotation marks 32
using type transformations 316
using_color variable 306
using_text_color variable 306
using_title_color variable 306
V
validate_callback 226
value for newly created action points 258
values, printing 118
Variable Window position 303
variables
assigning command output to 25
casting 121
changing values 32, 51, 96, 115, 130, 150, 153
default value for 135, 136
printing 118
removing 152
setting 136
watched 160
watching 159
vector transformation 308
VERBOSE variable 260
verbosity command-line

V
option 351, 355
verbosity setting replacement character 358
version variable 294, 306
version, TotalView 259
target()
calling 341
catching 275
viewing CLI variables 135, 136
visualizer_launch_enabled variable 294
visualizer_launch_string variable 294
visualizer_max_rank variable 295
vm_size property 200
vm_size size 107

W
w, see dwhere command
wa, see dwatch command
wait, see dwait command
walk subcommand 204
warn_step_throw variable 295
warning state 260
watch, see dwatch command
watchpoints 159
$newval 159
$oldval 159
conditional 159
information not saved 29
length of 160
supported systems 160
-WG,-cmpo=i option 362
WGROUP variable 260
wh, see dwhat command
what, see dwhat command
When barrier done, stop value 253
When barrier hit, stop value 253
where, see dwhere command
why type transformations 308
window position, forcing 300
worker group list variable 260
worker_list element 256
worker threads 260
inserting 169
removing 169