# Book Overview

## Part I - CLI Commands

1. CLI Command Summary ........................................................................... 3
2. CLI Commands .......................................................................................... 13
3. CLI Namespace Commands ......................................................................... 129
4. Batch Debugging Using tvscript ................................................................. 173
5. TotalView Variables .................................................................................... 185
6. Creating Type Transformations ..................................................................... 229

## Part II - Running TotalView

7. TotalView Command Syntax ....................................................................... 241
8. TotalView Debugger Server Command Syntax ........................................... 253

## Part III - Platforms and Operating Systems

9. Platforms and Compilers ........................................................................... 263
10. Operating Systems ..................................................................................... 275
11. Architectures ............................................................................................ 287

## Part IV - Appendix

A. MPI Startup ............................................................................................ 319
About This Book
TotalView Family Differences ............................................................. xi
How To Use This Book ...................................................................... xii
Conventions ..................................................................................... xiii
TotalView Documentation ............................................................... xiii

Part I: CLI Commands

1 CLI Command Summary

2 CLI Commands
Command Overview .......................................................................... 13
alias ................................................................................................... 16
capture .............................................................................................. 18
dactions ............................................................................................ 19
dassign ............................................................................................. 22
dattach .............................................................................................. 24
dbarrier ............................................................................................ 27
dbreak ............................................................................................... 31
dcache .............................................................................................. 34
dcheckpoint ...................................................................................... 35
dcont ................................................................................................. 38
ddelete .............................................................................................. 39
ddetach .............................................................................................. 40
ddisable ........................................................................................... 41
ddlopen ............................................................................................. 42
ddown ............................................................................................... 45
denable ............................................................................................ 46
dexamine ......................................................................................... 47
dflush ............................................................................................... 49
dfocus .............................................................................................. 52
dga ................................................................................................. 54
dgo ................................................................................................. 56
dgroups ............................................................................................ 57
3 CLI Namespace Commands

Command Overview ................................................................. 129
actionpoint .............................................................................. 131
dec2hex .................................................................................... 134
dll ............................................................................................. 135
errorCodes ............................................................................... 136
Part III: Platforms and Operating Systems

9 Platforms and Compilers

Compiling with Debugging Symbols ........................................... 263
Apple Running Mac OS X .......................................................... 264
HP Tru64 UNIX ........................................................................ 264
HP-UX for IA-64 ...................................................................... 264
IBM AIX on RS/6000 Systems .................................................. 265
IBM Bluegene ......................................................................... 266
IBM Power Linux ..................................................................... 266
Linux Running on an x86 Platform .......................................... 266
Linux Running on an x86-64 Platform ..................................... 267
Linux Running on an Itanium Platform ................................... 267
SGI IRIX-MIPS Systems .......................................................... 267
Sun Solaris ............................................................................. 268
Using gnu_debuglink Files ......................................................... 268
Total View Command-Line Options and CLI State Variables ... 269
Searching for the gnu_debug_link File .................................... 269
Using Exception Data on Tru64 UNIX .................................... 270
Linking with the dbfork Library ............................................... 271
Linking with dbfork and HP Tru64 UNIX ................................. 271
Linking with HP-UX ................................................................ 271
dbfork on IBM AIX on RS/6000 Systems ................................. 272
Linking C++ Programs with dbfork ........................................ 272
Linux or Mac OS X .................................................................. 273
SGI IRIX6-MIPS ....................................................................... 273
SunOS 5 SPARC ...................................................................... 274

10 Operating Systems

Supported Operating Systems .................................................... 275
Mounting the /proc File System ............................................... 276
Mounting /proc HP Tru64 UNIX and SunOS 5 ......................... 276
Mounting proc SGI IRIX ......................................................... 277
Swap Space ............................................................................. 277
Swap Space on HP Tru64 UNIX ............................................... 277
Swap Space on HP HP-UX ....................................................... 277
  Maximum Data Size .............................................................. 278
Swap Space on IBM AIX .......................................................... 278
Swap Space on Linux ............................................................. 279
Swap Space on SGI IRIX .......................................................... 279
Swap Space on SunOS 5 ......................................................... 279
Shared Libraries .................................................................... 280
  Changing Linkage Table Entries and LD_BIND_NOW ............ 281
  Using Shared Libraries on HP-UX ........................................ 281
Debugging Your Program’s Dynamically Loaded Libraries ....... 282
  Known Limitations .............................................................. 284
Remapping Keys ..................................................................... 284
Expression System ................................................................. 284
  Expression System on HP Alpha Tru64 UNIX ....................... 284
  Expression System on IBM AIX ............................................ 284
11 Architectures

AMD and Intel x86-64 .......................................................... 287
  x86-64 General Registers ................................................. 288
  x86-64 Floating-Point Registers ....................................... 288
  x86-64 FPCR Register ..................................................... 289
    Using the x86-64 FPCR Register ................................... 290
  x86-64 FPSR Register ..................................................... 290
  x86-64 MXCSR Register .................................................. 291

HP Alpha .............................................................................. 291
  Alpha General Registers .................................................. 291
    Alpha Floating-Point Registers ...................................... 292
    Alpha FPCR Register .................................................... 293

HP PA-RISC ........................................................................... 293
  PA-RISC General Registers ............................................. 293
  PA-RISC Process Status Word ........................................ 294
  PA-RISC Floating-Point Registers .................................... 295
  PA-RISC Floating-Point Format ....................................... 296

IBM Cell Broadband Engine ................................................ 296

Power Architectures .......................................................... 297
  Power General Registers .................................................. 297
  Power MSR Register ....................................................... 298
  Power Floating-Point Registers ....................................... 299
  Power FPSCR Register .................................................... 299
    Using the Power FPSCR Register ................................... 300

Intel IA-64 ............................................................................. 301
  Intel IA-64 General Registers .......................................... 301
  IA-64 Processor Status Register Fields (PSR) ....................... 302
  Current Frame Marker Register Fields (CFM) ...................... 303
  Register Stack Configuration Register Fields (RSC) .............. 303
  Previous Function State Register Fields (PFS) ...................... 303
  Floating Point Registers .................................................. 304
  Floating Point Status Register Fields ................................ 304

Intel x86 ................................................................................. 305
  Intel x86 General Registers .............................................. 305
  Intel x86 Floating-Point Registers ..................................... 306
  Intel x86 FPCR Register ................................................... 306
    Using the Intel x86 FPCR Register .................................. 307
  Intel x86 FPSR Register ................................................... 307
  Intel x86 MXCSR Register ................................................ 308

SGI MIPS ............................................................................... 308
  MIPS General Registers .................................................. 309
  MIPS SR Register ........................................................... 310
  MIPS Floating-Point Registers ....................................... 310
  MIPS FCSR Register ....................................................... 311
    Using the MIPS FCSR Register ...................................... 312
  MIPS Delay Slot Instructions ......................................... 312

Sun SPARC ........................................................................... 313
  SPARC General Registers ............................................... 313
  SPARC PSR Register ....................................................... 314
About This Book

This document is the reference guide for TotalView® Debugger. Unlike the TotalView Users Guide which presented GUI and CLI information together, the chapters in this book either are devoted to one interface or contain information that pertains to both.

TotalView Family Differences

This manual describes the TotalView Enterprise, TotalView Team, and TotalView Individual debuggers. Each of these allows you to use the CLI debugger as well. In all cases, TotalView Enterprise and TotalView Team have the same features. They differ in the way they are licensed. However, TotalView Individual is slightly different.

The most fundamental differences between TotalView Team and TotalView Enterprise 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 2 programmers, each debugging a 32 process job. In contrast, a TotalView Enterprise 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; Enterprise</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 Enterprise 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 Enterprise 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 computes 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>Programers 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>

---

### How To Use This Book

The information in this book is in three parts.

- **CLI Commands**
  This part contains descriptions of all the CLI commands, the variables that you can set using the CLI, and other CLI-related information.

- **Running TotalView**
  TotalView and the TotalView Debugger Server (`tvdsvr`) can accept many command-line options. This part describes these options.

- **Platforms and Operating Systems**
  Although the way in which you use TotalView is the same from system to system and from environment to environment, these systems and envi-
environments place some constraints on what you must do, and require that you compile programs differently on the various UNIX platforms. This part describes these differences.

## 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 parts of a command that are optional.</td>
</tr>
<tr>
<td><em>arguments</em></td>
<td>In a command description, text in italics represents information you type. Elsewhere, italics is used for emphasis.</td>
</tr>
<tr>
<td><em>Dark text</em></td>
<td>In a command description, dark text represents keywords or options that you must type exactly as displayed. Elsewhere, it represents words that are used in a programmatic way rather than their normal way.</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>In program listings, this indicates that you are seeing a program or something you’d type in response to a shell or CLI prompt. If this text is in bold, it’s indicating that what you’re seeing is what you’ll be typing. If you’re viewing this information online, example text is in color.</td>
</tr>
</tbody>
</table>

This graphic symbol indicates that the information that follows—which is printed in italics—is a note. This information is an important qualifier to what you just read.

## TotalView Documentation

The following table describes other TotalView documentation:

<table>
<thead>
<tr>
<th>Title</th>
<th>Contents</th>
<th>Online</th>
<th>HTML</th>
<th>PDF</th>
<th>Print</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluating TotalView</td>
<td>Brochure that leads you to basic TotalView features.</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>❌</td>
</tr>
<tr>
<td>TotalView Users Guide</td>
<td>Describes how to use the TotalView GUI and the CLI; this is the most used of all the TotalView books.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>TotalView New Features</td>
<td>Describes new features added to TotalView.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Please contact us if you have problems installing TotalView, questions that are not answered in the product documentation or on our Web site, or suggestions for new features or improvements.

Our Internet email address for support issues is:

support@totalviewtech.com

For documentation issues, the address is:

documentation@totalviewtech.com

Our phone numbers are:

1-800-856-3766 in the United States
(+1) 508-652-7700 worldwide

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

- The version of TotalView and the platform on which you are running TotalView.
- An example that illustrates the problem.
- A record of the sequence of events that led to the problem.
This part of the TotalView Reference Guide contains five chapters that describe the TotalView Command Line Interpreter (CLI).

Chapter 1: CLI Command Summary
Summarizes all the CLI commands.

Chapter 2: CLI Commands
Contains detailed descriptions of the CLI commands that are found 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
Contains descriptions of 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
Shows how you create batch scripts that run TotalView unattended.

Chapter 5: TotalView Variables
Describes all TotalView variables, including those that you use 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
If you do not wish to see all the members of a class or structure or if you would like to alter the way TotalView displays these elements, you can call a CLI routine that tells TotalView how you want it to display information. This chapter tells how to create these CLI routines.
This chapter contains a summary of all TotalView® Debugger CLI commands. The commands are described in Chapter 2, “CLI Commands,” on page 13 and Chapter 3, “CLI Namespace Commands,” on page 129.

**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 defn-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 ]`
- 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 CLI control
```bash
dattach [ -g gid ] [ -r fname ]
[ -replay | -no_replay ]
[ -ask_attach_parallel | -no_attach_parallel ]
[ -c corefile-name ] [ -rank num ]
[ -e ] fname pid-list
[ -parallel_attach_subset subset_specification ]
```

**dbarrier**
Creates a barrier breakpoint at a source location
```bash
dbarrier breakpoint-expr [ -stop_when_hit { group | process | none } ]
[ -stop_when_done { group | process | none } ] [ -pending ]
```
Creates a barrier breakpoint at an address
```bash
dbarrier –address addr
[ -stop_when_hit { group | process | none } ]
[ -stop_when_done { group | process | none } ] [ -pending ]
```

**dbreak**
Creates a breakpoint at a source location
```bash
dbreak breakpoint-expr [ -p | -g | -t ] [ [ -l lang ] –e expr ] [ –pending ]
```
Creates a breakpoint at an address
```bash
dbreak –address addr [ -p | -g | -t ] [ [ -l lang ] –e expr ] [ –pending ]
```

**dcache**
Clears the remote library cache
```bash
dcache –flush
```

**dcheckpoint**
Creates a checkpoint on SGI IRIX
```bash
dcheckpoint [ after_checkpointing ] [ -by process_set ] [ –no_park ]
[ -ask_attach_parallel | -no_attach_parallel ]
[ -no_preserve_ids ] [ –force ] checkpoint-name
```
Creates a checkpoint on IBM AIX
```bash
dcheckpoint [ –delete | –halt ]
```

**dcont**
Continues execution and waits for execution to stop
```bash
dcont
```

**ddelete**
Deletes some action points
```bash
ddelete action-point-list
```
Deletes all action points
```bash
ddelete –a
```

**ddetach**
Detaches from the processes
```bash
ddetach
```
**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

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

**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

\texttt{dheap –info [ –backtrace ] [ start\_address [ end\_address ] ]}

Indicates whether an address is within a deallocated block

\texttt{dheap –is\_dangling address}

Locates memory leaks

\texttt{dheap –leaks [ –check\_interior ]}

Enables or disables Memory Debugger event notification

\texttt{dheap –[no]notify}

Paints memory with a distinct pattern

\texttt{dheap –paint [ subcommands ]}

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

\texttt{dheap –red\_zones [ subcommands ]}

Enables and disables allocation and reallocation notification

\texttt{dheap –tag\_alloc subcommand start\_address [ end\_address ]}

Displays the Memory Debugger’s version number

\texttt{dheap –version}

\texttt{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.

\texttt{dhistory [ –info ] [ –get\_time ] [ –go\_time time ] [ –go\_live [ –enable [ –disable ]]]}

\texttt{dhold}

Holds processes

\texttt{dhold –process}

Holds threads

\texttt{dhold –thread}

\texttt{dkill}

Terminates execution of target processes

\texttt{dkill [ –remove ]}

\texttt{dlappend}

Appends list elements to a TotalView variable

\texttt{dlappend variable-name value [ ... ]}

\texttt{dlist}

Displays code relative to the current list location

\texttt{dlist [ –n num-lines ]}

Displays code relative to a named location

\texttt{dlist breakpoint-expr [ –n num-lines ]}

Displays code relative to the current execution location

\texttt{dlist –e [ –n num-lines ]}
dll
Manages shared libraries
   TV::dll action [ dll-id-list ] [ –all ]

dload
Loads debugging information
   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
   dmstat

dnext
Steps source lines, stepping over subroutines
   dnext [ –back ] [ num-steps ]

dnexti
Steps machine instructions, stepping over subroutines
   dnexti [ –back ] [ num-steps ]

dout
Executes until just after the place that called the current routine
   dout [ –back ] [ frame-count ]

dprint
Prints the value of a variable or expression
   dprint [ –nowait ] [ –slice slice_expr ] variable_or_expression

dptsets
Shows the status of processes and threads in an array of P/T expressions
   dptsets [ ptset_array ] ...

drerun
Restarts processes
   drerun [ cmd_arguments ] [ < infile ]
      [ > [ > ] [ & ] outfile ]
      [ 2> [ > ] errfile ]

drestart
Restarts a checkpoint on AIX
   drestart [ –halt ] [ –g gid ] [ –r host ] [ –no_same_hosts ]
   Restarts a checkpoint on SGI
   drestart [ process-state ] [ –no_unpark ] [ –g gid ] [ –r host ]
      [ –ask_attach_parallel | –no_attach_parallel ]
      [ –no_preserve_ids ] checkpoint-name

drun
Starts or restarts processes
   drun [ cmd_arguments ] [ < infile ]
      [ > [ > ] [ & ] outfile ]
      [ 2> [ > ] errfile ]
**dset**
Creates or changes a CLI state variable
**dset** *debugger-var value*
Views current CLI state variables
**dset** [ *debugger-var* ]
Sets the default for a CLI state variable
**dset** -set_as_default *debugger-var* *value*

**dstatus**
Shows current status of processes and threads
**dstatus**

**dstep**
Steps lines, stepping into subfunctions
**dstep** [ –back ] [ *num-steps* ]

**dstepi**
Steps machine instructions, stepping into subfunctions
**dstepi** [ –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

```
dwatch –address addr –length byte-count [ –p | –g | –t ]
[ [ –l lang ] –e expr ] [ –t type ]
```

**dwhat**
Determines what a name refers to

```
dwhat symbol-name
```

**dwhere**
Displays locations in the call stack

```
dwhere [ –level level-num ] [ num-levels ] [ –args ] [ –locals ] [ –registers ]
[ –noshow_pc ] [ –noshow_fp ] [ –show_image ]
```

Displays all locations in the call stack

```
dwhere –all [ –args ] [ –locals ] [ –registers ]
[ –noshow_pc ] [ –noshow_fp ] [ –show_image ]
```

**dworker**
Adds or removes a thread from a workers group

```
dworker { number | boolean }
```

**errorCode**
Returns a list of all error code tags

```
TV::errorCode
```

Returns or raises error information

```
TV::errorCode number_or_tag [ –raise [ message ] ]
```

**exit**
Terminates the debugging session

```
exit [ –force ]
```

**expr**
Manipulates values created by `dprint –nowait`

```
TV::expr action [ susp-eval-id ] [ other-args ]
```

**focus_groups**
Returns a list of groups in the current focus

```
TV::focus_groups
```

**focus_processes**
Returns a list of processes in the current focus

```
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
Sets and gets 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

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
```
This chapter contains detailed descriptions of CLI commands.

## Command Overview

This section lists all of the CLI commands. It also contains a short explanation of what each command does.

### General CLI Commands

The CLI commands in this group provide information on the general CLI operating environment:

- **alias**: Creates or views pseudonym for commands and arguments.
- **capture**: Allows commands that print information to instead send their output to a variable.
- **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.
Command Overview

- **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.
- **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 allow you to browse the program’s source files:

- **ddown**: Navigates through the call stack by manipulating the current frame.
- **dexamine**: Display 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 lets you work 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.
- **dstepi**: 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.

### 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 name you specify with text that you define. This text can contain one or more commands. After you create an alias, you can use it in the same way as a native TotalView or Tcl command. In addition, you can include an alias as part of a 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 only specify an `alias-name` argument, the CLI displays the definition of the alias.

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

The TotalView global startup file, `tvdinit.tvd`, defines a set of default aliases. All the common commands have one- or two-letter aliases. (You can obtain a list of these commands by typing `alias`—being sure not to use an 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”
}
```

The CLI does not parse `defn-body` (the command’s definition) until it is used. Therefore, you can create aliases that are nonsensical or incorrect. The CLI only detects errors when it tries to execute your alias.

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

**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 someone uses this alias.

alias step2 {s ; s}
Creates an alias that performs the same operations as the one in the previous example. It differs 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.
capture

Returns a command’s output as a string

**Format:**
capture [ –out | –err | –both ] [ –f filename ] command

**Arguments:**
- `–out` Tells the CLI to only capture output that is sent to `stdout`. This option is the default.
- `–err` Tells the CLI to send the output it captures to `stderr`.
- `–both` Tells the CLI to send the output it captures to `stdout` and `stderr`.
- `–f filename` Tells the CLI to send the output it captures to `filename`.
- `command` The CLI command (or commands) whose output is being captured. If you are specifying more than one command, you must enclose them within braces `{ }`.

**Description:** The `capture` command executes `command`, capturing all output that would normally go to the console into a string. After `command` completes, it returns the string. This command is analogous to the UNIX shell’s back-tick feature (``command``). The `capture` command lets you obtain 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 (0xfff8)`
  - `int2_array(2) = -6 (0xfffa)`
  - `int2_array(3) = -4 (0xfffc)`
  - `int2_array(4) = -2 (0xfffe)`
  Because the `capture` command records all of the 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.
dactions

Displays information, and saves and reloads action points

Format:

- `dactions [ ap-id-list ] [ –at source-loc ] [ –enabled | –disabled ]`
  Saves action points to a file.
- `dactions –save [ filename ]`
  Loads previously saved action points
- `dactions –load [ filename ]`

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.
  
  If you omit 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 just displays summary information for each.

- `–at source-loc` Displays the action points at `source-loc`.
- `–enabled` Shows only enabled action points.
- `–disabled` Shows only disabled action points.
- `–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.

This command also lets you obtain the action point identifier. You will need to use this identifier when you 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.
The –save option tells TotalView to write action point information to a file so that either you or TotalView can restore your action points later. The –load option tells TotalView to immediately read the saved file. If you use the file-name argument with either of these options, TotalView either writes to or reads from this file. If you do not use this argument, TotalView uses a file named program_name.TVD.v3breakpoints where program_name is the name of your program. TotalView writes this file into the directory in which your program resides.

The information saved includes expression information 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.

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 online Help.

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:
  
  1 addr=0x100012a8 [arrays.F#56] Enabled
  
  3 addr=0x100012c0 [arrays.F#57] Enabled

  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.
# 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. 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 Chapter 13 of the `TotalView Users Guide`.)

You need to know the following about 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.

**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.
dattach

Brings currently executing processes under CLI control

**Format:**

```
dattach [-g gid] [-r hname]
    [-ask_attach_parallel | -no_attach_parallel]
    [-replay | -no_replay]
    [-c corefile-name] [-rank num]
    [-e] filename [pid-list]
    [-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 188 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 there. See the Setting Up Parallel Debugging Sessions chapter of the TotalView Users 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**
  Asks whether TotalView should attach to parallel processes of a parallel job. The default is to automatically attach to processes. For additional information, see the Dynamic Libraries Page in the File > Preferences Dialog Box in the online Help.

- **-no_attach_parallel**
  Do not attach to any additional parallel processes in a parallel job. For additional information, see the Dynamic Libraries Page in the File > Preferences Dialog Box in the online Help.

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

- **-c corefile-name**
  Tells the CLI that it should load the core file named in the argument that follows. If you use this option, you must also specify a file name (`filename`).

- **-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 a file name. You need to use this argument if the file name begins with a dash (`-`) or only uses numeric characters.

- **filename**
  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.

**parallel_attach_subset** `subset_specification`

Defines a list of MPI ranks to be attached to when an MPI job is created or attached to. The list is space-separated; each element can have one of 3 forms:

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

A rank specification can be either a number or "$\text{max}\)" which represents the last rank in the MPI job.

For example:

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

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

**Description:**

The `dattach` command tells the CLI to attach to one or more processes, making it possible to continue process execution under CLI control.

**TotalView Individual:** You can only attach 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 `filename` 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.
dattach

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 CLI control</td>
</tr>
</tbody>
</table>

Examples:

**dattach mysys 10020**

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

**dattach –e 123 10020**

Loads file 123 and brings the process known to the run-time system by PID 10020 under CLI 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 CLI 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 CLI 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. It then attaches to *slave* running on the *otherhost* host and inserts them both in the same control group.
**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. In most cases, you will be typing a line number or a path-name containing a file name, function name, and line number, each separated by `#` characters (for example, `#file#line`). If you omit parts of this specification, the CLI creates them for you. For more information, see “Qualifying Symbol Names” in Chapter 13 of the TotalView Users Guide.

  For more information on breakpoint expressions, see “Breakpoint Expressions” on page 32.

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

- **–stop_when_hit width**
  
  Tells the CLI what else to stop when it stops the thread that arrives at a barrier point.

  If you do not use this option, the value of the BARRIER_STOP_ALL variable indicates what TotalView stops.

  This command’s width argument indicates what else TotalView stops. You can enter 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 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, the CLI releases all processes and threads held at the barrier. (Released means that these threads and processes can run.) Setting this option tells the CLI to stop additional threads contained in the same group or process.

  If you do not use this option, the value of the BARRIER_STOP WHEN DONE variable indicates what else TotalView stops.
The width argument indicates what else is stopped. You can enter one of the following three values:

- **group**: Stops the entire control group when the barrier is satisfied.
- **process**: Stops the processes that contain threads in the satisfaction set when the barrier is satisfied.
- **none**: Stops the satisfaction set. For process barriers, **process** and **none** have the same effect. This is the default if the BARRIER_STOP_WHEN_DONE variable is **none**.

**–pending** If TotalView cannot find a location to set the barrier, adding this option tells TotalView to create 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 **dbreak** on page 31.

**Description:** The **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.

You most often use the **dbARRIER** command 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, **dstep**, **dnext**, and **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. They are just released; they are 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 SHARE_ACTION_POINT variable. If it is set to true, the default is group. Otherwise, it is process.

TotalView determines what processes and threads 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.

Barriers can create deadlocks. For example, if two threads participate in two different barriers, each could be left waiting at different barriers, 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 tells TotalView what other threads to stop when a thread arrives at a barrier.

The `--stop_when_done` option controls the set of additional threads that TotalView stops 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.

The `none` argument for these options tells the CLI not to 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`.

```
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
   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. 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.
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. In most cases, you will be typing a line number or a path-name containing a file name, function name, and line number, each separated by `#` characters (for example, `#file#line`). If you omit parts of this specification, the CLI creates them for you. For more information, see "Qualifying Symbol Names" in Chapter 13 of the TotalView Users 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` Tells TotalView to stop 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 97 for more information.

- `–g` Tells TotalView to stop 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 97 for more information.

- `–t` Tells TotalView to stop 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 97 for more information.

- `–l lang` Sets the programming language used when you are entering expression `expr`. The languages you can enter are `c`, `c++`, `f77`, `f9`, and `asm` (for C, C++, FORTRAN 77, Fortran 9x, and assembler, respectively). If you do not specify a language, TotalView assumes that you wrote the expression in the same language as the routine at the breakpoint.

- `–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 16 of the TotalView Users Guide.

- `–pending` If TotalView cannot find a location to set the breakpoint, adding this option tells TotalView to create 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 that TotalView triggers when execution arrives at the specified location. The breakpoint tells each thread that it should stop when it arrives at a breakpoint. This command returns the ID of the new breakpoint. The CLI might not be able to set a breakpoint at the line you specify. This occurs when a line does not contain an executable statement.

Specifying a procedure name without a line number tells the CLI to set 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.

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.

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 tells TotalView that it should check shared libraries that it subsequently loads to see if it can set the breakpoint. 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 the CLI displays information on a breakpoint’s status, it displays the location where execution actually stops.

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.

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

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 barrier on each of the overloaded functions. The expression can begin with one of the following options:

`class class_name`  
Tells TotalView that the breakpoint expression is a class name and it should set a barrier on all methods of that class.

**Breakpoint Expressions**
virtual class::signature

Tells TotalView to set a a barrier on virtual function that override the signature in the class specified by the breakpoint expression.

Set a barrier at the line specified by \textit{breakpoint-expr} or the absolute address \textit{addr}. You can enter a breakpoint expression that are sets of addresses at which the barrier is placed, and are as follows:

- \texttt{[[##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 \textit{signature}. If parts of a function signature are missing, this expression can match more than one signature. For example, \texttt{“f”} matches \texttt{“f(void)”} and \texttt{“A::f(int)”}. You cannot specify a return type in a signature.
- \texttt{class\ class\_name}
  Names a set containing the addresses of all member functions of class \texttt{class\_name}.
- \texttt{virtual\ class::signature}
  Names the set of addresses of all virtual member functions that match \textit{signature}, and which are in the classes or derived from the class.

\textbf{Command alias:}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{Alias} & \textbf{Definition} & \textbf{Description} \\
\hline
b & \texttt{break} & Sets a breakpoint \\
bt & \{\texttt{dbreak t}\} & Sets a breakpoint only on the thread of interest \\
\hline
\end{tabular}
\end{table}

\textbf{Examples:}

For all examples, assume that the current process set is \texttt{d2.<} when the breakpoint is defined.

\texttt{dbreak 12}
Suspends process 2 when it reaches line 12. However, if the \texttt{STOP\_ALL} variable is set to \texttt{group}, all other processes in the group are stopped. In addition, if you set the \texttt{SHARE\_ACTION\_POINT} variable to \texttt{true}, the breakpoint is placed in every process in the group.

\texttt{dbreak -address 0x1000764}
Suspends process 2 when execution reaches address 0x1000764.

\texttt{b 12 -g}
Suspends all processes in the current control group when execution reaches line 12.

\texttt{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.

\texttt{dfocus p3 b 57 -e \{goto $63\}}
In process 3, sets the same evaluation point as the previous example.
**dcache**

Clears the remote library cache

**Format:**  
dcache –flush

**Arguments:**  
–flush  
Delete all files from the library cache that are not currently being used.

**Description:**  
The dcache –flush command tells TotalView to remove the library files that it places in your cache. This cache is located in the .totalview/lib_cache subdirectory contained 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, this command removes files not currently being used and that are not quite old enough for TotalView to automatically delete.
dcheckpoint

Creates a checkpoint image of processes (IBM and SGI)

**Format:**

```
dcheckpoint [ after_checkpointing ] [ –by process_set ] [ –no_park ]
   [ –ask_attach_parallel ] [ –no_attach_parallel ]
   [ –no_preserve_ids ] [ –force ] checkpoint-name
```

Creates a checkpoint on SGI IRIX

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

**Arguments:**

- **after_checkpointing**
  Defines the state of the process both before and after the checkpoint. Use one of the following options:
  - **–delete**
    Processes exit after the checkpoint occurs.
  - **–detach**
    (SGI only) Processes continue running after the checkpoint occurs. In addition, TotalView detaches from them.
  - **–go**
    (SGI only) Processes continue running after the checkpoint occurs.
  - **–halt**
    Processes halt after the checkpoint occurs.

- **–by process_set**
  Indicates the set of processes to checkpoint and a group of related processes. If you do not use a `process_set` option, TotalView only checkpoints the focus process. (On AIX, the checkpoint is the scope of the Parallel Environment job.) Your options are:
  - **ash**
    Checkpoints the array session.
  - **pgid**
    Checkpoints the entire UNIX process group.
  - **sid**
    Checkpoints the entire process session.

- **–no_park**
  (SGI only) Tells TotalView not to `park` all processes before it begins checkpointing them. If you use this option, you must use the `drestart` command’s `–no_unpark` option. If you restart the checkpoint from a shell, you must use this option.

- **–ask_attach_parallel**
  (SGI only) Asks whether TotalView reattaches to parallel processes of a parallel job. (Some systems automatically detach you from processes being checkpointed.)

- **–no_attach_parallel**
  (SGI only) Tells TotalView not to reattach to processes from which the checkpointing processes detached. (Some systems automatically detach you from processes being checkpointed.)

- **–no_preserve_ids**
  (SGI only) Lets TotalView assign new IDs when it restarts a checkpoint. If you do not use this option, TotalView uses the same IDs.

- **–force**
  (SGI only) Tells TotalView to overwrite an existing checkpoint.

- **checkpoint-name**
  (SGI only) Specifies the name being assigned to the checkpoint. TotalView ignores this name if you are creating a checkpoint on an IBM/RS6000 machine.
The `dcheckpoint` command saves program and process information to the `checkpoint-name` file. This information includes process and group IDs. Later, you use the `drestart` command to restart the program.

This command does not save TotalView breakpoint information. If you need to save this information, use the `dactions` command.

The following restrictions exist when you are trying to checkpoint IRIX processes:

- IRIX does not checkpoint a process that is running remotely and which communicates using sockets. Since the TotalView Server (`tvsrv`) uses sockets to redirect `stdin`, `stdout`, and `stderr`, you need to use the `drun` command to modify the way your processes send information to a `tty` before you create a checkpoint.
- If you are using SGI MPI, you need to use the `-cpr` command-line option.

Use the ASH option with MPI checkpoints

The `after_checkpointing` options let you specify what happens after the checkpoint operation concludes. If you do not specify an option, the CLI tells the checkpointed processes to stop. This lets you investigate a program's state at the checkpoint position. In contrast, the `-go` option tells the CLI to let the processes continue to run. You use the `-detach` and `-halt` options less frequently. The `-detach` option shuts down the CLI and leaves the processes running. This command's `-halt` option is similar to the `-detach` option, differing only in that processes started by the CLI and TotalView also terminate.

The `process_set` options tell TotalView which processes to checkpoint. Although the focus set can only contain one process, you can also include processes in the same process group, process session, process hierarchy, or array session in the same checkpoint. If you do not use one of the `-by` options, TotalView only checkpoints the focus process.

If the focus group contains more than one process, the CLI displays an error message.

Before TotalView begins checkpointing your program, it 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 get control of it.) If you plan to restart the checkpoint file outside of TotalView, you must use the `-no_park` option.

When you create the checkpoint, the CLI detaches from processes before they are checkpointed. By default, the CLI automatically reattaches to them. If you do not want this to occur, use the `-no_attach_parallel` option to tell the CLI not to reattach, or use the `-ask_attach_parallel` option to tell the CLI to ask you whether to reattach.
Examples:

- **dcheckpoint check1**
  
  Checkpoints the current process. TotalView writes the checkpoint information to the `check1` file. These processes stop.

- **f3 dcheckpoint check1**
  
  Checkpoints process 3. Process 3 stops. TotalView writes the checkpoint information to the `check1` file.

- **f3 dcheckpoint –go check1**
  
  Checkpoints process 3. Process 3 continues to run. TotalView writes the checkpoint information to the `check1` file.

- **f3 dcheckpoint –by pgid –detach check1**
  
  Checkpoints process 3 and all other processes in the same UNIX process group. All of the checkpointed processes continue running but they run detached from the CLI. TotalView writes the checkpoint information to the `check1` file.
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.

You can interrupt this action by typing Ctrl+C. This tells TotalView to stop executing these processes.

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.

**Command alias:**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>co</td>
<td><code>dcont</code></td>
<td>Resume</td>
</tr>
<tr>
<td>CO</td>
<td><code>{dfocus g dcont}</code></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 TotalView is holding 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 which 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 which belong to processes 1, 2, and 3.

- `CO`
  Resumes execution of all stopped threads that are not held and which belong to the current group.
**ddelete**

**Deletes action points**

**Format:**  
Deletes the specified action points

\[ \textit{ddelete action-point-list} \]

Deletes all action points

\[ \textit{ddelete -a} \]

**Arguments:**  
\[ \textit{action-point-list} \]
A list of the action points to delete.

\[ \textit{-a} \]
Tells TotalView to delete all action points in the current focus.

**Description:**  
The \textit{ddelete} command permanently removes one or more action points.  
The argument to this command tells the CLI which action points to delete.  
The \textit{-a} option tells the CLI to delete all 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.

**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:**  
\[ \textit{ddelete 1 2 3} \]
Deletes action points 1, 2, and 3.

\[ \textit{ddelete -a} \]
Deletes all action points associated with processes in the current focus.

\[ \textit{dfocus \{p1 p2 p3 p4\} ddelete -a} \]
Deletes all the breakpoints associated with processes 1 through 4. Breakpoints associated with other threads are not affected.

\[ \textit{dfocus a de -a} \]
Deletes all action points known to the CLI.
ddetach

**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 does not have to be originally 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.

**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.
**ddisable**

Temporarily disables action points

**Format:**

```
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 say which instances to disable. You can obtain a list of these numbers if you use the `dactions` command.
- `-a` Tells TotalView to disable all action points.

**Description:**

The `ddisable` command temporarily deactivates action points. This command does not, however, delete them.

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.

**Command alias:**

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

**Examples:**

- `ddisable 3 7` Disables the action points whose IDs are 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.
- `di 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.
- `ddisable 1 2 -block 3 4` Disables the action points associated with blocks 3 and 4 in action points 1 and 2.
ddlopen

Dynamically loads shared object libraries

Format: Dynamically loads a shared object library

\texttt{ddlopen [ \textit{–now} | \textit{–lazy} ] [ \textit{–local} | \textit{–global} ] [ \textit{–mode int} ] filespec}

Displays information about shared object libraries

\texttt{ddlopen [ \textit{–list { dll-ids...}} ]}

Arguments:

\textit{–now} \hspace{1cm} \text{Includes RTLD\_NOW in the dlopen command’s mode argument. (Now means immediately resolve all undefined symbols.)}

\textit{–lazy} \hspace{1cm} \text{Includes RTLD\_LAZY in the dlopen command’s mode argument. (Lazy means that the dlopen() function does not try to resolve unresolved symbols. Instead, symbols are resolved as code is executed.)}

\textit{–local} \hspace{1cm} \text{Includes RTLD\_GLOBAL in the dlopen command’s mode argument. (Local means that library symbols are not available to libraries that the program subsequently loads.) This argument is the default.}

\textit{–global} \hspace{1cm} \text{Includes RTLD\_LOCAL in the dlopen command’s mode argument. (Global means that library symbols are available to libraries that the program subsequently loads.)}

\textit{–mode int} \hspace{1cm} \text{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 \hspace{1cm} \text{The shared library to load.}

\textit{–list} \hspace{1cm} \text{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.}

\textit{dll-ids} \hspace{1cm} \text{A list of one or more DLL IDs.}

Description: The \texttt{ddlopen} command dynamically loads shared object libraries, or lists the shared object libraries that you loaded using this or the \texttt{Tools > Dynamic Libraries} command.

If use a \texttt{filespec} argument, TotalView performs a \texttt{dlopen} operation on this file in each process in the current P/T set. If you are running on the IBM AIX operating system, you can add a parenthesized library module name to the end of the \texttt{filespec} argument.

\texttt{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 \textit{–now} and \textit{–lazy} options indicate whether \texttt{dlopen} immediately resolves unresolved symbol references or defer 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 \texttt{RTLD\_LAZY} the same as \texttt{RTLD\_NOW}).

The \texttt{–local} and \texttt{–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. (HP Tru 64 UNIX doesn’t support either alternative; its operation is equivalent to using the \texttt{–global} option. IRIX, Solaris, and Linux only support the \texttt{–global} option; if you don’t specify an option, the default is the \texttt{–local} option.)

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

If an error occurs while executing the \texttt{dlopen()} function, TotalView calls the \texttt{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 \texttt{dlopen} command. You can use the \texttt{TV::dll} command to obtain information about and delete these objects. If all \texttt{dlopen()} calls return immediately, the \texttt{dlopen} command returns a unique DLL ID that you can also use with the \texttt{TV::dll} 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 \texttt{TV::expr} command.

If you do not use a \texttt{filespec} argument or if you use the \texttt{–list} option without using a DLL ID argument, TotalView prints information about objects loaded using \texttt{dlopen}. 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 \texttt{dlopen} command prints its output directly to the console.

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

**Examples:**

\begin{verbatim}
    dlopen "mpistat.so"
    Loads \texttt{mpistat.so} library file. The returned argument lists the process into which TotalView loaded the library.

    dfocus g dlopen "mpistat.so(mpistat.o)"
    Loads the module \texttt{mpistat.o} in the AIX DLL library \texttt{mpistat.so} into all members of the current process’s control group:
\end{verbatim}
**ddlopen**

- **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.
**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. It also 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:

```
0 check_fortran_arrays_ PC=0x10001254, FP=0x7fff2ed0 [arrays.F#48]
```

- `d 5` Moves the current frame down five levels in the call stack.
**denable**

**Format:** Enables some action points

```
denable action-point-list [ –block number-list]
```

Enables all disabled action points in the current focus

```
denable –a
```

**Arguments:**

- `action-point-list` The identifiers of the action points being enabled.
- `–a` Tells TotalView to enable all action points.
- `–block number-list` If you set a breakpoint on a line that is ambiguous, this option names which instances to enable. You can obtain a list of these numbers if you use the `dactions` command.

**Description:**

The `denable` command reactivates action points that you previously disabled with the `ddisable` command. The `–a` option tells the CLI to enable all action points in the current focus.

If you did not save the ID values of disabled action points, you can use the `dactions` command to obtain a list of this information.

If you do not indicate a focus, the default focus is the Process of Interest.

**Command alias:**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>en</td>
<td><code>denable</code></td>
<td>Enables action points</td>
</tr>
</tbody>
</table>

**Examples:**

- `denable 3 4` Enables two previously identified action points.
- `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.
- `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.
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. If you omit this option, the CLI determines this number of columns based on the data’s word size and format.
- `-c cnt`
- `-count cnt` Specifies the number of elements to examine. If you omit this option, the CLI displays the entire object. This number is determined by the object’s datatype. If no type is available, the default value for `cnt` is 1 element.
- `-d`
- `-data_only` Do 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` Interpret 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` Show information about the type of memory associated with the address. If you omit the option, the CLI does not display this information. This argument is incompatible with `-data_only`. When you use this option, the CLI annotates 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 automatically to include the pre-guard region if the user specified guards in the memory debugging configuration.

--sc
--show_chars

Show a trailing character dump for each line. If you do not use this option, the CLI does not show the trailing characters.

--sl len
--string_length len

Specifies the maximum size string to display. If you do not use 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).

Description:
Examine 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.

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>Examine (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:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>susp-eval-id</td>
<td>The ID returned or thrown by the <code>dprint</code> command or which is printed by the <code>dwhere</code> command.</td>
</tr>
<tr>
<td>–all</td>
<td>Flushes all suspended evaluations in the current focus.</td>
</tr>
</tbody>
</table>

**Description:**

The `dflush` command unwinds the stack to eliminate frames generated by suspended computations. Typically, these frames can occur if you used the `dprint -nowait` command. However, this situation can occur, for example, 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.

You can use this command in the following ways:

- 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.

```plaintext
# Create 5 suspended functions
#
d1.<> dprint -nowait nothing2(7)
7
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
d1.<> dprint -nowait nothing2(8)
8
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
d1.<> dprint -nowait nothing2(9)
9
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
```
d1.<> dprint -nowait nothing2(10)
10
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
d1.<> dprint -nowait nothing2(11)
11
Thread 1.1 hit breakpoint 4 at line 310 in "nothing2(int)"
...

# The top of the call stack looks like:
#
# Use the dflush command to remove the last item pushed onto the stack. Notice the frame associated with "11" is no longer there.
#
# 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.

#
# The top of the call stack looks like:
#
# Use the dflush command to remove the last item pushed onto the stack. Notice the frame associated with "11" is no longer there.
#
dflush

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]
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. This set defines the target upon which the CLI commands that follow will act.
- `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 it or just the command that immediately follows.

The `dfocus` command always expects a P/T value as its first argument. This value can either be 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 a `command` argument, the `dfocus` command returns the focus as a string value.

Instead of a P/T set, you can type a P/T set expression. These expressions are described in "Using P/T Set Operators" in Chapter 12 of the TotalView Users Guide.

**Command alias:**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td><code>dfocus</code></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
Sets and then resets command focus. A focus command that includes a dot and omits the process value tells the CLI to use 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 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 12 of the TotalView Users Guide contains additional dfocus examples.
Displayer Global Array variables

**Format:**

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

**Arguments:**

- **–lang**
  
  Tells TotalView which language conventions to use when displaying information. If you omit this option, TotalView uses the language used by the thread of interest.

- **lang_type**
  
  Tells TotalView which language type it uses when displaying a global array. Your choices are either “c” or “f”.

- **handle_or_name**
  
  Tells TotalView to display an array. This can either be a numeric handle or the name of the array. If you omit this argument, TotalView displays a list of all Global Arrays.

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

**Description:**

The `dga` command tells TotalView to display 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 it is 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**

The CLI displays a list of Global Arrays; for example:

```plaintext
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:

```plaintext
bc_mask(:,2) = {
    (1,1) = 1 (0x00000001)
    (2,1) = 1 (0x00000001)
    (1,2) = 1 (0x00000001)
    (2,2) = 0 (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.
**dgo**

**Format:**

```plaintext
dgo
```

**Arguments:**

- **–back** (ReplayEngine only) Runs the non-held 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 tells all nonheld processes and threads in the current focus to resume execution. 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.

**Command alias:**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>g</code></td>
<td><code>dgo</code></td>
<td>Resumes execution</td>
</tr>
<tr>
<td><code>G</code></td>
<td><code>{dfocus g dgo}</code></td>
<td>Resume 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 g g` Continues all processes in the control group. Only processes and threads that are not held are allowed to run.
- `f gL g` Continues all threads in the share group that are at the same PC as the thread of interest.
- `f pL g` Continues all threads in the current process that are at the same PC as the thread of interest.
- `f t g` Continues a single thread.
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:**

- `--` *gid*  
  The group ID upon 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 tell the CLI to create a new process or thread group. You can specify one of the following arguments: `t`, `thread`, `p`, or `process`.

**Description:** The `dgroups` command lets you perform the following functions:

- Add members to process and thread groups.
- Create a group.
- Intersect a group with a set of processes and threads.
- Delete groups.
- Display the name and contents of groups.
- Remove members from a group.
**dgroups –add**
The `dgroups –add` command 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. If you do not use an `id-list` argument, the CLI adds the threads and processes in the current focus. Similarly, you can name the group to which the CLI adds members if you use the `–g` option. If you omit 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.

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 only adds it once.

The CLI does not let you use this command to add a process to a control group. If you need to perform this operation, you can use the `CGROUP(dpid)` variable; for example:

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

**dgroups –delete**
The `dgroups –delete` command deletes the target group. You can only delete groups that you create; you cannot delete groups that TotalView creates.

**dgroups –intersect**
The `dgroups –intersect` command 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**
The `dgroups -list` command 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.

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

The CLI returns information from this command; the information is not returned.

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

**dgroups –new**
The `dgroups –new` command 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 by using 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.

*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**
The `dgroups –remove` command 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
 Tells the CLI to display information about all named groups; for example:

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

gr –l *
 Tells the CLI to display 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}
```
**dgroups**

**dgroups -new**

```
gr -n t -g mygroup $GROUP($CGROUP(1))
```

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

**set mygroup [dgroups -new]**

Creates a new process group that contains the current focus process.

**dgroups -remove**

```
dgroups -remove -g 3 3.2
```

Removes thread 3.2 from group 3.

**f W dgroups -add**

Marks the current thread as being a worker thread.

**f W dgroups -r**

Indicates that the current thread is not a worker thread.
dhalt

Suspended 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.

**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:**
dxalt
Suspended 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
Suspended 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.
**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 which 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
  
  ```
  dheap --red_zones [ subcommands ]
  ```

- Enables or disables allocation and reallocation notification
  
  ```
  dheap --tag_alloc subcommand [ start_address [ end_address ] ]
  ```

- Displays the Memory Debugger version number
  
  ```
  dheap --version
  ```
Description: The *dheap* command is described in the *Batch Scripting* section of Locating Memory Problems in the MemoryScape documentation.
dhistory

Performs actions upon ReplayEngine

**Format:**

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

**Arguments:**

- **–info**
  Writes ReplayEngine information including the current time, the live time and if 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 a 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 the ReplayEngine so that it will be active when you next restart the program.

- **–disable**
  Disables the ReplayEngine so that it will be inactive when you next restart the program.

**Description:**

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

In addition, you can enable and display the 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 dow.

**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. You can abbreviate the `–process` option to `–p`.  
- **–thread** Holds threads in the current focus. You can abbreviate the `–thread` option to `–t`.

**Description:**

The `dhold` command holds the threads and processes in the current focus. Unlike system manager threads, you cannot hold the 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`.

---

"TotalView Reference Guide: version 8.8"
dkill

Terminates execution of processes

**Format:**

`dkill [–remove ]`

**Arguments:**

`–remove`

Tells TotalView that it should remove 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 only restart the initial process.

If you do not indicate a focus, the default focus is the Process of Interest. 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><code>dkill</code></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.
**dlappend**

Appends list elements to a TotalView variable

**Format:**

```plaintext
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. The `dlappend` 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 following command displays an error message:

```tcl
dlappend foo 1 3 5
```

**Examples:**

```plaintext
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.
**dlist**

**Displays source code lines**

**Format:**

- **dlist** [ –n num-lines ]  
  Displays source code relative to the current list location

  - **dlist** breakpoint-expr [ –n num-lines ]  
    Displays source code relative to a named place

  - **dlist** –e [ –n num-lines ]  
    Displays source code relative to the current execution location

**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 succeeding 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, you will 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 “Qualifying Symbol Names” in Chapter 12 of the TotalView Users Guide.) The CLI creates defaults if you omit parts of this specification.
  
  If you enter a different file, this will be the file that the CLI uses for future displays. 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 32. If what you enter names more than one address, TotalView picks one.

- **–e**: Sets the display location to include the current execution point of the thread of interest. 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 place in the source code. (This position is 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.

The CLI expands tabs in the source file as blanks in the output. The `TAB_WIDTH` variable controls the tab stop width, which has a default value of 8. If `TAB_WIDTH` is set to –1, no tab processing is done, 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>l</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.

- **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 so that it is the twentieth 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 Chapters 5 and 6 in the TotalView Users Guide for information on the server launch commands.)

- **{ –np | –procs | –tasks } num**
  Indicates the number of processes or tasks that the starter program will create.

- **–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.

- **–mpi starter**
  Names the starter configuration. This can be the name of a configuration supplied by TotalView Technologies or one that you have created and have named using the TV::parallel_configs variable. For information on defining configurations, see Appendix A, “MPI Startup,” on page 319.

- **–starter_args argument**
  Indicates additional arguments that are 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 a file name. You need to use this argument if the file name begins with a dash or only uses numeric characters.

- **executable**
  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 be attached to when an MPI job is created or attached to. The list is space-separated; each element can have one of 3 forms:
rank: specifies that rank only
rank1-rank2: specifies all ranks between rank1 and rank2, inclusive
rank1-rank2:stride: specifies every stride rank between rank1 and rank2
A rank specification can be either a number or "$max" which represents the last rank in the MPI job.
For example:
dload -parallelAttachSubset \{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. The dload command returns the TotalView ID for the new object.

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 be running 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

    Loads another process based on image foo on
    machine slowhost. The CLI places this process in the
    same group as process 2.
The *dmstat* command displays information about how your program is using memory. The CLI returns memory information in three parts, as follows:

- **Memory usage summary**: Indicates 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**: Shows the amount of memory that each process is currently using.
- **Image information**: Lists 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 contains definitions of the columns the CLI displays:

<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 being used for data created at run time; for example, calls to the <em>malloc()</em> function allocate space on the heap while the <em>free()</em> function releases it.</td>
</tr>
<tr>
<td>stack</td>
<td>The amount of memory used by the currently executing routine and all of the routines in its backtrace. If this is a multithreaded process, TotalView only shows 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 <em>foo()</em>, 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 <em>foo()</em> function. If <em>foo()</em> invokes the <em>bar()</em> function, the stack contains three frames. When <em>foo()</em> finishes executing, the stack only contains one frame.</td>
</tr>
</tbody>
</table>
dmstat

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>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10432</td>
<td>[16384]</td>
<td>17.69M</td>
</tr>
<tr>
<td>5</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>[16384]</td>
<td>17.44M</td>
</tr>
<tr>
<td>6</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>[16384]</td>
<td>17.44M</td>
</tr>
<tr>
<td>7</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>[16384]</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</td>
</tr>
<tr>
<td>/lib/i686/libc.so.6</td>
<td>1050688</td>
<td>122338</td>
<td>16</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>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>[16384]</td>
<td>17.44M</td>
</tr>
<tr>
<td>5</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>[16384]</td>
<td>17.44M</td>
</tr>
<tr>
<td>6</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>[16384]</td>
<td>17.44M</td>
</tr>
<tr>
<td>7</td>
<td>1128.54K</td>
<td>16.15M</td>
<td>9976</td>
<td>10944</td>
<td>[16384]</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>

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 differ 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 nesting 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 value is this size difference.

The sum of the sizes of the mappings in the process’s address space.

Examples:

dmstat is sensitive to the focus, as this example from a four-process program shows:

```plaintext
process: text data heap stack [stack_vm] vm_size
1 (9271): 1128.54K 16.15M 9976 10432 [16384] 17.69M
image information:
image_name      text        data  dpids
....ry/forked_mem_exampleLINUX      2524    16778479      1 5 6 7
/lib/i686/libpthread.so.0           32172    27948            1
/lib/i686/libc.so.6                 1050688  122338           16
/lib/ld-linux.so.2                  70240    10813            1
```

dfocus a dmstat

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

```plaintext
process: text data heap stack [stack_vm] vm_size
1 (9979): 1128.54K 16.15M 9976 10944 [16384] 17.69M
5 (9982): 1128.54K 16.15M 9976 10944 [16384] 17.44M
6 (9983): 1128.54K 16.15M 9976 10944 [16384] 17.44M
7 (9984): 1128.54K 16.15M 9976 10944 [16384] 17.44M
maximum:
1 (9979): 1128.54K 16.15M 9976 10944 [278528] 17.69M
minimum:
5 (9982): 1128.54K 16.15M 9976 10944 [16384] 17.44M
```

image information:

```plaintext
image_name      text        data  dpids
....ry/forked_mem_exampleLINUX      2524    16778479           1 5 6 7
/lib/i686/libpthread.so.0           32172    27948           1 5 6 7
/lib/i686/libc.so.6                 1050688  122338           1 5 6 7
/lib/ld-linux.so.2                  70240    10813           1 5 6 7
```
**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 tells the CLI 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 100.

**Command alias:**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td><strong>dnext</strong></td>
<td>Runs the thread of interest 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 thread of interest, 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 thread of interest one statement and runs all threads in the process that are at the same PC as the thread of interest to the same statement. Other threads in the process run freely. The group of threads that is at the same PC is called the <strong>lockstep group</strong>. This alias does not force process width. If the default focus is set to <strong>group</strong>, this steps the group.</td>
</tr>
</tbody>
</table>
### Examples:

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dnext</td>
<td></td>
<td>Steps one source line.</td>
</tr>
<tr>
<td>n 10</td>
<td></td>
<td>Steps ten source lines.</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td>Steps one source line. It also runs all other processes in the group that are in the same lockstep group to the same line.</td>
</tr>
<tr>
<td>f t n</td>
<td></td>
<td>Steps the thread one statement.</td>
</tr>
<tr>
<td>dfocus 3. dnext</td>
<td></td>
<td>Steps process 3 one step.</td>
</tr>
</tbody>
</table>
**dnexi**

**Steps machine instructions, stepping over subroutines**

**Format:**

```
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 `dnexi` command executes machine-level instructions; that is, it advances the program by a single instruction. However, if the instruction invokes a subfunction, the `dnexi` command executes the subfunction as if it were one instruction; that is, it steps over the call. This command steps the thread of interest while allowing other threads in the process to run.

The optional `num-steps` argument tells the CLI how many `dnexi` operations to perform. If you do not specify `num-steps`, the default is 1.

The `dnexi` 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 100.

**Command alias:**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ni</td>
<td><code>dnexi</code></td>
<td>Runs the thread of interest one instruction while allowing other threads in the process to run.</td>
</tr>
<tr>
<td>NL</td>
<td><code>{dfocus g dnexti}</code></td>
<td>A group stepping command. This searches for threads in the share group that are at the same PC as the thread of interest, 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><code>{dfocus L dnexti}</code></td>
<td>Steps the process threads in lockstep. This steps the thread of interest one instruction, and runs all threads in the process that are at the same PC as the thread of interest to the same statement. Other threads in the process run freely. The group of threads that is at the same PC is called the <strong>lockstep group</strong>. This alias does not force process width. If the default focus is set to <strong>group</strong>, this steps the group.</td>
</tr>
</tbody>
</table>
**Alias** | **Definition** | **Description**
--- | --- | ---
NIL  | \{dfocus gL dnexti\} | Steps lockstep threads in the group. This steps all threads in the share group that are at the same PC as the thread of interest one instruction. Other threads in the control group run freely.

niw  | \{dfocus W dnexti\} | Steps worker threads in the process. This steps the thread of interest 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.

NIW | \{dfocus gW dnexti\} | Steps worker threads in the group. This steps the thread of interest 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.

**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.
- ft n: Steps the thread one machine-level instruction.
- dfocus 3. dnexti: Steps process 3 one machine-level instruction.
**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. If you omit 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 thread of interest 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 thread of interest, 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 thread of interest out of the current function, and also runs all threads in the process that are at the same PC as the thread of interest 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 thread of interest out of the current function. 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 100 command information.

**Examples:**

- `f t ou` Runs the current thread of interest 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.
dprint

Evaluates and displays information

**Format:**

Prints the value of a variable

\[\text{dprint} \ [ \text{–nowait} \ [ \text{–slice “slice_expr”} \ ] \ \text{variable}]\]

Prints the value of an expression

\[\text{dprint} \ [ \text{–nowait} \ [ \text{–slice “slice_expr”} \ ] \ \text{expression}]\]

**Arguments:**

–nowait

Tells TotalView to evaluate the expression in the background. You need to use \text{T:\texttt{\textbf{expr}}} to obtain the results, as they are not displayed.

–slice “slice_expr”

Defines an array slice—that is, a portion of the array—that will be printed. If the programming language is C or C++, you will need to use a backquote (') when you enter the array subscripts. For example, “\[100:110\]”.  

variable

A variable whose value is displayed. The variable can be local to the current stack frame or it can be global. If the variable being displayed is an array, you can qualify the variable’s name with a slice that tells the CLI to display 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 $\text{stop}$, SEGV, breakpoint occurs, the dprint command throws an exception that describes the event. The first exception subcode returned by \text{T:\texttt{\textbf{errorCodes}}} is the \text{ susp-eval-id} (a suspension-evaluation-ID). You can use this to manipulate suspended evaluations with the dflush and \text{T:\texttt{\textbf{expr}}} commands. For example:

\[\text{dfocus tdpid.dtid TV::expr get susp-eval-id}\]

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 \text{ susp-eval-id} that you can use to obtain the results of the evaluation using \text{T:\texttt{\textbf{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 tells the CLI to stop printing this information.
Since the dprint command can generate a considerable amount of output, you might want to use the capture command described on page 18 to save the output to a variable.

Structure output appears with one field printed per line; for example:

```plaintext
sbfo = {
    f3 = 0x03 (3)
    f4 = 0x04 (4)
    f5 = 0x05 (5)
    f20 = 0x000014 (20)
    f32 = 0x0000020 (32)
}
```

Arrays print in a similar manner; for example:

```plaintext
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:

```plaintext
d.1<> p -slice "\[10:20\]" random
random slice:(10:30) = {
    (10) = 0.479426
    (11) = 0.877583
    (12) = 0.564652
    (13) = 0.825336
    (14) = 0.644218
    (15) = 0.764842
    (16) = 0.717356
    (17) = 0.696707
    (18) = 0.783327
    (19) = 0.62161
    (20) = 0.841471
}
```

The following is another way of specifying the same slice:

```plaintext
d.1<> set my_var \[10:20\]
d.1<> p -slice $my_var random
random slice:(10:30) = {
```

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:

```c
float bint;
```

The following statement displays the variable as an array of one integer:

```c
dprint { (int [1]) bint; }
```

If the expression is a pointer, the first addresses is the value of the pointer. Here is an array declaration:

```c
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:

```c
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:

```c
dprint \$r1
```

You must precede the dollar sign (`$`) with a backslash to escape it since the register’s name includes the `$`. This `$` is not the standard indicator that tells Tcl to fetch a variable’s value. Appendix A, "Architectures," on page 287 lists the mnemonic names assigned to registers.

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><code>p</code></td>
<td><code>dprint</code></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()\}
1
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
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
dptsets

Shows the status of processes and threads

Format:  \texttt{dptsets} [ \texttt{ptset_array} ] ...

Arguments:  \texttt{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 12 of the \textit{TotalView Users Guide}.)

Description:  The \texttt{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 Chapter 12 of the \textit{TotalView Users Guide}), and the elements’ array indices are strings that label each element’s section in the output.

If you do not use the optional \texttt{ptset_array} argument, the CLI supplies a default array that contains all P/T set designators: \texttt{error}, \texttt{existent}, \texttt{held}, \texttt{running}, \texttt{stopped}, \texttt{unheld}, and \texttt{watchpoint}.

Examples:  The following example displays information about processes and threads in the current focus:

\begin{verbatim}
  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
  ...
\end{verbatim}
The following example creates a two-element P/T set array, and then displays the results. Notice the labels in this example.

```plaintext
d1.<> set set_info(0) breakpoint(1)
    breakpoint(1)
d1.<> set set_info(1) stopped(1)
    stopped(1)
d1.<> dptsets set_info
    0:
    1: 892484 Breakpoint [arraySGI]
       1.1: 892484.1 Breakpoint PC=0x10001544, [array.F#81]
    1:
    1: 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:

```plaintext
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:
    1: 882547 Breakpoint [arraysSGI]
       1.1: 882547.1 Breakpoint PC=0x10001544, [arrays.F#81]
    my_breakpoints:
    1: 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.

```plaintext
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**

Restarts 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. If you specify a file name instead of `stdin`, `in_operation` indicates a file from which the launched process reads information.

- **out_operations**
  
  Names the file to which the CLI writes output. If you specify a file name instead of `stdout`, `out_operations` indicates the file into which the launched processes writes information.

- **error_operations**
  
  Names the file to which the CLI writes error output. If you specify a file name instead of `stderr`, `error_operations` indicates the file into which the launched processes writes error information.

- **>& outfile**
  
  Sends output and error messages to `outfile` instead of `stdout` and `stderr`.

- **>& 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`.

- **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 95 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:

```bash
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:

```bash
drerun
```

Reruns the current process. Because it doesn’t use arguments, the process restarts using its previous values.

```bash
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.
drestart

Restarts a checkpoint (IBM and SGI only)

**Format:**

Restarts a checkpoint on IBM AIX

```
drestart [ –halt ] [ –g gid ] [ –r host ] [ –no_same_hosts ]
```

Restarts a checkpoint on SGI IRIX

```
drestart [ process-state ] [ –no_unpark ] [ –g gid ] [ –r host ]
  [ –ask_attach_parallel | –no_attach_parallel ]
  [ –no_preserve_ids ] checkpoint-name
```

**Arguments:**

`process_state` Defines the state of the process both before and after the checkpoint. If you do not specify a process state, parallel processes are held immediately after the place where the checkpoint occurred. The CLI attaches to these created parallel processes. You can use one of the following options:

- `–detach` (SGI only) Although TotalView starts checkpointed processes, it does not attach to them.
- `–go` (SGI only) TotalView starts checkpointed parallel processes and attaches to them.
- `–halt` TotalView stops checkpointed processes after it restarts them.
- `–no_unpark` (SGI only) Indicates that the checkpoint was created outside of TotalView or that you used the `dcheckpoint` command’s `–no_park` option when you created the checkpoint file.
- `–g gid` Names the control group into which TotalView places all created processes.
- `–r host` Names the remote host upon which the restart occurs.
- `–ask_attach_parallel` (SGI only) Asks if the CLI automatically attaches to the parallel processes being created. This is most often used in procedures.
- `–no_attach_parallel` (SGI only) Attach only to the base process. That is, the CLI does not attach to the parallel processes being created.
- `–no_preserve_ids` (SGI only) Use new IDs after it restarts the processes. If you omit this option, TotalView causes the process to use the same process, group, session, or ash IDs after it restarts.
- `–no_same_hosts` (IBM only) 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.

`checkpoint-name` The file name used when the checkpoint file was saved.
Description: The `drestart` command restores and restarts all of the checkpointed processes. By default, the CLI attaches to the base process. You have the following choices, among others:

- If there are parallel processes related to this base process, TotalView attaches to them.
- If you do not want the CLI to automatically attach to these parallel processes, use the `–no_attach_parallel` option.
- If you do not know if there are parallel processes, if you want the user to decide, or if you are using this command in a Tcl procedure, use the `–ask_parallel_process` option.

Restarting on AIX using LoadLeveler

On the RS/6000, if you want to debug a LoadLeveler `poe` job from the point at which the checkpoint was made, 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`.

When attaching to `poe`, parallel tasks are not yet created, so do not try to attach to any of them. Also, use the `–no_attach_parallel` option when using the `dattach` command.

You cannot restart the checkpoint using the `drestart` command. `poe` will tell TotalView when it is time to attach to the parallel task so that it can complete the restart operation.

Examples:

- `drestart check1`
  Restarts the processes checkpointed in the `check1` file. The CLI automatically attaches to parallel processes. This is an SGI checkpoint because it specifies a name.

- `drestart –no_unpark check1`
  Restarts the processes checkpointed in the `check1` file. This file was either created outside of TotalView or it was created using the `–no_park` option. This is an SGI checkpoint because it specifies a name.
**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`: Sends output and error messages to `outfile` instead of `stdout` and `stderr`.
  - `>> outfile`: Appends output and error messages 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`: Sends error messages to `errfile` instead of `stderr`.

**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.

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>r</td>
<td><code>drun</code></td>
<td>Starts or restarts processes</td>
</tr>
</tbody>
</table>

Examples:

- `drun` Tells the CLI to begin 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.
**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 Appendix A, “TotalView Variables,” on page 185.

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. If you omit 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/tvdsvr}
```

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 this 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.
dstatus

Shows current status of processes and threads

Format:  dstatus
Arguments:  This command has no arguments
Description:  The dstatus command prints information about the current state of each process and thread in the current focus.

If you have not changed the focus, the default width 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 that contains process 1.

Command alias:

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>st</td>
<td>dstatus</td>
<td>Shows current status</td>
</tr>
<tr>
<td>ST</td>
<td>{dfocus g dstatus}</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=0x0d9cae64
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 W and g specifiers in the ST alias. The result is the same as if you had entered f gW st.

f L ST  Shows status for every thread in the share group that is at the same PC as the thread of interest.
**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 tells the CLI how many **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 tells the CLI that to only step the thread of interest.

On systems having identifiable manager threads, the **dfocus t dstep** command allows the manager threads as well as the thread of interest 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 thread of interest. TotalView selects one matching thread from each matching process. TotalView then runs all processes in the group and waits until the thread of interest 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. Regardless of which threads are in the group of interest, TotalView only waits for threads that are in the same share group as the
thread of interest. This is because it is not useful to run threads executing in different images to the same goal.

**Process Width**

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 thread of interest 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 thread of interest reaches the goal.

**Thread group**

TotalView runs all threads in the process that are in that group to the same goal as the thread of interest. 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 thread of interest 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 thread of interest 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 thread of interest, 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 thread of interest one statement, and runs all threads in the process that are at the same PC as the thread of interest 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 <strong>lockstep group</strong>. This alias does not force process width. If the default focus is set to <strong>group</strong>, 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 thread of interest one statement. Other threads in the control group run freely.</td>
</tr>
</tbody>
</table>
## dstep

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sw</td>
<td>{dfocus W dstep}</td>
<td>Steps worker threads in the process. This steps the thread of interest 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 <strong>group</strong>, 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 thread of interest 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.
**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 tells the CLI how many `dstepi` operations to perform. If you do not specify `num-steps`, the default is 1.

For more information, see `dstep` on page 100.

**Command alias:**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>si</td>
<td><code>dstepi</code></td>
<td>Runs the thread of interest one instruction while allowing other threads in the process to run.</td>
</tr>
<tr>
<td>SI</td>
<td><code>{dfocus g dstepi}</code></td>
<td>Searches for threads in the share group that are at the same PC as the thread of interest, 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><code>{dfocus L dstepi}</code></td>
<td>Steps the process threads in lockstep. This steps the thread of interest one instruction, and runs all threads in the process that are at the same PC as the thread of interest 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 <code>group</code>, this steps the group.</td>
</tr>
<tr>
<td>SIL</td>
<td><code>{dfocus gl dstepi}</code></td>
<td>Steps lockstep threads in the group. This steps all threads in the share group that are at the same PC as the thread of interest one instruction. Other threads in the control group run freely.</td>
</tr>
</tbody>
</table>
### Examples:

- **dstepi**
  - **Description:** 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**
  - **Description:** Executes the next 15 instructions.

- **f p1.2 dstepi**
  - **Description:** 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**
  - **Description:** Steps thread 2 in process 1 by one instruction. No other threads in process 1 execute.

### Alias Table

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>siw</td>
<td>( {d\text{focus } W \text{ dstepi}} )</td>
<td>Steps worker threads in the process. This steps the thread of interest 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 <code>group</code>, this steps the group.</td>
</tr>
<tr>
<td>SIW</td>
<td>( {d\text{focus } gW \text{ dstepi}} )</td>
<td>Steps worker threads in the group. This steps the thread of interest 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>
dunhold

Releases a held process or thread

**Format:**

Releases a process

```
dunhold –process
```

Releases a thread

```
dunhold –thread
```

**Arguments:**

- `–process` Tells TotalView to release processes in the current focus. You can abbreviate the `–process` option argument to `–p`.
- `–thread` Tells TotalView to release 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</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</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
http; uht
```

Releases all worker threads in the focus process.

Holds all threads in the focus process except the thread of interest.
dunset

Restores default settings for variables

Format:

Restores a CLI variable to its default value

\texttt{\textit{dunset debugger-var}}

Restores all CLI variables to their default values

\texttt{\textit{dunset --all}}

Arguments:

\texttt{debugger-var} \hspace{1cm} \text{Name of the CLI variable whose default setting is being restored.}

\texttt{--all} \hspace{1cm} \text{Restores the default settings of all CLI variables.}

Description:

The \texttt{dunset} command reverses the effects of any previous \texttt{dset} commands, restoring CLI variables to their default settings. See Appendix A, "TotalView Variables," on page 185 for information on these variables.

Tcl variables (those created with the Tcl \texttt{set} command) are not affected by this command.

If you use the \texttt{--all} option, the \texttt{dunset} command affects all changed CLI variables, restoring them to the settings that existed when the CLI session began. Similarly, specifying \texttt{debugger-var} tells the CLI to restore that one variable.

Examples:

\texttt{\textit{dunset PROMPT}} \hspace{1cm} \text{Restores the prompt string to its default setting; that is, {\texttt{\{dfocus\}>}}.}

\texttt{\textit{dunset --all}} \hspace{1cm} \text{Restores all CLI variables to their default settings.}
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 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 that are not running to the goal are allowed to run. If one of the secondary threads arrives at the goal before the thread of interest, the thread continues running, ignoring this goal. In contrast, if you specify thread width, only the thread of interest runs.

The `duntil` command differs from other step commands when you apply it to a group, as follows:

- **Process group** TotalView runs the entire group, and the CLI waits until all processes in the group have at least one thread that has arrived at the goal breakpoint. This lets you sync up all the processes in a group in preparation for group-stepping them.

- **Thread group** TotalView 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 between how processes and threads run using the `duntil` command and other stepping commands. Here is how this command determines what it runs:

- **Process Group Operation**: TotalView examines the thread of interest to see if it is already at the goal. If it is, TotalView does not run the process of interest. Similarly, TotalView examines all other processes in the share group, and it only runs processes that do not have a thread at the goal. It also runs members of the control group that are 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 that are 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 that are 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><code>duntil</code></td>
<td>Runs the thread of interest until it reaches a target, while allowing other threads in the process to run.</td>
</tr>
<tr>
<td>UN</td>
<td><code>{dfocus g duntil}</code></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><code>{dfocus L duntil}</code></td>
<td>Runs the thread of interest until it reaches the target, and runs all threads in the process that are at the same PC as the thread of interest 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><code>{dfocus gL duntil}</code></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><code>{dfocus W duntil}</code></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><code>{dfocus gW duntil}</code></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.
**dup**

**Moves up the call stack**

**Format:**  
`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 your program enters a procedure, and decreases by one when your 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 lets you access 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 the `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:**

- `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:

  1 check_fortran_arrays_  PC=0x10001254,  
  FP=0x7fff2ed0  [arrays.F#48]

- `dfocus p1 u 5`  
  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.
dwait

Blocks command input until the target processes stop

**Format:**

```
dwait
```

**Arguments:**

This command has no arguments

**Description:**

The `dwait` command tells the CLI to wait 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:

```
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 only need to specify the amount of storage to watch if you want to override the default value.
- `–g` Tells TotalView to stop all processes in the process’s control group when the watchpoint triggers.
- `–p` Tells TotalView to stop the process that hit this watchpoint.
- `–t` Tells TotalView to stop 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 only specify 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 what you specify is 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.

Watchpoints are not available on Macintosh computers running OS X, IBM PowerPC computers running Linux Power, IBM Blue Gene/L, and Hewlett Packard (HP) computers running or HP-UX.
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.

The CLI lets you obtain a variable’s address in the following ways 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.

Chapter 15 of the TotalView Users Guide contains 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 tell TotalView to watch a certain number of bytes, starting at the named location.

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.
dwhat

Determines 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 tells the CLI to display information about a named entity in a program. The displayed information contains the name of the entity and a description of the name.

To view information on CLI variables or aliases, you need to 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 show what the CLI displays when you enter one of the indicated 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: 0x11fffefc0
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_sec time_t (32 bits)
        tv_usec int (32 bits)
    }

dlist  
float field3_float;
double field3_double;
en_check en1;

};

main ()
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)
        <padding> = '\000' (0x00, or 0)
        field2_long = 0x0000000000000000 (0)
    }

...
wh check_struct

In process 2:
Type name: struct check_struct: \
  Size: 80 bytes; Category: Structure
Fields in type:
{
  field1_int int (32 bits)
  field2_char char (8 bits)
  field2_chars $string[2] (16 bits)
  <padding> $char (8 bits)
  field3_int int (32 bits)
  field2_uchar unsigned char (8 bits)
  <padding> $char[3] (24 bits)
  field_sub struct sub_st (320 bits){
    field1_int int (32 bits)
    <padding> $char[4] (32 bits)
    field2_long long (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)
  }

  ...

  ...
}
**dwhere**

Displays the current execution location and call stack

**Format:**

Displays locations in the call stack

```
dwhere [ –level level-num ] [ num-levels ] [ –args ] [ –locals ] [ –registers ]
          [ –noshow_pc ] [ –noshow_fp ] [ –show_image ]
```

Displays all locations in the call stack

```
dwhere –all [ –args ] [ –locals ] [ –registers ]
          [ –noshow_pc ] [ –noshow_fp ] [ –show_image ]
```

**Arguments:**

- `–level level-num` Set the level at which `dwhere` starts displaying information.
- `num-levels` Restricts output to this number of levels of the call stack. If you omit this option, the CLI shows all levels in the call stack.
- `–args` Displays argument names and values in addition to program location information. If you omit this option, the CLI does not show argument names and values.
- `–locals` Display each frame’s local variables.
- `–noshow_pc` Do not show the PC.
- `–noshow_fp` Do not show the FP.
- `–registers` Displays each frame’s registers.
- `–show_image` Show the executable name as well as the file name.
- `–all` Shows all levels of the call stack.

**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, with the default being to show information at the thread level.

Arguments control the amount of command output in two ways:

- The `num-levels` argument lets you control how many levels of the call stacks are displayed, counting from the uppermost (most recent) level. If you omit this argument, the CLI shows all levels in the call stack, which is the default.
- The `–a` option tells the CLI that it should also display 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 the CLI displays when you do not use the `num-levels` argument.

Output is generated for each thread in the target focus.

**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 p1.5 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.
**dworker**

Adds or removes a thread from a workers group

**Format:**

```
dworker { number | boolean }
```

**Arguments:**

- **number**: If positive, marks the thread of interest as a worker thread by inserting it into the workers group.
- **boolean**: If `true`, marks the thread of interest as a worker thread by inserting it into the workers group. If `false`, marks the thread as being a nonworker thread by removing it from the workers group.

**Description:**

The `dworker` command inserts or removes a thread from the workers group.

If `number` is `0` or `false`, this command marks the thread of interest as a nonworker thread by removing it from the workers group. If `number` is `true` or is a positive value, this command marks the thread of interest 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><code>{dworker false}</code></td>
<td>Removes the focus thread from the workers group</td>
</tr>
<tr>
<td>wot</td>
<td><code>{dworker true}</code></td>
<td>Inserts the focus thread into the workers group</td>
</tr>
</tbody>
</table>
exit

Terminates the debugging session

Format:  exit [ –force ]

Arguments:  –force  Tells TotalView to exit 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 asks to exit. If you answer yes, TotalView exits. If you entered the CLI from the TotalView GUI, this command also closes the GUI window.

If you had 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; they do the same thing.

Examples:  exit  Exits TotalView, leaving any attached processes running.
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. If you do not use an 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. You can press Enter to see more data or enter q to return to the CLI prompt.

When you type a topic name, the CLI attempts to complete what you type. The help command also allows you to 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

...

d1.> he acc
The following commands provide access to the properties of TotalView objects:

You can 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

Tells the CLI that it should close all TotalView processes without asking permission.

**Description:**
The `exit` command terminates the TotalView session. After you enter this command, the CLI asks to exit. If you answer yes, TotalView exits. If you entered the CLI from the TotalView GUI, this command also closes the GUI window.

If you had 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; they do the same thing.

**Examples:**
`quit`

Exits TotalView, leaving any attached processes running.
spurs

Returns information on the spurs library use

**Format:**
Displays help information

```
spurs [ help ]
```

Add directories to the OBJECT_SEARCH_PATH variable

```
spurs add [ directory directory-list ... ]
```

Creates an image-qualified breakpoint

```
spurs break [ spu-image-name source-location-expression ]
```

Deletes breakpoints

```
spurs delete breakpoint-id-list
```

Shows the directories in which TotalView searches for SPURS SPU ELF executables.

```
spurs info [ directory | break ]
```

Prints information about the kernel, the taskset, and tasks

```
spurs print [ kernel [ eaSpurs ] |
  taskset eaTaskset ]
```

```
task eaTaskset taskID ]
```

Scans for information—this is a no-op

```
spurs scan
```

**Arguments:**

directories Names the directories that are to be added to the CLI’s OBJECT_SEARCH_PATH variable. Here’s an example of how you would use this argument:

```
spurs add directory directory1 directory2
```

Notice that the 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 into a particular line in the image. For more information on location expressions, see dbreak on page 31.

eaSpurs The kernel context at this PPU address.

eaTaskset

**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 within the CLI.

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 ]**

You can obtain information on how to use the `spurs` command in two ways:

- Typing `spurs` tells the CLI to write a one line description of its commands.
- Typing `spurs help` displays more information about all `spurs` commands.

**spurs add [ directory directory-list ]**

Either displays a one-line description of this command or adds directories to be searched when TotalView looks for SPURS SPU executables.

- Typing `spurs add` tells the CLI to write a one line description of this command.
- Typing `spurs add directory directory-list` adds directories to the CLI’s OBJECT_SEARCH_PATH variable. This variable contains the search path TotalView uses when it searches for SPU ELF executable files. The directories you are adding are placed at the beginning of the list in the order in which you name them. If a directory is already within the list, the previously named directory is removed.

  This command returns the modified OBJECT_SEARCH_PATH variable.

**spurs break [ spu-image-name source-location-expression ]**

Either displays a one-line description of this command or adds

- Typing `spurs break` tells the CLI to write a one line description of this command.
- Typing `spurs break spu-image-name source-loc-expr` creates an image-qualified breakpoint path. This is identical to typing the following CLI command:

  
  dbreak -pending ##spu-image-name##source-loc-expr

  This command a pending breakpoint that is only located with the image you name. However, if the image has already been loaded, TotalView will set an ordinary breakpoint rather than a pending breakpoint. When entering this command, the focus must be on a 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, enable, and others.

**spurs delete breakpoint-id-list**

Permanently removes one or more action points. The argument to this command tells the CLI 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 ]**
Either prints information of the contents of the OBJECT_SEARCH_PATH variable or prints action point information about action points in the thread in the current focus. If you do not use an argument, the CLI prints a line explaining what you can type.

**spurs print**
The spurs print command can be used in the following ways:

```plaintext
spurs print [ kernel [ eaSpurs ] ]
  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. This 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 task eaTaskset** 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 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.
- **print taskset eaTaskset** Prints the taskset at PPU address `eaTaskset`. The command focus must be one or more PPU threads.
  - `cell_spurs_print_taskset` 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 for scanning 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. This lets you set all of your terminal’s properties. However, this is most often used to set erase and kill characters.

If you start the CLI from a terminal by 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 displays information about your current settings.

The output from this command is returned as a string.

Examples: `stty` Prints information about your terminal settings. The information it prints is the same as if you had entered `stty` while interacting with a shell.

`stty -a` Prints information about all of 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. If you are having problems with command-line editing, use this command. (The `sane` argument is not available in all environments.)
**unalias**

Removes a previously defined alias

**Format:**

```
unalias alias-name
```

Removes all aliases

```
unalias --all
```

**Arguments:**

- `alias-name` The name of the alias to delete.
- `--all` Tells the CLI to remove all aliases.

**Description:**

The `unalias` command removes a previously defined alias. You can delete all aliases by 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.
unalias
This chapter contains detailed descriptions of the CLI commands that are not in the top-level mainspace.

Command Overview

This section lists all of CLI commands that are not in the top-level mainspace. It also contains a short explanation of what each command does.

Accessor 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.
Command Overview

**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.
- **errorCode**: 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 action point.
  - 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.

class 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.

class enabled

A value (either 1 or 0) indicating if the action point is enabled. A value of 1 means enabled. (settable)

class expression

The expression to execute at an action point. (settable)

class id

The ID of the action point.

class language

The language in which the action point expression is written.

class length

The length in bytes of a watched area. This property is only valid for watchpoints. (settable)

class line

The source line at which the action point is set. This property is not valid for watchpoints.

class satisfaction_group

The group that must arrive at a barrier for the barrier to be satisfied. (settable)

class 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)

class 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)

class 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)

class type

The object’s type. (See type_values for a list of possible types.)

class 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
```
**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.
**Format:**

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

**Arguments:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>The action to perform, as follows:</td>
</tr>
<tr>
<td>close</td>
<td>Dynamically unloads the shared object libraries that were dynamically loaded by the <code>ddlopen</code> commands corresponding to the list of dll-ids.</td>
</tr>
<tr>
<td></td>
<td>If you use the <code>-all</code> option, TotalView closes all of the libraries that it opened.</td>
</tr>
<tr>
<td>commands</td>
<td>Displays the subcommands that you can use. The CLI responds by displaying these four <code>action</code> subcommands. There are no arguments to this subcommand.</td>
</tr>
<tr>
<td>get</td>
<td>Retrieves the values of one or more TV::dll properties. The other-args argument can include one or more property names.</td>
</tr>
<tr>
<td></td>
<td>If you use the <code>-all</code> option as the <code>dll-id-list</code>, the CLI returns a list containing one (sublist) element for each object.</td>
</tr>
<tr>
<td>properties</td>
<td>Lists the TV::dll properties that TotalView can access. There are no arguments to this subcommand.</td>
</tr>
<tr>
<td>resolution_urgency_values</td>
<td>Returns a list of values that this property can take. This list is operating-system specific, but always includes <code>{lazy now}</code>.</td>
</tr>
<tr>
<td>symbol_availability_values</td>
<td>Returns a list of values that this property can take. This list is operating system specific, but always includes <code>{lazy now}</code>.</td>
</tr>
<tr>
<td>dll-id-list</td>
<td>A list of one or more dll-ids. There are the IDs returned by the <code>ddlopen</code> command.</td>
</tr>
<tr>
<td>-all</td>
<td>Closes all shared libraries that you opened using the <code>ddlopen</code> command.</td>
</tr>
</tbody>
</table>

**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.
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 tells the CLI to raise 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.
```
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 susp-eval-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 susp-eval-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:

  \[\texttt{dfocus p dfocus [TV::expr get } \texttt{ susp-eval-id } \texttt{ initially_suspended_processes] dwait}\]
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.<>
**focus_groups**

*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.
**focus_processes**

Returns a list of processes in the current focus

**Format:**

\[TV::focus\_processes \[ -all \mid -group \mid -process \mid -thread \]\]

**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.
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:**

```
f p1. < TV::focus_threads
```

Returns a list containing all threads in process 1.
group

Sets and gets group properties

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

Arguments:

- **action**
  - `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 being the number of process groups and the second being the number of thread groups. In many ways, this is related 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.
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.
**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 contain 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:

- `canonical_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_executable_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"
Memory Problems Using TotalView Guide” 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 returned 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 element 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.

**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 sizes of the mappings in the process’s address space.

Examples:

- `f g 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
  ```
**read_symbols**

Reads shared library symbols

**Format:**

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 "Debugging Programs" chapter of the TotalView Users Guide.
respond

Provides responses to commands

**Format:**

```text
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:

```
 dset LINES_PER_SCREEN 0
```

The most common values for `response` are `y` and `n`.

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.

```
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.
**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_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. For example:

```
TV::scope walk $scope_id by_properties \
  kind typedef base_name "^__BMN_.*$"
```

- **–pre_scope tcl_cmd**: Names the commands called before walking a scope.
- **–pre_sym tcl_cmd**: Names the commands called before walking a symbol.
- **–post_scope tcl_cmd**: Names the commands called after walking a scope.
- **–post_symbol tcl_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] \ 
  [required_property_regexp_pair]... \ 
  [non-required_property_regexp_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.
symbol

Gets and sets symbol properties

**Format:**

```latex
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.
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** The following table 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 TotalView Technologies provided will be the ones that you will use.

<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>4</td>
<td>4</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></td>
<td></td>
<td></td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>length</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>array_type</td>
<td>4</td>
<td>4</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>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>lower_bound</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stride_bound</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>block</td>
<td>4</td>
<td></td>
<td>address_class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>artificial</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>length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>location</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>char_type</td>
<td>4</td>
<td>4</td>
<td>artificial</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>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scope_owner</td>
</tr>
<tr>
<td>code_type</td>
<td>4</td>
<td>4</td>
<td>artificial</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>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scope_owner</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>common</td>
<td>4</td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>location</td>
</tr>
<tr>
<td>ds undiscovered_type</td>
<td>4</td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>scope_owner</td>
</tr>
<tr>
<td>enum_type</td>
<td>4</td>
<td>artificial</td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enumerators</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external_name</td>
<td>kind</td>
</tr>
<tr>
<td>error_type</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external_name</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td>file</td>
<td>4</td>
<td>artificial</td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compiler_kind</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>delayed_symbol</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>demangler</td>
<td>language</td>
</tr>
<tr>
<td>float_type</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external_name</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td>function_type</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external_name</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td>image</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>kind</td>
</tr>
<tr>
<td>int_type</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external_name</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td>label</td>
<td>4</td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>location</td>
</tr>
<tr>
<td>linenumber</td>
<td>4</td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>location</td>
</tr>
<tr>
<td>loader_symbol</td>
<td></td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td>member</td>
<td>4</td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>location</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td>module</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>kind</td>
</tr>
<tr>
<td>named_constant</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td>namespace</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</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>common</td>
<td>4</td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>location</td>
</tr>
<tr>
<td>ds_undiscovered_type</td>
<td>4</td>
<td>4</td>
<td>artificial</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>scope_owner</td>
</tr>
<tr>
<td>enum_type</td>
<td>4</td>
<td>4</td>
<td>artificial enumerators</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>value_size</td>
</tr>
<tr>
<td>error_type</td>
<td>4</td>
<td>4</td>
<td>artificial external_name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external_name</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td>file</td>
<td>4</td>
<td>artificial</td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compiler_kind</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>delayed_symbol</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>demangler</td>
<td>language</td>
</tr>
<tr>
<td>float_type</td>
<td>4</td>
<td>4</td>
<td>artificial external_name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external_name</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td>function_type</td>
<td>4</td>
<td>4</td>
<td>artificial external_name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>logical_scope_owner</td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>4</td>
<td>artificial</td>
<td>full_pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>kind</td>
</tr>
<tr>
<td>int_type</td>
<td>4</td>
<td>4</td>
<td>artificial external_name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external_name</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td>label</td>
<td>4</td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td>linenumber</td>
<td>4</td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>location</td>
</tr>
<tr>
<td>loader_symbol</td>
<td></td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td>member</td>
<td>4</td>
<td>address_class</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>artificial</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>location</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inheritance</td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kind</td>
<td>type_index</td>
</tr>
<tr>
<td>module</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>logical_scope_owner</td>
<td></td>
</tr>
<tr>
<td>named_constant</td>
<td>4</td>
<td>4</td>
<td>artificial</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</td>
<td>length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>logical_scope_owner</td>
<td>type_index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>value</td>
</tr>
<tr>
<td>namespace</td>
<td>4</td>
<td>artificial</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full_pathname</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>opaque_type</td>
<td>4</td>
<td>4</td>
<td>artificial</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>logical_scope_owner</td>
</tr>
<tr>
<td>pathname_-.</td>
<td>4</td>
<td></td>
<td>artificial</td>
</tr>
<tr>
<td>reference_symbol</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>lookup_scope</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_-.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>pathname</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scope_owner</td>
</tr>
<tr>
<td>pointer_type</td>
<td>4</td>
<td></td>
<td>artificial</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>length</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>4</td>
<td>4</td>
<td>artificial</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>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>qualification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>target_type_index</td>
</tr>
<tr>
<td>soid_reference_.</td>
<td>4</td>
<td></td>
<td>artificial</td>
</tr>
<tr>
<td>symbol</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>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>resolved_symbol_id</td>
</tr>
<tr>
<td>stringchar_type</td>
<td>4</td>
<td>4</td>
<td>artificial</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>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>target_type_index</td>
</tr>
<tr>
<td>subroutine</td>
<td>4</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></td>
<td></td>
<td></td>
<td>location</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>return_type_index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>static_chain</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>static_chain_height</td>
</tr>
<tr>
<td>typedef</td>
<td>4</td>
<td>4</td>
<td>artificial</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>length</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>target_type_index</td>
</tr>
<tr>
<td>variable</td>
<td>4</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></td>
<td></td>
<td></td>
<td>location</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>is_argument</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>location</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical_scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ordinal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scope_owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>type_index</td>
</tr>
<tr>
<td>void_type</td>
<td>4</td>
<td>4</td>
<td>artificial</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>length</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>wchar_type</td>
<td>4</td>
<td>4</td>
<td>artificial</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>length</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>target_type_index</td>
</tr>
</tbody>
</table>

The figure on the following page 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 1: Data Addressing*

```plaintext
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.

**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.
symbol

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 2: 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 3: Qualification

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_unexpected_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. The figure on the next page illustrates how these namespaces are related.

The following table lists the properties associated with a namespace.

<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></td>
<td>loader_name</td>
</tr>
<tr>
<td>c_global_symname</td>
<td>base_name</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>type_index</td>
</tr>
<tr>
<td>cplus_global_symname</td>
<td>base_name</td>
</tr>
<tr>
<td></td>
<td>cplus_class_name</td>
</tr>
<tr>
<td></td>
<td>cplus_local_name</td>
</tr>
<tr>
<td></td>
<td>cplus_overload_list</td>
</tr>
<tr>
<td></td>
<td>cplus_template_types</td>
</tr>
<tr>
<td></td>
<td>cplus_type_name</td>
</tr>
<tr>
<td></td>
<td>loader_file_path</td>
</tr>
<tr>
<td></td>
<td>loader_name</td>
</tr>
<tr>
<td>cplus_local_symname</td>
<td>base_name</td>
</tr>
<tr>
<td></td>
<td>cplus_class_name</td>
</tr>
<tr>
<td></td>
<td>cplus_local_name</td>
</tr>
<tr>
<td></td>
<td>cplus_overload_list</td>
</tr>
<tr>
<td></td>
<td>cplus_template_types</td>
</tr>
<tr>
<td></td>
<td>cplus_type_name</td>
</tr>
<tr>
<td></td>
<td>loader_file_path</td>
</tr>
<tr>
<td></td>
<td>loader_name</td>
</tr>
<tr>
<td>cplus_type_symname</td>
<td>base_name</td>
</tr>
<tr>
<td></td>
<td>cplus_class_name</td>
</tr>
<tr>
<td></td>
<td>cplus_local_name</td>
</tr>
<tr>
<td></td>
<td>cplus_overload_list</td>
</tr>
<tr>
<td></td>
<td>cplus_template_types</td>
</tr>
<tr>
<td></td>
<td>cplus_type_name</td>
</tr>
<tr>
<td></td>
<td>type_index</td>
</tr>
<tr>
<td>file_symname</td>
<td>base_name</td>
</tr>
<tr>
<td></td>
<td>directory_path</td>
</tr>
</tbody>
</table>
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 looking at the following function prototype:

```c
void c::foo<int>(int &)
```

<table>
<thead>
<tr>
<th>Symbol Namespaces</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>fortran_global_symname</td>
<td>base_name</td>
</tr>
<tr>
<td></td>
<td>fortran_module_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>
<tr>
<td></td>
<td>fortran_module_name</td>
</tr>
<tr>
<td>fortran_type_symname</td>
<td>base_name</td>
</tr>
<tr>
<td></td>
<td>fortran_module_name</td>
</tr>
<tr>
<td></td>
<td>fortran_parent_function_name</td>
</tr>
<tr>
<td></td>
<td>type_index</td>
</tr>
<tr>
<td>image_symname</td>
<td>base_name</td>
</tr>
<tr>
<td></td>
<td>directory_path</td>
</tr>
<tr>
<td></td>
<td>member_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>load_file_path</td>
</tr>
<tr>
<td></td>
<td>loader_name</td>
</tr>
<tr>
<td>module_symname</td>
<td>base_name</td>
</tr>
<tr>
<td>type_symname</td>
<td>type_index</td>
</tr>
</tbody>
</table>

The name of the function; for example, `foo`.

The C++ class name; for example, `c`.

Not used.

The function’s signature; for example, `int &`.

The template used to instantiate the function; for example: `<int>`.

The data type of the returned value; for example, `void`.

The directory to which you were attached when you started TotalView.

Your file’s pathname as it is named within your program.

The name of your module. Typically, this looks like `module`'var or `module`'sub'var.

The parent of the subroutine. For example, the parent is `module` in a reference such as `module'sub`. If you have an inner subroutine, the parent is the outer subroutine.

The line number at which something occurred.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loader_file_path</td>
<td>The file’s pathname.</td>
</tr>
<tr>
<td>loader_name</td>
<td>The mangled name.</td>
</tr>
<tr>
<td>member_name</td>
<td>In a library, you might have an object reference; for example, <code>libC.a(foo.so)</code>. <code>foo.so</code> is the member name.</td>
</tr>
<tr>
<td>node_name</td>
<td>Not used.</td>
</tr>
<tr>
<td>type_index</td>
<td>A handle that points to the type definition. It’s format is <code>&lt;number,number,number&gt;</code>.</td>
</tr>
</tbody>
</table>
**thread**

**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 finds 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:**
```
$ 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 \n    [f t TV::focus_threads] continue_sig $val
}
```
Set the starting signal for the focus thread.

```
proc show_signal {} {
    foreach th [TV::focus_threads] {
        puts "Continue_sig ($th): \n        [TV::thread get $th continue_sig]";
    }
}
```
Show all starting signals
**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. If you do not apply a command that uses this argument 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:

```
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:

```
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`, 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 crate 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:

- **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 `id` is the symbol ID of the symbol that was validated using the `validate_callback`’s procedure.
  - This callback defines a TotalView addressing expression that computes the starting address of an array’s first element.
### Type Transformation

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>compiler</strong></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><strong>id</strong></td>
<td>The type transformation ID returned from a <code>create</code> operation.</td>
</tr>
<tr>
<td><strong>language</strong></td>
<td>The language property specifies source language for the code of the aggregate type (class) to transform. This is always C++.</td>
</tr>
<tr>
<td><strong>list_element_count_addressing_callback</strong></td>
<td>Names the procedure that determines the total number of elements in a list. The call structure for this callback is:</td>
</tr>
<tr>
<td></td>
<td><code>list_element_count_addressing_callback id</code></td>
</tr>
<tr>
<td></td>
<td>where <code>id</code> is the symbol ID of the symbol that was validated using the <code>validate_callback</code>'s procedure.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>If your data structure does not have this element, you still must use this callback. In this case, simply return <code>{nop}</code> as the addressing expression and the transformation will count the elements by following all the pointers. This can be very time consuming.</td>
</tr>
<tr>
<td><strong>list_element_data_addressing_callback</strong></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><code>list_element_data_addressing_callback id</code></td>
</tr>
<tr>
<td></td>
<td>where <code>id</code> is the symbol ID of the symbol that was validated using the <code>validate_callback</code>'s procedure.</td>
</tr>
<tr>
<td><strong>list_element_next_addressing_callback</strong></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><code>list_element_next_addressing_callback id</code></td>
</tr>
<tr>
<td></td>
<td>where <code>id</code> is the symbol ID of the symbol that was validated using the <code>validate_callback</code>'s procedure.</td>
</tr>
<tr>
<td><strong>list_element_prev_addressing_callback</strong></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><code>list_element_prev_addressing_callback id</code></td>
</tr>
<tr>
<td></td>
<td>where <code>id</code> is the symbol ID of the symbol that was validated using the <code>validate_callback</code>'s procedure.</td>
</tr>
<tr>
<td></td>
<td>This property is optional. For example, you would not use it in a singly linked list.</td>
</tr>
<tr>
<td><strong>list_end_value</strong></td>
<td>Specifies if a list is terminated by NULL or the head of the list. Enter one of the following: <code>NULL</code> or <code>ListHead</code></td>
</tr>
</tbody>
</table>
list_first_element_addressing_callback
Names the procedure that defines an addressing
expression that specifies how to go from the head ele-
ment 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 call-
back is:

    list_element_first_element_addressing_callback id
where id is the symbol ID of the symbol that was vali-
dated using the validate_callback's 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 vali-
dated using the validate_callback's procedure.

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:

    lower_bounds_callback id
where id is the symbol ID of the symbol that was vali-
dated using the validate_callback's procedure.

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 transfor-
mation, 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 vali-
dated using the validate_callback's 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 trans-
formed; for example, you might enter "GNU Vector".
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:

`upper_bounds_callback id`

where `id` is the symbol ID of the symbol that was validated using the `validate_callback`'s procedure.

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 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 data type. The callback procedure must return a Boolean value, where `true` means the symbol is valid and can be transformed.
TotalView can run unattended if you start it using the `tvscript` command. This is called batch debugging because, like all batch programs, you do not need to use them interactively. In addition, you can invoke `tvscript` using `cron` to schedule debugging in the evening. In this way, you’ll have reports waiting for you in the morning.

The actions that `tvscript` can perform can be quite complicated if you include these actions in a script file because this script file can also 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 "nowshow_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
```

All these `tvscript` command-line options (and others) are explained in the remainder of this chapter.
The syntax for the `tvscript` command is:

```
tvscript [ options ] [ filename ] [ –a program_args ]
```

`options` TotalView and `tvscript` command-line options. You can use any of the options described in Chapter 7, "TotalView Command Syntax," on page 241. `tvscript` command-line options are described in the next section.

`filename` The program being debugged.

`–a program_args` Arguments that TotalView sends to your program.

The MPI compilers that you can run under `tvscript` control are those that you see when you select an MPI with the list in the Parallel page of the `File > New Program` dialog box. The command-line options that are most often used with `tvscript` are:

- `–mpi`
- `–starter_args`
- `–nodes`
- `–np` or `–procs` or `–tasks`

You will find more information on these command-line options in Chapter 7, "TotalView Command Syntax," on page 241.

You can run the following MPIs using `tvscript`:

- Intel MPI
- LAM
- MPICH2
- Open MPI
- OSU MVAPICH2
- poe on AIX and Linux
- Sun CT7
- Sun MPI

### `tvscript` Options

- `–create_actionpoint "source_location_expr [ = > action1 [, action2] ... ]"`  
  Creates an action point at a source location using an expression. (These expressions are discussed in "Action Point API" on page 181.) When this action point is hit, `tvscript` can trigger one or more actions. You will need to add one `–create_watchpoint` command-line option for each action point being set.

  See `–event_action` for information about actions.
tvscript Command Syntax

Performs an action when an event occurs. Events represent a condition that wasn’t anticipated. For example, `free_not_allocated` in the Memory Debugger. You can use more than one `-event_action` command-line option on the command line that invokes `tvscript`.

Here is how you enter an `event_action_list`:

```
event1 = action1, event2 = action2
```

or

```
event1 = >action1 , action2, action3
```

The following table lists the names of events that you can use:

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>General event</td>
<td><code>any_event</code></td>
<td>A generated event occurred.</td>
</tr>
<tr>
<td>Memory debugging event</td>
<td><code>addr_not_at_start</code></td>
<td>Program attempted to free a block using an incorrect address.</td>
</tr>
<tr>
<td></td>
<td><code>alloc_not_in_heap</code></td>
<td>The memory allocator returned a block not in the heap; the heap may be corrupt.</td>
</tr>
<tr>
<td></td>
<td><code>alloc_null</code></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><code>alloc_returned_bad_alignment</code></td>
<td>The memory allocator returned a misaligned block; the heap may be corrupt.</td>
</tr>
<tr>
<td></td>
<td><code>any_memory_event</code></td>
<td>A memory event occurred.</td>
</tr>
<tr>
<td></td>
<td><code>bad_alignment_argument</code></td>
<td>Program supplied an invalid alignment argument to the heap manager.</td>
</tr>
<tr>
<td></td>
<td><code>double_alloc</code></td>
<td>The memory allocator returned a block currently being used; the heap may be corrupt.</td>
</tr>
<tr>
<td></td>
<td><code>double_dealloc</code></td>
<td>Program attempted to free an already freed block.</td>
</tr>
<tr>
<td></td>
<td><code>free_not_allocated</code></td>
<td>Program attempted to free an address that is not in the heap.</td>
</tr>
<tr>
<td></td>
<td><code>guard_corruption</code></td>
<td>Program overwrote the guard areas around a block.</td>
</tr>
<tr>
<td></td>
<td><code>hoard_low_memory_threshold</code></td>
<td>Hoard low memory threshold crossed.</td>
</tr>
</tbody>
</table>
When an event occurs, you can tell TotalView what actions it should perform:

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory debugging actions</td>
<td>realloc_not_allocated</td>
<td>Program attempted to reallocate an address that is not in the heap.</td>
</tr>
<tr>
<td></td>
<td>rz_overrun</td>
<td>Program attempted to access memory beyond the end of an allocated block.</td>
</tr>
<tr>
<td></td>
<td>rz_underrun</td>
<td>Program attempted to access memory before the start of an allocated block.</td>
</tr>
<tr>
<td></td>
<td>rz_use_after_free</td>
<td>Program attempted to access a block of memory after it has been deallocated.</td>
</tr>
<tr>
<td></td>
<td>rz_use_after_free_overrun</td>
<td>Program attempted to access memory beyond the end of a deallocated block.</td>
</tr>
<tr>
<td></td>
<td>rz_use_after_free_underrun</td>
<td>Program attempted to access memory before the start of a deallocated block.</td>
</tr>
<tr>
<td></td>
<td>termination_notification</td>
<td>The target is terminating.</td>
</tr>
</tbody>
</table>

| Source code debugging event | actionpoint | A thread hit an action point. |
|                            | error       | An error occurred.           |

When an event occurs, you can tell TotalView what actions it should perform:

<table>
<thead>
<tr>
<th>Action Type</th>
<th>Action</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory debugging actions</td>
<td>check_guard_blocks</td>
<td>Check all guard blocks and write violations into the log file.</td>
</tr>
<tr>
<td></td>
<td>list_allocations</td>
<td>Write a list of all memory allocations into the log file.</td>
</tr>
<tr>
<td></td>
<td>list_leaks</td>
<td>Write a list of all memory leaks into the log file.</td>
</tr>
<tr>
<td></td>
<td>save_memory_debugging_file</td>
<td>Generate and save a memory debugging file.</td>
</tr>
<tr>
<td></td>
<td>save_text_heap_status_source_view</td>
<td>Generate and save a text version of the Heap Status Source View Report.</td>
</tr>
</tbody>
</table>
**Action Type** | **Action** | **Definition**
--- | --- | ---
Source code debugging actions | `display_backtrace` | Write the current stack backtrace into the log file.
--- | `-level level-num` | 
--- | `-level num_levels` | 
--- | `options` | 
--- | `print` | Write the value of a variable or an expression into the log file. If the variable is an array, the `-slice` option tells TotalView to limit the amount of data to that defined by `slice_exp`. A slice expression is a way to define the slice such as `var[100:130]` in C and C++. (This tells TotalView to display all values from `var[100]` to `var[130]`.) If you wanted to display every fourth value, you’d add an additional argument; for example, `var[100:130:4]`. For additional information, see Chapter 14 “Examining Arrays” in the TotalView Users Guide.
--- | `-slice {slice_exp}` | 
--- | `{variable | exp}` | 

By default, `tvscript` writes all of the information in the following table to the log file. You can exclude informa-
Eliminate memory debugging and memory event notification by using one of the following specifiers. You can eliminate:

<table>
<thead>
<tr>
<th>Type of Specifier</th>
<th>Specifier</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>General display specifiers</td>
<td>noshow_fp</td>
<td>Do not show the frame pointer (FP)</td>
</tr>
<tr>
<td></td>
<td>noshow_image</td>
<td>Do not show the process/library in backtrace</td>
</tr>
<tr>
<td></td>
<td>noshow_pc</td>
<td>Do not show the program counter (PC)</td>
</tr>
<tr>
<td></td>
<td>noshow_pid</td>
<td>Do not show the system process ID with process information</td>
</tr>
<tr>
<td></td>
<td>noshow_rank</td>
<td>Do not show the rank of a process, which is only shown for a parallel process</td>
</tr>
<tr>
<td></td>
<td>noshow_tid</td>
<td>Do not show the thread ID with process information</td>
</tr>
<tr>
<td>Memory debugging display specifiers</td>
<td>noshow_allocator</td>
<td>Do not show the allocator for the address space</td>
</tr>
<tr>
<td></td>
<td>noshow_backtrace</td>
<td>Do not show the backtraces for memory blocks</td>
</tr>
<tr>
<td></td>
<td>noshow_backtrace_id</td>
<td>Do not show the backtrace ID for memory blocks</td>
</tr>
<tr>
<td></td>
<td>noshow_block_address</td>
<td>Do not show the memory block start and end addresses</td>
</tr>
<tr>
<td></td>
<td>noshow_flags</td>
<td>Do not show the memory block flags</td>
</tr>
<tr>
<td></td>
<td>noshow_guard_id</td>
<td>Do not show the guard ID for memory blocks</td>
</tr>
<tr>
<td></td>
<td>noshow_guard_settings</td>
<td>Do not show the guard settings for memory blocks</td>
</tr>
<tr>
<td></td>
<td>noshow_leak_stats</td>
<td>Do not show the leaked memory block statistics</td>
</tr>
<tr>
<td></td>
<td>noshow_owner</td>
<td>Do not show the owner of the allocation</td>
</tr>
<tr>
<td></td>
<td>noshow_red_zones_settings</td>
<td>Do 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. You must use this option if you use any of the options that begin 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.
tvscript Command Syntax

-\texttt{-mem\_guard\_blocks}
  Adds guard blocks to an allocated memory block.

-\texttt{-mem\_hoard\_freed\_memory}
  Holds onto freed memory rather than return it to the heap.

-\texttt{-mem\_hoard\_low\_memory\_threshold nnnn}
  Sets the low memory threshold amount. When memory falls below this amount an event will be fired.

-\texttt{-mem\_paint\_all}
  Paints memory blocks with a bit pattern when a memory is allocated and when it is deallocated.

-\texttt{-mem\_paint\_on\_alloc}
  Paints memory blocks with a bit pattern when a memory block is allocated.

-\texttt{-mem\_paint\_on\_dealloc}
  Paints memory blocks with a bit pattern when a memory block is deallocated.

-\texttt{-mem\_red\_zones\_overruns}
  Turns on testing for Red Zone overruns.

-\texttt{-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:
  \begin{itemize}
  \item x:y allocations from x to y
  \item :y allocations from 1 to y
  \item x: allocations of x and higher
  \item x allocation of x
  \end{itemize}

-\texttt{-mem\_red\_zones\_underruns}
  Turns on testing for Red Zone underruns.

-\texttt{-maxruntime "hh:mm:ss"}
  Specifies how long the script can run.

-\texttt{-script\_file script\_file}
  Names a file containing \texttt{tvscript} API calls and Tcl callback procedures that you create. See for more information.

**tvscript Example:** The following example is similar to shown in “Batch Debugging Using tvscript” on page 173.

\begin{verbatim}
tvscript \  -create\_actionpoint "method1=>display\_backtrace \  -show\_arguments" \  -create\_actionpoint "method2#37=>display\_backtrace \  -show\_locals -level 1" \  -event\_action "error=>display\_backtrace \  -show\_arguments -show\_locals" \  -display\_specifiers "noshow\_pid,noshow\_tid" \  -maxruntime "00:00:30" \  filterapp -a 20
\end{verbatim}

This script tells \texttt{tvscript} to perform the following actions:
Create an actionpoint at the beginning of method1. When tvscript reaches that breakpoint, it logs a backtrace and the method’s arguments.

Create an actionpoint 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.

Log the backtrace, the current routine’s arguments, and its local variables when an error event occurs.

Exclude the process ID and thread ID from the information that tvscript logs.

Limit tvscript execution time to 30 sections.

Name the program being debugged and passes a value of 20 to the application.

---

The previous section discussed the command-line options that you use when invoking the tvscript command. A second method is to place commands in a file and feed these commands into tvscript using the --script_file command-line option. Within this script file, you can use 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
```
Log a message to the log file set up by tvscript.

```
tvscript_slog msg
```
Log a message to the summary log file set up by tvscript.

Process Functions API

```
tvscript_get_process_property process_id property
```
Get 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 145 for more information.

Thread Functions API

```
tvscript_get_thread_property thread_id property
```
Get 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 165 for more information.
### Action Point API

**tvscript_add_actionpoint_handler**  
*actionpoint_id actionpoint_handler*

Registers a procedure handler that is called when the action point associated with *actionpoint_id* is hit. This *actionpoint_id* is the value that is 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 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**

Set 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 that are 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 which 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 that is called 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 182.

```
# Get the process so we have some information about it
tvscript_log "PID: "
    [tvscript_get_process_property 1 "syspid"];
tvscript_log "Status: "
    [tvscript_get_process_property 1 "state"];
tvscript_log "Executable: "
    [tvscript_get_process_property 1 "executable"];

#############################################################
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 ll_actionpoint_handler {event_data} {
    tvscript_log "Inside ll_actionpnt_handler: $event_data"
    tvscript_slog "Inside ll_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

# 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"
This chapter contains a list of all CLI and TotalView variables. This chapter has three sections, each corresponding to a CLI namespace, as follows:

- **Top-Level (::) Namespace**
- **TV:: Namespace**
- **TV::MEMDEBUG:: Namespace**
- **TV::GUI:: Namespace**

### Top-Level (::) Namespace

**ARGS(dpid)**

Contains the arguments that TotalView passes to the process with TotalView ID *dpid* the next time you start the process.

- **Permitted Values:** A string
- **Default:** None

**ARGS_DEFAULT**

Contains the argument passed to a new process when no ARGSD(p) 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 tells TotalView what else it should stop when a barrier point is satisfied. This property also tells TotalView what else it should stop when 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 that you can use are as follows:
Top-Level (::) Namespace

**group**

TotalView will stop all processes in a thread’s control group when a thread reaches a barrier created using this as a default.

**process**

TotalView will stop the process in which the thread is running when a thread reaches a barrier created using this default.

**none**

TotalView just stops 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 that TotalView uses when a barrier point is satisfied. You can also set this value if you use 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 as follows:

**group**

When a barrier is satisfied, TotalView stops all processes in the control group.

**process**

When a barrier is satisfied, TotalView stops the processes in the satisfaction set.

**none**

TotalView only stops 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(dpid)**

Contains the control group for the process with the TotalView ID dpid. Setting this variable moves process dpid 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 that you can use while editing text in the CLI. These editing commands are always available in the CLI window of the TotalView GUI. However, they are only available in the stand-alone CLI if the terminal in which you are running it 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

EXECUTABLE_PATH

Contains a colon-separated list containing the directories that TotalView searches when it looks 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 you enter is the delimiter. This example uses a "+" as a delimiter. (Traditionally, forward slashes are used as delimiters. This is not a good idea as a forward slash is also used to separate components of a a pathname. For example, /home/my_dir contains forward slashes.)

When you enter information, be careful when you include special characters. When you do, these characters must follow standard Tcl rules and conventions.

Here is an example:

```
dset EXECUTABLE_SEARCH_MAPPINGS
{
+^/nfs/compiled/u2/(.*)$+ = +/nfs/host/u2/\1+
}
```

This expression tells TotalView that if it finds a directory named /nfs/compiled/u2/project/src1 in the expanded search path, it will become /nfs/host/u2/project/src1 after applying this mapping.

Default: { }
EXECUTABLE_SEARCH_PATH

Contains a list of paths, with paths separated by a colon. that TotalView searches when it looks for your executables. For information, consult the online help.

**Permitted Values:** Any directory or directory path.
**Default:** ${EXECUTABLE_PATH}:${PATH}:

GROUP(gid)

Contains a list containing the TotalView IDs for all members in group `gid`.

The first element in the list indicates what kind of group it is, as follows:

- **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 will not display lockstep groups when you use `dset` with no arguments—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 that contains 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**: This is 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 will 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 will display.

- **Permitted Values:** A positive integer
- **Default:** 512

**MAX_LIST**
Defines the number of lines that the `dlist` command will display.

- **Permitted Values:** A positive integer
- **Default:** 20

**OBJECT_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. For more information, see `EXECUTABLE_SEARCH_MAPPINGS`.

- **Default:** `{}`

**OBJECT_SEARCH_PATH**
Contains a list of paths, with paths separated by a colon, that TotalView searches when it looks for your program’s object files. For information, consult the online 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. If you use brackets ([ ]) in the prompt, TotalView assumes the information within the brackets is a Tcl command and evaluates this information before it creates the prompt string.

- **Permitted Values:** Any string. If you wish to access the value of `PTSET`, you must 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. TotalView decides which share group this is by looking at the control group for the process and the executable associated with this process. You cannot directly modify this group.

Permitted Values: A number
Default: None

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 this Boolean value is true, newly created action point are shared across the group. If it is false, a newly created action point is only active 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 using selecting the Plant in share group checkbox 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, that TotalView searches when it looks 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—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. For more information, see EXECUTABLE_SEARCH_MAPPINGS.

Default: {}

SOURCE_SEARCH_PATH

Contains a list of paths, with paths separated by a colon, that TotalView searches when it looks for your program’s source files. For information, consult the online 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. This property tells TotalView what else it should stop when it encounters this action point. The values you can set are as follows:

- **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**: `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 in which the TotalView executable is located. This is a read-only variable and cannot be set. This variable is exported as `TVROOT`, and is can be used in launch strings.

**Permitted Values**: The location of the TotalView installation directory

**TOTALVIEW_TCLLIB_PATH**
Contains a list containing 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 string
**Default**: Platform-specific

**VERBOSE**
Controls the error message information displayed by the CLI. The values for this variable can be:

- **info**: Prints errors, warnings, and informational messages. Informational messages include data on dynamic libraries and symbols.
- **warning**: Only print errors and warnings.
- **error**: Only print error messages.
TV:: Namespace

silent

Does not print error, warning, and informational messages. This also shuts off the printing of results from CLI commands. This should only be used 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 that you had 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, for 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 set to true, TotalView might ask you if you want to stop the process when a Cell SPU image is loaded.

If set to false, TotalView does not stop execution when a Cell SPU image is loaded.

Permitted Values: true or false
Default: true

TV::ask_on_dlopen

Setting this variable to true tells TotalView that it should ask you about stopping processes that use the dlopen or load (AIX only) system calls dynamically load a new shared library.
If this is set to false, TotalView will 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 that are displayed when the **TV::auto_array_cast_enabled** variable is set to 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 this is set to true, TotalView will automatically dereference 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**

Tells TotalView if and how it should dereference C and C++ pointers when you perform a View > Dive in All operation, as follows:

- **yes_dont_push**  
  While automatic dereferencing will occur, you can’t use the Undive command to see the undereferenced value when performing a Dive in All operation.

- **yes**  
  You will be able to use the Undive control to see undereferenced values.

- **no**  
  Do 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, you can’t use the Undive command to see the undereferenced value when performing a Dive in All operation.

- **yes**  
  You will be able to use the Undive control to see undereference values.

- **no**  
  Do 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, you can’t use the Undive command to see the undereferenced value.

- **yes**
  - You will be able to use the Undive control to see undereferenced values.

- **no**
  - Do 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**

Tells TotalView if and how it should dereference Fortran pointers when they are displayed, as follows:

- **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 undereferenced values.

- **no**
  - Do 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 undereferenced 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**

Tells TotalView if and how it should dereference Fortran pointers when they are displayed, as follows:

- **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 undereferenced 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_load_breakpoints**

Setting this variable to true tells TotalView that it should automatically load action points from the file named filename.TVD.v3breakpoints where filename is the name of the file being debugged. If the variable is set to false, TotalView does not automatically load your breakpoints. If you set this to false, you can still load breakpoints if you use the Action Point > Load All or the dactions –load command.

**Permitted Values:** true or false
**Default:** true

**TV::auto_read_symbols_at_stop**

Setting this variable to false tells TotalView that it should not automatically read symbols if execution stops when the program counter is in a library whose symbols were not read. Setting it to true tells TotalView that it should read 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**

Setting this variable to true tells TotalView that it should automatically write 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 this variable is set 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 for the “stop_all” property for newly created action points. This property tells TotalView what else it should 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**

TotalView will stop all processes in a thread’s control group when a thread reaches a barrier created using this as a default.

**process**

TotalView will stop the process in which the thread is running when a thread reaches a barrier created using this default.
thread TotalView just stops 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 tells TotalView what else it should stop when a barrier point is satisfied. You can also set this value if you use 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 as follows:

- group: When a barrier is satisfied, TotalView stops all processes in the control group.
- process: When a barrier is satisfied, TotalView stops the processes in the satisfaction set.
- thread: TotalView only stops 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_end_hostname-io

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 TotalView uses when it launches tvdsrv 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 that TotalView waits to hear back from the TotalView Debugger Server (tvdsrv) that it launches.

Permitted Values:  An integer from 1 to 3600 (1 hour)  
Default:  240

TV::bulk_launch_base_timeout  
Defines the base timeout period used when TotalView executes a bulk launch.
**TV::Namespace**

**TV::bulk_launch_enabled**

When this is set to `true`, tells TotalView that it should use its bulk launch features when it automatically launches 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 that TotalView waits for process to launch when it automatically launches 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_string**

Defines the command that will be used to launch the TotalView Debugger Server (`tvdsvr`) when remote processes are created. For information on this launch string, see "Replacement Characters" on page 257.

- **Permitted Values:** A string, usually contained within braces `{}`
- **Default:** The default value depends upon the platform—use the `dset` command to see what this default is

**TV::bulk_launch_tmpfile1_header_line**

Defines the header line used in the first temporary file when TotalView does a bulk server launch operation. For information on this launch string, see "Replacement Characters" on page 257.

- **Permitted Values:** A string
- **Default:** None

**TV::bulk_launch_tmpfile1_host_lines**

Defines the host line used in the first temporary file when TotalView performs a bulk server launch operation. For information on this launch string, see "Replacement Characters" on page 257.

- **Permitted Values:** A string
- **Default:** %R

**TV::bulk_launch_tmpfile1_trailer_line**

Defines the trailer line used in the first temporary file when TotalView performs a bulk server launch operation. For information on this launch string, see "Replacement Characters" on page 257.

- **Permitted Values:** A string
- **Default:** None

**TV::bulk_launch_tmpfile2_header_line**

Defines the header line used in the second temporary file when TotalView performs a bulk server launch operation. For information on this launch string, see "Replacement Characters" on page 257.

- **Permitted Values:** A string
- **Default:** None
### TV::bulk_launch_tmpfile2_host_lines
Defines the host line used in the second temporary file when TotalView does a bulk server launch operation. For information on this launch string, see "Replacement Characters" on page 257.

- **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 TotalView does a bulk server launch operation. For information on this launch string, see "Replacement Characters" on page 257.

- **Permitted Values:** A string
- **Default:** None

### TV::c_type_strings
When set to `true`, TotalView uses C type string extensions when it displays character arrays. When `false`, TotalView uses its 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. If it matches, TotalView does not ask if you want to stop the process and 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_spu_images_stop_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. If it matches, TotalView asks if you want to stop the process.

If the image name does not match this regular expression or the regular expression contained within `TV::cell_spu_images_ignore_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
Contains a strings that names the file that contain 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 it does, 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

Contains a string that names the file that contains 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

Contains a string that names the file that contains 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: auto
Default: auto

TV::comline_patch_area_base

Allocates the patch space dynamically at the given address. See “Allocating Patch Space for Compiled Expressions” in Chapter 15 of the TotalView 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 Chapter 15 of the TotalView Users Guide.

Permitted Values: A positive number
Default: 0

TV::command_editing

Enables some Emacs-like commands that you can use while editing text in the CLI. These editing commands are always available in the CLI window of TotalView GUI. However, they are only available within the stand-alone CLI.
When the terminal in which you are running it 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.

TV::compile_expressions

When this variable is set to true, TotalView enables compiled expressions. If this is set to false, TotalView interprets your expression.

If you are running on an IBM AIX system, you can use the –use_aix_fast_trap command line option to speed up the performance of compiled expressions. Check the TotalView Release Notes to determine if your version of the operating system supports this feature.

Permitted Values: true or false
Default: false

TV::compiler_vars

(HP Alpha, HP and SGI only) When this variable is set to true, TotalView shows variables created by your Fortran compiler as well as the variables in your program. When set to false (which is the default), TotalView does not show the variables created by your compiler.

Some Fortran compilers (HP f90/f77, SGI 7.2 compilers) write debugging information that describes variables that 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 by using the following 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 this variable is set to **true**, TotalView kills attached processes and exits. When set to false, TotalView can sometimes 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.

If you set the **TV::ignore_control_c** variable to **true**, TotalView ignores this variable.

*Permitted Values:* true or false

*Default:* true

**TV::copyright_string**

This is a read-only string containing the copyright information displayed when you start the CLI and TotalView.

**TV::current_cplus_demangler**

Setting this variable overrides the C++ demangler that TotalView uses. TotalView will ignore what you set the value of this variable to unless you also set the value of the **TV::force_default_cplus_demangler** variable. You can set this variable to the following values:

- compaq: HP cxx on running Linux-Alpha
- dec: HP Tru64 C++
- gnu: GNU C++ on Linux Alpha
- gnu_dot: GNU C++ Linux x86
- gnu_v3: GNU C++ Linux x86
- hp: HP aCC compiler
- irix: SGI IRIX C++
- 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 that TotalView uses. TotalView will ignore what you set the value of this variable to unless you also set the value of the **TV::force_default_f9x_demangler** variable. You can set this variable to the following values:

- xlf90: IBM Fortran
- dec: HP Tru64 Fortran
- decf90: HP Tru64 Fortran 90
- fujitsu_f9x: Fujitsu Fortran 9x
- hpux11_64_f9x: HP Fortran 9x
- intel: Intel Fortran 9x
- mipspro_f9x: SGI IRIX Fortran
- 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 TotalView uses when displaying double-precision values. This is one of a series of variables that define how TotalView displays data. The format of each is similar and is as follows:

\{presentation format-1 format-2 format-3\}

*presentation*

Selects which format TotalView uses 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 will either be **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 that is 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 that is used in the `printf()` function.
# (pound) Tells TotalView to display a 0x prefix for hexadecimal and 0 for octal formats. This isn’t used within floating-point formats.

0 (zero) Tells TotalView to pad 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) Tells TotalView to left-justify 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 TotalView uses 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 TotalView uses 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 TotalView uses 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 TotalView uses 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 TotalView uses 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 TotalView uses 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.
**TV::data_format_single**
Defines the format TotalView uses 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::data_format_long_stringlen**
Defines the maximum number of characters displayed for a long string.

- **Permitted Values:** A positive integer number
- **Default:** 8000

**TV::dbfork**
When this variable is set to **true**, TotalView catches the `fork()`, `vfork()`, and `execve()` system calls if your executable is linked with the `dbfork` library.

- **Permitted Values:** **true** or **false**
- **Default:** **true**

**TV::default_stderr_append**
When this variable is set to **true**, TotalView appends `stderr` information to the file set in the GUI or in the **TV::default_stderr_filename** variable. If no pathname is set, the value of this variable is ignored.

- **Permitted Values:** **true** or **false**
- **Default:** **false**

**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:** SiCortex: `ssh -x`
  HPUX: `remsh`
  All other platforms: `rsh`

**TV::default_stderr_filename**
Names the file to which TotalView writes `stderr` information.

- **Permitted Values:** A string indicating a pathname
- **Default:** None

**TV::default_stderr_is_stdout**
When this variable is set to true, TotalView writes `stderr` information to the same location as it writes `stdout` information.

- **Permitted Values:** **true** or **false**
- **Default:** **false**

**TV::default_stdin_filename**
Names the file from which TotalView reads `stdin` information.

- **Permitted Values:** A string indicating a pathname
- **Default:** None
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Permitted Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TV::default_stdout_append</td>
<td>When this variable is set to <code>true</code>, TotalView appends <code>stdout</code> information to the file set in the GUI or in the TV::default_stdout_filename variable. If no pathname is set, the value of this variable is ignored.</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>TV::default_stdout_filename</td>
<td>Names the file to which TotalView writes <code>stdout</code> information.</td>
<td>A string indicating a pathname</td>
<td>None</td>
</tr>
<tr>
<td>TV::display_assembler_symbolically</td>
<td>When this variable is set to <code>true</code>, TotalView displays assembler locations as label+offset. When it is set to <code>false</code>, these locations are displayed as hexadecimal addresses.</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>TV::dll_ignore_prefix</td>
<td>Defines a list of library files for which you are not asked to stop the process when it is loaded. This list contains a colon-separated list of prefixes. Also, TotalView will not ask if you would like to stop a process if:</td>
<td>A list of path names, each item of which is separated from another by a colon</td>
<td>/lib:/usr/lib:/usr/lpp:/usr/ccs/lib:/usr/dt/lib:/tmp/</td>
</tr>
<tr>
<td>TV::dll_read_all_symbols</td>
<td>TotalView will always read loader and debugging symbols of libraries named within this variable.</td>
<td>One or more library names separated by colons</td>
<td>None</td>
</tr>
<tr>
<td>TV::dll_read_loader_symbols_only</td>
<td>When TotalView loads libraries named in this variable, it only reads loader symbols. Because TotalView checks and processes the names in TV::dll_read_all_symbols list before it processes this list, it will ignore names that are in that list and in this one.</td>
<td>One or more library names separated by colons</td>
<td>None</td>
</tr>
</tbody>
</table>
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 are not interested in debugging 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 will 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 will ignore 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 will not be 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 tell TotalView that it should stop the current process when it loads a library file having this suffix.

TotalView will ask you if you would like 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::dpvm

When this is set to true, TotalView enables support for debugging HP Tru64 UNIX Parallel Virtual Machine applications. This value can only be set in a startup script. You can override this variable’s value by using the following command-line options switches:

- –dpvm sets this variable to true
- –no_dpvm sets this variable to false

**Permitted Values:** true or false  
**Default:** false
**TV::dump_core**  
When this is set to **true**, TotalView will create a core file when an internal TotalView error occurs. This is only used 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 as follows:

- **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:

- `dset TV::dwhere_qualification_level all-parent_function`

This is saying that all elements are displayed except for a parent function.

For more information on these arguments, see "symbol" on page [153](#).

**Permitted Values:**  
One or more of the arguments listed above.  
**Default:**  
`class_name+template_args+module+
parent_function+member+node`
**TV::dynamic**

When this is set to `true`, TotalView loads symbols from shared libraries. This variable is available on all platforms supported by TotalView Technologies. (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. It is set by using the `–env` command-line option. For more information, see “–env variable=value” on page 244.

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 set to either to a value greater than 0, allows TotalView to pickup threads created using the `clone()` system call. The values can set are as follows:

- **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.

*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.*
**TV::force_default_cplus_demangler**

When this is set to `true`, TotalView uses the demangler set in the `TV::current_cplus_demangler` variable. You would set this variable when TotalView uses the wrong demangler. TotalView can use the wrong demangler if you are using an unsupported compiler, and 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 this is set to `true`, TotalView uses the demangler set in the `TV::current_fortran_demangler` variable. You would set this variable when TotalView uses the wrong demangler. TotalView can use the wrong demangler if you are using an unsupported compiler, and unsupported language preprocessor, or if your vendor has made changes to your compiler.

Permitted Values: **true** or **false**  
Default: **false**

**TV::global_typenames**

When this is set to `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 will automatically display the variable by using the non-opaque type definition when displaying variables declared with the opaque type.

If you set this variable to `false`, TotalView does **not** assume that type names are globally unique within a program. Only use this variable 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 this is set to `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 it is set to 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 268.

Permitted Values: **true** or **false**  
Default: **true**
TV::gnu_debuglink_checksum

When this is set to 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 268.

Permitted Values: true or false
Default: true

TV::gnu_debuglink_global_directory

Names the directory in which you store gnu_debuglink files. For more information, see "Using gnu_debuglink Files" on page 268.

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 that TotalView uses to search 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 268.

Permitted Values: A string containing directory paths.
Default: %D:%D.debug:%G%/%D

TV::hpf

Deprecated.

TV::hpf_node

Deprecated.

TV::host_platform

This is a read-only value that returns the architecture upon which TotalView is running.

TV::ignore_control_c

When this is set to 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 only be invoked while TotalView is being initialized. This is a read-only variable.

*Permitted Values:* true or false  
*Default:* false

**TV::ipv6_support**

When you set this variable to **true**, you are enabling ipv6 support. If you set this to **false**, ipv6 support is disabled.

*Permitted Values:* true or false  
*Default:* false

**TV::jnibridge**

Internal use only.

**TV::kcc_classes**

When this is set to **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 this is set to **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 of the 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 only 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 that will be run before TotalView kills a process. The contents of this variable is a list of pairs. For example:

```dset TV::kill_callbacks {```
The first element in the pair is a regular expression and the second is the name of a TCL function. If the processes name matches the regular expression, TotalView runs the TCL procedure, giving it the D PID 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 then kill 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 into which TotalView writes 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 TVSVR_LAUNCHCMD if set, otherwise the value of default_launch_command. Note: changing the value of TVSVR_LAUNCHCMD 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) 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. This variable tells TotalView which local server it should launch.

**Permitted Values:** A file or path name to the local server

**Default:** tvdsrv

**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. For information on this launch string, see “Replacement Characters” on page 257.
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 this is set to true, TotalView displays MPI message queues when you are debugging an MPI program. When the variable is set to false, these queues are not displayed. You would disable these queues if something is overwriting the message queues, thereby confusing TotalView.

Permitted Values: true or false
Default: true

**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 except for SiCortex TotalView, where TV::platform is "linux-x86-64-x-linux-mipsel64" and TV::native_platform is "linux-x86-64."

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. Setting this variable to true tells TotalView that it is using NPTL threads. false means that the program isn’t using this package.

Permitted Values: true, false, or auto
Default: auto

**TV::open_cli_window_callback**

The CLI executes the string that is this variable’s value after you open the CLI by selecting the Tools > Command Line command. It is ignored when you open the CLI from the command line. It 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 .tvdr file:

```
dset TV::open_cli_window_callback "stty sane"
```

Permitted Values: A string representing a Tcl or CLI command
Default: Null

**TV::parallel**

When set to true, you are enabling 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**

Tells TotalView if it should automatically attach to processes. Your choices are as follows:
- yes: Attach to all started processes.
- no: Do not attach to any started processes.
TV:: Namespace

- **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
  *Default*: yes

**TV::parallel_configs**

Defines a new parallel configuration or overwrites an existing one. You can define this variable in a global .tvrc to add new configurations or overwrite those provided by TotalView Technologies. You can also define this variable in a tvrc file contained within your .totalview directory.

For more information, please go to [http://www.totalviewtech.com/Documentation/mpi_startup.html](http://www.totalviewtech.com/Documentation/mpi_startup.html).

**TV::parallel_stop**

Tells TotalView if it should automatically run processes when your program launches them. Your choices are as follows:

- **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 upon 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 immediately after it loads a program and just before it runs it. TotalView executes these procedures after it invokes the procedures in the TV::image_load_callbacks list.

The procedures in this list are only called once even though your executable may use many programs and libraries.

  *Permitted Values*: A list of 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::pvm**

When this is set to true, TotalView lets you debug the ORNL (Oak Ridge National Laboratory) implementation of Parallel Virtual Machine (PVM) applications. This variable can only be set in a start up script. However, you can override this value by using the following command-line options:

- **–pvm** sets this variable to true
- **–no_pvm** sets this variable to false

  *Permitted Values*: true or false
  *Default*: false
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TV::replay_history_mode</code></td>
<td>Controls how ReplayEngine handles the history buffer when it is full, as follows:</td>
</tr>
<tr>
<td></td>
<td>- 1: Discard the oldest history and continue.</td>
</tr>
<tr>
<td></td>
<td>- 2: Stop the process.</td>
</tr>
<tr>
<td></td>
<td><strong>Permitted Values:</strong> 1 or 2</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> 1</td>
</tr>
<tr>
<td><code>TV::replay_history_size</code></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td><strong>Permitted Values:</strong> An integer or an integer followed by K or M</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> 0 (Limited only by available memory)</td>
</tr>
<tr>
<td><code>TV::restart_threshold</code></td>
<td>When you kill a multi-threaded or multiprocess program, TotalView asks if you really want to kill the program if there is more than one thread or process running. Changing this value tells TotalView how many threads or processes need to be running before it asks.</td>
</tr>
<tr>
<td></td>
<td><strong>Permitted Values:</strong> a positive integer</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> 1</td>
</tr>
<tr>
<td><code>TV::save_global_dialog_defaults</code></td>
<td>Obsolete.</td>
</tr>
<tr>
<td><code>TV::save_search_path</code></td>
<td>Obsolete.</td>
</tr>
<tr>
<td><code>TV::save_window_pipe_or_filename</code></td>
<td>Names the file to which TotalView writes or pipes the contents of the current window or pane when you select the File &gt; Save Pane command.</td>
</tr>
<tr>
<td></td>
<td><strong>Permitted Values:</strong> A string naming a file or pipe</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> None, until something is saved. Afterward, the saved string is the default.</td>
</tr>
<tr>
<td><code>TV::search_case_sensitive</code></td>
<td>When this is set to true, text searches only succeed if a string exists whose case exactly matches what you enter in the Edit &gt; Find Dialog Box. For example, searching for Foo won’t find foo if this variable is set to true. It will be found if this variable is set to false.</td>
</tr>
<tr>
<td></td>
<td><strong>Permitted Values:</strong> true or false</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> false</td>
</tr>
<tr>
<td><code>TV::server_launch_enabled</code></td>
<td>When this is set to true, TotalView uses its single-process server launch procedure when launching remote tvdsvr processes. When the variable is set to false, tvdsvr is not automatically launched.</td>
</tr>
<tr>
<td></td>
<td><strong>Permitted Values:</strong> true or false</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> true</td>
</tr>
</tbody>
</table>
**TV::server_launch_string**

Names the command string that TotalView uses to automatically launch the TotalView Debugger Server (**tvdsvr**) when you start to debug a remote process. This command string is executed by `/bin/sh`. By default, TotalView uses the `rsh` command to start the server, but you can use any other command that can invoke `tvdsvr` on a remote host. If no command is available for invoking a remote process, you can’t automatically launch the server; therefore, you should set this variable to `/bin/false`. If you cannot automatically launch a server, you should also set the `TV::server_launch_enabled` variable to `false`. For information on this launch string, see "Replacement Characters" on page 257.

**Permitted Values:** A string

**Default:**

```
{%C %R –n "%B/tvdsvr –working_directory %D –callback %L –set_pw %P –verbosity %V %F`}
```

**TV::server_launch_timeout**

Specifies the number of seconds that TotalView waits to hear back from the TotalView Debugger Server (**tvdsvr**) that it launches.

**Permitted Values:** An integer from 1 to 3600 (1 hour)

**Default:** 30

**TV::server_response_wait_timeout**

Specifies how long TotalView will 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 this Boolean value is `true`, newly created action point are shared across the group. If it is `false`, a newly created action point is only active 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 using 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**

The list that you assign to this variable 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:

```
action=signal_list
```

An `action` can be one of the following: Error, Stop, Resend, or Discard.
A \textit{signal\_list} is a list of one or more signal specifiers, separated by commas:

\begin{verbatim}
signal\_specifier[, signal\_specifier] ...
\end{verbatim}

A \textit{signal\_specifier} can be a signal name (such as \texttt{SIGSEGV}), a signal number (such as \texttt{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 \textit{action\_list}:

\begin{itemize}
  \item If you specify an action for a signal in an \textit{action\_list}, TotalView changes the default action for that signal.
  \item If you do not specify a signal in the \textit{action\_list}, TotalView does not change its default action for the signal.
  \item If you specify a signal that does not exist for the platform, TotalView ignores it.
  \item 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.
\end{itemize}

If you need to revert the settings for signal handling to built-in defaults, use the \texttt{Defaults} button in the File > Signals Dialog Box.

For example, to set the default action for the \texttt{SIGTERM} signal to \texttt{Resend}, you specify the following action list:

\begin{verbatim}
{Resend=SIGTERM}
\end{verbatim}

As another example, to set the action for \texttt{SIGSEGV} and \texttt{SIGBUS} to \texttt{Error}, the action for \texttt{SIGHUP} and \texttt{SIGTERM} to \texttt{Resend}, and all remaining signals to \texttt{Stop}, you specify the following action list:

\begin{verbatim}
{Stop=*, Error=SIGSEGV, SIGBUS, Resend=SIGHUP, SIGTERM}
\end{verbatim}

This action list shows how TotalView applies the actions from left to right.

1. Sets the action for all signals to \texttt{Stop}.
2. Changes the action for \texttt{SIGSEGV} and \texttt{SIGBUS} from \texttt{Stop} to \texttt{Error}.
3. Changes the action for \texttt{SIGHUP} and \texttt{SIGTERM} from \texttt{Stop} to \texttt{Resend}.

\begin{itemize}
  \item \texttt{Permitted Values}: A list of signals, as was just described
  \item \texttt{Default}: This differs from platform to platform; type \texttt{dset TV::signal\_handling\_mode} to see what a platform’s default values are
\end{itemize}

\textbf{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.

\begin{itemize}
  \item \texttt{Permitted Values}: An integer
  \item \texttt{Default}: 8
\end{itemize}

\textbf{TV::spell\_correction}

When you use the \texttt{View > Lookup Function} or \texttt{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 (\texttt{verbose}), TotalView displays a dialog box before it corrects spelling. You can set this resource to \texttt{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 as follows:

- **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 this is set to 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 one 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  
**Default:** {:c:include s:asm S:asm c:c h:c:include lex:c:include y:include bmap:c:include f:f77 F:f77 f90:f9x F90:f9x hpf:hpf HPF:hpf cxx:c++ cpp:c++ cc:c++ c++:c++ C:+ C++:c++ hxx:c++:include hpp:c++:include hh:c++:include h++:c++:include HXX:c++:include HPP:c++:include HH:c++:include H:c++:include ih:c++:include th:c++}

### 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-cell 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 set to `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. As of TotalView 8.7, when this variable was introduced, only AIX supported the fast trap mechanism for breakpoints, but we anticipate other operating systems adding support. 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 Release Notes on our web site for more information.

**Permitted Values:** `true` or `false`  
**Default:** `true`

**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. As of TotalView 8.7, when this variable was introduced, only AIX supported the fast trap mechanism for watchpoints, but we anticipate other operating systems adding support. 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 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 this is set to 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 this is set to 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 this is set to 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::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 set to 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 this is set to 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_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

---

**TV::GUI:: Namespace**

The variables in this section only have meaning (and in some cases, a value) 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 this is set to **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 this is set to **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 tell TotalView to 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 0}

TV::GUI::geometry_expressions
Specifies the position at which TotalView displays the Tools > Expression List Window.
TV::GUI::namespace

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_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 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

**Default:** {0 0 0 0}

**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 coordinate’s 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 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 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 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 0\}

TV::GUI::geometry_ptset

No longer used.

TV::GUI::geometry_pvm

Specifies the position at which TotalView displays the Tools > PVM 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_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::heap_summary_refresh

Not user settable.

TV::GUI::inverse_video

Not implemented.
TV::GUI::keep_expressions

Deprecated.

TV::GUI::keep_search_dialog

When this is set to 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::pop_at_breakpoint

When this is set to 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 this is set to 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
Default: true

TV::GUI::show_startup_parameters

Setting this value to true tells TotalView to display that it should 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

The value set for this variable tell TotalView how it should display the toolbar.

Permitted Values: icons_above_text, icons_besides_text, icons, or text
Default: icons_above_text

TV::GUI::tooltips_enabled

When set tp true, TotalView displays more information about variables 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:

-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

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 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. If TotalView can’t create this expression, you will need to read the Creating Type Transformations Guide, which is located on our web site at http://www.totalviewtech.com/Documentation/.

Topics in this chapter are:
- “Why Type Transformations” on page 229
- “Creating Structure and Class Transformations” on page 231

Why Type Transformations

Modern programming languages allow you to use abstractions such as structures, class, and STL data types such as lists, maps, 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, the following figure shows 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.

Transforming STL strings, vectors, lists, and maps is the TotalView default. If you do not want TotalView to transform your information, select the Options Tab within the File > Preferences Dialog Box and then 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 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 231
- “build struct transform Function” on page 232
- “Type Transformation Expressions” on page 233
- “Using Type Transformations” on page 236

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:

```tcl
::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:
  ```tcl
  source stuff.tvd
  ```
- You can place the transformation source file into the same directory 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 following Variable Window when you dive on the *info* structure:

![Figure 5: Transforming a Structure](image)

**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:

```
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:

```
{ 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:

```
{ 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:

```
{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.

{* 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.

{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 -> expr} Names a subelement in a structure accessed using a pointer. This is used in the same way as the -> operator that exists 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 6 on page 235 shows the Variables Windows that TotalView displays for the info class and the x struct.

The following transformation remaps this information:

::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 \texttt{x1} class as shown in Figure 7.

The members of this transformation are as follows:

- \textbf{pmonth}: The \texttt{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.

- \textbf{pName}: The \texttt{pName} member is added. The transformation contains two operations. The first dereferences the pointer. In addition, as \texttt{x1} is derived from \texttt{xbase}, you need to upcast the variable when you want to include it.

Notice that one expression is nested within another.

- \textbf{pStreet}: The \texttt{pStreet} member is added. The operations that are performed are the same as for \texttt{pName}.

- \textbf{pVoid1}: The \texttt{v} member is added. Because the application’s definition of the data is \texttt{void *}, casting tells TotalView how it should interpret the information. In this example, the data is being cast into a pointer to a string.

- \textbf{pVoid2}: The \texttt{q} member is added. The transformation contains two operations. The first casts \texttt{q} into a pointer to the \texttt{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:

```bash
  dset TOTALVIEW_TCLLIB_PATH
```

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```
This section of the TotalView Reference Guide contains information about command-line options you use when starting TotalView and the TotalView Debugger Server.

Chapter 7: TotalView Command Syntax
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
This chapter describes how you modify the behavior of the tvdsrv. 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.
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 ] [ filename [ corefile ] ] [ -a [ args ] ]
```

or

```
totalview [ options ] -args filename [ args ]
```

**Arguments:**

- `options`: TotalView options.
- `filename`: 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.
- `corefile`: Specifies the name of a core file. Use this argument in addition to `filename` when you want to examine a core file 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`.

  **default:** light blue

- **-bg color**
  
  Same as `-background`.

- **-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.

-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 consoleoutputfile
Redirects TotalView console output to a file named consoleoutputfile.
All TotalView console output is written to stderr.

-default_parallel_attach_subset=subset_specification
Specifies a list of MPI ranks to be attached to when an MPI job is created or attached to. 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 strideth rank between rank1 and rank2

A rank specification can be either a number or "$max" (the last rank in the MPI job).

--demangler=compiler
Overrides the demangler and mangler TotalView uses by default. The following indicate override options.

-demangler=compaq HP cxx on Linux (alpha)
-demangler=dec HP Tru64 C++ or Fortran
-demangler=gnu GNU C++ on Linux Alpha
-demangler=gnu_dot GNU C++ on Linux x86
-demangler=gnu_v3 GNU C++ Linux x86
-demangler=hp HP aCC compiler
-demangler=irix SGI IRIX C++
-demangler=kai KAI C++
Command-Line Options

- **–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.

- **–dpvm**  
  HP Tru64 UNIX only: Enable support for debugging the HP Tru64 UNIX implementation of Parallel Virtual Machine (PVM) applications.

- **–dump_core**  
  Allows TotalView to dump a core file of itself when an internal error occurs. This is used to help TotalView Technologies 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`.

*default*: black

`-fg color`  
Same as `-foreground`.

`-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.
Command-Line Options

-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.

-lb
(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.

-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_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.

–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"

–nodes
   Specifies the number of nodes upon which the MPI job will run.

–np
   Specifies how many tasks that TotalView should launch for the job. This argument usually follows a –mpi com-
   mand-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 appli-
cation is using it and it isn’t.

–parallel
   (Default) Enables handling of parallel program run-time libraries such as MPI, PE, and UPC.
Command-Line Options

–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 of TotalView when starting a parallel program. Possible options are:

  yes (default)  Attach to all processes
  no  Attach to no processes
  ask  Ask which to attach to

–patch_area_base address
Allocates the patch space dynamically at address. See “Allocating Patch Space for Compiled Expressions” in Chapter 15 of the TotalView Users 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 Users Guide.

–pid pid
Tells TotalView to attach to process pid after it starts executing.

–procs
Specifies how many tasks that TotalView should launch for the job. This argument usually follows a –mpi command-line option.

–pvm
Enables support for debugging the ORNL implementation of Parallel Virtual Machine (PVM) applications.

–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 Users Guide.

–r hostname[:portnumber]
Same as –remote.

–replay
Tells TotalView that it should enable 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 Chapter 4 of the TotalView 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 Users Guide.

A signalSpecifier 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.
Command-Line Options

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 your program will write **stderr** information while executing within TotalView. If the file exists, TotalView overwrites it.

**–stderr_append**

Tells TotalView to append the information if writes to **stderr** to the file named in the **–stderr** command or the file named in GUI or a TotalView variable. If the file does not exist, TotalView creates it.

**–stderr_is_stdout pathname**

Tells TotalView to redirect **stderr** to **stdout**.

**–stdin pathname**

Names the file from which your program will read information while executing within TotalView.

**–stdout pathname**

Names the file to which your program will write **stdout** information while executing within TotalView. If the file exists, TotalView overwrites it.

**–stdout_append**

Tells TotalView to append the information if writes to **stdout** to the file named in the **–stdout** command or the file named in GUI or a TotalView 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 **–noteamplus**.

**–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**.
Command-Line Options

–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 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 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.

–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.

–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.

–verbosity level  Sets the verbosity level of TotalView messages to level, which may be one of silent, error, warning, or info.

  default: info
This chapter summarizes the syntax of the TotalView Debugger Server command, `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 ”Setting Up Remote Debugging Sessions” in the TotalView Users Guide.

Topics in this chapter are:
- The `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.)
The tvdsvr Command and Its Options

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:

\[
pw = \text{hexnumhigh:hexnumlow}
\]

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_ports 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 tvdsvr. This option is most often used with a bulk launch.
  For more information, see Chapter 4, "Setting Up Remote Debugging Sessions" in the TotalView Users Guide.

- **--debug_file console_outputfile**
  Redirects TotalView Debugger Server console output to a file named console_outputfile.
  **Default:** All console output is written to stderr.
The tvdsvr Command and Its Options

-dpvm
Uses the HP Tru64 UNIX implementation of the Parallel Virtual Machine (DPVM) library process as its input channel and registers itself as the DPVM tasker.

This option is not intended for users launching tvdsvr manually. When you enable DPVM support within TotalView, TotalView automatically uses this option when it launches tvdsvr.

-nodes_allowed num
Explicitly tells tvdsvr 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 tvdsvr should communicate with TotalView. If this port is busy, tvdsvr does not select an alternate port number (that is, it won’t communicate with anything) unless you also specify –search_port.

Default: 4142

-pvm
Uses the ORNL implementation of the Parallel Virtual Machine (PVM) library process as its input channel and registers itself as the ORNL PVM tasker.

This option is not intended for users launching tvdsvr manually. When you enable PVM support within TotalView, TotalView automatically uses this option when it launches tvdsvr.

-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, tvdsvr 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 Chapter 4 of the TotalView 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 tvdsvr.conf in your /etc directory. This file must be owned by user ID 0 (root). When tvdsvr
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: \texttt{--search\_port} or \texttt{--port}. To stop \texttt{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 \texttt{kill} command.

\textbf{\texttt{--set\_pw hexnumhigh:hexnumlow}}

Sets the password to the 64-bit number specified by the \texttt{hexnumhigh} and \texttt{hexnumlow} 32-bit numbers. When a connection is established between \texttt{tvdsvr} and TotalView, the 64-bit password passed by TotalView must match this password set with this option. \texttt{tvdsvr} displays the selected number in the following message:

\begin{verbatim}
pw = hexnumhigh:hexnumlow
\end{verbatim}

We recommend using this option to avoid connections by other users.

If necessary, you can disable password checking by specifying the "\texttt{--set\_pw 0:0}" option with the \texttt{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.

\textbf{\texttt{--set\_pws password-list}}

Sets 64-bit passwords. TotalView must supply these passwords when \texttt{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 Chapter 4, "Setting Up Remote Debugging Sessions" in the TotalView Users Guide.

\textbf{\texttt{--verbosity level}}

Sets the verbosity level of TotalView Debugger Server-generated messages to \texttt{level}, which may be one of \texttt{silent}, \texttt{error}, \texttt{warning}, or \texttt{info}.

\textbf{Default: info}

\textbf{\texttt{--working\_directory directory}}

Makes \texttt{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.
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:

- **%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 `rsh`. On HP, this command is `remsh`. 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 start-up 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** (Red Storm and BlueGene architectures) If TotalView must use an alternative name for `tvdsvr`, specify its name here. For example, on BlueGene, the server name is `tvdsvr_bgl`. On Red Storm, it is `tvdsvr_rs`.
- **%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 Chapter 4, "Setting Up Remote Debugging Sessions" in the TotalView 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.
Replacement Characters
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
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
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
When debugging assembly-level functions, you will need to know how TotalView refers to your machines registers.
This chapter describes the compilers and parallel runtime environments used on platforms supported by TotalView. You must refer to the TotalView 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 275.

Topics in this chapter are:

- Compiling with Debugging Symbols
- Using gnu_debuglink Files
- Using Exception Data on Tru64 UNIX
- 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.

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

The following table lists the procedures to compile programs on HP Alpha running Linux.

<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>GCC C</td>
<td>gcc -g program.c</td>
</tr>
<tr>
<td>GCC C++</td>
<td>g++ -g program.cxx</td>
</tr>
<tr>
<td>GCC Fortran</td>
<td>g77 -g program.f</td>
</tr>
<tr>
<td>IBM xlc C</td>
<td>xlc -g program.c</td>
</tr>
<tr>
<td>IBM xlc C++</td>
<td>xlC -g program.cxx</td>
</tr>
<tr>
<td>IBM xlF Fortran 77</td>
<td>xlF -g program.f</td>
</tr>
<tr>
<td>IBM xlF90 Fortran 90</td>
<td>xlF90 -g program.f90</td>
</tr>
</tbody>
</table>

On Mac OS X 10.4, you can create 64-bit applications using GCC 4 by adding the `–m64` command-line option.

HP Tru64 UNIX

The following table lists the procedures to compile programs on HP Tru64 UNIX.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP Tru64 UNIX C</td>
<td>cc -g program.c</td>
</tr>
<tr>
<td>HP Tru64 UNIX C++</td>
<td>cxx -g program.cxx</td>
</tr>
<tr>
<td>HP Tru64 UNIX Fortran 77</td>
<td>f77 -g program.f</td>
</tr>
<tr>
<td>HP Tru64 UNIX Fortran 90</td>
<td>f90 -g program.f90</td>
</tr>
<tr>
<td>HP Tru64 UPC compiler</td>
<td>upc -g [-fthreads n] program.upc</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>
</tbody>
</table>

HP-UX for IA-64

The following table lists the procedures to compile programs on HP HP-UX for IA-64.

<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>GCC Fortran</td>
<td>g77 -g program.f</td>
</tr>
<tr>
<td>HP C</td>
<td>cc -g program.cc</td>
</tr>
<tr>
<td>HP aCC</td>
<td>aCC -g program.cxx</td>
</tr>
<tr>
<td>HP Fortran 77</td>
<td>f77 -g program.cc</td>
</tr>
<tr>
<td>HP Fortran 90</td>
<td>f90 -g program.cc</td>
</tr>
</tbody>
</table>
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>gcc -g program.c</td>
</tr>
<tr>
<td>GCC C++</td>
<td>g++ -g program.cxx</td>
</tr>
<tr>
<td>IBM xlc C</td>
<td>xlc -g program.c</td>
</tr>
<tr>
<td>IBM xlc C++</td>
<td>xlc -g program.cxx</td>
</tr>
<tr>
<td>IBM xlf Fortran 77</td>
<td>xlf -g program.f</td>
</tr>
<tr>
<td>IBM xlf90 Fortran 90</td>
<td>xlf90 -g program.f90</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:

```bash
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.

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 xI 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 Bluegene

The following table lists the procedures to compile programs on the IBM Blue Gene/L computer.

<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.cc</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.f</td>
</tr>
<tr>
<td>Absoft Fortran 90</td>
<td>f77 –g program.for</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.f</td>
</tr>
<tr>
<td>Absoft Fortran 90</td>
<td>f77 –g program.for</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>
</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><code>f77 -g program.f</code></td>
</tr>
<tr>
<td></td>
<td><code>f77 -g program.for</code></td>
</tr>
<tr>
<td>Absoft Fortran 90</td>
<td><code>f90 -g program.f90</code></td>
</tr>
<tr>
<td>Absoft Fortran 95</td>
<td><code>f95 -g program.f95</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>G77</td>
<td><code>g77 -g program.f</code></td>
</tr>
<tr>
<td>Intel C++ Compiler</td>
<td><code>icc -g program.cxx</code></td>
</tr>
<tr>
<td>Intel Fortran Compiler</td>
<td><code>ifc -g program.f</code></td>
</tr>
<tr>
<td>Pathscale EKO C</td>
<td><code>pathcc -g program.f</code></td>
</tr>
<tr>
<td>Pathscale EKO C++</td>
<td><code>pathCC -g program.f</code></td>
</tr>
<tr>
<td>Lahey/Fujitsu Fortran</td>
<td><code>lf95 -g program.f</code></td>
</tr>
<tr>
<td>PGI C++</td>
<td><code>pcCC -g program.f</code></td>
</tr>
<tr>
<td>PGI Fortran 77</td>
<td><code>pgf77 -g program.f</code></td>
</tr>
<tr>
<td>PGI Fortran 90</td>
<td><code>pgf90 -g program.f</code></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><code>gcc -g program.c</code></td>
</tr>
<tr>
<td>GCC C++</td>
<td><code>g++ -g program.cxx</code></td>
</tr>
<tr>
<td>G77</td>
<td><code>g77 -g program.f</code></td>
</tr>
<tr>
<td>Intel C++ Compiler</td>
<td><code>icc -g program.cxx</code></td>
</tr>
<tr>
<td>Intel Fortran Compiler</td>
<td><code>ifc -g program.f</code></td>
</tr>
</tbody>
</table>

### SGI IRIX-MIPS Systems

The following table lists the procedures to compile programs on SGI MIPS systems running IRIX.

<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>G77</td>
<td><code>g77 -g program.f</code></td>
</tr>
<tr>
<td>Intrepid (GCC UPC)</td>
<td><code>upc -g [-fthreads n] program.upc</code></td>
</tr>
<tr>
<td>SGI MIPSpro C</td>
<td><code>cc -n32 -g program.c</code></td>
</tr>
<tr>
<td></td>
<td><code>cc -64 -g program.c</code></td>
</tr>
<tr>
<td>SGI MIPSpro C++</td>
<td><code>CC -n32 -g program.cxx</code></td>
</tr>
<tr>
<td></td>
<td><code>CC -64 -g program.cxx</code></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.

TotalView does not support compiling with -m32, which is the default for some compilers. You must specify either -m32 or -m64.

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>SGI MIPSprow77</td>
<td>f77 -m32 -g program.f</td>
</tr>
<tr>
<td></td>
<td>f77 -m64 -g program.f</td>
</tr>
<tr>
<td>SGI MIPSprow 90</td>
<td>f90 -m32 -g program.f90</td>
</tr>
<tr>
<td></td>
<td>f90 -m64 -g program.f90</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.
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_flag`
  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_flag` 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
Using Exception Data on Tru64 UNIX

If you receive the following error message when you load an executable into TotalView, you may need to compile your program so that it includes exception data.

```
Cannot find exception information. Stack backtraces may not be correct.
```

To provide a complete stack backtrace in all situations, TotalView needs for you to include exception data with the compiled executable. To compile with exception data, you need to use the following options:

```
cc –Wl,–u,_fpdata_size program.c
```

where:

- `–Wl` Passes the arguments that follow to another compilation phase (`–Wl`), which in this case is the linker (`l`). Each argument is separated by a comma (,).
- `–u,_fpdata_size` Causes the linker to mark the next argument (`_fpdata_size`) as undefined. This forces the exception data into the executable.
- `program.c` Is the name of your program.

Compiling with exception data increases the size of your executable slightly. If you choose not to compile with exception data, TotalView can provide correct stack backtraces in most situations, but not in all situations.
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.

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.

Add one of the following command-line options to the command that you use to link your programs:

- `/opt/totalview/alpha/lib/libdbfork.a`
- `–L/opt/totalview/alpha/lib –ldbfork`

For example:

```
cc -o program program.c  
    -L/opt/totalview/alpha/lib -ldbfork
```

As an alternative, you can set the `LD_LIBRARY_PATH` environment variable and omit the `–L` option on the command line:

```
setenv LD_LIBRARY_PATH /opt/totalview/alpha/lib
```

### Linking with HP-UX

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 one of the following arguments:

- `/opt/totalview/lib/hpux11-hppa/libdbfork.a`
- `–L/opt/totalview/hpux11-hppa/lib –ldbfork`

For example:

```
cc -n32 -o program program.c  
    -L/opt/totalview/hpux11-hppa/lib -ldbfork
```

If you are compiling 64-bit code, use the following arguments:

- `/opt/totalview/lib/hpux11-hppa/libdbfork_64.a`
- `–L/opt/totalview/hpux11-hppa/lib –ldbfork_64`

For example:

```
cc -64 -o program program.c  
    -L/opt/totalview/hpux11-hppa/lib 
    -ldbfork_64
```

As an alternative, you can set the `LD_LIBRARY_PATH` environment variable and omit the `–L` command-line option. For example:

```
setenv LD_LIBRARY_PATH  
    /opt/totalview/hpux11-hppa/lib
```
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
  - bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
- /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
  - bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
- /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
    -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:

  #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:

```
cc –o program program.c \
   –L/usr/totalview/linux-x86/lib –ldbfork
```

However, `linux-ia64` uses `libdbfork` for 64-bit programs.

**SGI IRIX6-MIPS**

Add one of the following arguments or command-line options to the command that you use to link your programs.

If you are compiling your code with `–n32`, use the following arguments:

- `/opt/totalview/irix6-mips/lib/libdbfork_n32.a`
- `–L/opt/totalview/irix6-mips/lib –ldbfork_n32`

For example:

```
cc –n32 –o program program.c \ 
   –L/opt/totalview/irix6-mips/lib –ldbfork_n32
```

If you are compiling your code with `–64`, use the following arguments:

- `/opt/totalview/irix6-mips/lib/libdbfork.a_n64.a`
- `–L/opt/totalview/irix6-mips/lib –ldbfork_n64`

For example:

```
cc –64 –o program program.c \ 
   –L/opt/totalview/irix6-mips/lib –ldbfork_n64
```

As an alternative, you can set the `LD_LIBRARY_PATH` environment variable and omit the `–L` option on the command line:

```
setenv LD_LIBRARY_PATH /opt/totalview/irix6-mips/lib
```
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

For example:

```
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:

```
setenv LD_LIBRARY_PATH /opt/totalview/sun5/lib
```
Operating Systems

Chapter 10

This chapter describes the operating system features that can be used with TotalView. This chapter includes the following topics:

- Supported Operating Systems
- Mounting the /proc File System (HP Tru64 UNIX, IRIX, and 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 TotalView Platforms and Systems Requirements document on our web site. You can locate this document by going to http://www.totalviewtech.com/Documentation/".

- Apple Macintosh OS X 10.4.
- Cray XT3 (UNICOS/lc environment) running SuSE Linux Enterprise Server 8.2.99 and 9.
- HP Alpha workstations running HP Tru64 UNIX versions V4.0F and V5.1B.
- HP HP-UX for IA-64 systems running version 2.0 (11.23).
- IBM Blue Gene systems running SuSE Linux Enterprise Server 9 for the front end nodes.
Mounting the /proc File System

To debug programs on HP Tru64 UNIX, SunOS 5, and IRIX 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>HP Tru64 UNIX</td>
<td>% /sbin/mount -t procfs /proc on /proc type procfs (rw)</td>
</tr>
<tr>
<td>SunOS 5</td>
<td>% /sbin/mount</td>
</tr>
<tr>
<td>IRIX</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 HP Tru64 UNIX and 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>HP Tru64 UNIX</td>
<td>/etc/fstab</td>
<td>/proc /proc procfs rw 0 0</td>
</tr>
<tr>
<td>SunOS 5</td>
<td>/etc/vfstab</td>
<td>/proc - /proc proc - no -</td>
</tr>
</tbody>
</table>
Then, to mount the `/proc` file system, enter the following command:

```
/sbin/mount /proc
```

### Mounting proc SGI IRIX

To make sure that the `/proc` file system is mounted each time your system boots, make sure that `/etc/rc2` issues the `/etc/mntproc` command. Then, to mount the `/proc` file system, enter the following command:

```
/etc/mntproc
```

---

### 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:

  ```
  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 HP Tru64 UNIX

To find out how much swap space has been allocated and is currently being used, use the `swapon` command on HP Tru64 UNIX.

To find out how much swap space is in use while you are running TotalView:

```
/bin/ps -o LFMT
```

To add swap space, use the `/sbin/swapon(8)` command. You must be logged in as `root` to use this command. For more information, refer to the online manual page for this command.

### Swap Space on HP HP-UX

The `swapinfo` command on an HP-UX system lets you find out how much swap space is allocated and is being used.

To find out how much swap space is being used while TotalView is running, enter:

```
/usr/bin/ps -lf
```
Here is an example of what you might see. The \texttt{SZ} column shows the pages occupied by a program.

To add swap space, use the \texttt{/usr/sbin/swapon(1M)} command or the \texttt{SAM} (System Administration Manager) utility. If you use \texttt{SAM}, invoke the \texttt{Swap} command in the \texttt{Disks and File Systems} menu.

**Maximum Data Size**

To see the current data size limit in the C shell, enter:

\begin{verbatim}
limit datasize
\end{verbatim}

The following command displays the current \textit{hard} limit:

\begin{verbatim}
limit -h datasize
\end{verbatim}

If the current limit is lower than the hard limit, you can easily raise the current limit. To change the current limit, enter:

\begin{verbatim}
limit datasize new_data_size
\end{verbatim}

If the hard limit is too low, you must reconfigure and rebuild the kernel, and then reboot. This is most easily done using \texttt{SAM}.

To change \texttt{maxdsiz}, use the following path through the \texttt{SAM} menus:

\begin{verbatim}
Kernel Configuration > Configurable Parameters > maxdsiz > Actions > Modify Configurable Parameter > Specify New Formula/Value > Formula/Value
\end{verbatim}

You can now enter the new maximum data segment size.

You may also need to change the value for \texttt{maxdsiz_64}.

Here is the command that lets you rebuild the kernel with these changed values:

\begin{verbatim}
Configurable Parameter > Actions > Process New Kernel
\end{verbatim}

Answer \texttt{yes} to process the kernel modifications, \texttt{yes} to install the new kernel, and \texttt{yes} again to reboot the machine with the new kernel.

When the machine reboots, the value you set for \texttt{maxdsiz} should be the new hard limit.

**Swap Space on IBM AIX**

To find out how much swap space has been allocated and is currently being used, use the \texttt{/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:

\begin{verbatim}
totalview executable
\end{verbatim}

Press Ctrl+Z to suspend TotalView.

2. Use the following command to see how much swap space TotalView is using:

\begin{verbatim}
ps u
\end{verbatim}
For example, in this case the value in the SZ column is 5476 KB:

```
USER    PID   %CPU  %MEM   SZ   RSS  TTY  ...
smith   15080  0.0   6.0   5476  5476  pts/1  ...
```

To add swap space, use the AIX system management tool, `smit`. Use the following path through the `smit` menus:

```
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 SGI IRIX

To find out how much swap space has been allocated and is currently being used, use the `swap` 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.
   ```
2. Use the following command to see how much swap space TotalView is using:
   ```
   /bin/ps -l
   
   Use the following command to determine the number of bytes in a page:
   ```
   ```
   sysconf PAGESIZE
   ```

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.

### 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.
   ```
2. 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.
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:

- 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.

On HP-UX, TotalView cannot stop the loading of shared libraries until after static constructors on shared library initialization routines have been run.

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 241 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.

On most operating systems (except HP), 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.

HP-UX does not have this (or an equivalent) variable. On HP systems, you can avoid this problem by using the -B immediate option the executable being debugged, or by invoking chatr with the -B immediate option. (See the chatr documentation for complete information on how to use this command.)

You will also have to enter `pxdb –s on`.

Using Shared Libraries on HP-UX

The dynamic library loader on HP-UX loads shared libraries into shared memory. Writing breakpoints into code sections loaded in shared memory can cause programs not under TotalView control to fail when they execute an unexpected breakpoint.

If you need to single-step or set breakpoints in shared libraries, you must set your application to load those libraries in private memory. This is done using HP’s pxdb command.

```
pxdb -s on appname  (load shared libraries into private memory)
pxdb -s off appname  (load shared libraries into shared memory)
```

For 64-bit platforms, use `pxdb64` instead of `pxdb`. If the version of `pxdb64` supplied with HP’s compilers does not work correctly, you may need to install an HP-supplied patch. You will find additional information in the TotalView Release Notes.
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 ones that are 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.

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. (See Figure 9.)

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>HP Tru64 UNIX Alpha</td>
<td>/usr/shlib/</td>
</tr>
<tr>
<td></td>
<td>/usr/lib/cmplrs/cc/</td>
</tr>
<tr>
<td></td>
<td>/usr/local/lib/</td>
</tr>
<tr>
<td></td>
<td>/var/shlib/</td>
</tr>
<tr>
<td>HP-UX</td>
<td>/usr/lib/</td>
</tr>
<tr>
<td></td>
<td>/opt/langtools/lib/</td>
</tr>
<tr>
<td>IBM AIX</td>
<td>/lib/</td>
</tr>
<tr>
<td></td>
<td>/usr/lpp/</td>
</tr>
<tr>
<td></td>
<td>/usr/dt/lib/</td>
</tr>
<tr>
<td>SGI IRIX</td>
<td>/lib/</td>
</tr>
<tr>
<td></td>
<td>/usr/local/lib/</td>
</tr>
<tr>
<td></td>
<td>/lib32/</td>
</tr>
<tr>
<td></td>
<td>/usr/lib32/</td>
</tr>
<tr>
<td></td>
<td>/lib64/</td>
</tr>
<tr>
<td></td>
<td>/usr/local/lib64</td>
</tr>
<tr>
<td>SUN Solaris 2.x</td>
<td>/lib/</td>
</tr>
<tr>
<td></td>
<td>/usr/ccs/lib/</td>
</tr>
<tr>
<td>Linux x86</td>
<td>/lib</td>
</tr>
<tr>
<td>Linux Alpha</td>
<td>/lib</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.

**Known Limitations**

Dynamic library support has the following known limitations:

- TotalView does not deal correctly with parallel programs that call `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 `dlclose`) and then reloading them at a different address using `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 `keysym` 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'
# Remap F35 to Next
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 HP Alpha Tru64 UNIX**

On HP Tru64 UNIX, TotalView supports compiled and interpreted expressions. TotalView also supports assembly language in expressions.

**Expression System on IBM AIX**

On IBM AIX, 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);
```

**Expression System on SGI IRIX**

On IRIX, TotalView supports compiled and interpreted expressions. TotalView also supports assembler in expressions.

TotalView includes the SGI IRIX expression compiler. This feature does not use any MIPS-IV specific instructions. It does use MIPS-III instructions freely. It fully supports –n32 and –64 executables.

Due to limitations in dynamically allocating patch space, compiled expressions are disabled by default on SGI IRIX. To enable compiled expressions, use the `TV::compile_expressions` CLI variable to set the option to true. This variable tells TotalView to find or allocate patch space in your program for code fragments generated by the expression compiler.

If you enable compiled patches on SGI IRIX with a multiprocess program, you must use static patches. For example, if you link a static patch space into a program and run the program under TotalView control, TotalView should let you debug it. If you attach to a previously started MPI job, however, even static patches will not let the program run properly. If TotalView still fails to work properly with the static patch space, then you probably cannot use compiled patches with your program.

For general instructions on using patch space allocation controls with compiled expressions, see "Allocating Patch Space for Compiled Expressions" in Chapter 15 of the TotalView Users Guide.
Architectures

This chapter describes the architectures TotalView supports, including:
- "AMD and Intel x86-64" on page 287
- "HP Alpha" on page 291
- "HP PA-RISC" on page 293
- "IBM Cell Broadband Engine" on page 296
- "Power Architectures" on page 297
- "Intel IA-64" on page 301
- "Intel x86" on page 305 (Intel 80386, 80486 and Pentium processors)
- "SGI MIPS" on page 308
- "Sun SPARC" on page 313

AMD and Intel x86-64

This section describes AMD’s 64-bit processors and the Intel EM64T processors. It contains the following information:
- "x86-64 General Registers" on page 288
- "x86-64 Floating-Point Registers" on page 288
- "x86-64 FPCR Register" on page 289
- "x86-64 FPSR Register" on page 290
- "x86-64 MXCSR Register" on page 291

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>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$rdx</td>
</tr>
<tr>
<td>RCX</td>
<td>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$rcx</td>
</tr>
<tr>
<td>RBX</td>
<td>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$rbx</td>
</tr>
<tr>
<td>RSI</td>
<td>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$rsi</td>
</tr>
<tr>
<td>RDI</td>
<td>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$rdi</td>
</tr>
<tr>
<td>RBP</td>
<td>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$rbp</td>
</tr>
<tr>
<td>RSP</td>
<td>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$rsp</td>
</tr>
<tr>
<td>R8-R15</td>
<td>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$r8-$r15</td>
</tr>
<tr>
<td>RA</td>
<td>Selector registers</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$ra</td>
</tr>
<tr>
<td>SS</td>
<td>$int</td>
<td>no</td>
<td></td>
<td></td>
<td>$ss</td>
</tr>
<tr>
<td>DS</td>
<td>$int</td>
<td>no</td>
<td></td>
<td></td>
<td>$ds</td>
</tr>
<tr>
<td>ES</td>
<td>$int</td>
<td>no</td>
<td></td>
<td></td>
<td>$es</td>
</tr>
<tr>
<td>FS</td>
<td>$int</td>
<td>no</td>
<td></td>
<td></td>
<td>$fs</td>
</tr>
<tr>
<td>GS</td>
<td>$int</td>
<td>no</td>
<td></td>
<td></td>
<td>$gs</td>
</tr>
<tr>
<td>EFLAGS</td>
<td>$int</td>
<td>no</td>
<td></td>
<td></td>
<td>$eflags</td>
</tr>
<tr>
<td>RIP</td>
<td>Instruction pointer</td>
<td>$code[]</td>
<td>no</td>
<td>yes</td>
<td>$eip</td>
</tr>
<tr>
<td>FS_BASE</td>
<td>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$fs_base</td>
</tr>
<tr>
<td>GS_BASE</td>
<td>$long</td>
<td>yes</td>
<td></td>
<td></td>
<td>$gs_base</td>
</tr>
<tr>
<td>TEMP</td>
<td>$long</td>
<td>no</td>
<td></td>
<td></td>
<td>$temp</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>
</tbody>
</table>
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>
<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>
</tbody>
</table>
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 3 of the TotalView 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.

### x86-64 FPSR Register

The bit settings of the x86-64 FPSR register are outlined in the following table.
x86-64 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 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>

HP Alpha

This section contains the following information:

- Alpha General Registers
- Alpha Floating-Point Registers
- Alpha FPCR Register

The Alpha processor supports the IEEE floating-point format.

Alpha General Registers

TotalView displays the Alpha general registers in the Stack Frame Pane of the Process Window. The next table describes how TotalView treats each general register, and the actions you can take with each register.
Alpha Floating-Point Registers

TotalView displays the Alpha floating-point registers in the Stack Frame Pane of the Process Window. Here is a table that 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 – $f1</td>
<td>Floating-point registers (f registers), used singly</td>
<td>$double</td>
<td>yes</td>
<td>yes</td>
<td>$f0 – $f1</td>
</tr>
<tr>
<td>$f2 – $f9</td>
<td>Conventional saved registers</td>
<td>$double</td>
<td>yes</td>
<td>yes</td>
<td>$f2 – $f9</td>
</tr>
<tr>
<td>$f10 – $f15</td>
<td>Conventional scratch registers</td>
<td>$double</td>
<td>yes</td>
<td>yes</td>
<td>$f10 – $f15</td>
</tr>
<tr>
<td>$f16 – $f21</td>
<td>Argument registers</td>
<td>$double</td>
<td>yes</td>
<td>yes</td>
<td>$f16 – $f21</td>
</tr>
<tr>
<td>$f22 – $f30</td>
<td>Conventional scratch registers</td>
<td>$double</td>
<td>yes</td>
<td>yes</td>
<td>$f22 – $f30</td>
</tr>
<tr>
<td>$f31</td>
<td>ReadAsZero/Sink register</td>
<td>$double</td>
<td>yes</td>
<td>yes</td>
<td>$f31</td>
</tr>
<tr>
<td>FPCR</td>
<td>Floating-point control register</td>
<td>$long</td>
<td>no</td>
<td>yes</td>
<td>$fpcr</td>
</tr>
</tbody>
</table>
Alpha FPCR Register

For your convenience, TotalView interprets the bit settings of the Alpha FPCR register. You can edit the value of the FPCR 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>SUM</td>
<td>0x8000000000000000</td>
<td>Summary bit</td>
</tr>
<tr>
<td>DYN=CHOP</td>
<td>0x0000000000000000</td>
<td>Rounding mode — Chopped rounding mode</td>
</tr>
<tr>
<td>DYN=MINF</td>
<td>0x0400000000000000</td>
<td>Rounding mode — Negative infinity</td>
</tr>
<tr>
<td>DYN=NORM</td>
<td>0x0800000000000000</td>
<td>Rounding mode — Normal rounding</td>
</tr>
<tr>
<td>DYN=PINF</td>
<td>0x0c00000000000000</td>
<td>Rounding mode — Positive infinity</td>
</tr>
<tr>
<td>IOV</td>
<td>0x0200000000000000</td>
<td>Integer overflow</td>
</tr>
<tr>
<td>INE</td>
<td>0x0100000000000000</td>
<td>Inexact result</td>
</tr>
<tr>
<td>UNF</td>
<td>0x0080000000000000</td>
<td>Underflow</td>
</tr>
<tr>
<td>OVF</td>
<td>0x0040000000000000</td>
<td>Overflow</td>
</tr>
<tr>
<td>DZE</td>
<td>0x0020000000000000</td>
<td>Division by zero</td>
</tr>
<tr>
<td>INV</td>
<td>0x0010000000000000</td>
<td>Invalid operation</td>
</tr>
</tbody>
</table>

HP PA-RISC

This section contains the following information:

- PA-RISC General Registers
- PA-RISC Process Status Word
- PA-RISC Floating-Point Registers
- PA-RISC Floating-Point Format

PA-RISC General Registers

TotalView displays the PA-RISC 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 take with them.

<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>Always contains zero</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$r0</td>
</tr>
<tr>
<td>r1-r31</td>
<td>General registers</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$r1-$r31</td>
</tr>
<tr>
<td>pc</td>
<td>Current instruction pointer</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$pc</td>
</tr>
<tr>
<td>nxtpc</td>
<td>Next instruction pointer</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$nxtpc</td>
</tr>
<tr>
<td>pcs</td>
<td>Current instruction space</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$pcs</td>
</tr>
<tr>
<td>nxtpcs</td>
<td>Next instruction space</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$nxtpcs</td>
</tr>
</tbody>
</table>
## PA-RISC Process Status Word

For your convenience, TotalView interprets the bit settings of the PA-RISC Processor Status Word. You can edit the value of this word and set some of the bits listed in the following table.

<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>psw</td>
<td>Processor status word</td>
<td>$long</td>
<td>yes</td>
<td>no</td>
<td>$psw</td>
</tr>
<tr>
<td>sar</td>
<td>Shift amount register</td>
<td>$long</td>
<td>yes</td>
<td>no</td>
<td>$sar</td>
</tr>
<tr>
<td>sr0-sr7</td>
<td>Space registers</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$sr0-$sr7</td>
</tr>
<tr>
<td>recov</td>
<td>Recovery counter</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$recov</td>
</tr>
<tr>
<td>pid1-pid8</td>
<td>Protection IDs</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$pid1-$pid8</td>
</tr>
<tr>
<td>ccr</td>
<td>Coprocessor configuration</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$ccr</td>
</tr>
<tr>
<td>scr</td>
<td>SFU configuration register</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$scr</td>
</tr>
<tr>
<td>eiem</td>
<td>External interrupt enable mask</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$sieiem</td>
</tr>
<tr>
<td>iir</td>
<td>Interrupt instruction</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$iir</td>
</tr>
<tr>
<td>isr</td>
<td>Interrupt space</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$isr</td>
</tr>
<tr>
<td>ior</td>
<td>Interrupt offset</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$ior</td>
</tr>
<tr>
<td>cr24-cr26</td>
<td>Temporary registers</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$cr24-$cr26</td>
</tr>
<tr>
<td>tp</td>
<td>Thread pointer</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$tp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>0x0000000000800000</td>
<td>64-bit addressing enable</td>
</tr>
<tr>
<td>E</td>
<td>0x0000000004000000</td>
<td>Little-endian enable</td>
</tr>
<tr>
<td>S</td>
<td>0x0000000002000000</td>
<td>Secure interval timer</td>
</tr>
<tr>
<td>T</td>
<td>0x0000000001000000</td>
<td>Taken branch flag</td>
</tr>
<tr>
<td>H</td>
<td>0x0000000008000000</td>
<td>Higher-privilege transfer trap enable</td>
</tr>
<tr>
<td>L</td>
<td>0x0000000004000000</td>
<td>Lower-privilege transfer trap enable</td>
</tr>
<tr>
<td>N</td>
<td>0x0000000002000000</td>
<td>Nullify current instruction</td>
</tr>
<tr>
<td>X</td>
<td>0x0000000001000000</td>
<td>Data memory break disable</td>
</tr>
<tr>
<td>B</td>
<td>0x0000000008000000</td>
<td>Taken branch flag</td>
</tr>
<tr>
<td>C</td>
<td>0x0000000004000000</td>
<td>Code address translation enable</td>
</tr>
<tr>
<td>V</td>
<td>0x0000000002000000</td>
<td>Divide step correction</td>
</tr>
<tr>
<td>M</td>
<td>0x0000000001000000</td>
<td>High-priority machine check mask</td>
</tr>
<tr>
<td>O</td>
<td>0x0000000000800000</td>
<td>Ordered references</td>
</tr>
<tr>
<td>F</td>
<td>0x0000000000200000</td>
<td>Performance monitor interrupt unmask</td>
</tr>
<tr>
<td>R</td>
<td>0x0000000000100000</td>
<td>Recovery counter enable</td>
</tr>
<tr>
<td>Q</td>
<td>0x0000000000000008</td>
<td>Interrupt state collection enable</td>
</tr>
<tr>
<td>P</td>
<td>0x0000000000000004</td>
<td>Protection identifier validation enable</td>
</tr>
<tr>
<td>D</td>
<td>0x0000000000000002</td>
<td>Data address translation enable</td>
</tr>
<tr>
<td>I</td>
<td>0x0000000000000001</td>
<td>External interrupt unmask</td>
</tr>
<tr>
<td>C/B</td>
<td>0x000000000FF0000FF</td>
<td>Carry/borrow bits</td>
</tr>
</tbody>
</table>
PA-RISC Floating-Point Registers

The PA-RISC has 32 floating-point registers. The first four are used for status and exception registers. The rest can be addressed as 64-bit doubles, as two 32-bit floats in the right and left sides of the register, or even-odd pairs of registers as 128-bit extended floats.

<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>status</td>
<td>Status register</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$status</td>
</tr>
<tr>
<td>er1-er7</td>
<td>Exception registers</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$er1-$er7</td>
</tr>
<tr>
<td>fr4-fr31</td>
<td>Double floating-point registers</td>
<td>$double</td>
<td>yes</td>
<td>yes</td>
<td>$fr4-$fr31</td>
</tr>
<tr>
<td>fr4l-fr31l</td>
<td>Left half floating-point registers</td>
<td>$float</td>
<td>yes</td>
<td>yes</td>
<td>$fr4l-$fr31l</td>
</tr>
<tr>
<td>fr4r-fr31r</td>
<td>Right half floating-point registers</td>
<td>$float</td>
<td>yes</td>
<td>yes</td>
<td>$fr4r-$fr31r</td>
</tr>
<tr>
<td>fr4/5-fr30/fr31</td>
<td>Extended floating-point register pairs</td>
<td>$extended</td>
<td>yes</td>
<td>yes</td>
<td>$fr4/5-fr30/fr31</td>
</tr>
</tbody>
</table>

The floating-point status word controls the arithmetic rounding mode, enables user-level traps, enables floating-point exceptions, and indicates the results of comparisons.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rounding Mode</td>
<td>0</td>
<td>Round to nearest</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Round toward zero</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Round toward +infinity</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Round toward –infinity</td>
</tr>
<tr>
<td>Exception Enable and Exception Flag Bits</td>
<td>V</td>
<td>Invalid operation</td>
</tr>
<tr>
<td></td>
<td>Z</td>
<td>Division by zero</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>Overflow</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>Underflow</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>Inexact result</td>
</tr>
<tr>
<td>Comparison Fields</td>
<td>C</td>
<td>Compare bit; contains the result of the most recent queued compare instruction.</td>
</tr>
<tr>
<td></td>
<td>CQ</td>
<td>Compare queue; contains the result of the second-most recent queued compare through the twelfth-most recent queued compare. Each queued compare instruction shifts the CQ field right one bit and copies the C bit into the left-most position. This field occupies the same bits as the CA field and is undefined after a targeted compare.</td>
</tr>
<tr>
<td></td>
<td>CA</td>
<td>Compare array; an array of seven compare bits, each of which contains the result of the most recent compare instruction targeting that bit. This field occupies the same bits as the CQ field and is undefined after a queued compare.</td>
</tr>
<tr>
<td>Other Flags</td>
<td>T</td>
<td>Delayed trap</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>Denormalized as zero</td>
</tr>
</tbody>
</table>
PA-RISC Floating-Point Format

The PA-RISC processor supports the IEEE floating-point format.

IBM Cell Broadband Engine

TotalView displays Cell general registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each register, and the actions you can take with it.

<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–R127</td>
<td>General Registers</td>
<td>union $spu_vec128</td>
<td>yes</td>
<td>yes</td>
<td>$r0–$r127</td>
</tr>
<tr>
<td>FPSCR</td>
<td>Floating point status and control register</td>
<td>$int128</td>
<td>yes</td>
<td>yes</td>
<td>$fpscr</td>
</tr>
<tr>
<td>OBJECT_ID</td>
<td>Context ID created by application; used later by profiling tools to uniquely identify context</td>
<td>$long_long</td>
<td>no</td>
<td>yes</td>
<td>$object_id</td>
</tr>
<tr>
<td>ID</td>
<td>Instruction dispatch register</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$id</td>
</tr>
<tr>
<td>PC</td>
<td>Program counter</td>
<td>$code[272]</td>
<td>yes</td>
<td>yes</td>
<td>$pc</td>
</tr>
<tr>
<td>SP</td>
<td>Stack pointer</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$sp</td>
</tr>
<tr>
<td>FP</td>
<td>Floating point register</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$fp</td>
</tr>
<tr>
<td>LR</td>
<td>Link register</td>
<td>$code[657]</td>
<td>yes</td>
<td>yes</td>
<td>$lr</td>
</tr>
<tr>
<td>SRR0</td>
<td>Interrupt return address register</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$srr0</td>
</tr>
<tr>
<td>LSLR</td>
<td>Local storage limit register</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$lslr</td>
</tr>
<tr>
<td>DECR</td>
<td>SPU decrementer</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$decr</td>
</tr>
<tr>
<td>DECR_STATUS</td>
<td>Decrementer status</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$decr_status</td>
</tr>
<tr>
<td>CNTL</td>
<td>Read access to the status register</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$cntl</td>
</tr>
<tr>
<td>EVENT_MASK</td>
<td>Event mask for SPU interrupts</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$event_mask</td>
</tr>
<tr>
<td>EVENT_STATUS</td>
<td>Number of pending SPU events</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$event_status</td>
</tr>
<tr>
<td>IBOX</td>
<td>Second SPU to CPU communication mailbox</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$ibox</td>
</tr>
<tr>
<td>MBOX</td>
<td>First SPU to CPU communication mailbox</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$mbox</td>
</tr>
<tr>
<td>WBOX</td>
<td>CPU to SPU communication mailbox</td>
<td>$int128</td>
<td>no</td>
<td>yes</td>
<td>$wbox</td>
</tr>
<tr>
<td>IBOX_STAT</td>
<td>Length of the IBOX queue</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$ibox_stat</td>
</tr>
<tr>
<td>MBOX_STAT</td>
<td>Length of the MBOX queue</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$mbox_stat</td>
</tr>
<tr>
<td>WBOX_STAT</td>
<td>Length of the WBOX queue</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$wbox_stat</td>
</tr>
<tr>
<td>SIGNAL1</td>
<td>Signal notification channel</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$signal1</td>
</tr>
<tr>
<td>SIGNAL1_TYPE</td>
<td>Behavior of signal1 notification channel</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$signal1_type</td>
</tr>
<tr>
<td>SIGNAL2</td>
<td>Signal notification channel</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$signal2</td>
</tr>
<tr>
<td>SIGNAL2_TYPE</td>
<td>Behavior of signal1 notification channel</td>
<td>$int</td>
<td>no</td>
<td>yes</td>
<td>$signal2_type</td>
</tr>
<tr>
<td>MFC</td>
<td>Access to the MFC DMA area</td>
<td>$long_long</td>
<td>no</td>
<td>yes</td>
<td>$mfc</td>
</tr>
</tbody>
</table>
This section contains the following information:

- Power General Registers
- Power MSR Register
- Power Floating-Point Registers
- Power FPSCR Register
- Using the Power FPSCR Register

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>$int</td>
<td>yes</td>
<td>yes</td>
<td>$r0</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>RTOC</td>
<td>TOC pointer</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$rtoc</td>
</tr>
<tr>
<td>R3 – R31</td>
<td>General registers 3 – 31</td>
<td>$int</td>
<td>yes</td>
<td>yes</td>
<td>$r3 – $r31</td>
</tr>
<tr>
<td>INUM</td>
<td></td>
<td>$int</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</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</td>
<td>yes</td>
<td>no</td>
<td>$ctr</td>
</tr>
<tr>
<td>CR</td>
<td>Condition register (see below)</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$cr</td>
</tr>
<tr>
<td>XER</td>
<td>Integer exception register (see below)</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$xer</td>
</tr>
<tr>
<td>DAR</td>
<td>Data address register</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$dar</td>
</tr>
<tr>
<td>MQ</td>
<td>MQ register</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$mq</td>
</tr>
<tr>
<td>MSR</td>
<td>Machine state register</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$msr</td>
</tr>
<tr>
<td>SEG0 – SEG9</td>
<td>Segment registers 0 – 9</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$seg0 – $seg9</td>
</tr>
<tr>
<td>SG10 – SG15</td>
<td>Segment registers 10 – 15</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$sg10 – $sg15</td>
</tr>
<tr>
<td>SCNT</td>
<td>SS_COUNT</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$scnt</td>
</tr>
<tr>
<td>SAD1</td>
<td>SS_ADDR 1</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$sad1</td>
</tr>
<tr>
<td>SAD2</td>
<td>SS_ADDR 2</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$sad2</td>
</tr>
<tr>
<td>SCD1</td>
<td>SS_CODE 1</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$scd1</td>
</tr>
</tbody>
</table>
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:

\[0x20000002 (536870914) (STD:0x00,SL:2,CA)\]

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>0x0000000000040000</td>
<td>POW</td>
<td>Power management enable</td>
</tr>
<tr>
<td>0x0000000000020000</td>
<td>TGPR</td>
<td>Temporary GPR mapping</td>
</tr>
<tr>
<td>0x0000000000010000</td>
<td>ILE</td>
<td>Exception little-endian mode</td>
</tr>
<tr>
<td>0x0000000000008000</td>
<td>EE</td>
<td>External interrupt enable</td>
</tr>
<tr>
<td>0x0000000000004000</td>
<td>PR</td>
<td>Privilege level</td>
</tr>
<tr>
<td>0x0000000000002000</td>
<td>FP</td>
<td>Floating-point available</td>
</tr>
<tr>
<td>0x0000000000001000</td>
<td>ME</td>
<td>Machine check enable</td>
</tr>
<tr>
<td>0x0000000000000800</td>
<td>FE0</td>
<td>Floating-point exception mode 0</td>
</tr>
<tr>
<td>0x0000000000000400</td>
<td>SE</td>
<td>Single-step trace enable</td>
</tr>
<tr>
<td>0x0000000000000200</td>
<td>BE</td>
<td>Branch trace enable</td>
</tr>
<tr>
<td>0x0000000000000100</td>
<td>FE1</td>
<td>Floating-point exception mode 1</td>
</tr>
<tr>
<td>0x0000000000000040</td>
<td>IP</td>
<td>Exception prefix</td>
</tr>
<tr>
<td>0x0000000000000200</td>
<td>IR</td>
<td>Instruction address translation</td>
</tr>
<tr>
<td>0x0000000000000100</td>
<td>DR</td>
<td>Data address translation</td>
</tr>
<tr>
<td>0x0000000000000020</td>
<td>RI</td>
<td>Recoverable exception</td>
</tr>
<tr>
<td>0x0000000000000001</td>
<td>LE</td>
<td>Little-endian mode 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>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: $\infty - \infty$</td>
</tr>
<tr>
<td>0x00400000</td>
<td>VXIDI</td>
<td>Floating-point invalid operation exception: $\infty / \infty$</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: $\infty * \infty$</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>
</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 FPSCR 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 3 of the TotalView 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.
Intel IA-64

This section contains the following information:

- Intel IA-64 General Registers
- "IA-64 Processor Status Register Fields (PSR)" on page 302
- "Current Frame Marker Register Fields (CFM)" on page 303
- "Register Stack Configuration Register Fields (RSC)" on page 303
- "Previous Function State Register Fields (PFS)" on page 303
- "Floating Point Registers" on page 304
- "Floating Point Status Register Fields" on page 304

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.

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>N</td>
<td>Y</td>
<td>$rsc</td>
</tr>
<tr>
<td>bsp</td>
<td>rse backing store pointer (AR 17)</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$bsp</td>
</tr>
<tr>
<td>bspstore</td>
<td>rse backing store pointer for memory stores (AR 18)</td>
<td>$long</td>
<td>N</td>
<td>Y</td>
<td>$bspstore</td>
</tr>
<tr>
<td>rnat</td>
<td>rse NAT collection register (AR 19)</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$rnat</td>
</tr>
<tr>
<td>ccv</td>
<td>compare and exchange value register (AR 32)</td>
<td>$long</td>
<td>Y</td>
<td>Y</td>
<td>$ccv</td>
</tr>
</tbody>
</table>
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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>be</td>
<td>big-endian enable</td>
</tr>
<tr>
<td>2</td>
<td>up</td>
<td>user performance monitor enable</td>
</tr>
<tr>
<td>3</td>
<td>ac</td>
<td>alignment check</td>
</tr>
<tr>
<td>4</td>
<td>mfl</td>
<td>lower (f2-f31) floating point registers written</td>
</tr>
<tr>
<td>5</td>
<td>mfh</td>
<td>upper (f32-f127) floating point registers written</td>
</tr>
<tr>
<td>13</td>
<td>ic</td>
<td>interruption collection</td>
</tr>
<tr>
<td>14</td>
<td>i</td>
<td>interrupt bit</td>
</tr>
<tr>
<td>15</td>
<td>pk</td>
<td>protection key enable</td>
</tr>
<tr>
<td>17</td>
<td>dt</td>
<td>data address translation</td>
</tr>
<tr>
<td>18</td>
<td>dfl</td>
<td>disabled lower floating point register set</td>
</tr>
<tr>
<td>19</td>
<td>dfh</td>
<td>disabled upper floating point register set</td>
</tr>
<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>
</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.

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6:0</td>
<td>sof</td>
<td>Size of frame</td>
</tr>
<tr>
<td>13:7</td>
<td>sol</td>
<td>Size of locals portion of stack frame</td>
</tr>
<tr>
<td>17:14</td>
<td>sor</td>
<td>Size of rotating portion of stack frame (number of rotating registers is sor*8)</td>
</tr>
<tr>
<td>24:18</td>
<td>rrb.gr</td>
<td>Register rename base for general registers</td>
</tr>
<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>

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>1:0</td>
<td>00</td>
<td>enforced lazy</td>
</tr>
<tr>
<td>1:0</td>
<td>01</td>
<td>load intensive</td>
</tr>
<tr>
<td>1:0</td>
<td>10</td>
<td>store intensive</td>
</tr>
<tr>
<td>1:0</td>
<td>11</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>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/unnormar 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>
This section contains the following information:

- “Intel x86 General Registers” on page 305
- “Intel x86 Floating-Point Registers” on page 306
- “Intel x86 FPCR Register” on page 306
- “Intel x86 FPSR Register” on page 307
- “Intel x86 MXCSR Register” on page 308

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></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$ecx</td>
</tr>
<tr>
<td>EDX</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$edx</td>
</tr>
<tr>
<td>EBX</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$ebx</td>
</tr>
<tr>
<td>EBP</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$ebp</td>
</tr>
<tr>
<td>ESP</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$esp</td>
</tr>
<tr>
<td>ESI</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$esi</td>
</tr>
<tr>
<td>EDI</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$edi</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></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>
<tr>
<td>ES</td>
<td></td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$es</td>
</tr>
<tr>
<td>FS</td>
<td></td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fs</td>
</tr>
<tr>
<td>GS</td>
<td></td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$gs</td>
</tr>
<tr>
<td>EFLAGS</td>
<td></td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$eflags</td>
</tr>
<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></td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$fault</td>
</tr>
<tr>
<td>TEMP</td>
<td></td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$temp</td>
</tr>
<tr>
<td>INUM</td>
<td></td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$inum</td>
</tr>
<tr>
<td>ECODE</td>
<td></td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$ecode</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>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>FPSELT</td>
<td>Instruction selector</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpisel</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>FPDSEL</td>
<td>Data selector</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpdssel</td>
</tr>
<tr>
<td>MXCSR</td>
<td>SSE status and control</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$mxcsv</td>
</tr>
<tr>
<td>MXCSR_M</td>
<td>MXCSR mask</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$mxcsr_mask</td>
</tr>
<tr>
<td>ASK</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$xmm0_l</td>
</tr>
<tr>
<td>XMM0_L</td>
<td>Streaming SIMD Extension: left half</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$xmm0_l</td>
</tr>
<tr>
<td>XMM7_L</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$xmm7_l</td>
</tr>
<tr>
<td>XMM0_H</td>
<td>Streaming SIMD Extension: right half</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$xmm0_h</td>
</tr>
<tr>
<td>XMM7_H</td>
<td></td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$xmm7_h</td>
</tr>
</tbody>
</table>

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 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 “Schar[16]”, “Sfloat[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.
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 3 of the TotalView 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;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>
</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>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>

The MIPS processor supports the IEEE floating-point format.
**MIPS General Registers**

TotalView displays the MIPS general-purpose 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.

Programs compiled with `-64` or `-n32` have 64-bit registers. TotalView uses `$long` for `-64` compiled programs and `$long long` for `-n32` compiled programs.

<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>ZERO</td>
<td>Always has the value 0</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$zero</td>
</tr>
<tr>
<td>AT</td>
<td>Reserved for the assembler</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$at</td>
</tr>
<tr>
<td>V0 – V1</td>
<td>Function value registers</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$v0 – $v1</td>
</tr>
<tr>
<td>A0 – A7</td>
<td>Argument registers</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$a0 – $a7</td>
</tr>
<tr>
<td>T0 – T3</td>
<td>Temporary registers</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$t0 – $t3</td>
</tr>
<tr>
<td>S0 – S7</td>
<td>Saved registers</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$s0 – $s7</td>
</tr>
<tr>
<td>T8 – T9</td>
<td>Temporary registers</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$t8 – $t9</td>
</tr>
<tr>
<td>K0 – K1</td>
<td>Reserved for the operating system</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$k1 – $k2</td>
</tr>
<tr>
<td>GP</td>
<td>Global pointer</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$gp</td>
</tr>
<tr>
<td>SP</td>
<td>Stack pointer</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$sp</td>
</tr>
<tr>
<td>S8</td>
<td>Hardware frame pointer</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$s8</td>
</tr>
<tr>
<td>RA</td>
<td>Return address register</td>
<td>$code[]</td>
<td>no</td>
<td>yes</td>
<td>$ra</td>
</tr>
<tr>
<td>MDLO</td>
<td>Multiply/Divide special register, holds least-significant bits of multiply, quotient of divide</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$mdlo</td>
</tr>
<tr>
<td>MDHI</td>
<td>Multiply/Divide special register, holds most-significant bits of multiply, remainder of divide</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$mdhi</td>
</tr>
<tr>
<td>CAUSE</td>
<td>Cause register</td>
<td>$long</td>
<td>yes</td>
<td>yes</td>
<td>$cause</td>
</tr>
<tr>
<td>EPC</td>
<td>Program counter</td>
<td>$code[]</td>
<td>no</td>
<td>yes</td>
<td>$epc</td>
</tr>
<tr>
<td>SR</td>
<td>Status register</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$sr</td>
</tr>
<tr>
<td>VFP</td>
<td>Virtual frame pointer</td>
<td>$long</td>
<td>no</td>
<td>no</td>
<td>$vfp</td>
</tr>
</tbody>
</table>

The virtual frame pointer is a software register that TotalView maintains. It is not an actual hardware register. TotalView computes the VFP as part of stack backtrace.
**MIPS SR Register**

For your convenience, TotalView interprets the bit settings of the SR register as outlined in the next table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Bit Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000001</td>
<td>IE</td>
<td>Interrupt enable</td>
</tr>
<tr>
<td>0x00000002</td>
<td>EXL</td>
<td>Exception level</td>
</tr>
<tr>
<td>0x00000004</td>
<td>ERL</td>
<td>Error level</td>
</tr>
<tr>
<td>0x00000008</td>
<td>S</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>0x00000010</td>
<td>U</td>
<td>User mode</td>
</tr>
<tr>
<td>0x00000018</td>
<td>U</td>
<td>Undefined (implemented as User mode)</td>
</tr>
<tr>
<td>0x00000000</td>
<td>K</td>
<td>Kernel mode</td>
</tr>
<tr>
<td>0x00000020</td>
<td>UX</td>
<td>User mode 64-bit addressing</td>
</tr>
<tr>
<td>0x00000040</td>
<td>SX</td>
<td>Supervisor mode 64-bit addressing</td>
</tr>
<tr>
<td>0x00000080</td>
<td>KX</td>
<td>Kernel mode 64-bit addressing</td>
</tr>
<tr>
<td>0x0000FF00</td>
<td>IM=i</td>
<td>Interrupt Mask value is i</td>
</tr>
<tr>
<td>0x00010000</td>
<td>DE</td>
<td>Disable cache parity/ECC</td>
</tr>
<tr>
<td>0x00020000</td>
<td>CE</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x00040000</td>
<td>CH</td>
<td>Cache hit</td>
</tr>
<tr>
<td>0x00080000</td>
<td>NMI</td>
<td>Non-maskable interrupt occurred</td>
</tr>
<tr>
<td>0x00100000</td>
<td>SR</td>
<td>Soft reset or NMI exception</td>
</tr>
<tr>
<td>0x00200000</td>
<td>TS</td>
<td>TLB shutdown has occurred</td>
</tr>
<tr>
<td>0x00400000</td>
<td>BEV</td>
<td>Bootstrap vectors</td>
</tr>
<tr>
<td>0x02000000</td>
<td>RE</td>
<td>Reverse-Endian bit</td>
</tr>
<tr>
<td>0x04000000</td>
<td>FR</td>
<td>Additional floating-point registers enabled</td>
</tr>
<tr>
<td>0x08000000</td>
<td>RP</td>
<td>Reduced power mode</td>
</tr>
<tr>
<td>0x10000000</td>
<td>CU0</td>
<td>Coprocessor 0 usable</td>
</tr>
<tr>
<td>0x20000000</td>
<td>CU1</td>
<td>Coprocessor 1 usable</td>
</tr>
<tr>
<td>0x40000000</td>
<td>CU2</td>
<td>Coprocessor 2 usable</td>
</tr>
<tr>
<td>0x80000000</td>
<td>XX</td>
<td>MIPS IV instructions usable</td>
</tr>
</tbody>
</table>

**MIPS Floating-Point Registers**

TotalView displays the MIPS floating-point registers in the Stack Frame Pane of the Process Window. Here is a table that 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, F2</td>
<td>Hold results of floating-point type function; ( f0 ) has the real part, ( f2 ) has the imaginary part</td>
<td>$double\</td>
<td>yes</td>
<td>yes</td>
<td>$f0, $f2</td>
</tr>
<tr>
<td>F1 – F3, F4 – F11</td>
<td>Temporary registers</td>
<td>$double\</td>
<td>yes</td>
<td>yes</td>
<td>$f1 – $f3, $f4 – $f11</td>
</tr>
<tr>
<td>F12 – F19</td>
<td>Pass single- or double-precision actual arguments</td>
<td>$double\</td>
<td>yes</td>
<td>yes</td>
<td>$f12 – $f19</td>
</tr>
</tbody>
</table>
For your convenience, TotalView interprets the bit settings of the MIPS FCSR register. You can edit the value of the FCSR 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>RM = RN</td>
<td>0x00000000</td>
<td>Round to nearest</td>
</tr>
<tr>
<td>RM = RZ</td>
<td>0x00000001</td>
<td>Round toward zero</td>
</tr>
<tr>
<td>RM = RP</td>
<td>0x00000002</td>
<td>Round toward positive infinity</td>
</tr>
<tr>
<td>RM = RM</td>
<td>0x00000003</td>
<td>Round toward negative infinity</td>
</tr>
<tr>
<td>flags = (I)</td>
<td>0x00000004</td>
<td>Flag = inexact result</td>
</tr>
<tr>
<td>flags = (U)</td>
<td>0x00000008</td>
<td>Flag = underflow</td>
</tr>
<tr>
<td>flags = (O)</td>
<td>0x00000010</td>
<td>Flag = overflow</td>
</tr>
<tr>
<td>flags = (Z)</td>
<td>0x00000020</td>
<td>Flag = divide by zero</td>
</tr>
<tr>
<td>flags = (V)</td>
<td>0x00000040</td>
<td>Flag = invalid operation</td>
</tr>
<tr>
<td>enables = (I)</td>
<td>0x00000080</td>
<td>Enables = inexact result</td>
</tr>
<tr>
<td>enables = (U)</td>
<td>0x0000100</td>
<td>Enables = underflow</td>
</tr>
<tr>
<td>enables = (O)</td>
<td>0x0000200</td>
<td>Enables = overflow</td>
</tr>
<tr>
<td>enables = (Z)</td>
<td>0x0000800</td>
<td>Enables = divide by zero</td>
</tr>
<tr>
<td>enables = (V)</td>
<td>0x0001000</td>
<td>Enables = invalid operation</td>
</tr>
<tr>
<td>cause = (I)</td>
<td>0x0000100</td>
<td>Cause = inexact result</td>
</tr>
<tr>
<td>cause = (U)</td>
<td>0x0000200</td>
<td>Cause = underflow</td>
</tr>
<tr>
<td>cause = (O)</td>
<td>0x0001000</td>
<td>Cause = overflow</td>
</tr>
<tr>
<td>cause = (Z)</td>
<td>0x0000800</td>
<td>Cause = divide by zero</td>
</tr>
<tr>
<td>cause = (V)</td>
<td>0x0002000</td>
<td>Cause = invalid operation</td>
</tr>
<tr>
<td>cause = (E)</td>
<td>0x00000800</td>
<td>Cause = unimplemented</td>
</tr>
<tr>
<td>FCC = 0/c</td>
<td>0x00800000</td>
<td>FCC = Floating-Point Condition Code 0; c = Condition bit</td>
</tr>
<tr>
<td>FS</td>
<td>0x01000000</td>
<td>Flush to zero</td>
</tr>
<tr>
<td>FCC = (1)</td>
<td>0x02000000</td>
<td>FCC = Floating-Point Condition Code 1</td>
</tr>
<tr>
<td>FCC = (2)</td>
<td>0x04000000</td>
<td>FCC = Floating-Point Condition Code 2</td>
</tr>
<tr>
<td>FCC = (3)</td>
<td>0x08000000</td>
<td>FCC = Floating-Point Condition Code 3</td>
</tr>
<tr>
<td>FCC = (4)</td>
<td>0x10000000</td>
<td>FCC = Floating-Point Condition Code 4</td>
</tr>
<tr>
<td>FCC = (5)</td>
<td>0x20000000</td>
<td>FCC = Floating-Point Condition Code 5</td>
</tr>
<tr>
<td>FCC = (6)</td>
<td>0x40000000</td>
<td>FCC = Floating-Point Condition Code 6</td>
</tr>
<tr>
<td>FCC = (7)</td>
<td>0x80000000</td>
<td>FCC = Floating-Point Condition Code 7</td>
</tr>
</tbody>
</table>
Using the MIPS FCSR Register
You can change the value of the MIPS FCSR register 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 FCSR register in the Stack Frame Pane. In this case, you would change the bit setting for the FCSR to include 0x400. The string displayed next to the FCSR register should now include enables=(Z). Now, when your program divides by zero, it receives a SIGFPE signal, which you can catch with TotalView. See “Setting Up a Debugging Session” in the TotalView Users Guide for more information.

MIPS Delay Slot Instructions
On the MIPS architecture, jump and branch instructions have a “delay slot”. This means that the instruction after the jump or branch instruction is executed before the jump or branch is executed.

In addition, there is a group of “branch likely” conditional branch instructions in which the instruction in the delay slot is executed only if the branch is taken.

The MIPS processors execute the jump or branch instruction and the delay slot instruction as an indivisible unit. If an exception occurs as a result of executing the delay slot instruction, the branch or jump instruction is not executed, and the exception appears to have been caused by the jump or branch instruction.

This behavior of the MIPS processors affects both the TotalView instruction step command and TotalView breakpoints.

The TotalView instruction step command will step both the jump or branch instruction and the delay slot instruction as if they were a single instruction. If a breakpoint is placed on a delay slot instruction, execution will stop at the jump or branch preceding the delay slot instruction, and TotalView will not know that it is at a breakpoint. At this point, attempting to continue the thread that hit the breakpoint without first removing the breakpoint will cause the thread to hit the breakpoint again without executing any instructions. Before continuing the thread, you must remove the breakpoint. If you need to reestablish the breakpoint, you might then use the instruction step command to execute just the delay slot instruction and the branch.

A breakpoint placed on a delay slot instruction of a branch likely instruction will be hit only if the branch is going to be taken.
This section has the following information:
- SPARC General Registers
- SPARC PSR Register
- SPARC Floating-Point Registers
- SPARC FPSR Register
- Using the SPARC FPSR Register

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>$int</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>Program counter</td>
<td>$code[]</td>
<td>no</td>
<td>yes</td>
<td>$pc</td>
</tr>
<tr>
<td>nPC</td>
<td>Next program counter</td>
<td>$code[]</td>
<td>no</td>
<td>yes</td>
<td>$npc</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,</td>
<td>Floating-point registers (f registers), used singly</td>
<td>$float</td>
<td>no</td>
<td>yes</td>
<td>$f0, $f1, $f0_f1</td>
</tr>
<tr>
<td>F0_F1</td>
<td>Floating-point registers (f registers), used singly</td>
<td>$float</td>
<td>yes</td>
<td>yes</td>
<td>$f2–$f31</td>
</tr>
<tr>
<td>F0, F1,</td>
<td>Floating-point registers (f registers), used as pairs</td>
<td>$double</td>
<td>no</td>
<td>yes</td>
<td>$f0, $f1, $f0_f1</td>
</tr>
<tr>
<td>F0_F1</td>
<td>Floating-point registers (f registers), used as pairs</td>
<td>$double</td>
<td>yes</td>
<td>yes</td>
<td>$2–$f30_f31</td>
</tr>
<tr>
<td>FPCR</td>
<td>Floating-point control register</td>
<td>$int</td>
<td>no</td>
<td>no</td>
<td>$fpcr</td>
</tr>
<tr>
<td>FPSR</td>
<td>Floating-point status register</td>
<td>$int</td>
<td>yes</td>
<td>no</td>
<td>$fpsr</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.
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.

### SPARC FPSR Register

<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>0x00000200</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>0x00000400</td>
<td>Floating-point trap type IEEE Exception</td>
</tr>
<tr>
<td>UFIN</td>
<td>0x00000800</td>
<td>Floating-point trap type Unfinished FPop</td>
</tr>
<tr>
<td>UIMP</td>
<td>0x00000c00</td>
<td>Floating-point trap type Unimplemented FPop</td>
</tr>
<tr>
<td>SEQE</td>
<td>0x00001000</td>
<td>Floating-point trap type Sequence Error</td>
</tr>
<tr>
<td>NS</td>
<td>0x00040000</td>
<td>Nonstandard floating-point FAST mode</td>
</tr>
<tr>
<td>TEM=NX</td>
<td>0x00080000</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>

### 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 \texttt{0x01000000} so that TotalView traps the “divide by zero” bit. The string displayed next to the FPSR register should now include \texttt{TEM=(DZ)}. Now, when your program divides by zero, it receives a \texttt{SIGFPE} signal, which you can catch with TotalView. See “Handling Signals” in Chapter 3 of the TotalView 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 \texttt{AEXC=(DZ)} bit.
Appendix A: MPI Startup
Here you will find information that will allow you to create startup profiles for environments that TotalView Technologies doesn’t define. These definitions will show up in the Parallel tab of the New Program dialog box.
Overview

TotalView Technologies products know about different 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.

The following explanation is for TotalView and MemoryScape.

The only way MemoryScape users can alter the way an MPI program starts up is by altering the parallel_support.tvd file, which is contained within the totalview/lib installation directory area. TotalView users can also alter this file and they can create a local definition.

If you are using a locally-installed MPI implementation, you should add it to your PATH variable. By default, our products 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 and MemoryScape 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, you would need to 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 is to copy a similar configuration from the TV::private::parallel_configs_base variable to the TV::parallel_configs variable, then make changes.
When you add configurations, they are simply added to a list. This means that if TotalView Technologies supplies a definition named `foo` and you create a definition named `foo`, both exist and your product chooses the first one in the list. Because both are displayed, you must be careful to give each new definition a unique name.

### Customizing Your Parallel Configuration

The **File > New** dialog box (TotalView) or the **Add parallel program** screen (MemoryScape) lets you select a parallel configuration. If the default configurations that TotalView Technologies provides do not meet your needs, you can either overwrite these configurations or create new ones by setting the `TV::parallel_configs` variable. Here are three examples:

```plaintext
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 -tvsv %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;
    service_tids:  2 3 4;
}
```

Customizing Your Parallel Configuration

```c
comm_world: 0;
pretest: test -x /bin/poe
msg_lib: /usr/lpp/ppe.poe/lib/%m
```

All lines (except for comments) end with a semi-colon (;). Add spaces freely to make what you enter readable 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.

When running TotalView, you can set this variable in two places:

- Your system's `.tvdrc` file. If you set this variable here, everyone using this TotalView version will see the definition.
- Your `.totalview/tvdrc` file. You will be the only person to see this definition when you start TotalView.

The fields that you can set are as follows:

- **comm_world** (optional) 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,

\[ \text{NAME1=VALUE1, NAME2=VALUE2, NAME3=VALUE3} \]

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,

\[ \text{NAME1, NAME2, NAME3} \]

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 tar-
get. While this option does not work with all MPI implementa-
tions, 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 everything is OK. If the shell command exit is not as expected, TotalView or MemoryScape complains and asks for permission before continuing.

**pretext_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 TotalView, 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 processes.

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.
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 process lifetime. Our products attach to this process and communicate with it using its debug interface. For example, 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, environment, and temporary files. However, the script does not run as part of the parallel job. This script must understand the `-tv` command-line option and the TOTALVIEW environment variable.

**bootstrap**: The parallel system attempts to launch an uninstrumented MPI by interposing TotalView or MemoryScape 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.
Index

-serial device 255
—--add-gnu-debuglink command-line
   option 268

Symbols

# scoping separator character 27, 31, 70
$mpiexec variable 321
$newval variable in watchpoints 111
$oldval variable in watchpoints 111
$stop function 49, 84
%B server launch replacement character 257
%C server launch replacement character 257
%D path name replacement character 257
%H hostname replacement character 257
%L host and port replacement character 257
%N line number replacement character 258
%P password replacement character 258
%S source file replacement character 258
%t1 file replacement character 258
%t2 file replacement character 258
%V verbosity setting replacement character 259
* expr 233
.debug.gnu_debuglink copy 268
totalview/lib_cache subdirectory 34
tvd files 152
/proc file system 276, 277
= symbol for PC of current buried stack frame 71
> symbol for PC 71
@ symbol for action point 71

A

--a option to totalview command 242
ac, see dactions command
acquiring processes 204
action point
   identifiers 19
Action Point > Save All command 195
action point identifiers 46
action points
   autoloading 195
   default for newly created 190
   default property 191
   deleting 39, 131, 142, 145, 167
   disabling 19, 41, 41
   displaying 19
   enabling 19
   identifiers 19
   information about 19, 125
   loading 19
   loading automatically 246
   loading saved information 20
   reenabling 46
   saving 19
   saving information about 20
   scope of what is stopped 191
   setting at location 19
   sharing 190
   stopping when reached 218
actionpoint
   properties 131
actionpoint command 131
actions, see dactions command
activating type transformations 237
Add parallel program screen 320
adding group members 58
adding groups 57
address 153
address property 131
addressing_callback 169
advancing by steps 100
after_checkpointing options 36
aggregate data 229
AIX
   compiling on 265
   linking C++ to dbfork library 272
   linking to dbfork library 272
   swap space 278
aix_use_fast_trap command-line option 242
aix_use_fast_trap variable 192
alias command 16
aliases
   default 16
   removing 127
Alpha
   architecture 291
   floating-point registers 292
   FPCR register 293
   general registers 291
append, see dlappend command
appending to CLI variable lists 69
architectures 191
   Alpha 291
   HP PA-RISC 293
   Intel IA-64 301
   Intel-x86 287, 305
   MIPS 308
   PowerPC 297
   SPARC 313
arenas 52, 80
ARGS variable 185
ARGS(dpmid) variable 91, 95
ARGS_DEFAULT variable 91, 95, 185
arguments
   command line 95
   default 185
   for totalview command 241
barrier_stop_all variable

barrier, barrier is satisfied

back-tick analogy

background command-line option

ba, b, B

automatically attaching to processes

automatic dereferencing of arrays

auto_save_breakpoints variable

auto_read_symbols_at_stop variable

auto_load_breakpoints variable

auto_deref_initial_c variable

auto_deref_initial_fortran variable

auto_deref_nested_c variable

auto_deref_nested_fortran variable

auto_load_breakpoints variable

auto_read_symbols_at_stop variable

auto_save_breakpoints variable

automatic dereferencing of arrays

automatically attaching to processes

auto_array_cast_bounds variable

auto_array_cast_enabled variable

auto_deref_in_all_c variable

auto_deref_in_all_fortran variable

auto_deref_initial_c variable

auto_deref_initial_fortran variable

auto_deref_nested_c variable

auto_deref_nested_fortran variable

auto_array_cast_bounds variable

auto_array_cast_enabled variable

auto_deref_in_all_c variable

auto_deref_in_all_fortran variable

auto_deref_initial_c variable

auto_deref_initial_fortran variable

auto_deref_nested_c variable

auto_deref_nested_fortran variable

auto_load_breakpoints variable

auto_read_symbols_at_stop variable

auto_save_breakpoints variable

automatic dereferencing of arrays

automatically attaching to processes

by_language_rules 150

by_path 150

by_type_index 150

C

C language escape characters 22

C shell 277

C++

demangler 201, 243

including libdbfork.h 272

STL instantiation 230

c_type_strings 198

c_type_strings variable 198

cache, flushing 34

cache, see dcache command

call stack 109

displaying 117

see also, stack frame

call tree saved position 224

callback command-line option 253, 254

callback list 211

callback_host 254

callback_ports 254

callbacks 210

after loading a program 214

when opening the CLI 213

capture command 18, 121

case sensitive searching 215

cast subcommand 150

Cast to array with bounds checkbox 193

casting variables 86

ccq command-line option 242

CGROUP variable 186

changing CLI variables 97

changing dynamic context 109

changing focus 52

changing value of program variable 22, 36, 59, 67, 82, 94, 105, 107

chase_mouse variable 222

checking interior pointers variable 221

checkpoint, see dcheckpoint command

checkpointing

preserving IDs 35

process state 35

reattaching to parallel 35

restarting 93

scope 35

class transformations 231

classes, transforming 233

CLI

activated from GUI flag 222

sourcing files 232
CLI commands
action points 15
alias 16
capture 18, 121
dactions 19
dassign 22
dattach 24
dbarrier 27
dbreak 31
dcache 34
dcheckpoint 35
dcont 38
ddelete 39
ddetach 40
disable 41
ddlopen 42
ddown 45
dec2hex 134
denable 46
dexamine 47
dflush 49, 84
dfocus 52
dga 54
dgo 56
dgroups 57
dhalt 62
dheap 63
dhold 67
dkill 68
dappend 69
dlist 70
dlist command 189
dload 73
dmstat 76
dnext 78
dnexti 80
dout 82
dprint 84
dptsets 88
drun 91
drestart 93
drun 68, 95
drun, reissuing 96
dset 97
dstatus 99
dstep 100
dstepi 103
dunhold 105
dunset 106
duntil 107
dup 109
dwait 110
dwatch 111
dwhat 114
dwhere 117
dworker 119
environment 13
executing immediately 244
execution control 14
exit 120
help 121
initialization 13
overview 13, 129
program information 14
quit 122
responding to 149
stty 126
summary 3
termination 13
TV::actionpoint 131
TV::dll command 43, 135
TV::errorCodes 136
TV::expr 84, 137
TV::focus_groups 139
TV::focus_processes 140
TV::focus_threads 141
TV::group 142
TV::hex2dec 144
TV::process 145
TV::read_symbols command 148
TV::respond 149
TV::scope 150
TV::source_process_startup 152
TV::symbol 153
TV::thread 165
TV::type 167
TV::type_transformatio 169
unalias 127
CLI variables
aix_use_fast_trap 192
ARGS 185
ARGS_DEFAULT 185
ask_on_dlopen 193
auto_array_cast_bounds 193
auto_array_cast_enabled 193
auto_deref_in_all_c 193
auto_deref_in_all_fortran 193
auto_deref_initial_fortran 194
auto_deref_initial_c 194
auto_deref_nested_c 194
auto_deref_nested_fortran 194
auto_load_breakpoints 195
auto_read_symbols_at_stop 195
auto_save_breakpoints 195
BARRIER_STOP_ALL 27, 29, 185
barrier_stop_all 195
BARRIER_STOP_WHEN_DONE 27, 186
barrier_stop_when_done 196
blue_gene_launch_string 196
bluegene_io_interface 196
bluegene_server_launch_timeout 196
bulk_launch_base_timeout 197
bulk_launch_enabled 197
bulk_launch_incr_timeout 197
bulk_launch_string 197
bulk_launch_tmpfile1_header_line 197
bulk_launch_tmpfile1_host_lines 197
bulk_launch_tmpfile1_trailer_line 197
bulk_launch_tmpfile2_header_line 197
bulk_launch_tmpfile2_host_lines 198
bulk_launch_tmpfile2_trailer_line 198
c_type_strings 198
CGROUP 186
changing 97
chase_mouse 222
comline_patch_area_base 199
comline_path_area_length 199
COMMAND_EDITING 186
command_editing 199
compile_expressions 200
compiler_vars 200
control_c_quick_shutdown 201
copyright_string 201
current_cplusplus_demangler 201
current_fortran_demangler 201
data_format_double 202
data_format_ext 203
data_format_int16 203
data_format_int32 203
data_format_int64 203
data_format_int8 203
data_format_single 204
data_format_singles 204
dbork 204
default_value 97
default_snippet_extent 220
default_stderr_append 204
default_stderr_filename 204
default_stderr_is_stdout 205
default_stdin_filename 204
default_stdout_append 205
default_stdout_filename 205
deleting 97
display_assembler_symbolically 205
display_bytes_kb_mb 222
display_font_dpi 222
dll_ignore_prefix 205
dll_read_all_symbols 205
dll_read_loader_symbols_only 205
dll_read_no_symbols 206
dll_stop_suffix 206
dpvm 206
dump_core 207
dwhere_qualification_level 207
dynamic 208
ditor_launch_string 208
enabled 222
local_server 212
local_server_launch_string 212
MAX_LIST 70, 189
message_queue 213
npt_threads 213
OBJECT_SEARCH_MAPPINGS 187, 189
OBJECT_SEARCH_PATH 189
parallel 213
parallel_attach 213
parallel_configs 319, 320
parallel_stop 214
platform 214
pop_at_breakpoint 226
pop_on_error 226
process_load_callbacks 214
PROMPT 189
PTSET 189
pvm 214, 215
restart_threshold 215
save_window_pipe_or_filename 215
search_case_sensitive 215
server_launch_enabled 215
server_launch_string 216
server_launch_timeout 216
server_response_wait_timeout 216
SGROUP 190
SHARE_ACTION_POINT 190
share_action_point 216
shared_data_filters 221
show_startup_parameters 226
show_sys_thread_id 227
signal_handling_mode 216
single_click_dive_enabled 227
source_pane_tab_width 217
SOURCE_SEARCH_MAPPINGS 190
SOURCE_SEARCH_PATH 190
spell_correction 217
stack_trace_qualification_level 218
STOP_ALL 31, 191
stop_all 218
stop_relates_on_proc_error 218
suffix 218
TAB_WIDTH 71, 191
THREADS 191
ttf 219
ui_font 227
ui_font_family 227
ui_font_size 227
user_threads 219
using_color 227
using_text_color 227
using_title_color 228
VERBOSE 191
version 219, 228
viewing 97
visualizer_launch_enabled 219
visualizer_launch_string 219
visualizer_max_rank 219
warn_step_throw 220
WGROUP 192
wrap_on_search 220
closes shared libraries 135
clusterid property 145
c0, see dcont command
code snippets 221
code, displaying 70
color
foreground 244
comand-line options
–ccq 242
comline_patch_area_base_variable 199
comline_patch_area_length_variable 199
command arguments 185
command focus 52
command input, blocking 110
command line arguments 95
command output 18
command prompt 189
command summary 3
command verb
actionpoint command 135
command, Tools > Dynamic Libraries 42
COMMAND_EDITING variable 186
command_editing variable 199
command-line options
–aix use_fast_trap 242
–background 242
–bg 242
–ccq 242
–compiler_vars 242
–control_c_quick_shutdown 242
–dbfork 243
–debug_file 243
–demangler 243
–display 244
–dll_ignore_prefix 244
–dll_stop_suffix 244
–dpvm 244
–dump_core 244
–e 244
–ent 244
–env 244
–f9x_demangler 245
–fg 245
–foreground 244
–global_types 245

C

EXECUTABLE_PATH 25, 71, 187
EXECUTABLE_SEARCH_PATH 188
fixed_font 222
fixed_font_family 223
fixed_font_size 223
follow_clone 208
font 223
force_default_cplus_demangler 209
force_default_f9x_demangler 209
force_window_position 223
frame_offset_x 223
frame_offset_y 223
group 210
geometry_call_tree 224
group 210
geometry_cli 224
group 210
group 210
geometry_globals 224
group 210
geometry_help 224
geometry_memory_stats 224
geometry_message_queue 225
group 210
group 210
geometry_message_queue_graph 225
group 210
group 210
geometry_modules 225
group 210
group 210
geometry_ptset 225
group 210
group 210
geometry_pvm 225
group 210
geometry_root 225
group 210
geometry_thread_objects 225
group 210
geometry_variable_stats 226
group 210
group 210
global_typenames 209
group 210
group 210
gnu_debuglink 209, 210, 269
group 210
group 210
gnu_debuglink_checksum_flag 269
group 210
group 210
gnu_debuglink_global_directory 210, 269
group 210
group 210
gnu_debuglink_search_path 210
GROUP 188
GROUPS 24, 73, 188
hia_allow_ibm_poe 221
group 210
group 210
ignore_control_c 210
group 210
ignore_snippets 221
group 210
image_load_callbacks 210
in_setup 210
group 210
group 210
kcc_classes 211
group 210
keep_expressions 226
group 210
keep_search_dialog 226
group 210
kernel_launch_string 211
group 210
kill_callbacks 211
group 210
leak_check_interior_pointers 221
group 210
leak_max_cache 221
leak_max_chunk 221
library_cache_directory 212
LINES_PER_SCREEN 188
local_interface 212
local_server 212
local_server_launch_string 212
-gnu_debuglink 245
-gnu_debuglink_checksum 245
-ipv6_support 245
-kcc_classes 245
-lb 246
-message_queue 246, 247
-mqd 247
-ncq 243
-nlb 246
-no_compiler_vars 242
-no_control_c_quick_shutdown 243
-no_dbfork 243
-no_ent 244
-no_global_types 245
-no_gnu_debuglink 245
-no_gnu_debuglink_checksum 245
-no_ipv6_support 245
-no_kcc_classes 246
-no_message_queue 247
-no_mqd 247
-no_nptl_threads 247
-no_parallel 247
-no_team 250
-no_teamplus 250
-no_user_threads 250
-nptl_threads 247
-parallel 247
-patch_area_base 247
-patch_area_length 248
-pid 248
-r 248
-remote 248
-s 248
-search_path 248
-serial 248
-shm 249
-signal_handling_mode 248
-stder 249
-stderr_append 250
-stderr_is_stdout 250
-stdin 250
-stdout 250
-stdout_append 250
-team 250
-teamplus 250
-tvhome 250
-user_threads 250
-verbosity 250
-xterm_name 250
commands
responding to 149
totalview 241
tvdsrv, syntax and use 253
user-defined 16
commands verb
actionpoint command 131
expr command 137
group command 142
process command 145
thread command 165
type command 167
compile_expressions variable 200
compiler property 170
compiler_vars command-line option 242
compiler_vars variable 200
compilers, KCC 211
compiling
debugging symbols 263
-g compiler command-line option 263
on Bluegene 266
on HP IA-64 264
on HP Tru64 UNIX 264
on IBM Power Linux 266
on IRIX 267
on Itanium 267
on Mac OS X 264
on SunOS 268
on x86 266
on x86-64 267
options 263
conditional watchpoints 111
connection directory 257
console output for tvdsrv 254
console output redirection 243
cont, see dcont command
continue_sig property 165
continuing execution 56
control group variable 186
control group, stopping 218
control list element 188
control_c_quick_shutdown command-line option 242
control_c_quick_shutdown variable 201
copyright_string variable 201
core dumped for TotalView 244
do not load this core files, loading 24
count property 142
core, see dcont command
create subcommand 169
creating barrier breakpoints 28
creating commands 16
creating groups 57, 59
creating new process objects 74
creating threads 56
creating type transformations 229
Ctrl+C, ignoring 210
Ctrl+D to exit CLI 120, 122
current data size limit 278
current frame marker register
Intel IA-64 303
current list location 45
current_cplus_demangler variable 201
current_fortran_demangler variable 201
current_list_location 253
current_thread 165
core, see dcont command
deactivating action points 41
deadlocks at barriers 29
debug gnu_debuglink copy 268
debug_file_command-line option 243
debugger server 216, 253
d debugging remote systems 34
d debugging session, ending 120
dec2hex command 134
default aliases 16
default arguments 95, 185
modify 96
default focus 52
default preferences, setting 97
default value of variables, restoring 106

d, see ddown command
dactions command 19
dassign command 22
data format, presentation styles 202
data size 76
data size limit in C shell 277
data_format_double variable 202
data_format_ext variable 203
data_format_int16 variable 203
data_format_int32 variable 203
data_format_int64 variable 203
data_format_int8 variable 203
data_format_single variable 204
data_format_stringlen variable 204
datatype cast expr 234
datatype incompatibilities 22
dattach command 24
dbar command 27
dbfork command-line option 243
dbfork library 271
linking with 271
syntax 243
dbfork variable 204
dbreak command 31
setting expression in 31
thread level breakpoint 31
dcache command 34
dcheck command 35
preserving IDs 35
process 35
reattaching to parallel 35
scope 35
dcont command 38
delete command 39
ddetach command 40
ddisable command 41
ddll_read_all_symbols variable 205
ddlopen command 42
ddown command 45
d, see ddelete command
dec2hex command 134
debug gnu_debuglink copy 268
debug_file_command-line option 243, 254
debugger server 216, 253
debugging remote systems 34
debugging session, ending 120
dec2hex command 134
default aliases 16
default arguments 95, 185
modify 96
default focus 52
default preferences, setting 97
default value of variables, restoring 106

current_fortran_demangler variable 201

d, see ddown command
dactions command 19
dassign command 22
data format, presentation styles 202
data size 76
data size limit in C shell 277
data_format_double variable 202
data_format_ext variable 203
data_format_int16 variable 203
data_format_int32 variable 203
data_format_int64 variable 203
data_format_int8 variable 203
data_format_single variable 204
data_format_stringlen variable 204
datatype cast expr 234
datatype incompatibilities 22
dattach command 24
dbar command 27
dbfork command-line option 243
dbfork library 271
linking with 271
syntax 243
dbfork variable 204
dbreak command 31
setting expression in 31
thread level breakpoint 31
dcache command 34
dcheck command 35
preserving IDs 35
process 35
reattaching to parallel 35
scope 35
dcont command 38
delete command 39
ddetach command 40
ddisable command 41
ddll_read_all_symbols variable 205
ddlopen command 42
ddown command 45
d, see ddelete command
dec2hex command 134
debug gnu_debuglink copy 268
debug_file_command-line option 243, 254
debugger server 216, 253
debugging remote systems 34
debugging session, ending 120
dec2hex command 134
default aliases 16
default arguments 95, 185
modify 96
examine... examining memory 47, 48  
using data on HP Tru64 270  
extection subcodes 84  
extection, warning when thrown 220  
extecutable property 145  
EXECUTABLE_PATH variable 25, 71, 187  
EXECUTABLE_SEARCH_PATH variable 188  
extecuting as one instruction 80  
extecuting as one statement 78  
extecuting assembler instructions 103  
extecuting source lines 100  
extecution  
continuing 56  
displaying location 117  
halting 62  
resuming 38  
exec() 271  
calling 243  
catching 204  
ext_command 120  
expr . expr 233  
expr -> expr 233  
expr command 84, 137  
extression property 132, 137  
extression system  
AIX 284  
Alpha 284  
IRIX 285  
extression values, printing 84  
extpressions in breakpoint 31  
extpressions, compiling 200  
extpressions, type transformation 233  
extensions for file names 218  
F  
f, see dfocus command  
f9x_demangler command-line option 245  
fast_trap, setting 192  
fatal errors 277  
fg command-line option 245  
figures  
Vector Transformation 230  
File > Preferences command 237  
file name extensions 218  
files  
initialization 232  
libdbfork.h 272  
filters, sharing memory filters 221  
fixed_font variable 222  
fixed_font_family variable 223  
fixed_font_size variable 223  
floating point data format 202  
double-precision 202  
extended floating point 203  
PA-RISC 296  
single-precision 204  
floating point status register  
Intel IA-64 304  
flush, see dlflush command  
flushing cache 34  
flushing suspended evaluations 49  
focus  
see also dfocus command  
default 52  
defining 189  
temporarily changing 52  
focus_groups command 139  
focus_processes command 140  
focus_threads command 141  
focus_threads_property 137  
follow_clone variable 208  
font variable 223  
fonts 222  
fixed 222, 223  
ui 223, 227  
iu font family 227  
iu font size 227  
force_default_cplus_demangler variable 209  
force_default_f9x_demangler variable 209  
force_window_position variable 223  
foreground command-line option 244  
for() 271  
calling 243  
catching 204  
Fortran demangler 201  
frame_offset_x variable 223  
frame_offset_y variable 223  
functions  
evaluating 84  
setting breakpoints at 32  
G  
g, see dgo command  
genral registers  
Intel IA-64 301  
geometry_cal... 224  
geometry_cli variable 224  
geometryGlobals variable 224  
geometry_help variable 224  
geometry_memory_stats variable 224  
geometry_message_queue variable 225  
geometry_message_queue_graph variable 225  
geometry_modules variable 225  
geometry_process variable 225  
geometry_ptset variable 225  
geometry_pvm variable 225  
geometry_root variable 225  
geometry_thread_objects variable 225  
geometry_variable variable 226  
geometry_variable_stats variable 226  
get subcommand 150  
get verb  
actionpoint command 131, 135  
expr command 137  
group command 142  
process command 145  
thread command 165  
type command 167  
Global Arrays 54  
setting language for display 54  
global_typenames variable 209  
global_types, command-line option 245  
GNU C++ STL instantiation 230  
gnu_debuglink command-line option 245, 269  
gnu_debuglink files 268  
gnu_debuglink variable 209, 269  
gnu_debuglink_checksum command-line option 245, 269  
gnu_debuglink_checksum variable 210  
gnu_debuglink_checksum_flag variable 269  
gnu_debuglink_global_directory variable 210  
gnu_debuglink_global_directory command-line option 269  
gnu_debuglink_global_directory command-line option 269  
gnu_debuglink_search_path variable 210  
go, see dgo command  
goal breakpoint 101  
gr, see dggroups command  
grup command 142  
grup ID 192  
grup members, stopping flag 191  
grup of interest 101  
GROUP variable 188  
grup width stepping behavior 100  
grups  
accessing properties 142  
adding 57  
adding members 58  
creating 57, 59  
deleting 57, 58  
intersecting 57, 58  
listing 57, 59  
naming 58  
placing processes in 25  
removing 59  
removing members 57  
returning list of 139
setting properties 142
GROUPS variable 24, 73, 188
groups, see dgroups command

H
h, see dhalt command
halt, see dhalt command
halting execution 62
handling signals 248
handling user-level (M:N) thread packages 219
heap size 76
heap_size property 145
held property 145, 165
help command 121
help window position 224
hex2dec command 144
hexadecimal conversion 134
hia_allow_ibm_poe 221
hold, see dhold command
holding processes 67
holding threads 28, 67
host ports 254
hostname
  expansion 257
  for tvdsvr 254
  property 146
  replacement 258
HP-UX
  /proc file system 276
  linking to dbfork library 271
  swap space 277
hp, see dhold command
HP-UX
  architecture 293
  shared libraries 281
  swap space 277
ht, see dhold command
htp, see dhold command

I
I/O redirection 95
id property 132, 137, 142, 146, 165, 167, 170
ignore_control_c variable 210
ignore_snippets false 221
ignoring libraries by prefix 244
image browser window position 224
image file, stripped copy 268
Image information 76
image_id property 167
image_ids property 146
image_load_callbacks variable 210
in_setup variable 210
inet interface name 212, 246
info state 191
information on a name 114
initialization file 127, 232
initially_suspended_process property 137
inserting working threads 119
instructions, stepping 103
integer (64-bit) data format 203
integer data format
  16-bit 203
  32-bit 203
  8-bit 203
Intel IA-64
  architecture 301
  current frame marker register 303
  floating point status register 304
general registers 301
Intel IA-64
  floating point registers 304
  general registers 301
  previous function state register 303
  processor status register 302
  register stack configuration register 303
Intel-x86
  architecture 287, 305
  floating-point registers 288, 306
  FPCR register 289, 306
  using 290, 307
  FPSR register 290, 291, 307, 308
  general registers 288, 305
  interface name for server 212, 246
  interiopinters, checking 221
  intersecting groups 57, 58
ipv6_support command-line option 245
ipv6_support option 245
IRIX
  /proc file system 276
  linking to dbfork library 273
  swap space 279
J
job_t::launch 276
K
k, see dkill command
  -kcc_classes command-line option 245
  kcc_classes command-line option 245
  kcc_classes variable 211
  keep_expressions variable 226
  keep_search_dialog variable 226
  kernel_launch_string variable 211
  keys, remapping 284
  keysym 284
  kill, see dkill command
  kill_callbacks variable 211
  killing attached processes 201
  killing processes 68
L
l, see dlst command
language property 132, 167, 170
lappend, see dlappend command
launch string
  for editor 208
  for server (Sun only) 212
  for Visualizer 219
Launch Strings page 219
launching
  local server 212
  processes 95
  single process server launch string 216
  tvdsvr 253
  Visualizer 219
lb command-line option 246
LD_BIND_NOW environment variable 281
LD_LIBRARY_PATH 271, 273, 274
  -ldbfork linker option 271, 274
  -ldbfork option 272, 273
  -ldbfork 64 linker option 271
  -ldbfork_64 option 272, 273
  -ldbfork_n32 linker option 273
  -ldbfork_n64 linker option 273
  leak_check_interior_pointers 221
  leak_check_interior_pointers variable 221
  leak_max_cache 221
  leak_max_chunk 221
  length property 132, 167
  levels for dwhere 189
  levels, moving down 45
  libdbfork.a 271, 273, 274
  libdbfork.a_n64.a 273
  libdbfork.h file 272
  libdbfork_64.a 273
  libdbfork_n32.a 273
  libraries
    dbfork 243
    ignoring by prefix 244
    loading by suffix 206
    loading symbols from 208
    not loading based on prefix 205
    shared 280
  library cache data 212
  library cache, flushing 34
  library_cache_directory variable 212
  line property 132
  line_number 154
  LINES_PER_SCREEN variable 188
  linking to dbfork library 271
  AIX 272
  C++ and dbfork 272
  HP Tru64 UNIX 271
  IRIX 273
  SunOS 5 274
  Linux swap space 279
list location 45
list, see dl list command
list_element_count_addressing_callback 170
list_element_data_addressing_callback 170
list_element_next_addressing_callback 170
list_element_prev_addressing_callback 170
list_end_value property 170
list_first_element_addressing_callback 171
list_head_addressing_callback 171
listing groups 57, 59
listing lines 189
load and loadbind 282
load, see dl load command
load, see dl load command
load, see dl load command
load, see dl load command
load, see dl load command
load, see dl load command
load, see dl load command
loading
action point information 20
action points 19, 246
libraries 205, 206
programs 74
shared libraries 135
symbols from shared libraries 208
tvd files 152
local_interface variable 212
local_server variable 212
local_server_launch_string_variable 212
lockstep list element 188
logical model 230
lookup subcommand 150
lookup_keys subcommand 151
lower_bounds_callback 171

M
machine instructions, stepping 103
manager property 165
manager threads, running 100
managing shared libraries 135
mangler, overriding 243, 245
mappings, search path 190
MAX_LIST variable 70, 189
maxdsize 64, 278
maximum data segment size 278
Maximum permissible rank field 219
mem_detect_leaks memory sub-option 178
mem_guard_blocks memory sub-option 179
mem_hoard_freed_memory memory sub-option 179
mem_paint_all memory sub-option 179
mem_paint_on_alloc memory sub-option 179
mem_paint_on_dealloc memory sub-option 179
member_type property 143
member_type_values property 143
members argument, build_struct_transform 233
members property 143
memory
data size 76
heap 76
stack 76
text size 76
memory statistics window position 224
memory use 76
message queue graph window position 225
message queue window position 225
message verbosity variable 191
message_queue command-line option 246, 247
message_queue_variable 213
MIPS
architecture 308
delay slot instructions 312
FCSR register 311
using 312
floating-point registers 310
general registers 309
SR register 310
mkswap command 279
modules window position 225
more processing 84
more prompt 121, 188
mounting /proc file system 276
MPI message queues 213
MPI startup 319
mqd_command-line-option 247
multiprocess programs, attaching to processes 25

N
N upcast expr 234
n, see dnext command
name argument, build_struct_transform 233
name property 167, 171
name, information about 114
namespaces 97
TV:: 97
TV::GUI: 97
using wildcards 97
Namespaces, Symbol 161
naming the host 254
cqc_command-line-option 243
nested subroutines, stepping out of 82
new groups 59
newval variable in watchpoints 111
next, see dnext command
nexti, see dnexti command
ni, see dnexti command
nil, see dnexti command
niw, see dnexti command
nl, see dnext command
nlb command-line option 246
no_ask_on_dlopen command-line option 283
no_compiler_vars command-line option 242
no_control_c_quick_shutdown command-line option 243
no_dbfork command-line option 243
no_dynamic command-line option 280
no_ent command-line option 244
no_global_types command-line option 245
no_gnu_debuglink command-line option 245
no_gnu_debuglink_checksum_command-line-option 245
no_ipv6_support command-line option 245
no_kcc_classes command-line-option 246
no_message_queue command-line option 247
no_mqd command-line option 247
no_nptl_threads command-line option 247
no_parallel_command-line-option 247
no_team command-line option 250
no_teamplus_command-line option 250
no_user_threads command-line-option 250
nodeid property 146
nodes_allowed command-line-option 255
nptl_threads_command-line-option 247
nptl_threads_variable 213
nw, see dnext command

O
OBJECT_SEARCH_MAPPINGS variable 187, 189
OBJECT_SEARCH_PATH variable 189
oldval variable in watchpoints 111
Open (or raise) process window at breakpoint checkbox 226
Open process window on error signal check box 226
opening shared libraries 42
option 255
options
  –aix_use_fast_trap 242
tvdsr
    –callback 253
    –serial 253
    –server 253
    –set_pw 254
    –user_threads 250
ou, see dout command
oul, see dout command
out, see dout command
ouw, see dout command

P
p, see dprint command
p/T set information 88
panes, width 217
parallel command-line option 247
parallel processes, attaching to 24
parallel runtime libraries 213
parallel variable 213
Parallel Virtual Machine 206, 214, 244
parallel_attach variable 213
parallel_configs variable 73, 319, 320
parallel_stop variable 214
parallel_support.tvd file 319
PA-RISC
  floating-point format 296
  floating-point registers 295
  general registers 293
  process status word 294
passwords 256
  checking 256
  generated by tvdsr 254
patch space 199
patch_area_base command-line option 247
patch_area_length command-line option 248
PATH environment variable
  for tvdsr 253
pc property 165
picking up threads 208
pid command-line option 248
Plant in share group checkbox 190, 216
platform variable 214
pop_at_breakpoint variable 226
pop_on_error variable 226
popping Process Window on error variable 226
port 4142 255
port command-line option 255
port number 255
  for tvdsr 254
  replacement 258
  searching 255
ports on host 254
post_scope 151
post_symbol 151
PowerPC
  architecture 297
  floating-point registers 299
  FPSCR register 299
  using the 300
  FPSCR register, using 300
general registers 297
MSR register 298
pre_scope 151
pre_sym 151
preferences, setting defaults for 97
preserving IDs in checkpoint 35
previous function state register
  Intel IA-64 303
print, see dprint command
printing expression values 84
printing information about current
state 99
printing registers 86
printing slices 85
printing variable values 84
proc file system problems 276
Procedure Linkage Table (PLT) 281
Process > Startup command 56
process barrier breakpoint, see barrier
breakpoint
process command 145
process groups, see groups
process information, saving 36
process list element 188
process objects, creating new 74
process statistics 76
process width stepping behavior 101
process window position 225
process/thread sets, changing 52
process_load_callbacks variable 214
process_set_checkpoint options 36
processes
  attaching to 24, 73
  automatically acquiring 204
  automatically attaching to 213
  current status 99
  destroyed when exiting CLI 120, 122
  detaching from 40
  holding 67
  killing 68
  properties 145
  releasing 105
  releasing control 40
  restarting 91, 95
  returning list of 140
  starting 91, 95
  terminating 68
processor status register
  Intel IA-64 302
program control groups, placing pro-
cesses in 25
program stepping 100
program variable, changing value 22,
36, 59, 67, 82, 94, 105, 107
programs, loading 74
PROMPT variable 189
prompting when screen is full 84
properties
  address 131
  clusterid 145
  continuation_sig 165
  count 142
  done 137
  dpid 165
  duid 145, 165
  enabled 132
  enum_values 167
  executable 145
  expression 132, 137
  focus_threads 137
  heap_size 145
  held 145, 165
  hostname 146
id 132, 137, 142, 146, 165, 167
image_id 167
image_ids 146
initially_suspended_process 137
language 132, 167
length 132, 167
line 132
manager 165
member_type 143
member_type_values 143
members 143
name 167
nodeid 146
pc 165
prototype 167
rank 167
result 138
satisfaction_group 132
share 132
sp 165
stack_size 146
stack_vm_size 147
state 147, 165
state_values 147, 166
status 138
stop_when_done 132
stop_when_hit 132
struct_fields 167
symbol 154
syspid 147
systid 166
text_size 147
threadcount 147
threads 147
type 143
type_values 132, 143
vm_size 147
properties verb
  actionpoint command 131, 135
  expr command 137
  group command 142
  process command 145
  thread command 165
  type command 167
prototype property 167
PTSET variable 189
ptsets, see dptsets
PVM 255
pvm command-line option 248, 255
pvm variable 214, 215
pvm window position 225
pxdb command 281
pxdb64 command 281
Q
qnofullpath command-line option 265
qualifying symbol names 70
quit command 122
quotation marks 22
R
r command-line option 248
r, see drun command
raising errors 136
rank property 167
ranks, attaching to 24
raw memory display 47
read_delayed subcommand 153
read_symbols command 148
reading action points file 19
reading symbols 148, 195, 205, 206
rebind subcommand 153
reenabling action points 46
register stack configuration register
Intel-IA-64 303
registers
  Alpha FPCR 293
  floating-point
    Alpha 292
    Intel-x86 288, 306
    MIPS 310
  PowerPC 299
  SPARC 314
  general
    Alpha 291
    Intel-x86 288, 305
    MIPS 309
    PowerPC 297
    SPARC 313
  Intel-x86 FPCR 289, 306
  using the 290, 307
Intel-x86 FPCR 290, 291, 307, 308
MIPS FCSR 311
  using the 312
MIPS SR 310
Power FPCR 299
Power MSR 298
PowerPC FPCR 299
  using 300
PowerPC FPCRs, using 300
PowerPC MSR 298
printing 86
SPARC FPCR 315
SPARC FPCRs, using 315
SPARC PSR 314
registers, using in evaluations 32
regular expressions within name argument 233
release 186
releasing control 40
releasing processes and threads 27, 105
remapping keys 284
remote command-line option 248
remote debugging, tvdsvr command syntax 253
remote systems, debugging 34
removing
  aliases 127
  group member 57
  groups 59
  variables 106
worker threads 119
remsh command 257
replacement characters 257
replacing tabs with spaces 191
rerun, see rerun command
resolve_final subcommand 153
resolve_next subcommand 153
respond 149
restart, see drestart command
restart_threshold variable 215
restarting processes 91, 95
restoring variables to default values 106
result property 138
resuming execution 38, 56, 68
returning error information 136
root path 191
  of TotalView 191
Root Window position 225
routines, stepping out of 82
rr, see drrun command
RS/6000, compiling on 265
rsh command, with tvdsvr 216
RTLD_GLOBAL 42
RTLD_LAZY 42, 43
RTLD_LOCAL 42
RTLD_NOW 42, 43
run, see drun command
running to an address 107
S
s command-line option 248
s, see dstep command
saturation set 28, 186, 196
saturation_group property 132
save_window_pipe_or_filename variable 215
saved position
  Call Tree Window 224
  CLI Window 224
  Help Window 224
  Image Browser Window 224
  Memory Statistics Window 224
  Message Queue Graph Window 225
  Message Queue window 225
  Modules Window 225
  Process Window 225
  PVM Window 225
  Root Window 225
  Thread Objects Window 225
  Variable Window 226
saving action point information 20
saving action points 19
saving process information 36
scope 154
scope command 150
scope of action point 190
screen size 188
search dialog, remaining displayed 226
search path 187
  mappings 190
  setting 187, 188, 189, 190
search_case_sensitive variable 215
search_path command-line option 248
search_port command-line option 255
searching
  case sensitive 215
  wrapping 220
serial command-line option 248, 253, 255
serial line connection 255
server command-line option 253, 255
server launch command 257
server_launch_enabled variable 215
server_launch_string variable 216
server_launch_timeout variable 216
server_response_wait_timeout variable 216
servers, number of 258
set verb
  actionpoint command 131
  group command 142
unl, see duntil command
unset, see dusset command
until, see duntil command
unw, see duntil command
unwinding the stack 49
up, see dup command
upper_bounds_callback 172
user_threads command-line option 250
user_threads variable 219
user-defined commands 16
user-level (M:N) thread packages 219
using quotation marks 22
using type transformations 236
using_color_variable 227
using_text_color_variable 227
using_title_color_variable 228
validate_callback 172
value for newly created action points 190
values, printing 84
Variable Window position 226
variables
  assigning command output to 18
  casting 86
  changing values 22, 36, 59, 67, 82, 94, 105, 107
  default value for 97
  printing 84
  removing 106
  setting 97
  watched 112
  watching 111
Vector Transformation figure 230
VERBOSE variable 191
verbosity command-line option 250, 256
verbosity setting replacement character 259
version variable 219, 228
version, TotalView 191
vfork()
  calling 243
  catching 204
viewing CLI variables 97
visualizer_launch_enabled variable 219
visualizer_launch_string variable 219
visualizer_max_rank_variable 219
vm_size property 147
vm_size size 77
X
xterm_name command-line option 250

V
validate_callback 172
value for newly created action points 190
values, printing 84
Variable Window position 226
variables
  assigning command output to 18
  casting 86
  changing values 22, 36, 59, 67, 82, 94, 105, 107
  default value for 97
  printing 84
  removing 106
  setting 97
  watched 112
  watching 111
Vector Transformation figure 230
VERBOSE variable 191
verbosity command-line option 250, 256
verbosity setting replacement character 259
version variable 219, 228
version, TotalView 191
vfork()
  calling 243
  catching 204
viewing CLI variables 97
visualizer_launch_enabled variable 219
visualizer_launch_string variable 219
visualizer_max_rank_variable 219
vm_size property 147
vm_size size 77

W
w, see dwhere command
wa, see dwatch command
wait, see dwait command
walk subcommand 151
warn_step_throw variable 220
warning state 191
watch, see dwatch command
watchpoints 111
  $newval 111
  $oldval 111
  conditional 111
  information not saved 20
  length of 112
  supported systems 111
–WG,–cmpo=i option 265
WGROUP variable 192
wh, see dwhat command
what, see dwhat command
When barrier done, stop value 186
When barrier hit, stop value 185
where, see dwhere command
why type transformations 229
window position, forcing 223
worker group list variable 192
worker threads 192
  inserting 119
  removing 119
worker, see dwworker command
workers list element 188
working_directory command-line option 256
wot, see dwworker command
wrap_on_search variable 220

X
xterm_name command-line option 250