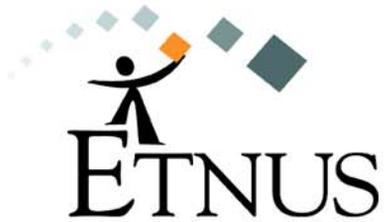


# TOTALVIEW

## REFERENCE GUIDE



APRIL 2003

VERSION 6.2

Copyright © 1998–1999, 2003 by Etnus Inc. All rights reserved.

Copyright © 1999–2003 by Etnus LLC. All rights reserved.

Copyright © 1996–1998 by Dolphin Interconnect Solutions, Inc.

Copyright © 1993–1996 by BBN Systems and Technologies, a division of BBN Corporation.

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

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

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

TotalView and Etnus are registered trademarks of Etnus LLC.

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



# Book Overview

## part I – CLI Commands

1	CLI Command Summary.....	3
2	CLI Commands.....	15
3	CLI Namespace Commands.....	129
4	TotalView Variables.....	161
5	Default Arena Widths.....	201

## part II – Running TotalView

6	TotalView Command Syntax.....	207
7	TotalView Debugger Server (tvdsvr) Command Syntax.....	215

## part III – Platforms and Operating Systems

8	Compilers and Platforms.....	225
9	Operating Systems.....	237
10	Architectures.....	251





# Contents

## About This Book

How To Use This Book .....	xi
Conventions .....	xii
TotalView Documentation .....	xiii
Contacting Us .....	xiv

<b>1 CLI Command Summary .....</b>	<b>3</b>
------------------------------------	----------

## **2 CLI Commands**

Command Overview .....	15
alias .....	19
capture .....	21
dactions .....	23
dassign .....	26
dattach .....	28
dbarrier .....	31
dbreak .....	36
dcache .....	39
dcheckpoint .....	40
dcont .....	43
ddelete .....	45
ddetach .....	46
ddisable .....	47
ddown .....	48
denable .....	50
dflush .....	51
dfocus .....	54
dgo .....	57
dgroups .....	58

dhalt.....	64
dhold.....	65
dkill .....	66
dlappend.....	67
dlist .....	68
dload.....	71
dmstat.....	73
dnext .....	75
dnexti .....	77
dout .....	79
dprint .....	81
dptsets.....	86
drerun .....	89
drestart .....	91
drun.....	93
dset .....	96
dstatus .....	98
dstep.....	100
dstepi.....	103
dunhold.....	105
dunset.....	106
duntil.....	107
dup.....	110
dwait .....	112
dwatch .....	113
dwhat .....	117
dwhere .....	121
dworker .....	123
exit .....	124
help.....	125
quit.....	126
stty .....	127
unalias.....	128

### 3 CLI Namespace Commands

Command Overview .....	129
actionpoint .....	131
dec2hex .....	133
errorCodes .....	134
expr.....	136

focus_groups .....	138
focus_processes .....	139
focus_threads .....	140
group .....	141
hex2dec .....	143
image .....	144
process .....	147
respond .....	150
scope .....	151
source_process_startup .....	152
symbol .....	153
thread .....	154
type .....	156
type_transformation .....	159
<b>4 TotalView Variables</b>	
Top-Level (::) Namespace .....	161
TV:: Namespace .....	168
TV::GUI:: Namespace .....	193
<b>5 Default Arena Widths</b> .....	201
<b>6 TotalView Command Syntax</b>	
Syntax .....	207
Options .....	208
<b>7 TotalView Debugger Server (tvdsvr) Command Syntax</b>	
The tvdsvr Command and Its Options .....	215
Replacement Characters .....	219
<b>8 Compilers and Platforms</b>	
Compiling with Debugging Symbols .....	225
HP Alpha Running Linux .....	226
HP Tru64 UNIX .....	226
HP-UX .....	227
IBM AIX on RS/6000 Systems .....	227
Linux Running on an x86 Platform .....	228
SGI IRIX-MIPS Systems .....	229
SunOS 5 on SPARC .....	230
Using Exception Data on HP Tru64 UNIX .....	231
Linking with the dbfork Library .....	231

Linking with dbfork and HP Tru64 UNIX .....	232
Linking with HP-UX .....	232
dbfork on IBM AIX on RS/6000 Systems .....	233
<i>Linking C++ Programs with dbfork</i> .....	233
Linux .....	234
SGI IRIX6-MIPS .....	234
SunOS 5 SPARC .....	235

## 9 Operating Systems

Supported Operating Systems .....	237
Mounting the /proc File System .....	238
Mounting /proc HP Tru64 UNIX and SunOS 5 .....	239
Mounting proc SGI IRIX .....	239
Swap Space .....	239
Swap Space on HP Tru64 UNIX .....	240
Swap Space on HP HP-UX .....	240
<i>Maximum Data Size</i> .....	240
Swap Space on IBM AIX .....	241
Swap Space on Linux .....	242
Swap Space on SGI IRIX .....	242
Swap Space on SunOS 5 .....	242
Shared Libraries .....	243
Changing Linkage Table Entries and LD_BIND_NOW .....	244
Using Shared Libraries on HP-UX .....	245
Debugging Dynamically Loaded Libraries .....	245
Known Limitations .....	248
Remapping Keys .....	248
Expression System .....	248
Expression System on HP Alpha Tru64 UNIX .....	248
Expression System on IBM AIX .....	249
Expression System on SGI IRIX .....	249

## 10 Architectures

HP Alpha .....	251
Alpha General Registers .....	252
Alpha Floating-Point Registers .....	253
Alpha FPCR Register .....	253
HP PA-RISC .....	254
PA-RISC General Registers .....	254
PA-RISC Process Status Word .....	255

PA-RISC Floating-Point Registers .....	256
PA-RISC Floating-Point Format .....	258
IBM Power .....	258
Power General Registers .....	258
Power MSR Register .....	259
Power Floating-Point Registers .....	260
Power FPSCR Register .....	260
<i>Using the Power FPSCR Register</i> .....	262
Intel-x86 .....	262
Intel-x86 General Registers .....	263
Intel-x86 Floating-Point Registers .....	264
Intel-x86 FPCR Register .....	265
<i>Using the Intel-x86 FPCR Register</i> .....	265
Intel-x86 FPSR Register .....	266
Intel-x86 MXSCR Register .....	266
SGI MIPS .....	267
MIPS General Registers .....	268
MIPS SR Register .....	269
MIPS Floating-Point Registers .....	270
MIPS FCSR Register .....	271
<i>Using the MIPS FCSR Register</i> .....	272
MIPS Delay Slot Instructions .....	272
Sun SPARC .....	273
SPARC General Registers .....	273
SPARC PSR Register .....	274
SPARC Floating-Point Registers .....	275
SPARC FPSR Register .....	275
<i>Using the SPARC FPSR Register</i> .....	276
<b>Index</b> .....	279





# About This Book

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

## How To Use This Book



The information in this book is in three parts.

- **CLI Commands**

Here you will find a description of all of the CLI's commands, the variables that you can set using the CLI, and other CLI-related information.

- **Running TotalView**

TotalView and the TotalView Debugger Server can accept a great many command-line options. These options are described here.

- **Platforms and Operating Systems**

While the way in which you use TotalView is the same from system to system and from environment to environment, these systems and environments place some constraints on what you must do and require that you compile programs differently on the various UNIX platforms. These differences are described here.

## Conventions

The following table describes the conventions used in this book:

TABLE I: Book Conventions

Convention	Meaning
[ ]	Brackets are used when describing parts of a command that are optional.
<i>arguments</i>	Within a command description, text in italic represent information you type. Elsewhere, italic is used for emphasis. You won't have any problems distinguishing between the uses.
<b>Dark text</b>	Within a command description, <b>dark text</b> represent key words 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.
Example text	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. Bolding this kind of text is done only when it's important. You'll usually be able to differentiate what you type from what the system prints.
	This graphic symbol indicates that a feature is only available in the GUI. If you see it on the first line of a section, all the information in the section is just for GUI users. When it is next to a paragraph, it tells you that just the sentence or two being discussed applies to the GUI.
CLI EQUIVALENT:	The primary emphasis of this book is on the GUI. It shows the windows and dialog boxes that you use. This symbol tells you how to do the same thing using the CLI.

## TotalView Documentation

The following table describes other TotalView documentation:

TABLE II: TotalView Documentation

Title	Contents	Online			
		Help	HTML	PDF	Print
TotalView Users Guide	Describes how to use the TotalView GUI and the CLI	✓	✓	✓	
TotalView QuickView	Presents what you need to know to get started using TotalView				✓
Creating Type Transformations	Tells how to create Tcl CLI macros that change the way structures and STL containers appear		✓	✓	
TotalView Graphic User Interface Command Reference	Defines all TotalView GUI commands	✓	✓	✓	
TotalView Installation Guide	Contains the procedures to install TotalView and the FLEX/license manager	✓	✓	✓	
TotalView New Features	Tells you about new features added to TotalView	✓	✓	✓	
TotalView Release Notes	Lists known bugs and other information related to the current release	✓	✓	✓	
IBM Considerations	Briefly describes things you should know when run on IBM RS6000 machines	✓	✓	✓	
Linux Considerations	Briefly describes things you should know when running on Linux platforms	✓	✓	✓	
Patching Platforms	Describes how to apply vendor supplied patches to operating systems and compilers	✓	✓	✓	
Platforms and System Requirements	Lists the platforms upon which TotalView runs and the compilers it supports	✓	✓	✓	

## Contacting Us

---

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 E-Mail address for support issues is **support@etnus.com**

For documentation issues, the address is: **documentation@etnus.com**

Call: 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



# Part I: CLI Commands

This part of the TotalView Reference Guide contains five chapters describing the CLI. While Chapter 4 describes all CLI variables, these variables include those used to set GUI behaviors.

## Chapter 1: CLI Command Summary

There are a great number of CLI commands. This chapter summarizes them for you.

## Chapter 2: CLI Commands

Here you will find a detailed treatment of all CLI commands that are found in the CLI's unqualified (top-level) namespace. These are the commands that you will use day-in and day-out and those that are most often used interactively.

## Chapter 3: CLI Namespace Commands

This chapter 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: TotalView Variables

TotalView places variables in three namespaces: unqualified (top-level), **TV::** and **TV::GUI**. For the most part, you set these variables to alter TotalView behaviors.

## Chapter 5: Default Arena Widths

Many of the commands described in Chapter 2 have a default arena. That is, the scope of their action is across a set of groups, processes, and threads unless you tell the command otherwise. Here you'll find a listing of all these default arenas.



# Chapter 1

## CLI Command Summary

This chapter contains a summary of all TotalView® CLI commands. Type Transformation macros and convenience routines are not listed here.

### 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

```
dattach [ -g gid ] [ -r hname ]  
        [ -ask_attach_parallel | -no_attach_parallel ]  
        [ -c corefile-name ]  
        [ -e ] fname pid-list
```

## dbarrier

Creates a barrier breakpoint at a source location

```
dbarrier source-loc [ -stop_when hit { group | process | none } ]  
        [ -stop_when_done { group | process | none } ]
```

Creates a barrier breakpoint at an address

```
dbarrier -address addr  
        [ -stop_when_hit { group | process | none } ]  
        [ -stop_when_done { group | process | none } ]
```

## dbreak

Creates a breakpoint at a source location

```
dbreak source-loc [ -p | -g | -t ] [ [ -l lang ] -e expr ]
```

Creates a breakpoint at an address

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

## dcache

Clears the remote library cache

```
dcache
```

## dcheckpoint

Creates a checkpoint image of processes (SGI only)

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

**dcont**

Continues execution and waits for execution to stop

**dcont**

**ddelete**

Deletes some action points

**ddelete** *action-point-list*

Deletes all action points

**ddelete** **-a**

**ddetach**

Detaches from the processes

**ddetach**

**ddisable**

Disables some action points

**ddisable** *action-point-list*

Disables all action points

**ddisable** **-a**

**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**

## dflush

Removes the top-most suspended **dprint**

**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**

## 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 ]**

Creates a new thread or process group

**dgroups -new [ thread\_or\_process ] [ -g gid ] [ id-list ]**

Removes members for thread or process groups

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

**dhalt**

Suspends execution of processes

**dhalt**

**dhold**

Holds processes

**dhold -process**

Holds threads

**dhold -thread**

**dkill**

Terminates execution of target processes

**dkill**

**dlappend**

Appends list elements to a TotalView variable

**dlappend** *variable-name value* [ ... ]

**dlist**

Displays code relative to the current list location

**dlist** [ **-n** *num-lines* ]

Displays code relative to a named location

**dlist** *source-loc* [ **-n** *num-lines* ]

Displays code relative to the current execution location

**dlist -e** [ **-n** *num-lines* ]

**dload**

Loads debugging information

**dload** [ **-g** *gid* ] [ **-r** *hname* ] [ **-e** ] *executable*

**dmstat**

Displays memory use information

**dmstat**

**dnext**

Steps source lines, stepping over subroutines

```
dnext [ num-steps ]
```

**dnexti**

Steps machine instructions, stepping over subroutines

```
dnexti [ num-steps ]
```

**dout**

Runs out from current subroutine

```
dout [ frame-count ]
```

**dprint**

Prints the value of a variable or expression

```
dprint variable_or_expression
```

**dptsets**

Shows 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 (SGI only)

```
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 [ -new ] 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 [ num-steps ]
```

**dstepi**

Steps machine instructions, stepping into subfunctions

```
dstepi [ 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 line-number
```

Runs to an address

```
duntil -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 [ num-levels ] [ -a ]
```

**dworker**

Adds or removes a thread from a workers group

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

## errorCodes

Returns a list of all error code tags

**TV::errorCodes**

Returns or raises error information

**TV::errorCodes** *number\_or\_tag* [ **-raise** [ *message* ] ]

## expr

Manipulates values created by **dprint -nowait**

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

## exit

Terminates the debugging session

**exit** [ **-force** ]

## 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*

## image

Gets and sets image properties

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

## process

Gets and sets process properties

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

## quit

Terminates the debugging session

```
quit [ -force ]
```

## respond

Provides responses to commands

```
TV::respond response command
```

## scope

Returns information about a a symbol's scope.

```
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 examine properties

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

## unalias

Removes an alias

```
unalias alias-name
```

Removes all aliases

```
unalias -all
```



## Chapter 2

# CLI Commands

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

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.
- *dggroups*: 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

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.
- *dflush*: Unwinds stack from **dprint** computations.
- *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 status of processes and threads in a P/T set.
- *dstatus*: Shows 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

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.
- *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 are responsible for defining and manipulating 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; it 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*: Reenables an action point that has been disabled.
- *dwatch*: Defines a watchpoint.

## Other Commands

The commands in the 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.
- *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 will substitute 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 just do not enter an *alias-name* argument, the CLI displays the names and definitions of all aliases. If you just 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.

TotalView's 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, here is a procedure that 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. Thus, 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's 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 will apply 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**. Note that you could also create an alias that does the same thing as follows: **alias step2 {s 2}**.

`alias step1 {f p1. dstep}`  
Defines an alias called **step1** that steps the first user thread in process 1. Note that 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:

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

### 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; that is, ``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 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 some sample output:

```
int2_array(1) = -8 (0xffff8)
int2_array(2) = -6 (0xffffa)
int2_array(3) = -4 (0xffffc)
int2_array(4) = -2 (0xffffe)
```

Because **capture** records all of the information sent to it by the commands in the **foreach**, you do not have to use a **dlist** command.

```
exec cat << [ capture help commands ] > cli_help.txt
```

Writes the help text for all TotalView 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 TotalView 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, saves, and reloads action points

### Format:

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 ]
```

### Arguments:

*ap-id-list*

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

If you omit this argument, TotalView displays summary information about all action points in the processes in the focus set. If one ID is entered, TotalView displays full information for it. If more than one ID is entered, TotalView just displays summary information for each.

**-at** *source-loc*

Displays the action points at *source-loc*.

**-enabled**

Only shows enabled action points.

**-disabled**

Only shows 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 will read and write action point information. If you omit this file name, TotalView writes them 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. 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.

**NOTE** The identifier is returned when the action point is created. It is also displayed when the target stops at an action point.

You can include specific action point identifiers as arguments to the command when detailed information is required. 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 that it should write action point information to a file so that either you or TotalView can restore your action points at a later time. The **–load** option tells TotalView that it should immediately read in the saved file. If you use the *filename* argument with either of these options, TotalView either writes to or reads from this file. If you do not use this argument, it uses a file named *programname.TVD.v3breakpoints* where *programname* is the name of your program. This file is written to the same directory as your program.

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**, it writes this information to **foo.TVD.v3breakpoints**.

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

If a file with the default name exists, TotalView can read this information when it starts your program. When TotalView exits, it can create the default. For more information, see **File > Preferences** in TotalView's Help system.

#### Command alias:

You may find the following alias useful:

Alias	Definition	Meaning
<b>ac</b>	<b>{dactions}</b>	Displays all action points

#### Examples:

```
ac -at 81
```

Displays information about the action points on line 81. (Notice that 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
```

`dfocus p1 dactions`

Displays information on all action points defined within process 1.

`dfocus p1 dactions -enabled`

Displays information on all enabled action points within process 1.

## dassign

Changes the value of a scalar variable

*Format:*

**dassign** *target value*

*Arguments:*

*target*

The name of a scalar variable within 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 may be a scalar variable, a dereferenced pointer variable, or an element in an array or structure.

The default focus for **dassign** is *thread*. So, 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, **dassign** 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 may 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 12 of the TOTALVIEW USERS GUIDE.)

Here are some things you should know 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**, **\t**, and the like. 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 expression is evaluated by TotalView's expression system. This system is discussed in Chapter 12 of the TOTALVIEW USERS GUIDE.

*Command alias:*

You may find the following alias useful:

Alias	Definition	Meaning
as	{dassign}	Changes a scalar variable's value

*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 could 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 ]
        [ -c corefile-name ]
        [ -e ] filename pid-list
```

### Arguments:

- g *gid*** Sets the control group for the processes being added to be group *gid*. This group must already exist. (The CLI **GROUPS** variable contains a list of all groups. See **GROUPS** on page 164 for more information.)
- r *hname*** The host on which the process is running. The CLI will launch a TotalView Debugger Server on the host machine if one is not already running there. Consult 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 if TotalView should attach to parallel processes of a parallel job. The default is to automatically attach to processes. For additional information, see **File > Preferences** in TotalView's Help.
- no\_attach\_parallel** Do not attach to any additional parallel processes in a parallel job. For additional information, see **File > Preferences** in TotalView's Help.
- 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*).
- 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 will control. All PIDs must reside on the same system, and they will all be placed into 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.

#### Description:

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

This command returns the TotalView process ID (DPID) as a string. If you specify more than one process in a command, **dattach** 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 allows you to 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** for each.

If you have not already loaded *filename*, the CLI searches for it. The search will include all directories in the **EXECUTABLE\_PATH** CLI variable.

The process identifiers specified in the *pid-list* must refer to existing processes in the run-time environment. TotalView attaches to the processes, regardless of their execution states.

#### Command alias:

You may find the following alias useful:

Alias	Definition	Meaning
<b>at</b>	<b>{dattach}</b>	Brings the process under CLI control

*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 Debugger 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 slaveprog 456
```

Begins by attaching to **mainprog** running on the local host. It then attaches to **slaveprog** running on **otherhost** 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 source-loc [ -stop_when_hit width ]
                  [ -stop_when_done width ]
```

Creates a barrier breakpoint at an address

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

### Arguments:

*source-loc*

The barrier breakpoint location as a line number or as a string 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 will create them for you. For more information, see "Qualifying Symbol Names" in Chapter 12 of the TOTALVIEW USERS GUIDE.

**-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 it should stop when it stops the thread arriving at a barrier.

If you do not use this option, the value of the **BARRIER\_STOP\_ALL** variable indicates what TotalView will stop.

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 barrier is hit.

**process**

Stops the process that hit the barrier.

**none**

Stops the thread that hit the barrier; that is, the thread will be held and all other threads continue running. If you apply this *width* to a process barrier, TotalView will stop 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 that it should 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 containing threads in the satisfaction set when the barrier is satisfied.

You will find information on the "satisfaction set" in this topic's *Description* section.

- none** Stops the satisfaction set. For process barriers, **process** and **none** have the same effect. This is the default if **BARRIER\_STOP\_WHEN\_DONE** is **none**.

#### Description:

The **dbarrier** command sets a process or thread barrier breakpoint that is triggered when execution arrives at a location. This command returns the ID of the newly created breakpoint.

The **dbarrier** command is most often used to synchronize a set of threads. The P/T set defines which threads are affected by the barrier. 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 reaching 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 now continue the process, those threads stopped at the barrier also run along with any other threads that were not participating with the barrier. After they 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 may also stop the thread's process or control group. Neither are 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.

**NOTE** 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 will never be 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 it should stop when a thread arrives at a barrier.

The `-stop_when_done` option controls the set of additional threads that TotalView will stop 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 will 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 `dset`.

The **none** argument for these options indicate that the CLI should not stop additional threads.

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

TotalView plants the barrier point in the processes or groups specified in the current focus. If the current focus:

- Does not indicate an explicit group, the CLI creates a process barrier across the share group.
- Indicates a process group, the CLI creates a process barrier that is satisfied when all members of that group reach the barrier.
- 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 (which is 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, **stop\_when\_hit** 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:*

You may find the following aliases useful:

Alias	Definition	Meaning
<code>ba</code>	<code>{dbarrier}</code>	Defines a barrier.
<code>baw</code>	<code>{dfocus pW dbarrier -stop_when_done process}</code>	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 containing the satisfaction set.
<code>BAW</code>	<code>{dfocus gW dbarrier -stop_when_done group}</code>	Creates a thread barrier across the worker threads in the share group of interest. The set of threads stopped when the barrier is satisfied will be the entire control group.

*Examples:*

`dbarrier 123` Stops each process in the control group when it arrives at line 123. After all 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 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 on 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 source-loc [-p | -g | -t] [[-l lang] -e expr]
```

Creates a breakpoint at an address

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

### Arguments:

- |                      |  |
|----------------------|--|
| <i>source-loc</i>    | The breakpoint location specified as a line number or as a string containing a file name, function name, and line number, each separated by # characters; for example, #file#line. Defaults are constructed if you omit parts of this specification. For more information, see "Qualifying Symbol Names" in Chapter 12 of the TOTALVIEW USERS GUIDE.                           |
| -address <i>addr</i> | 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 <b>STOP_ALL</b> variable to <b>process</b> . See <b>dset</b> on page 96 for more information.  |
| -g                   | Tells TotalView to stop all processes in the process's control group when the breakpoint is hit. You can set this option as the default by setting the <b>STOP_ALL</b> variable to <b>group</b> . See <b>dset</b> on page 96 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 <b>STOP_ALL</b> variable to <b>thread</b> . See <b>dset</b> on page 96 for more information.  |
| -l <i>lang</i>       | Sets the programming language used when you are entering expression <i>expr</i> . The languages you can enter are <b>c</b> , <b>c++</b> , <b>f7</b> , <b>f9</b> , and <b>asm</b> (for C, C++, FORTRAN 77, Fortran 9x, and assembler). 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 <i>expr</i>       | When the breakpoint is hit, TotalView will evaluate expression <i>expr</i> in the context of the thread that hit the breakpoint. The language statements and operators you can use are described in Chapter 14 of the TOTALVIEW USERS GUIDE.   |

*Description:*

The **dbreak** command defines a breakpoint or evaluation point that TotalView triggers when execution arrives at the specified location. The ID of the new breakpoint is returned.

Each thread stops when it arrives at a breakpoint.

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.

The CLI may not be able to set a breakpoint at the line you specify. This occurs when a line does not contain an executable statement.

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

When the CLI displays information on a breakpoint's status, it displays the location where execution will actually stop.

If the CLI encounters a *stop group* breakpoint, it suspends each process in the group as well as the process containing 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 will need to embed code written in Fortran, C, or assembler within Tcl commands. In addition, your expressions will often include TotalView built-in functions. For example, if you want to use the TotalView **\$tid** built-in function, you will need to type it as **\\$tid**.

*Command alias:*

You may find the following aliases useful:

Alias	Definition	Meaning
<b>b</b>	<b>{break}</b>	Sets a breakpoint
<b>bt</b>	<b>{dbreak t}</b>	Sets a breakpoint just on the thread of interest

*Examples:*

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

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

**dbreak -address 0x1000764**  
Suspends process 2 when address 0x1000764 is reached.

**b 12 -g**                        Suspends all processes in the current control group when line 12 is reached.

**dbreak 57 -l f9 -e {goto \$63}**  
Causes the thread that struck the breakpoint to transfer to line 63. The host language for this statement is Fortran 90 or Fortran 95.

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

## 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 will remove files not currently being used and that are not quite old enough for TotalView to automatically delete.

## dcheckpoint

Creates a checkpoint image of processes (SGI only)

### Format:

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

### 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 being checkpointed.
- detach** Processes continue running after being checkpointed. In addition, TotalView detaches from them.
- go** Processes continue running after being checkpointed.
- halt** Processes halt after they are checkpointed.

**-by** *process\_set* Indicates the set of processes that will be checkpointed. If you do not use a *process\_set* option, TotalView only checkpoints the focus process. Your options are:

- ash** Checkpoint the array session.
- hid** Checkpoint the hierarchy rooted in the focus process.
- pgid** Checkpoint the entire UNIX process group.
- sid** Checkpoint the entire process session.

**-no\_park** Tells TotalView that it should not *park* all processes before TotalView begins checkpointing them. If you use this option, you will also need to use the **drestart** command's **-no\_unpark** option. Checkpoints that will be restarted from a shell must use this option.

**-ask\_attach\_parallel** Asks if TotalView should reattach to parallel processes of a parallel job. (Some systems automatically detach you from processes being checkpointed.)

**-no\_attach\_parallel** Tells TotalView that it should not reattach to processes from which the checkpointing processes detached. (Some systems automatically detach you from processes being checkpointed.)

- no\_preserve\_ids** Lets TotalView assign new IDs when it restarts a checkpoint. If you do not use this option, the same IDs are used.
- force** Tells TotalView to overwrite an existing checkpoint.
- checkpoint-name* The name being assigned to the checkpoint.

**Description:**

The **dcheckpoint** command saves program and process information into the *checkpoint-name* file. This information includes process and group IDs. Some time later, you will use the **drestart** command to restart the program.

**NOTE** This command does not save TotalView breakpoint information.

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

- IRIX will not checkpoint a process that is running remotely and which communicates using sockets. As the TotalView Debugger Server (**tvdsvr**) uses sockets to redirect **stdin**, **stdout**, and **stderr**, you will need to use the **drun** command to modify the way your processes send information to a **tty** before creating a checkpoint.
- Because SGI MPI makes extensive use of sockets, you cannot checkpoint SGI MPI programs.

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 that they should stop. This lets you investigate a program's state at the checkpoint position. In contrast, **-go** tells the CLI that it should let the processes continue to run. The **-detach** and **-halt** options are used less frequently. The **-detach** option shuts down the CLI and leaves the processes running. This command's **-halt** option is similar to **-detach**, differing only in that processes started by the CLI and TotalView are also terminated.

The *process\_set* options tell TotalView which processes it should checkpoint. While the focus set can only contain one process, processes in the same process group, process session, process hierarchy, or array session can also be included 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.

Just 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 will be restarting the checkpoint file outside of TotalView, you must use the **-no\_park** option.

On some operating systems (including SGI), 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 that it shouldn't reattach, or use the **-ask\_attach\_parallel** option to indicate that it should ask you if it should reattach.

*Examples:*

```
dcheckpoint check1
```

Checkpoint the current process. TotalView writes the checkpoint information into the *check1* file. These processes stop.

```
f3 dcheckpoint check1
```

Checkpoint process 3. Process 3 stops. TotalView writes the checkpoint information into the *check1* file.

```
f3 dcheckpoint -go check1
```

Checkpoint process 3. Process 3 continues to run. TotalView writes the checkpoint information into the *check1* file.

```
f3 dcheckpoint -by ppid -detach check1
```

Checkpoint 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 into the *check1* file.

## dcont

Continues execution and waits for execution to stop

*Format:*

**dcont**

*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 whose definition is as follows:

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

This behavior is often what you want to do in scripts. It is seldom what you want to do interactively.

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

This command has no arguments.

A **dcont** command completes when all threads in the focus set of processes stop executing.

*Command alias:*

You may find the following aliases useful:

Alias	Definition	Meaning
co	{dcont}	Resume
CO	{dfocus g dcont}	Group-level resume

*Examples:*

**dcont** Resumes execution of all *stopped/runnable* threads belonging to processes in the current focus. (Threads held at barriers are not affected.) 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/runnable* threads belonging to process 1. The CLI does not accept additional commands until the process stops.

dfocus {p1 p2 p3} co

Resumes execution of all *stopped/runnable* threads belonging to processes 1, 2, and 3.

C0

Resumes execution of all *stopped/runnable* threads belonging to the current group.

## ddelete

Deletes action points

### Format:

Deletes some action points

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

Deletes all action points

```
ddelete -a
```

### Arguments:

*action-point-list* A list of the action points being deleted.

**-a** Tells TotalView to delete all action points in the current focus.

### Description:

The **ddelete** command permanently removes one or more action points. The argument to this command lets you specify which action points the CLI should delete. The **-a** option indicates that the CLI should delete all action points.

If you delete a barrier point, the CLI releases the processes and threads held at it.

### Command alias:

You may find the following alias useful:

Alias	Definition	Meaning
<b>de</b>	<b>{ddelete}</b>	Deletes action points

### Examples:

```
ddelete 1 2 3    Deletes breakpoints 1, 2, and 3.
```

```
ddelete -a      Deletes all action points associated with processes in the current focus.
```

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

```
dfocus a de -a
Deletes all action points known to the CLI.
```

## ddetach

Detaches from processes

### Format:

**ddetach**

### 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.

This command has no arguments.

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 containing the thread.

### Command alias:

You may find the following alias useful:

Alias	Definition	Meaning
<b>det</b>	<b>{ddetach}</b>	Detaches from processes

### 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:

Disables some action points

```
ddisable action-point-list
```

Disables all action points

```
ddisable -a
```

### Arguments:

*action-point-list*      A list of the action points being disabled.

-a                        Tells TotalView to disable all action points.

### Description:

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

The first form of this command lets you explicitly name the IDs of the action points being disabled. The second form lets you disable all action points.

### Command alias:

You may find the following alias useful:

Alias	Definition	Meaning
<b>di</b>	<b>{ddisable}</b>	Temporarily disables action points

### 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 action points associated with processes 1 through 4.  
Action points associated with other processes are not  
affected.
```

## 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 **ddown** effectively "moves down" in the call stack. (If "up" is in the direction of **main()**, then "down" is back to where you were before moving 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 one, 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. 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 surrounding 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, **dstep**), or until you enter 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:

You may find the following alias useful:

Alias	Definition	Meaning
<b>d</b>	{ <b>ddown</b> }	Moves down the call stack

*Examples:*`ddown`

Moves down one level in the call stack. As a result, for example, **dlist** commands that follow will refer to the procedure that invoked this one. Here is an example of what is printed 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

Enables action points

### Format:

Enables some action points

```
denable action-point-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.

### Description:

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

If you have not saved the ID values of disabled action points, you can use the **dactions** command to obtain a list of this information.

### Command alias:

You may find the following alias useful:

Alias	Definition	Meaning
en	{denable}	Reenables action points

### Examples:

```
denable 3 4
```

 Enables two previously identified action points. These action points were previously disabled with the **ddisable** command.

```
dfocus {p1 p2} denable -a
```

 Enables all action points associated with processes 1 and 2. Settings associated with other processes are not affected.

```
en -a
```

 Enables all action points associated with the current focus.

```
f a en -a
```

 Enables all actions points in all processes.

## dflush

### Unwinds stack from suspended computations

#### Format:

Removes the top-most suspended expressions evaluations

#### **dflush**

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

#### **dflush** *susp-eval-id*

Removes all suspended computations

#### **dflush** **-all**

#### Arguments:

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

**-all**                      Flushes all suspended evaluations within the current focus.

#### Description:

The **dflush** command unwinds the stack to eliminate frames generated by suspended computations. Typically, these can occur if you had used the **dprint** **-nowait** command. However, this situation can occur, for example, if an error occurred in a function call in an eval point or in an expression within a **Tools > Evaluate** window or if you use a **\$stop** function.

You can use this command in three 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 width, the default focus is thread level.

#### Examples:

The following example uses **dprint** to place five suspended routines on the stack. It then uses **dflush** to remove them. As you follow the example, you'll see the three different ways you can use the **dflush** command.

```

#
# 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)"
...

#
# Here's what the top of the call stack looks like:
#

d1.<> dwhere
0 nothing2          PC=0x00012520, FP=0xffbef130 [fork_loop_2.cxx#310]
1 ***** Eval Function Call (11) *****
2 nothing2          PC=0x00012520, FP=0xffbef220 [fork_loop_2.cxx#310]
3 ***** Eval Function Call (10) *****
4 nothing2          PC=0x00012520, FP=0xffbef310 [fork_loop_2.cxx#310]
5 ***** Eval Function Call (9) *****
6 nothing2          PC=0x00012520, FP=0xffbef400 [fork_loop_2.cxx#310]
7 ***** Eval Function Call (8) *****
8 nothing2          PC=0x00012520, FP=0xffbef4f0 [fork_loop_2.cxx#310]
9 ***** Eval Function Call (7) *****
10 forker           PC=0x00013fd8, FP=0xffbef648 [fork_loop_2.cxx#1120]
11 fork_wrapper     PC=0x00014780, FP=0xffbef6c8 [fork_loop_2.cxx#1278]
...

#
# Use dflush to remove the last item pushed onto the stack.
# Notice the frame associated with "11" is no longer there
#

d1.<> dflush
d1.<> dwhere
0 nothing2          PC=0x00012520, FP=0xffbef220 [fork_loop_2.cxx#310]
1 ***** Eval Function Call (10) *****
2 nothing2          PC=0x00012520, FP=0xffbef310 [fork_loop_2.cxx#310]
3 ***** Eval Function Call (9) *****
4 nothing2          PC=0x00012520, FP=0xffbef400 [fork_loop_2.cxx#310]
5 ***** Eval Function Call (8) *****

```

```

6 nothing2          PC=0x00012520, FP=0xffbef4f0 [fork_loop_2.cxx#310]
7 ***** Eval Function Call (7) *****
8 forker            PC=0x00013fd8, FP=0xffbef648 [fork_loop_2.cxx#1120]
9 fork_wrapper      PC=0x00014780, FP=0xffbef6c8 [fork_loop_2.cxx#1278]

#
# Use dflush with a suspended ID argument to remove all frames up to
# and including the one associated with suspended ID 9. This means
# that 7 and 8 remain.
#

d1.<> dflush 9
# Top of call stack after dflush 9
d1.<> dwhere
0 nothing2          PC=0x00012520, FP=0xffbef400 [fork_loop_2.cxx#310]
1 ***** Eval Function Call (8) *****
2 nothing2          PC=0x00012520, FP=0xffbef4f0 [fork_loop_2.cxx#310]
3 ***** Eval Function Call (7) *****
4 forker            PC=0x00013fd8, FP=0xffbef648 [fork_loop_2.cxx#1120]
5 fork_wrapper      PC=0x00014780, FP=0xffbef6c8 [fork_loop_2.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_loop_2.cxx#1120]
1 fork_wrapper      PC=0x00014780, FP=0xffbef6c8 [fork_loop_2.cxx#1278]

```

## dfocus

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

#### Format:

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*

#### Arguments:

*p/t-set*

A set of processes and threads. This set defines the target upon which the CLI commands that follow will act. You can also use a P/T set filter as one or more of the elements in this list.

*command*

A CLI command, which when it executes, operates upon its own local focus.

#### Description:

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

The **dfocus** command always expects a P/T value as its first argument. This value can 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 *command* in the new focus. After *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, **dfocus** returns the result of the command. If you do not enter a command, **dfocus** returns the focus as a string value.

**NOTE** 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 11 of the TotalView Users Guide.

*Command alias:*

You may find the following alias useful:

Alias	Definition	Meaning
<code>f</code>	<code>{dfocus}</code>	Changes the object upon which a command acts

*Examples:*

<code>dfocus g dgo</code>	Continues the TotalView group containing the focus process.
<code>dfocus p3 {dhalt; dwhere}</code>	Stops process 3 and displays backtraces for each of its threads.
<code>dfocus 2.3</code>	Sets the focus to thread 3 of process 2, where the "2" and the "3" are TotalView's process and thread identifier values. The focus is set to <b>d2.3</b> .
<code>dfocus 3.2</code> <code>dfocus .5</code>	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 <i>process 3, thread 5</i> ( <b>d3.5</b> ).
<code>dfocus g dstep</code>	Steps the current group. Note that while the thread of interest is determined by the current focus, the command acts on the entire group containing that thread.
<code>dfocus {p2 p3} {dwhere ; dgo}</code>	Performs a backtrace on all threads in processes 2 and 3 and then tells these processes to execute.
<code>f 2.3 {f p w; f t s; g}</code>	Executes a backtrace ( <b>dwhere</b> ) 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.
<code>dfocus p1</code>	Changes the current focus to include just those threads currently in process 1. The width is set to <b>process</b> . The CLI sets the prompt to <b>p1.&lt;</b> .
<code>dfocus a</code>	Changes the current set to include all threads in all processes. When you execute this command, you will notice that your prompt changes to <b>a1.&lt;</b> . This command alters the CLI's

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.

You will find many other **dfocus** examples in Chapter 11 of the TOTALVIEW USERS GUIDE.

## dgo

Resumes execution of processes

*Format:*

**dgo**

*Description:*

The **dgo** command tells all non-held 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 **drrun** command. If you are also using the TotalView GUI, this value can be set by using the **Process > Startup** command.

This command has no arguments.

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.

*Command alias:*

You may find the following aliases useful:

Alias	Definition	Meaning
<b>g</b>	<b>{dgo}</b>	Resumes execution
<b>G</b>	<b>{dfocus g dgo}</b>	Group resume

*Examples:*

<b>dgo</b>	Resumes execution of all <i>stopped/runnable</i> threads belonging to processes in the current focus. (Threads held at barriers are not affected.)
<b>G</b>	Resumes execution of all threads in the current control group.
<b>f p g</b>	Continues the current process. Only threads that are not held are actually allowed to run.
<b>f g g</b>	Continues all processes in the control group. Only processes and threads that are not held are allowed to run.
<b>f gL g</b>	Continues all threads in the share group that are at the same PC as the thread of interest.
<b>f pL g</b>	Continues all threads in the current process that are at the same PC as the thread of interest.
<b>f t g</b>	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:

**-g** *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 containing 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 TotalView 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 indicating that TotalView will 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. TotalView adds each of these threads and processes to the group. If you add a:

- Process to a thread group, TotalView adds all of its threads.
- Thread to a process group, TotalView adds the thread's parent process.

You can abbreviate **--add** to **-a**.

The CLI returns the ID of this group.

The items being added can be explicitly named using an *id-list*. If you do not use an *id-list*, 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 this option, the CLI uses the groups in the current focus.

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

**NOTE** If you specify an *id-list* and use the **-g** option, the CLI ignores the focus.

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

TotalView does not let you use this command to add a process to a control group. If you need to perform this operation, you can add it by using 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 uses 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 **–intersect** 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 TotalView's list of group names by using the Tcl **regex** command.

**NOTE** 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; it 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 **–g**, 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.

The items being added can be explicitly named using an *id-list*. If you do not use an *id-list*, the CLI adds the threads and processes in the current focus.

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

**NOTE** If you specify an *id-list* and use the `-g` option, the CLI ignores the focus.

If you are adding more than one object and one of these objects is a duplicate, TotalView will add the nonduplicate objects to the group.

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

### **dgroups --remove**

The `dgroups --remove` command removes members from one or more thread or process groups. If you ask to remove a process from a thread group, TotalView removes all of its threads. If you ask to remove a thread from a process group, TotalView 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 `--remove` to `-r`.

#### *Command alias:*

You may find the following alias useful:

Alias	Definition	Meaning
<code>gr</code>	<code>dgroups</code>	Manipulates a group

*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 containing 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 **mygroup**.

**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 it; 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 its workers group.
- f gW gr -i -g mygroup  
Removes all nonworker threads from **mygroup**.

**dgroups -list**

- dgroups -list      Tells TotalView to display information about all named groups.  
For example:
- ```
ODD_P: {process 1 3}
EVEN_P: {process 2 4}
```
- gr -l \*      Tells TotalView 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 -new**

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

Creates a new thread group named **mygroup** containing 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 workers thread.

## dhalt

Suspends execution of processes

*Format:*

**dhalt**

*Description:*

The **dhalt** command stops all processes and threads in the current focus. The command has no arguments.

*Command alias:*

You may find the following aliases useful:

Alias	Definition	Meaning
<b>h</b>	{ <b>dhalt</b> }	Suspends execution
<b>H</b>	{ <b>dfocus g dhalt</b> }	Group stop

*Examples:*

**dhalt**

Suspends execution of all *running* threads belonging to processes in the current focus. (Threads that are held at barriers are not affected.)

**f t 1.1 h**

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

## dhold

Holds threads or processes

### Format:

Holds processes

**dhold -process**

Holds threads

**dhold -thread**

### Arguments:

**-process** Indicates that processes in the current focus will be held. You can abbreviate **-process** to **-p**.

**-thread** Indicates that threads in the current focus will be held. You can abbreviate **-thread** to **-t**.

### Description:

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

**NOTE** You cannot hold system manager threads.

### Command alias:

You may find the following aliases useful:

Alias	Definition	Meaning
hp	{dhold -process}	Holds the focus process
HP	{f g dhold -process}	Holds all processes in the focus group
ht	{f t dhold -thread}	Holds the focus thread
HT	{f g dhold -thread}	Holds all threads in the focus group
htp	{f p dhold -thread}	Holds all threads in the focus process

### 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.

## dkill

Terminates execution of processes

*Format:*

**dkill**

*Description:*

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

This command has no arguments.

Because the executables associated with the defined processes are still “loaded,” typing 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.

*Command alias:*

You may find the following aliases useful:

Alias	Definition	Meaning
k	{dkill}	Terminates a process's execution

*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:*

```
dlappend variable-name value [ ... ]
```

*Arguments:*

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

*value*                    The values being appended.

*Description:*

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

```
lappend foo 1 3 5
```

In contrast, the following command displays an error message:

```
dlappend foo 1 3 5
```

*Examples:*

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

## dlist

Displays source code lines

### Format:

Displays code relative to the current list location

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

Displays code relative to a named place

```
dlist source-loc [ -n num-lines ]
```

Displays code relative to the current execution location

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

### Arguments:

**-n** *num-lines*

Requests that this number of lines be displayed rather than the default value. (The default is the value of the **MAX\_LIST** variable.) If *num-lines* is negative, lines before the current location are shown, and additional **dlist** commands will show preceding lines in the file rather than succeeding lines.

This option also sets the value of the **MAX\_LIST** variable to *num-lines*.

*source-loc*

Sets the location at which the CLI begins displaying information. This location is specified as a line number or as a string containing 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 11 of the TOTALVIEW USERS GUIDE. Defaults are constructed if you omit parts of this specification.

**-e**

Sets the display location to include the current execution point of the thread of interest. If you used **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 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 neither *source-loc* nor **-e** is specified, the command continues where the previous list command left off. To display the thread's execution point, use **dlist -e**.

If you enter a file or procedure name, the listing begins at the file or procedure's first line.

The default focus for this command is thread level. If your focus is at process level, TotalView acts on each thread in the process.

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

Tabs in the source file are expanded as blanks in the output. The tab stop width is controlled by the **TAB\_WIDTH** variable, which has a default value of 8. If **TAB\_WIDTH** is set to `-1`, no tab processing is done, and tabs are displayed using their ASCII value.

All lines are shown 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 **dwhere** that indicates the selected frame.

Here are some general rules:

- The initial display location is **main()**.
- The display location is set 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:

You may find the following alias useful:

Alias	Definition	Meaning
	{dlist}	Displays lines

*Examples:*

These examples assume that **MAX\_LIST** is at its initial value of 20.

- |  |   |
|--|---|
| <code>dlist</code>                     | Displays 20 lines of source code, beginning at the current list location. The list location is incremented by 20 when the command completes.  |
| <code>dlist 10</code>                  | Displays 20 lines, starting with line 10 of the file corresponding to the current list location. Because an explicit value was used, the CLI ignores the previous command. The list location is changed to line 30.       |
| <code>dlist -n 10</code>               | Displays 10 lines, starting with the current list location. The value of the list location is incremented by 10.  |
| <code>dlist -n -50</code>              | Displays source code preceding the current list location; 50 lines are shown, ending with the current source code location. The list location is decremented by 50.   |
| <code>dlist do_it</code>               | Displays 20 lines in procedure <b>do_it</b> . The list location is changed so that it is the 20th line of the procedure.  |
| <code>dfocus 2.&lt; dlist do_it</code> | Displays 20 lines in the routine <b>do_it</b> associated with process 2. If the current source file were named <b>foo</b> , this could also be specified as <b>dlist foo#do_it</b> , naming the executable for process 2. |
| <code>dlist -e</code>                  | Displays 20 lines starting 10 lines above the current execution location.   |
| <code>f 1.2 l -e</code>                | Lists the lines around the current execution location of thread 2 in process 1.   |
| <code>dfocus 1.2 dlist -e -n 10</code> | Produces essentially the same listing as the previous example, differing in that 10 lines are displayed.  |
| <code>dlist do_it.f#80 -n 10</code>    | Displays 10 lines, starting with line 80 in file <b>do_it.f</b> . The list location is updated to line 90.  |

## dload

### Loads debugging information

#### Format:

**dload** [ **-g** *gid* ] [ **-r** *hname* ] [ **-e** ] *executable*

#### Arguments:

<b>-g</b> <i>gid</i>	Sets the control group for the process being added to the group ID specified by <i>gid</i> . This group must already exist. (The CLI <b>GROUPS</b> variable contains a list of all groups.)
<b>-r</b> <i>hname</i>	The host on which the process will run. The CLI will launch a TotalView Debugger Server on the host machine if one is not already running there. See Chapter 5, "Setting Up Parallel Debugging Sessions" of the TOTALVIEW USERS GUIDE for information on the server launch commands.
<b>-e</b>	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.
<i>executable</i>	A fully or partially qualified file name for the file corresponding to the program.

#### Description:

The **dload** command creates a new TotalView process object for *executable*. The **dload** command returns the TotalView ID for the new object.

#### Command alias:

You may find the following alias useful:

Alias	Definition	Meaning
<b>lo</b>	<b>{dload}</b>	Loads debugging information

#### Examples:

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

**lo -g 3 -r other\_computer do\_this**  
Loads the debugging information for executable **do\_this** executing on the **other\_computer** machine into the CLI. This process is placed into group 3.

## dload

```
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.

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

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

## dmstat

Displays memory use information

*Format:*

dmstat

*Description:*

The **dmstat** command displays information about how your program is using memory. After you enter this command, the CLI returns memory information. This information is displayed 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 values shown in the six columns are:

<b>text</b>	The amount of memory used to store your program's machine code instructions. The "text segment" is sometimes called the "code segment."
<b>data</b>	The amount of memory used to store initialized data.
<b>heap</b>	The amount of memory currently being used for data created at runtime. The heap is an area of memory that your program uses when it needs to dynamically allocate memory. For example, calls to <b>malloc()</b> allocate space on the heap while <b>free()</b> releases it.
<b>stack</b>	The amount of memory used by the currently executing block or routines and all the blocks or routines that have invoked it. For example, if your main routine invokes function <b>foo()</b> , 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 <b>foo()</b> . If <b>foo()</b> invokes <b>bar()</b> , the stack contains three frames. When <b>foo()</b> finishes executing, the stack only contains one frame.
<b>stack_vm</b>	The logical size of the stack is the difference between the current value of the stack pointer and 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 value here is this size difference.

**vm\_size**

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

*Examples:*

**dmstat**

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

```

    process:   text      data      heap      stack  [stack_vm]  vm_size
  1 (18549):  2257.08K   32.31M   17179869184.00G   9888 [ 278528]   22.70M

image information:
      image_name      text      data  dpids
...ry/forked_mem_exampleLINUX      5048   33556958  1
  /lib/i686/libpthread.so.0      64344   55896  1
  /lib/i686/libc.so.6      2101376   244676  1
  /lib/ld-linux.so.2      140480   21626  1

```

**dfocus a dmstat**

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

```

    process:   text      data      heap      stack  [stack_vm]  vm_size
  1 (18549):  2257.08K   32.31M   17179869184.00G   9888 [ 278528]   22.70M
  2 (18550):  2257.08K   32.31M   17179869183.99G   534192 [ 540672]   17.94M
  3 (18551):  2257.08K   32.31M   17179869183.99G   796336 [ 802816]   18.19M
  4 (18552):  2257.08K   32.31M   17179869183.99G   1033.67K [1040.00K]   18.44M

maximum:
  1 (18549):  2257.08K   32.31M   17179869184.00G   9888 [ 278528]   22.70M
minimum
  2 (18550):  2257.08K   32.31M   17179869183.99G   534192 [ 540672]   17.94M

image information:
      image_name      text      data  dpids
...ry/forked_mem_exampleLINUX      5048   33556958  1 2 3 4
  /lib/i686/libpthread.so.0      64344   55896  1 2 3 4
  /lib/i686/libc.so.6      2101376   244676  1 2 3 4
  /lib/ld-linux.so.2      140480   21626  1 2 3 4

```

## dnext

### Steps source lines, stepping over subroutines

#### Format:

**dnext** [ *num-steps* ]

#### Arguments:

*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, **dnext** 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 it should 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:

You may find the following aliases useful:

Alias	Definition	Meaning
n	{ <b>dnext</b> }	Runs the thread of interest one statement while allowing other threads in the process to run.
N	{ <b>dfocus g dnext</b> }	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.

Alias	Definition	Meaning
nl	{dfocus L dnext}	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 are at the same PC is called the <i>lockstep group</i> .  This alias does not force process width. If the default focus is set to <b>group</b> , this steps the group.
NL	{dfocus gL dnext}	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.
nw	{dfocus W dnext}	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 <b>group</b> , this steps the group.
NW	{dfocus gW dnext}	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 statement. All other threads in the control group run freely.

**Examples:**

dnext	Steps one source line.
n 10	Steps ten source lines.
N	Steps one source line. It also runs all other processes in the group that are in the same lockstep group to the same line.
f t n	Steps the thread one statement.
dfocus 3. dnext	Steps process 3 one step.

## dnexti

### Steps machine instructions, stepping over subroutines

*Format:*

**dnexti** [ *num-steps* ]

*Arguments:*

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

*Description:*

The **dnexti** command executes machine-level instructions; that is, it advances the program by a single instruction. However, if the instruction invokes a subfunction, **dnexti** 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 **dnexti** operations it should perform. If you do not specify *num-steps*, the default is 1.

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

For more information on stepping in processes and threads, see **dstep** on page 100.

*Command alias:*

You may find the following aliases useful:

Alias	Definition	Meaning
ni	{ <b>dnexti</b> }	Runs the thread of interest one instruction while allowing other threads in the process to run.
NI	{ <b>dfocus g dnexti</b> }	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.

Alias	Definition	Meaning
nil	{dfocus L dnexti}	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 are at the same PC is called the <i>lockstep group</i> .  This alias does not force process width. If the default focus is set to <b>group</b> , this steps the group.
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 <b>group</b> , 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 as well.
f t n	Steps the thread one machine-level instruction.
dfocus 3. dnexti	Steps process 3 one machine-level instruction.

## dout

Runs out from the current subroutine

*Format:*

**dout** [ *frame-count* ]

*Arguments:*

*frame-count*

An integer that specifies that the thread return 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 the current subroutine.
- From one or more nested subroutines.

When process width is specified, TotalView allows all threads in the process that are not running to this goal to run free. Note that specifying process width is the default.

*Command alias:*

You may find the following aliases useful:

Alias	Definition	Meaning
ou	{ <b>dout</b> }	Runs the thread of interest out of the current function while allowing other threads in the process to run.
OU	{ <b>dfocus g dout</b> }	A group stepping command. This 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.

Alias	Definition	Meaning
oul	{dfocus L dout}	<p>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 are at the same PC is called the <i>lockstep group</i>.</p> <p>This alias does not force process width. If the default focus is set to <b>group</b>, this steps the group.</p>
OUL	{dfocus gL dout}	<p>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.</p>
ouw	{dfocus W dout}	<p>Runs worker threads in the process. This runs the thread of interest out of the current function and runs all worker threads in the process to the same (goal) statement. The nonworker threads in the process run freely.</p> <p>This alias does not force process width. If the default focus is set to <b>group</b>, this steps the group.</p>
Ouw	{dfocus gW dout}	<p>Runs worker threads in the group. This runs the thread of interest out of the current function and also runs all worker threads in the same share group out of the current function. All other threads in the control group run freely.</p>

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

```
dprint [ -nowait ] variable
```

Prints the value of an expression

```
dprint [ -nowait ] expression
```

### Arguments:

**-nowait** Tells TotalView to evaluate the expression in the background. You will need to use **TV::expr** to obtain the results as they are not displayed.

*variable* A variable whose value will be 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 be evaluated and printed. Because *expression* must also conform to Tcl syntax, you must quote it if it includes any blanks, and it must be enclosed in braces (**{}**) if it includes brackets (**[ ]**), dollar signs (**\$**), quote characters (**"**), or any other Tcl special characters.

### Description:

The **dprint** command evaluates and displays a variable or an expression. The CLI interprets the expression by looking up the values associated with each symbol and applying the operators. The result of an expression can be a scalar value or an aggregate (array, array slice, or structure).

If an event such as a **\$stop**, SEGV, breakpoint, or the like occurs, the **dprint** command throws an exception describing the event. The first exception subcode returned by **TV::errorCodes** is the *susp-eval-id* (a suspension-evaluation-ID). You can use this to manipulate suspended evaluations with the **dflush** and **TV::expr** commands.

If you use **-nowait**, TotalView evaluates the expression in the background. It also returns a *susp-eval-id* that you can use to obtain the results of the evaluation.

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 may want to use the **capture** command described on page 21 to save the output into a variable.

Structure output appears with one field printed per line. For example:

```
sbfo = {
  f3 = 0x03 (3)
  f4 = 0x04 (4)
  f5 = 0x05 (5)
  f20 = 0x000014 (20)
  f32 = 0x00000020 (32)
}
```

Arrays are printed in a similar manner. For example:

```
foo = {
  [0][0] = 0x00000000 (0)
  [0][1] = 0x00000004 (4)
  [1][0] = 0x00000001 (1)
  [1][1] = 0x00000005 (5)
  [2][0] = 0x00000002 (2)
  [2][1] = 0x00000006 (6)
  [3][0] = 0x00000003 (3)
  [3][1] = 0x00000007 (7)
}
```

You can append a slice to the variable's name to tell the CLI that it should display a portion of an array. For example:

```
d.1<> p {master_array[::10]}
master_array(::10) = {
  (1) = 1 (0x00000001)
  (11) = 1331 (0x00000533)
  (21) = 9261 (0x0000242d)
  (31) = 29791 (0x0000745f)
  (41) = 68921 (0x00010d39)
  (51) = 132651 (0x0002062b)
  (61) = 226981 (0x000376a5)
```

```

(71) = 357911 (0x00057617)
(81) = 531441 (0x00081bf1)
(91) = 753571 (0x000b7fa3)
}

```

Note that the slice was placed within `{}` symbols. This prevents Tcl from trying to evaluate the information in the `[]` characters. You could, of course, instead escape the brackets; for example, `\[ \]`.

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

```

You can also use the **dprint** command to obtain values for your computer's registers. For example, on most architectures, **\$r1** is register 1. You would obtain the contents of this register by typing:

```
dprint \$r1
```

Notice that you must quote the **\$** since the name of the register's name includes the **\$**. This **\$** is not the standard indicator that tells Tcl to fetch a variable's value. Chapter 12, "Architectures" on page 251 lists the mnemonic names assigned to registers.

**NOTE** Do not use a **\$** when asking **dprint** to display your program's variables.

#### *Command alias:*

You may find the following alias useful:

Alias	Definition	Meaning
<b>p</b>	<code>{dprint}</code>	Evaluates and displays information

*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 just 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() }
```

The **dprint** command evaluates a function contained within three threads. Note that 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

This example evaluates a function without waiting. At a later time, the results are obtained using **TV::expr**. The number displayed immediately after the command, which is "1", is the *susp-eval-id*. Here is how this is done:

```
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 status of processes and threads

### Format:

```
dptsets [ ptset_array ] ...
```

### Arguments:

*ptset\_array* An optional array that indicates the P/T sets that will be shown. 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 11 of the TOTALVIEW USERS GUIDE.

### Description:

The **dptsets** command shows the status of each process and thread in a Tcl array of P/T expressions. These array elements are P/T expressions (see Chapter 11 of the TOTALVIEW USERS GUIDE) and the elements' array indices are strings that label each element's section in the output. Using this array syntax is explored in the *Examples* section.

If you do not use the optional *ptset\_array* argument, the CLI supplies a default array containing all P/T set designators. These designators are **error**, **existent**, **held**, **running**, **stopped**, **unheld**, and **watchpoint**.

### Examples:

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

```
d.1<> dptsets
unheld:
1:      808694      Stopped  [fork_loopSGI]
  1.1:  808694.1   Stopped  PC=0x0d9cae64
  1.2:  808694.2   Stopped  PC=0x0d9cae64
  1.3:  808694.3   Stopped  PC=0x0d9cae64
  1.4:  808694.4   Stopped  PC=0x0d9cae64

existent:
1:      808694      Stopped  [fork_loopSGI]
  1.1:  808694.1   Stopped  PC=0x0d9cae64
  1.2:  808694.2   Stopped  PC=0x0d9cae64
  1.3:  808694.3   Stopped  PC=0x0d9cae64
  1.4:  808694.4   Stopped  PC=0x0d9cae64
```

```
watchpoint:
```

```
running:
```

```
held:
```

```
error:
```

```
stopped:
```

```
1:      808694      Stopped   [fork_loopSGI]
  1.1:  808694.1    Stopped   PC=0x0d9cae64
  1.2:  808694.2    Stopped   PC=0x0d9cae64
  1.3:  808694.3    Stopped   PC=0x0d9cae64
  1.4:  808694.4    Stopped   PC=0x0d9cae64
...

```

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

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

1:
1:      892484      Breakpoint [arraysSGI]
  1.1:  892484.1    Breakpoint PC=0x10001544, [arrays.F#81]

```

The array index to **set\_info** becomes a label identifying the kind of information being displayed. In contrast, the information within parentheses in the **breakpoint** and **stopped** functions identify the arena for which the function will return information.

Using numbers as array indices almost ensures that you will not remember what is being printed. The following almost identical example shows a better way to use these array indices.

```

d1.<> set set_info(my_breakpoints) breakpoint(1)
breakpoint(1)
d1.<> set set_info(my_stopped) stopped(1)
stopped(1)
d1.<> dptsets set_info
my_stopped:
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. It differs in that the second element is the difference between three P/T sets.

```

d.1<> set mystat(system) a-gW
d.1<> set mystat(reallystopped) \
      stopped(a)-breakpoint(a)-watchpoint(a)
d.1<> dptsets t mystat
system:
Threads in process 1 [regress/fork_loop]:
1.-1:   21587. [-1] Running PC=0x3ff805c6998
1.-2:   21587. [-2] Running PC=0x3ff805c669c
...
Threads in process 2 [regress/fork_loop.1]:
2.-1:   15224. [-1] Stopped PC=0x3ff805c6998
2.-2:   15224. [-2] Stopped PC=0x3ff805c669c
...

reallystopped:
2.2:    15224.2    Stopped PC=0x3ff800d5758
2.-1:   15224. [-1] Stopped PC=0x3ff805c6998
2.-2:   15224. [-2] Stopped PC=0x3ff805c669c
...

```

## drerun

Restarts processes

### Format:

```
drerun [ cmd_args ][ in_operation infile ]
      [ out_operations outfile ]
      [ error_operations errfile ]
```

### Arguments:

<i>cmd_args</i>	The arguments to be used for restarting a process.
<i>operations</i>	The <i>in_operation</i> , <i>out_operations</i> , and <i>error_operations</i> are discussed in the <i>Description</i> section.
<i>infile</i>	If specified, indicates a file from which the launched processes will read information.
<i>outfile</i>	If specified, indicates the file into which the launched processes will write information.
<i>errfile</i>	If specified, indicates the file into which the launched processes will write error information.

### 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** and **ARGS\_DEFAULT** variables. These are set every time the process is run with different arguments. Consequently, if you do not specify the arguments to be used when restarting the process, the CLI uses the arguments specified when the process was previously run. (See **drun** on page 93 for more information.)

The **drerun** command differs from the **drun** command in that

- If you do not specify an argument, **drerun** 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 no arguments will be used.
- If the process already exists, **drun** will not restart it. (If you must use the **drun** command, you must first kill the process.) In contrast, the **drerun** command will kill and then restart the process.

The arguments to this command are similar to the arguments used in the Bourne shell.

The *in\_operation* is follows:

< *infile*                      Reads from *infile* instead of **stdin**.

The *out\_operations* are as follows:

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

The *error\_operations* are:

- 2> *errfile*                      Sends error messages to *errfile* instead of **stderr**.
- 2>>*errfile*                      Appends error messages to *errfile*.

#### Command alias:

You may find the following alias useful:

Alias	Definition	Meaning
rr	{drerun}	Restarts processes

#### Examples:

- drerun                              Reruns the current process. Because arguments are not used, the process is restarted using its previous values.
- rr -firstArg an\_argument -aSecondArg a\_second\_argument  
Reruns the current process. The default arguments are not used because replacement arguments are specified.

## drestart

Restarts a checkpoint (SGI only)

*Format:*

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

*Arguments:*

<i>process_state</i>	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:
<b>-detach</b>	While TotalView starts checkpointed process, it does not attach to them.
<b>-go</b>	TotalView starts checkpointed parallel processes and attaches to them.
<b>-halt</b>	TotalView stops checkpointed processes after it restarts them.
<b>-no_unpark</b>	Indicates that the checkpoint was created outside of TotalView or that you used the <b>dcheckpoint</b> command's <b>-no_park</b> option when you created the checkpoint file.
<b>-g</b> <i>gid</i>	Names the control group into which TotalView places all created processes.
<b>-r</b> <i>host</i>	Names the remote host upon which the restart will occur.
<b>-ask_attach_parallel</b>	Asks if the CLI should automatically attach to the parallel processes being created. This is most often used in procedures.
<b>-no_attach_parallel</b>	Tells TotalView to attach only to the base process. That is, the CLI will not attach to the parallel processes being created.
<b>-no_preserve_ids</b>	Tells TotalView that it should 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 <b>ash</b> IDs after restarting.
<i>checkpoint-name</i>	The name used when the checkpoint file was saved.

*Description:*

The **drestart** command restores and restarts all of the checkpointed processes. By default, the CLI will attach to the base process. Here are some of your choices.

- If there are parallel processes related to this base process, TotalView will attach 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 within a Tcl procedure, you should use the `-ask_parallel_process` option.

*Examples:*

```
drestart check1
```

Restarts the processes checkpointed in the **check1** file. The CLI automatically attaches to parallel processes.

```
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.

# drun

Starts or restarts processes

## Format:

```
drun [ cmd_arguments ] [ in_operation infile ]
      [ out_operations outfile ]
      [ error_operations errfile ]
```

## Arguments:

<i>cmd_arguments</i>	The argument list passed to the process.
<i>operations</i>	The <i>in_operation</i> , <i>out_operations</i> , and <i>error_operations</i> are discussed in the <i>Description</i> section.
<i>infile</i>	If specified, indicates a file from which the launched processes will read information.
<i>outfile</i>	If specified, indicates the file into which the launched processes will write information.
<i>errfile</i>	If specified, indicates the file into which the launched processes will write error information.

## Description:

The **drun** command launches each process in the current focus and starts it running. The command arguments are passed to the processes, and I/O redirection for the program, if specified, will occur. Later in the session, you can use the **drrun** command to restart the program.

The arguments to this command are similar to the arguments used in the Bourne shell.

The *in\_operation* is as follows:

< *infile*                Reads from *infile* instead of **stdin**.

The *out\_operations* are as follows:

> *outfile*                Sends output to *outfile* instead of **stdout**.

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

>>& *outfile*                Appends output and error messages to *outfile*.

>> *outfile*                Appends output to *outfile*.

The *error\_operations* are:

2> *errfile*                Sends error messages to *errfile* instead of **stderr**.

2> >*errfile*                Appends error messages to *errfile*.

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 will use 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(*n*)**                An array variable containing the command-line arguments. The index *n* is the process ID *n*. 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 that it should execute initial processes because TotalView cannot directly run processes that your program spawns. When you enter this command, the initial process must be have terminated; if it was not terminated, you are told to kill it and retry. (You can, of course, use the **drerun** command.)

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 reenter this command for processes that TotalView previously started—or issued for the first time for a process that was attached to using the **dattach** command—the CLI reinitializes your program.

*Issues With Using IBM's poe*

Both **poe** and the CLI can interfere with one another because each believes that it owns **stdin**. Because **poe** 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 will never see these characters. If your target process does not use **stdin**, you can use the **-stdinmode none** option. Unfortunately, this option is incompatible with **poe's -cmdfile** option that is used when specifying **-pgmmodel mpm**.

If you encounter these problems, you should redirect **stdin** within the CLI. For example:

```
drun < in.txt
```

*Command alias:*

You may find the following alias useful:

Alias	Definition	Meaning
<b>r</b>	<b>{drun}</b>	Starts or restarts processes

*Examples:*

<b>drun</b>	Tells the CLI to begin executing processes represented in the current focus.
<b>f {p2 p3} drun</b>	Begins execution of processes 2 and 3.
<b>f 4.2 r</b>	Begins execution of process 4. Note that this is the same as <b>f 4 drun</b> .
<b>dfocus a drun</b>	Restarts execution of all processes known to the CLI. If they were not previously killed, you are told to use the <b>dkill</b> command and then try again.
<b>drun &lt; in.txt</b>	Restarts execution of all processes in the current focus, setting them up to get standard input from file <b>in.txt</b> .

## 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**

Set the value to be used as the variable's default. This option is most often used by system administrators to set site-specific defaults in the global **.tvdr** startup script. Values set using this option replace the Etnus-supplied default.

### Description:

The **dset** command sets the value of CLI debugger variables. You'll find a listing and description of CLI and TotalView variables in Chapter 4, "TotalView Variables" on page 161.

If you type **dset** with no arguments, the CLI displays the names and current values for all TotalView CLI variables in the global namespace. If you use only one argument, the CLI returns and displays the variable's value.

The second argument defines the value that will replace a variable's previous value. It must be enclosed in quotes 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, just enter the namespace name immediately followed by a double colon; for example, **TV::**.

You can use an asterisk (\*) as a wildcard to indicate that the CLI should match more than one string; for example, **TV::g\*** matches all variables beginning with **g** in the **TV** namespace. For example, to view all variables in the **TV::** namespace, you would enter:

```
dset TV::
```

or:

```
dset TV::GUI::
```

The rightmost double colons are required 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.

#### Examples:

```
dset PROMPT "Fixme% "
```

Sets the prompt to be **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 a user changes this value in any way, the user will be able to select the **Defaults** button within the **File > Preferences's Launch String** Page to reset it to this value.

```
dset TV::GUI::fixed_font_size 12
```

Sets the TotalView GUI so that it displays fixed fonts using a 12 font. Commands such as this are often found in a startup file.

## dstatus

Shows current status of processes and threads

### Format:

**dstatus**

### Description:

The **dstatus** command prints information about the current state of each process and thread in the current focus. The **ST** command is an alias for **dfocus g dstatus**, and acts as a group-status command.

This command has no arguments.

If you have not changed the focus, the default width is *process*. In this case, **dstatus** 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 containing process 1.

### Command alias:

You may find the following aliases useful:

Alias	Definition	Meaning
<b>st</b>	{ <b>dstatus</b> }	Shows current status
<b>ST</b>	{ <b>dfocus g dstatus</b> }	Group status

### 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 containing the focus process. For example:
	<pre> 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   1.5:      773686.5   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   2.5:      779490.5   Stopped   PC=0x0d9cae64 </pre>
f W st	Shows status for all worker threads in the focus set. If the focus is set to <b>d1.&lt;</b> , 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 <b>W</b> and <b>g</b> specifiers in the <b>ST</b> alias. The results is the same as if you had entered <b>f gW st</b> .
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** [ *num-steps* ]

### Arguments:

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

### 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, **dstep** steps into the function.

The optional *num-steps* argument tells the CLI how many **dstep** operations it should 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 containing the thread to be stepped. Thus, while only one thread is stepped, all other threads contained in the same process also resume executing. In contrast, the **dfocus t dstep** command tells the CLI that it should just step the *thread of interest*.

**NOTE** On systems having identifiable manager threads, the “**dfocus t dstep**” command allows the manager threads to run as well as the thread of interest.

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.

### *thread width*

Only the thread of interest is allowed to run. (This is not supported on all systems.)

### *process width (default)*

The behavior depends on the group specified in the arena.

**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, it also stops there. The group of interest specifies the set of threads for which TotalView will wait. This means that the command does not complete until all threads in the group of interest are at the goal.

### *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. One matching thread from each matching process is selected. 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 will run, rather than just the process of interest. Regardless of what 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.

### *Command alias:*

You may find the following aliases useful:

Alias	Definition	Meaning
s	{dstep}	Runs the thread of interest one statement while allowing other threads in the process to run.
S	{dfocus g dstep}	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.

Alias	Definition	Meaning
sl	{dfocus L dstep}	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 are at the same PC is called the <i>lockstep group</i> .  This alias does not force process width. If the default focus is set to <b>group</b> , this steps the group.
SL	{dfocus gL dstep}	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.
sw	{dfocus W dstep}	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 <b>group</b> , this steps the group.
SW	{dfocus gW dstep}	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.

**Examples:**

dstep	Executes the next source line, stepping into any procedure call that is encountered. While only the current thread is stepped, 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 are halted 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** [ *num-steps* ]

*Arguments:*

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

*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 it should perform. If you do not specify *num-steps*, the default is 1.

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

*Command alias:*

You may find the following aliases useful:

Alias	Definition	Meaning
si	{dstepi}	Runs the thread of interest one instruction while allowing other threads in the process to run.
SI	{dfocus g dstepi}	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.
sil	{dfocus L dstepi}	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 are at the same PC is called the <i>lockstep group</i> .  This alias does not force process width. If the default focus is set to <b>group</b> , this steps the group.

Alias	Definition	Meaning
SIL	{dfocus gL dstepi}	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.
siw	{dfocus W dstepi}	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 <b>group</b> , this steps the group.
SIW	{dfocus gW dstepi}	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:*

dstepi	Executes the next machine instruction, stepping into any procedure call that is encountered. While only the current thread is stepped, other threads in the process are allowed to run.
si 15	Executes the next 15 instructions.
f p1.2 dstepi	Steps thread 2 in process 1 by one instruction and resumes execution of all other threads in process 1; they are halted as soon as thread 2 in process 1 executes its instruction.
f t1.2 si	Steps thread 2 in process 1 by one instruction. No other threads in process 1 execute.

## dunhold

Releases a held process or thread

### Format:

Releases a process

**dunhold -process**

Releases a thread

**dunhold -thread**

### Arguments:

**-process**

Indicates that TotalView should release processes in the current focus. You can abbreviate the **-process** argument to **-p**.

**-thread**

Indicates that TotalView should release threads in the current focus. You can abbreviate the **-thread** to **-t**.

### Description:

The **dunhold** command releases the threads or processes in the current focus.

Note that system manager threads cannot be held or released.

### Command alias:

You may find the following aliases useful:

Alias	Definition	Meaning
uhp	{dfocus p dunhold -process}	Releases the process of interest
UHP	{dfocus g dunhold -process}	Releases the processes in the focus group
uht	{dfocus t dunhold -thread}	Releases the thread of interest
UHT	{dfocus g dunhold -thread}	Releases all threads in the focus group
uhtp	{dfocus p dunhold -thread}	Releases the threads in the current process

### Examples:

f w uhtp

Unholds all worker threads in the focus process.

htp; uht

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

```
dunset debugger-var
```

Restores all CLI variables to their default values

```
dunset -all
```

### Arguments:

*debugger-var*            Name of the CLI variable whose default setting is being restored.

-all                      Restores the default settings of the CLI variables.

### Description:

The **dunset** command reverses the effects of any previous **dset** commands, restoring CLI variables to their default settings. See Chapter 4, "TotalView Variables" on page 161 for information on these variables.

Tcl variables (those created with the Tcl **set** command) are, of course, unaffected by this command.

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

### Examples:

```
dunset PROMPT       Restores the prompt string to its default setting; that is,  
                          {[dfocus]> }.
```

```
dunset -all         Restores all CLI variables to their default settings.
```

## duntil

Runs the process until a target place is reached

### Format:

Runs to a line

**duntil** *line-number*

Runs to an address

**duntil** **-address** *addr*

Runs into a function

**duntil** *proc-name*

### Arguments:

*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, it 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.

**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.

The differences between this command and other stepping commands are:

- **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 are run. While 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:*

You may find the following aliases useful:

Alias	Definition	Meaning
un	{duntil}	Runs the thread of interest until it reaches a target while allowing other threads in the process to run.
UN	{dfocus g duntil}	Runs the entire control group until every process in the share group has at least one thread at the goal. Processes have a thread at the goal do not run.
unl	{dfocus L duntil}	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 are at the same PC is called the <i>lockstep group</i> .  This does not force process width. If the default focus is set to <b>group</b> , this runs the group.
UNL	{dfocus gL duntil}	Runs "lockstep" threads in the share group until they reach the target. Other threads in the control group are allowed to run freely.

Alias	Definition	Meaning
unw	{dfocus W duntil}	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 <b>group</b> , this runs the group.
UNW	{dfocus gW duntil}	Runs worker threads in the same share group to a target. All other threads in the control group run freely.

*Examples:*

UNW 580            Lines up all worker threads at 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 one, 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 effect of **dup** 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 **dlist** displays the code surrounding this location. Entering **dup 2** (while in frame 0) followed by a **dlist**, for instance, displays source lines centered around the location from which the current routine’s parent was invoked. These lines will be in frame 2.

*Command alias:*

You may find the following alias useful:

Alias	Definition	Meaning
<b>u</b>	{ <b>dup</b> }	Moves up the call stack

*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**

*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 identically to 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.

This command has no arguments.

Unlike most other CLI commands, this command blocks additional CLI input until the blocking action is complete.

*Examples:*

**dwait**

Blocks further command input until all processes in the current focus have stopped (that is, none of their threads are still **running**).

**dfocus {p1 p2} dwait**

Blocks command input until processes 1 and 2 stop.

## dwatch

Defines a watchpoint

### Format:

Defines a watchpoint for a variable

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

Defines a watchpoint for an address

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

### Arguments:

<i>variable</i>	A symbol name corresponding to a scalar or aggregate identifier, an element of an aggregate, or a dereferenced pointer.
<b>-address</b> <i>addr</i>	An absolute address in the file.
<b>-length</b> <i>byte-count</i>	The number of bytes to watch. If a variable is named, 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.
<b>-g</b>	Tells TotalView to stop all processes in the process's control group when the watchpoint is hit.
<b>-p</b>	Tells TotalView to stop the process that hit this watchpoint.
<b>-t</b>	Tells TotalView to stop the thread that hit this watchpoint.
<b>-l</b> <i>lang</i>	Specifies the language used when you are writing an expression. The values you can use for <i>lang</i> are <b>c</b> , <b>c++</b> , <b>f7</b> , <b>f9</b> , and <b>asm</b> , 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 only an address is used, TotalView uses the C language.  Not all languages are supported on all systems.
<b>-e</b> <i>expr</i>	When the watchpoint is triggered, evaluates <i>expr</i> in the context of the thread that hit the watchpoint. In most cases, you need to enclose the expression in braces { }.
<b>-t</b> <i>type</i>	The data type of <b>\$oldval</b> / <b>\$newval</b> in the expression.

**Description:**

A **dwatch** command defines a watchpoint on a memory location where the specified variables are stored. The watchpoint triggers whenever the value of the variables changes. The CLI returns the ID of the newly created watchpoint.

**NOTE** Watchpoints are not available on Alpha Linux and HP.

The default action is controlled by the **STOP\_ALL** variable.

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 two 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.

**NOTE** Chapter 14 of the *TotalView Users Guide* contains additional information on watchpoints.

If you do not use the **-length** modifier, 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 specify that a certain number of bytes be watched, starting at the named location.

**NOTE** In all cases, the CLI watches addresses. If you specify a variable as the target of a watchpoint, the CLI resolves the variable to an absolute address. If you are watching a local stack variable, the position being watched is just where the variable happened to be when space for the variable was allocated.

The focus establishes the processes (not individual threads) for which the watchpoint is in effect.

The CLI prints a message showing the action point identifier, the location being watched, the current execution location of the triggering thread, and the identifier of the triggering threads.

One possibly confusing aspect of using expressions is that their syntax differs from that of Tcl. This is because you will need to embed code written in Fortran, C, or assembler within Tcl commands. In addition, your expressions will often include TotalView built-in functions.

#### Command alias:

You may find the following alias useful:

Alias	Definition	Meaning
<code>wa</code>	<code>{dwatch}</code>	Defines a watchpoint

#### Examples:

For these examples, assume that the current process set at the time of the **dwatch** command consists only of process 2, and that **p** is a global variable that is a pointer.

- `dwatch *p` Watches the address stored in pointer **p** at the time the watchpoint is defined, for changes made by process 2. Only process 2 is stopped. Note that the watchpoint location does not change when the value of **p** changes.
- `dwatch {*p}` Performs the same action as the previous example. Because the argument to **dwatch** contains a space, Tcl requires that you place the argument within braces.
- `dfocus {p2 p3} wa *p` Watches the address pointed to by **p** 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 *p` Watches the address pointed to by **p** in processes 2, 3, and 4. The **-p** option indicates that only the process triggering the watchpoint is stopped.
- `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 into it.
```

```
wa my_vbl -e {if (my_vbl == 0x11ffff38) $stop;}
    Performs the same function as the previous example. Note
    that this tests the variable directly rather than by using
    $newval.
```

## 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. The examples that follow show many of the kinds of elements that this command can display.

**NOTE** 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:*

You may find the following alias useful:

Alias	Definition	Meaning
<b>wh</b>	<b>{dwhat}</b>	Determines what a name refers to

*Examples:*

These 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 l.l:
```

```
Name: timeout; Type: struct timeval; Size: 8
bytes; Addr: 0x11ffffc0
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           20     float field3_float;
                21     double field3_double;
                22 en_check en1;
                23
                24 };
                25
                26 main ()
                27 {
                28     en_check vbl;
                29     check_struct s_vbl;
                30 >  vbl = big;
                31     s_vbl.field2_char = 3;
                32     return (vbl + s_vbl.field2_char);
                33 }

p vbl           vbl = big (0)
wh vbl         In thread 2.3:

                 Name: vbl; Type: enum en_check; \
                 Size: 4 bytes; Addr: Register 01
                 Scope: #check_structs.cxx#main \
                 (Scope class: Any)
                 Address class: register_var (Register \
                 variable)

wh en_check     In process 2:

                 Type name: enum en_check; Size: 4 bytes; \
                 Category: Enumeration
                 Enumerated values:
                 big      = 0
                 little   = 1
                 fat      = 2
                 thin     = 3

```

```

p s_vbl          s_vbl = {
                  field1_int = 0x800164dc (-2147392292)
                  field2_char = '\377' (0xff, or -1)
                  field2_chars = "\003"
                  <padding> = '\000' (0x00, or 0)
                  field3_int = 0xc0006140 (-1073716928)
                  field2_uchar = '\377' (0xff, or 255)
                  <padding> = '\003' (0x03, or 3)
                  <padding> = '\000' (0x00, or 0)
                  <padding> = '\000' (0x00, or 0)

                  field_sub = {
                      field1_int = 0xc0002980 (-1073731200)
                      <padding> = '\377' (0xff, or -1)
                      <padding> = '\003' (0x03, or 3)
                      <padding> = '\000' (0x00, or 0)
                      <padding> = '\000' (0x00, or 0)
                      field2_long = 0x0000000000000000 (0)
                      ...
                  }
                }

wh s_vbl          In thread 2.3:

                  Name: s_vbl; Type: struct check_struct; \
                  Size: 80 bytes; Addr: 0x11ffff240
                  Scope: #check_structs.cxx#main \
                  (Scope class: Any)
                  Address class: auto_var (Local variable)

wh check_struct  In process 2:

                  Type name: struct check_struct; \
                  Size: 80 bytes; Category: Structure
                  Fields in type:
                  {
                  field1_int      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_struct(320 bits){

```

dwhat

```
        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 locations in the call stack

*Format:*

**dwhere** [ *num-levels* ] [ **-a** ]

*Arguments:*

- 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.
- a**                                      Displays argument names and values in addition to program location information. If you omit this option, arguments are not shown.

*Description:*

The **dwhere** command prints the current execution locations and the call stacks—or sequences of procedure calls—which led to that point. Information is shown 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. Showing all levels 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 will display when you don't use the *num-levels* argument.

Output is generated for each thread in the target focus.

*Command alias:*

You may find the following alias useful:

Alias	Definition	Meaning
<b>w</b>	{ <b>dwhere</b> }	Displays the current location

*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.
- `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 procedures.
- `f p1.< w 5` Displays the most recent five levels of the call stacks for all threads involved in process 1. If the depth of any call stack is less than five levels, all of its levels are shown.
- This command is a slightly more complicated way of saying `f p1 w 5` because specifying a process width tells `dwhere` to ignore the thread indicator.

## 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.

Note that 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:*

You may find the following aliases useful:

Alias	Definition	Meaning
wof	{ <b>dworker false</b> }	Removes the focus thread from the workers group
wot	{ <b>dworker true</b> }	Inserts the focus thread into the workers group

## exit

Terminates the debugging session

*Format:*

**exit** [ **-force** ]

*Arguments:*

**-force**                      Tells the CLI that TotalView should exit without asking permission.

*Description:*

The **exit** command terminates the TotalView session.

After executing this command, the CLI asks if it is all right to exit. If you answer yes, TotalView exits. If you had entered the CLI from the TotalView GUI, this command also closes the GUI window.

**NOTE** Press Ctrl+D to exit from the CLI window without exiting from TotalView.

Any processes and threads that were created by the CLI are destroyed. Any processes that existed prior to the debugging session (that is, were attached by the CLI as part of a **dattach** operation) are detached and left executing.

The **exit** and **quit** commands are interchangeable; they both do exactly the same thing.

*Examples:*

**exit**                              Exits from the CLI, leaving any "attached" processes running.

# help

Displays help information

*Format:*

**help** [ *topic* ]

*Arguments:*

*topic*                      The topic or command for which the CLI prints information.

*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 more than one screen of data would be displayed, the CLI fills the screen with data and then displays a *more* prompt. You can then press Enter to see more data or enter **q** to return to the CLI prompt.

After 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's 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:*

You may find the following aliases useful:

Alias	Definition	Meaning
he	{help}	Displays help information

*Examples:*

help help                      Prints information describing 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 **quit** command terminates the CLI session.

The **exit** command terminates the TotalView session.

After executing the **quit** command, the CLI asks if it is all right to exit. If you answer yes, TotalView exits. If you had entered the CLI from the TotalView GUI, this command also closes the GUI window.

**NOTE** Enter Ctrl+D to exit from the CLI window without exiting from TotalView.

Any processes and threads that were created by the CLI are destroyed. Any processes that existed prior to the debugging session (that is, were attached by the CLI as part of a **dattach** operation) are detached and left executing.

The **quit** and **exit** commands are interchangeable; they both do exactly the same thing.

*Examples:*

`quit` Exits the CLI, leaving any "attached" processes running (in the run-time environment).

# 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 *stty-args*, the CLI displays information describing your current settings.

The output from this command is returned as a string.

### Examples:

**stty**

Prints information about your terminal settings. The information printed 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** option is not available in all environments.)

## unalias

Removes a previously defined alias

### Format:

Removes an alias

```
unalias alias-name
```

Removes all aliases

```
unalias -all
```

### Arguments:

*alias-name* The name of the alias being deleted.

**-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 now undefined and can no longer be used. If **step2** was included as part of the definition of another command, that command will no longer work correctly. However, the CLI will only display an error message when you try to execute the alias that contains this removed alias.

```
unalias -all
```

Removes all aliases.

## Chapter 3

# CLI Namespace Commands

This chapter contains detailed descriptions of CLI namespace commands.

## Command Overview

This section lists all of the CLI namespace commands. 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.
- *thread*: Accesses and sets thread properties.
- *type*: Accesses and sets data type properties.

The following functions are discussed in the CREATING TYPE TRANSFORMATIONS GUIDE:

- *scope*
- *symbol*
- *type\_transformation*

## Helper Functions

The following functions, all within the **TV::** namespace, are most often used in scripts:

- *dec2hex*: Converts a decimal number into hexadecimal format.
- *errorCodes*: Returns or raises TotalView error information.
- *hex2dec*: Converts a hexadecimal number into decimal format.
- *respond*: Sends a response to a command.
- *source\_process\_startup*: "Sources" a **.tvd** file when a process is loaded.

## actionpoint

### Sets and gets action point properties

#### Format:

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

#### Arguments:

<i>action</i>	The action to perform, as follows:
<b>commands</b>	Displays the subcommands that you can use. The CLI responds by displaying the four subcommands shown here. No other arguments are used with this subcommand.
<b>get</b>	Retrieves the values of one or more action point properties. The <i>other-args</i> 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 property names you entered.  If you use the <b>-all</b> option as the <i>object-id</i> , the CLI returns a list containing one (sublist) element for each object.
<b>properties</b>	Lists the action point properties that TotalView can access. No other arguments are used with this subcommand.
<b>set</b>	Sets the values of one or more properties. The <i>other-args</i> argument contains property name and value pairs.
<i>object-id</i>	An identifier for the action point.
<i>other-args</i>	Are arguments that the <b>get</b> and <b>set</b> actions use.

#### Description:

The **TV::actionpoint** command lets you examine and set action point properties and states. These states and properties are:

<b>address</b>	The address of the action point.
<b>enabled</b>	A value (either 1 or 0) indicating if the action point is enabled. A value of 1 means enabled. (settable)
<b>expression</b>	The expression to be executed at an action point. (settable)
<b>id</b>	The ID of the action point.
<b>language</b>	The language in which the action point expression is written.
<b>length</b>	The length in bytes of a watched area. This property is only valid for watchpoints. (settable)
<b>line</b>	The source line at which the action point is set. This is not valid for watchpoints.

## actionpoint

<b>satisfaction_group</b>	The group that must arrive at a barrier for the barrier to be <i>satisfied</i> . (settable)
<b>share</b>	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)
<b>stop_when_done</b>	Indicates what is stopped when a barrier is satisfied (in addition to the satisfaction set). Values are <b>process</b> , <b>group</b> , or <b>none</b> . (settable)
<b>stop_when_hit</b>	Indicates what is stopped when an action point is hit (in addition to the thread that hit the action point). Values are <b>process</b> , <b>group</b> , or <b>none</b> . (settable)
<b>type</b>	The object's type. See <b>type_values</b> for a list of possible types.
<b>type_values</b>	Lists values that can be assigned to the <b>type</b> property: <b>break</b> , <b>eval</b> , <b>process_barrier</b> , <b>thread_barrier</b> , and <b>watch</b> .

*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.

*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** on the host system.

## 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:

- |                      |  |
|----------------------|--|
| <i>number_or_tag</i> | Enters an error code mnemonic tag or its numeric value.  |
| <b>-raise</b>        | Raises the corresponding error. If you append a <i>message</i> , TotalView returns this string. Otherwise, TotalView uses the human-readable string for the error. |
| <i>message</i>       | 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, **errorCode** is set to a list whose format is:

**TOTALVIEW** *error-code subcodes... string*

The contents of this lists are as follows:

- The first list element is always **TOTALVIEW**.
- The second is always the error code.
- *subcodes* are not used at this time.
- The last 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 TotalView-style errors from a script.

*Examples:*

```
foreach e [TV::errorCodes] {  
    puts [eval format {"%20s %2d %s"} [TV::errorCodes $e]]  
}
```

Displays a list of all TotalView error codes.

**expr**Manipulates values created by `dprint --nowait`*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 specify **--done**, the CLI deletes the data for all completed expressions; that is, those where **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 as an *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 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, as just discussed.*Description:*The **TV::expr** command, in addition to showing you command information, returns and deletes values returned by a **dprint --nowait** command. The properties that you can use for this command are:**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.

<b>id</b>	The <i>susp-eval-id</i> of the object.
<b>initially_suspended_process</b>	<p>A list of dpid's 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:</p> <pre>dfocus p dfocus [TV::expr get <i>susp-eval-id</i> \   initially_suspended_processes] await</pre>
<b>result</b>	<p>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:</p> <pre>d1.&lt;&gt; dfocus {1.1 2.1} TV::expr get <i>susp-eval-id</i> result {{1.1 2} {2.1 3}}</pre> <pre>d1.&lt;&gt;</pre> <p>The result of expression <i>susp-eval-id</i> in thread 1.1 is 2 and in thread 2.1 is 3:</p>
<b>status</b>	<p>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:</p> <p><b>done</b>, <b>suspended</b>, and <b>{error <i>diag</i>}</b></p> <p>For example, if expression <i>susp-eval-id</i> 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 <b>TV::expr</b> is focused on threads 1.1, 2.1, and 3.1:</p> <pre>d1.&lt;&gt; 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}}}</pre> <pre>d1.&lt;&gt;</pre>

## focus\_groups

Returns a list of groups in the current focus

*Format:*

TV::focus\_groups

*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 will be 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 | -group | -process | -thread ]
```

### Arguments:

<code>-all</code>	Changes the default width to <b>all</b> .
<code>-group</code>	Changes the default width to <b>group</b> .
<code>-process</code>	Changes the default width to <b>process</b> .
<code>-thread</code>	Changes the default width to <b>thread</b> .

### 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.
```

## focus\_threads

Returns a list of threads in the current focus

### Format:

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

### Arguments:

<code>-all</code>	Changes the default width to <b>all</b> .
<code>-group</code>	Changes the default width to <b>group</b> .
<code>-process</code>	Changes the default width to <b>process</b> .
<code>-thread</code>	Changes the default width to <b>thread</b> .

### 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**, `TV::focus_threads` 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*

The action to perform, as follows:

**commands**

Displays the subcommands that you can use. The CLI responds by displaying the four subcommands shown here. 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 these values for these properties in a list in the same order as you entered the property names.

If you use the **–all** option as an *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, the operation is carried out 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 group properties and states. These states and properties are:

**count**

The number of members in a group.

**id**

The ID of the object.

**member\_type**

The type of the group's members, either **process** or **thread**.

**member\_type\_values**

Returns a list of all possible values for the **member\_type** property.

**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**

Returns a list of all possible values for the **type** property.

group

*Examples:*

```
TV::group get 1 count
```

Returns the number of objects in group 1.

## hex2dec

Converts to decimal

*Format:*

**TV::hex2dec** *number*

*Arguments:*

*number*                      A hexadecimal number.

*Description:*

The **TV::hex2dec** Converts a hexadecimal number into decimal. You can type **0x** before this value. The CLI correctly manipulates 64-bit values, regardless of the size of a **long**.

## image

## Sets and gets image properties

### MiniContents:

Description:

Examples:

### Format:

TV::image *action* [ *object-id* ] [ *other-args* ]

### Arguments:

*action*

The action to perform, as follows:

**add**

Adds an object to an image. The *object-id* argument is required; *other-args* is followed by two arguments. The first is the object class of the image. At this release, this type can only be **type\_transformation**. The second is the prototype's ID. (This is illustrated in the *Examples* section.)

**commands**

Displays the subcommands that you can use. The CLI responds by displaying the six subcommands shown here. Do not use additional arguments with this subcommand.

**get**

Gets the values of one or more image properties. The *other-args* argument can include one or more property names. The CLI returns these values for these properties in a list in the same order as you entered the property names.

If you use the **-all** option as an *object-id*, the CLI returns a list containing one (sublist) element for each object.

**lookup**

Looks up an object in the image and returns a list of IDs of matching objects. The *object-id* argument is required; *other-args* contains two arguments. The first is the object class of the image and the second is the name of the object. For example:

```
TV::image lookup 1|15 type "int *"
```

If no matching objects are found, the CLI returns an empty list. You can obtain a list of class objects by using the **lookup\_keys** property.

**lookup\_keys**

A list containing the object classes that can be used in a *by name* lookup; for example:

```
TV::image lookup 1|20 types foo
```

Currently, the only value returned is **{types}**.

<b>properties</b>	Displays the properties that the CLI can access. Do not use additional arguments with this option.
<b>set</b>	Sets the values of one or more image properties. The <i>other-args</i> argument contains property name and value pairs. type_transformations
<i>object-id</i>	The ID of an image. An image ID is two integers that identify the base executable and an associated DLL. You can obtain a list of all image IDs by using the following command:  TV::image get -all id  If you use the <b>-all</b> option, TotalView carries out this operation on all images in the current focus.
<i>other-args</i>	Arguments required by the <b>get</b> and <b>set</b> subcommands.

**Description:**

The **TV::image** command lets you examine and set the image properties and states. "Image" refers to all the programs, libraries, and other components that make up your executable. These states and properties are:

<b>data_size</b>	The amount of memory used to store initialized data.
<b>dpids</b>	IDs of the process associated with a thread
<b>id</b>	The ID of the object.
<b>is_dll</b>	A true/false value where <b>1</b> indicates the image is a shared library and <b>0</b> if it is an executable.
<b>name</b>	The name of the image.
<b>type_transformations</b>	A list of all of the type transformations that apply to an image.
<b>text_size</b>	The amount of memory used to store your program's machine code instructions. The "text segment" is sometimes called the "code segment."

**Examples:**

```
TV::image lookup 1|15 type "int *"
```

Finds the type identifiers for the **int \*** type in image **1|15**. The result might be:

```
1|25 1|76
```

## image

There can be more than one type with the same name in an image since many debugging formats provide separate type definitions in each source file.

```
foreach i [TV::image get -all id] {  
  puts [format "%40s; %s" [TV::image get $i name] $i]}  
  Lists all current images along with their IDs.
```

## process

## Sets and gets process properties

### Format:

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

### Arguments:

<i>action</i>	The action to perform, as follows:
<b>commands</b>	Displays the subcommands that you can use. The CLI responds by displaying the four subcommands shown here. Do not use other arguments with this subcommand.
<b>get</b>	Gets the values of one or more process properties. The <i>other-args</i> 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 <b>–all</b> option as an <i>object-id</i> , the CLI returns a list containing one (sublist) element for each object.
<b>properties</b>	Displays the properties that the CLI can access. Do not use other arguments with this subcommand.
<b>set</b>	Sets the values of one or more properties. The <i>other-args</i> arguments contains pairs of property names and values.
<i>object-id</i>	An identifier for a process. For example, <b>1</b> represents process 1. If you use the <b>–all</b> option, the subcommand is carried out on all objects of this class in the current focus.
<i>other-args</i>	Arguments required by the <b>get</b> and <b>set</b> subcommands.

### Description:

The **TV::process** command lets you examine and set process properties and states. These states and properties are:

<b>clusterid</b>	The ID of the cluster containing a 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 <b>0</b> (zero).
<b>duid</b>	The internal unique ID associated with an object.
<b>executable</b>	The program's name.
<b>heap_size</b>	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 <b>malloc()</b> allocate space on the heap while <b>free()</b> releases it.
<b>held</b>	A value (either <b>1</b> or <b>0</b> ) indicating if the process is held; <b>1</b> means that the process is held. (settable)
<b>hostname</b>	The name of the process's host system.
<b>id</b>	The process ID.
<b>image_ids</b>	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.
<b>nodeid</b>	The ID of the node upon which the process is running. The ID of each processor node is unique within a cluster.
<b>stack_size</b>	The amount of memory used by the currently executing block or routines and all the blocks routines that have invoked it. For example, if your main routine invokes function <b>foo()</b> , 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 <b>foo()</b> . If <b>foo()</b> invokes <b>bar()</b> , the stack contains three frames. When <b>foo()</b> finishes executing, the stack only contains one frame.
<b>stack_vm_size</b>	<p>The logical size of the stack is the difference between the current value of the stack pointer and 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.</p> <p>The value here is this difference in size.</p>
<b>state</b>	Current state of the process. See <b>state_values</b> for a list of states.
<b>state_values</b>	Lists all possible values for the <b>state</b> property. These values can be <b>break</b> , <b>error</b> , <b>exited</b> , <b>running</b> , <b>stopped</b> , or <b>watch</b> .
<b>syspid</b>	The system process ID.

<b>text_size</b>	The amount of memory used to store your program's machine code instructions. The "text segment" is sometimes called the "code segment."
<b>threadcount</b>	The number of threads in the process.
<b>threads</b>	A list of threads in the process.
<b>vm_size</b>	The sum of the sizes of the mappings in the process's address space.

*Examples:*

```
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

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}}
```

```
foreach i [TV::process get 1 image_ids] {
  puts [TV::image get $i name]}
Prints the name of the executable and all shared libraries cur-
rently linked into the focus process. For example, the output
of this command might be:
arraysAIX
/usr/lib/libxlf90.a
/usr/lib/libcrypt.a
/usr/lib/libc.a
```

## respond

Provides responses to commands

### Format:

```
TV::respond response command
```

### Arguments:

*response*                      The response to one or more commands. If you include more than one response, separate the responses with newline characters.

*command*                      One or more commands that the CLI will execute.

### 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 will place this information within 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 it. If more than one question is asked and *response* is used up, **TV::respond** starts over at the beginning of the *response* string. If *response* does not end with a newline, **TV::respond** appends one.

Do not use this command to suppress the **MORE** prompt in macros. You should instead use the following command:

```
dset LINES_PER_SCREEN 0
```

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

**NOTE** If you are using the TotalView GUI and the CLI at the same time, your CLI command may 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.

```
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. Neither of these two uses is recommended. Instead, you can use **exit -force**.

## scope

Does foobar

*Format:*

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

*Description:*

The **TV::scope** command lets you examine and set the scope properties and states. This command is explained in the CREATING TYPE TRANSFORMATION GUIDE.

## source\_process\_startup “Sources” 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 sources it.

## symbol

Does foobar

*Format:*

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

*Description:*

The **TV::symbol** command lets you examine and set the symbol properties and states. This command is explained in the CREATING TYPE TRANSFORMATION GUIDE.

## thread

### Gets and sets thread properties

*MiniContents:*

*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 the four subcommands shown here. 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 property names you entered.

If you use the **–all** option as an *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 thread properties and states. These states and properties are:

**continuation\_sig**

The signal that should be passed to a thread the next time it runs. On some systems, the thread receiving the signal may not always be the one for which this property was set.

**dpid**

The ID of the process associated with a thread.

**duid**

The internal unique ID associated with the thread.

**held**

A 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.

<b>manager</b>	A value (either <b>1</b> or <b>0</b> ) indicating if this is a system manger thread; <b>1</b> means that it is.
<b>pc</b>	Current PC at which the target is executing. (settable)
<b>sp</b>	The value of the stack pointer.
<b>state</b>	Current state of the target. See <b>state_values</b> for a list of states.
<b>state_values</b>	A list of values for the <b>state</b> property. These values are <b>break</b> , <b>error</b> , <b>exited</b> , <b>running</b> , <b>stopped</b> , and <b>watch</b> .
<b>systid</b>	The system thread ID.

*Examples:*

```
f p3 TV::thread get -all id
Returns a list of thread IDs for process 3. For example:
1.1 1.2 1.4
```

## type

## Gets and sets type properties

### Format:

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

### Arguments:

<i>action</i>	The action to perform, as follows:
<b>commands</b>	Displays the subcommands that you can use. The CLI responds by displaying the four subcommands shown here. Do not use other arguments with this option.
<b>get</b>	Gets the values of one or more type properties. The <i>other-args</i> 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 property names you entered.  If you use the <b>–all</b> option as an <i>object-id</i> , the CLI returns a list containing one (sublist) element for each object.
<b>properties</b>	Lists a type’s properties. Do not use other arguments with this option.
<b>set</b>	Sets the values of one or more type properties. The <i>other-args</i> argument contains paired property names and values.
<i>object-id</i>	An identifier for an object. For example, <b>1</b> represents process 1, and <b>1.1</b> represents thread 1 in process 1. If you use the <b>–all</b> option, the operation is carried out on all objects of this class in the current focus.
<i>other-args</i>	Arguments required by the <b>get</b> and <b>set</b> subcommands.

### Description:

The **TV::type** command lets you examine and set the type properties and states. These states and properties are:

<b>enum_values</b>	For an enumerated type, a list of <b>{name value}</b> pairs giving the definition of the enumeration. If you apply this to a non-enumerated type, the CLI returns an empty list.
<b>id</b>	The ID of the object.
<b>image_id</b>	The ID of the image in which this type is defined.
<b>language</b>	The language of the type.
<b>length</b>	The length of the type.
<b>name</b>	The name of the type; for example, <b>class foo</b> .

<b>prototype</b>	The ID for the prototype. If the object is not prototyped, the returned value is <code>{}</code> .
<b>rank</b>	(array types only) The rank of the array.
<b>struct_fields</b>	( <b>class/struct/union</b> types only). A list of lists giving the description of all the type's fields. Each sublist contains the following fields: <pre>{ name type_id addressing properties }</pre> where: <i>name</i> is the name of the field. <i>type_id</i> is simply the <i>type_id</i> of the field. <i>addressing</i> contains additional addressing information that points to the base of the field. <i>properties</i> contains an additional list of properties in the following format: <b>"[virtual] [public private protected] base class"</b> If no properties apply, this string is null. If you use <b>get struct_fields</b> for a type that is not a <b>class</b> , <b>struct</b> , or a <b>union</b> , the CLI returns an empty list.
<b>target</b>	For an array or pointer type, returns the ID of the array member or target of the pointer. If this is not applied to one of these types, the CLI returns an empty list.
<b>type</b>	Returns a string describing this type. For example, <b>signed integer</b> .
<b>type_values</b>	Returns all possible values for the <b>type</b> 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 may look something like:

```
4 1|12
```

The following example uses the **TV::type properties** command to obtain the list of properties:

type

```
d1.<> \  
proc print_type {id} {  
    foreach p [TV::type properties] {  
        puts [format "%13s %s" $p [TV::type get $id $p]]  
    }  
}  
d1.<> 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}...  
  
d1.<>
```

## type\_transformation

Creates type transformations  
and examine properties

*Format:*

**TV::type\_transformation** *action* [ *object-id* ] [ *other-args* ]

*Description:*

The **TV::type\_transformation** command lets you examine and set the scope properties and states. This command is explained in the CREATING TYPE TRANSFORMATION GUIDE.



## Chapter 4

# TotalView Variables

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::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 **ARGS(*dpid*)** 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:

**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)

**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:

- |                 |   |
|-----------------|---|
| <b>control</b>  | The group of all processes in a program   |
| <b>lockstep</b> | A group of threads that share the same PC |
| <b>process</b>  | A user-created process group              |

<b>share</b>	The group of processes in one program that share the same executable image
<b>thread</b>	A user-created thread group
<b>workers</b>	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:

<b>0</b>	No <i>more</i> processing occurs, and the printing does not stop when the screen fills with data.
<b>NONE</b>	This is a synonym for 0.
<b>AUTO</b>	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 <b>tty</b> value to 0. If <b>AUTO</b> 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

**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**

**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**

**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:

<b>INFO</b>	Prints errors, warnings, and informational messages. Informational messages include data on dynamic libraries and symbols.
<b>WARNING</b>	Only print errors and warnings.
<b>ERROR</b>	Only print error messages.
<b>SILENT</b>	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::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 dereferenced C pointers to array string** 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 **Back** command to see the undereferenced value when performing a **Dive in All** operation.

- yes** You will be able to use the **Back** 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 **Back** command to see the undereferenced value when performing a **Dive in All** operation.
- yes** You will be able to use the **Back** 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 **Back** command to see the undereferenced value.
- yes** You will be able to use the **Back** 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 **Back** command to see the undereferenced value.
- yes** You will be able to use the **Back** 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 **Back** command to see the undereferenced value.
- yes** You will be able to use the **Back** 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 **Back** command to see the undereferenced value.
- yes** You will be able to use the **Back** 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\_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.

**none:** 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.

**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 **BARRIER\_STOP\_WHEN\_DONE** variable.

*Permitted Values:* **group, process, or thread**  
*Default:* **group**

**TV::bulk\_launch\_base\_timeout:** Defines the base timeout period used when TotalView executes a bulk server launch.

*Permitted Values:* A number from 1 to 3600 (1 hour)  
*Default:* 20

**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 219.

*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 219.

*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 219.

*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 219.

*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 219.

*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 219.

*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 219.

*Permitted Values:* A string

*Default:* None

**TV::c\_type\_strings:** When this is set to **true**, TotalView uses C type string extensions when it displays character arrays. When set to **false**, TotalView instead uses its string type extensions.

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

*Default:* **true**

**TV::comline\_patch\_area\_base:** Allocates the patch space dynamically at the given *address*. See “Allocating Patch Space for Compiled Expressions” in Chapter 14 of the TOTALVIEW USERS GUIDE.

*Permitted Values:* A hexadecimal value indicating space accessible to TotalView.

*Default:* 0xfffffffffffffff

**TV::comline\_path\_area\_length:** Sets the length of the dynamically allocated patch space to the specified *length*. See “Allocating Patch Space for Compiled Expressions” in Chapter 14 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 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**

**TV::compiler\_expressions**: When this variable is set to **true**, TotalView enables compiled expressions. If this is set to **false**, TotalView interprets your expression.

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

*Default:* HP Alpha and IBM AIX: **true**

SGI IRIX: **false**

Not settable on other platforms

**TV::compiler\_vars**: (HP Alpha, HP, and SGI only) When this 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::copyright\_string:** This is a read-only string containing the copyright information displayed when you start the CLI and TotalView.

**TV::current\_cplusplus\_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\_cplusplus\_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 XL C/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.

**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.

You can display floating point information using **dec**, **hex**, and **oct** formats. You can display integers using **auto**, **dec**, and **sci** formats.

*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\_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\_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\_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::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::display\_assembler\_symbolically:** When this variable is set to **true**, TotalView displays assembler locations as **label+offset**. When it is set to **false**, these locations are displayed as hexadecimal addresses.

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

*Default:* **false**

**TV::dll\_ignore\_prefix:** Contains a colon-separated list of prefixes that indicates which library files TotalView should not load.

TotalView will not ask you if you would like to stop a process:

- If you also set the **TV::ask\_on\_dlopen** variable to **true**.
- If the suffix of the library being loaded does *not* match a suffix contained in the **TV::dll\_stop\_suffix** variable.
- If one or more of the prefixes in this list match the name of the library being loaded.

*Permitted Values:* A list of path names, each item of which is separated from another by a colon

*Default:* `/lib:/usr/lib:/usr/lpp:/usr/ccs/lib:/usr/dt/lib:/tmp/`

**TV::dll\_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::dynamic:** When this is set to **true**, TotalView loads symbols from shared libraries. This variable is available on all platforms supported by Etnus. (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::force\_default\_cplusplus\_demangler:** When this is set to **true**, TotalView uses the demangler set in the **TV::current\_cplusplus\_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 defi-

nitions 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::ignore\_control\_c:** When this is set to **true**, TotalView ignores Ctrl+C characters. This prevents you from inadvertently terminating the TotalView process. You would set this option to **false** when your program catches the Ctrl+C (**SIGINT**) signal.

*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 image. 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::propagate\_prototypes**

**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::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::library\_cache\_directory:** Specifies the directory into which TotalView writes library cache information.

*Permitted Values:* A string indicating a path.  
*Default:* **\$USERNAME/.totalview/lib\_cache**

**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:* **tvdsvr**

**TV::local\_server\_launch\_string:** (Sun only) If TotalView will not be using the server contained in the same working directory as the TotalView executable, the contents of this string indicate the shell command that TotalView uses to launch this alternate server. For information on this launch string, see “Replacement Characters” on page 219.

*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::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 **.tvdrc** file:

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

*Permitted Values:* A string representing a Tcl or CLI command

*Default:* **Null**

**TV::parallel:** When this is 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.
- **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\_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:* One of the following values: **alpha**, **hpux11-hppa**, **irix6-mips**, **linux-x86**, **linux-alpha**, **rs6000**, and **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**

**TV::save\_window\_pipe\_or\_filename:** Names the file to which TotalView writes or pipes the contents of the current window or pane when you select the **File > Save Pane** command.

*Permitted Values:* A string naming a file or pipe

*Default:* None, until something is saved. Afterward, the saved string is the default.

**TV::search\_case\_sensitive:** When this is set to **true**, text searches only succeed if a string exists whose case exactly matches what you enter in the **Edit > 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**.

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

*Default:* **false**

**TV::server\_launch\_enabled:** 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.

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

*Default:* **true**

**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 219.

*Permitted Values:* A string

*Default:* `{%C %R -n "tvdsvr -working_directory %D -callback %L -set_pw %P -verbosity %V"}`

**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::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 *signal\_list* is a list of one or more signal specifiers, separated by commas:

*signal\_specifier[,signal\_specifier] ...*

A *signal\_specifier* can be a signal name (such as **SIGSEGV**), a signal number (such as **11**), or a star (\*), which specifies all signals. We recommend using the signal name rather than the number because number assignments vary across UNIX versions.

The following rules apply when you are specifying an *action\_list*:

- If you specify an action for a signal in an *action\_list*, TotalView changes the default action for that signal.
- If you do not specify a signal in the *action\_list*, TotalView does not change its default action for the signal.
- If you specify a signal that does not exist for the platform, TotalView ignores it.
- If you specify an action for a signal twice, TotalView uses the last action specified. In other words, TotalView applies the actions from left to right.

If you need to revert the settings for signal handling to TotalView's built-in defaults, use the **Defaults** button in the **File > Signals** Dialog Box.

For example, to set the default action for the **SIGTERM** signal to *Resend*, you specify the following action list:

```
{Resend=SIGTERM}
```

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

```
{Stop=* Error=SIGSEGV,SIGBUS Resend=SIGHUP,SIGTERM}
```

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

- 1 Sets the action for all signals to *Stop*.
- 2 Changes the action for **SIGSEGV** and **SIGBUS** from *Stop* to *Error*.
- 3 Changes the action for **SIGHUP** and **SIGTERM** from *Stop* to *Resend*.

*Permitted Values:* A list of signals, as was just described

*Default:* This differs from platform to platform; type **dset TV::signal\_handling\_mode** to see what a platform's default values are

**TV::source\_pane\_tab\_width:** Sets the width of the tab character that is displayed in the Process Window's Source Pane. You may want to set this value to the same value as you use in your text editor.

*Permitted Values:* An integer

*Default:* 8

**TV::spell\_correction:** When you use the **View > Lookup Function** or **View > Lookup Variable** commands in the Process Window or edit a type string in a Variable Window, the debugger checks the spelling of your entries. By default (**verbose**), the debugger displays a dialog box before it corrects spelling. You can set this resource to **brief** to run the spelling corrector silently. (TotalView makes the spelling correction without displaying it in a dialog box first.) You can also set this resource to **none** to disable the spelling corrector.

*Permitted Values:* **verbose**, **brief**, or **none**

*Default:* **verbose**

**TV::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 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:c:include bmap:c:include f:f77 F:f77 f90:f9x F90:f9x hpf:hpfp HPF:hpfp cxx:c++ cpp:c++ cc:c++ 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++:include p:pascal P:pascal pas:pascal PAS:pascal}**

**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 routine.

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

*Default:* **true**

**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:* **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 TotalView 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 TotalView 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::GUI:: Namespace

**NOTE** The variables in this section only have meaning (and in some cases, a value) when you are displaying TotalView's 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\_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:* 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\_position:** 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 **Enlightment** 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::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 windows 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 windows **x** and **y** coordinates and the windows 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\_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 windows 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 windows **x** and **y** coordinates and the windows **width** and **height**.

*Default:* {0 0 0 0}

**TV::GUI::geometry\_ptset:** Specifies the position at which TotalView displays the Tools > P/T Set 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\_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::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::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::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



## Chapter 5

# Default Arena Widths

This chapter lists all CLI commands and their default arena widths. Commands in the TV:: namespace and commands that are not debugger related (for example, alias and help) are not shown as either focus is not relevant or is obvious (for example, focus\_groups).

Command	Default Arena Width
dactions	process
dassign	thread; if the current width is "process", <b>dassign</b> acts on each thread in the process
dattach	—
dbarrier	Obtains focus width from the setting of the <b>SHARE_ACTION_POINT</b> variable; if <b>true</b> , the default is "group;" if <b>false</b> , the default is "process"
dbreak	Obtains focus width from the setting of the <b>SHARE_ACTION_POINT</b> variable; if <b>true</b> , the default is "group;" if <b>false</b> , the default is "process"
dcache	—
dcheckpoint	process
dcont	process
ddelete	process
ddetach	process
ddisable	process
ddown	thread; if the current width is "process," <b>ddown</b> acts on each thread in the process
denable	process

Command	Default Arena Width
<code>dflush</code>	thread
<code>dfocus</code>	—
<code>dgo</code>	process
<code>dgroups</code>	The <code>-list</code> option ignores the focus; other options use group width to find the groups being operated on and thread width to find the operands
<code>dhalt</code>	process
<code>dhold</code>	depends on the <code>-thread</code> or <code>-process</code> option
<code>dkill</code>	process; note that killing the primary process for a control group always kills all of its slaves
<code>dlappend</code>	—
<code>dlist</code>	thread; if the current width is "process," <code>dlist</code> iterates over all threads in the process
<code>dload</code>	—
<code>dmstat</code>	process
<code>dnext</code>	process
<code>dnexti</code>	process
<code>dout</code>	process
<code>dprint</code>	thread; if the current width is "process", <code>dprint</code> acts on each thread in the process
<code>dptsets</code>	—
<code>drerun</code>	process
<code>drestart</code>	—
<code>drun</code>	process
<code>dset</code>	—
<code>dstatus</code>	process
<code>dstep</code>	process
<code>dstepi</code>	process
<code>dunhold</code>	depends on the <code>-thread</code> or <code>-process</code> option
<code>dunset</code>	—

Command	Default Arena Width
<b>duntil</b>	process
<b>dup</b>	thread; if the current width is "process, <b>dup</b> acts on each thread in the process
<b>dwait</b>	process
<b>dwatch</b>	Obtains focus from the <b>SHARE_ACTION_POINT</b> variable's setting <b>true</b> : default to group <b>false</b> : default to process
<b>dwhat</b>	thread; if the current width is "process," <b>dwhat</b> acts on each thread in the process
<b>dwhere</b>	thread; if the current width is "process," <b>dwhere</b> acts on each thread in the process
<b>dworker</b>	thread





## Part II: Running TotalView

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

### Chapter 6: TotalView Command Syntax

TotalView contains a great number of command-line options. Many of these options allow you to override TotalView's default behavior or a behavior that you've set in a preference or a startup file.

In previous releases, using options was the best way to set TotalView's behavior. Beginning with Release 6.0, you are better served by setting a preference or a CLI variable.

### Chapter 7: TotalView Debugger Server (tvdsvr) Command Syntax

This chapter describes how you modify the behavior of the `tvdsvr`. These options are most often used if a problem occurs in launching the server or if you have some very specialized need. In most cases, you can ignore the information in this chapter.



## Chapter 6

# TotalView Command Syntax

This chapter describes the syntax of the `totalview` command. Topics in this chapter are:

- Syntax
- Options

## Syntax

**Synopsis:** `totalview [ filename [ corefile ] ] [ options ]`

**Description:** The TotalView debugger 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.

### Arguments:

<i>filename</i>	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 <code>-g</code> compiler option. Any multiprocess programs that call <code>fork()</code> , <code>vfork()</code> , or <code>execve()</code> should be linked with the <code>dbfork</code> library.
<i>corefile</i>	Specifies the name of a core file. Use this argument in addition to <i>filename</i> when you want to examine a core file with TotalView.

**Using Options :** If you specify mutually exclusive options on the same command line (for example, `-dynamic` and `-no_dynamic`), the last option listed is used.

## 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.
- 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.
- 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*.  
  
Default: All TotalView console output is written to **stderr**.
- demangler=***compiler* Overrides the demangler and mangler TotalView uses by default. The following indicate override options.

<code>-demangler=compaq</code>	HP cxx on Linux (alpha)
<code>-demangler=dec</code>	HP Tru64 C++ or Fortran
<code>-demangler=gnu</code>	GNU C++ on Linux Alpha
<code>-demangler=gnu_dot</code>	GNU C++ on Linux x86
<code>-demangler=gnu_v3</code>	GNU C++ Linux x86
<code>-demangler=hp</code>	HP aCC compiler
<code>-demangler=irix</code>	SGI IRIX C++
<code>-demangler=kai</code>	KAI C++
<code>-demangler=kai3_n</code>	KAI C++ version 3.n
<code>-demangler=kai_4_0</code>	KAI C++
<code>-demangler=spro</code>	SunPro C++ 4.0 or 4.2
<code>-demangler=spro5</code>	SunPro C++ 5.0 or later
<code>-demangler=sun</code>	Sun CFRONT C++
<code>-demangler=xl</code>	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*s 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.

- no\_dpvm** HP Tru64 UNIX *only*: (Default) Disables support for debugging the HP Tru64 UNIX implementation of PVM applications.
- dump\_core** Allows TotalView to dump a core file of itself when an internal error occurs. This is used to help Etnus debug TotalView problems.
- no\_dumpcore** (Default) Does not allow TotalView to dump a core file when it gets an internal error.
- 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.
- 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=sp<sub>ro</sub>\_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. In C++, the 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.
- 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 *file-name.TVD.v3breakpoints* file, providing the file exists.
- nlb** Tells TotalView that it should not automatically load action points from an action points file.
- 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**.
- parallel** (Default) Enables handling of parallel program run-time libraries such as MPI, PE, and UPC.
- no\_parallel** Disables handling of parallel program run-time libraries such as MPI, PE, and UPC. This is useful for debugging parallel programs as if they were single-process programs.
- patch\_area\_base** *address* Allocates the patch space dynamically at the given *address*. See "Allocating Patch Space for Compiled Expressions" in Chapter 14 of the TOTALVIEW USERS GUIDE.

- patch\_area\_length** *length*  
Sets the length of the dynamically allocated patch space to the specified *length*. See “*Allocating Patch Space for Compiled Expressions*” in Chapter 14 of the TOTALVIEW USERS GUIDE.
- pid** *pid*  
Tells TotalView to attach to process *pid* after it starts executing.
- pvm**  
Enables support for debugging the ORNL implementation of Parallel Virtual Machine (PVM) applications.
- no\_pvm**  
(Default) Disables support for debugging the ORNL implementation of 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 Chapter 14 of the TOTALVIEW USERS GUIDE.
- r** *hostname[:portnumber]*  
Same as **-remote**.
- 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 have 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 *signal\_specifier* can be a signal name (such as **SIGSEGV**), a signal number (such as 11), or a star (\*), which specifies all signals. We recommend that you use the signal name rather than the number because number assignments vary across UNIX sessions.

The following rules apply when you are specifying an *action\_list*:

- (1) If you specify an action for a signal in an *action\_list*, TotalView changes the default action for that signal.
- (2) If you do not specify a signal in the *action\_list*, TotalView does not change its default action for the signal.
- (3) If you specify a signal that does not exist for the platform, TotalView ignores it.
- (4) If you specify an action for a signal more than once, TotalView uses the last action specified.

If you need to revert the settings for signal handling to TotalView's built-in defaults, use the **Defaults** button in the **File > Signals** dialog box.

For example, here's how to set the default action for the **SIGTERM** signal to resend:

"Resend=SIGTERM"

—  
-shm "action\_list"

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**.

-tvhome *pathname*

The directory from which TotalView reads preferences and other related information and the directory to which it writes this 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.

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

Default: **info**

## Chapter 7

# TotalView Debugger Server (tvdsvr) Command Syntax

This chapter summarizes the syntax of the TotalView Debugger Server command, `tvdsvr`, which is used for remote debugging. 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

**Synopsis:** `tvdsvr` {`-server` | `-callback` *hostname:port* | `-serial` *device*}  
*[other options]*

**Description:** The `tvdsvr` debugger server 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*.)

—callback *hostname:port*

If you prefer not to automatically launch the server, you can start **tvdsvr** manually and specify the **–server** option. Be sure to note the password that **tvdsvr** prints out with the message:

**pw** = *hexnumhigh:hexnumlow*

TotalView will prompt you for *hexnumhigh:hexnumlow* later. By default, **tvdsvr** automatically generates a password that it uses when establishing connections. If desired, you can set your own password by using the **–set\_pw** option.

To connect to the **tvdsvr** from TotalView, you use the **Fille > 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** *consoleo\_outputfile*  
 Redirects TotalView Debugger Server console output to a file named *console\_outputfile*.  
*Default:* All console output is written to **stderr**.
- 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.  
**NOTE** 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.
- port** *number*  
 Sets the TCP/IP port number on which **tvdsvr** should communicate with **totalview**. If this TCP/IP port number is busy, **tvdsvr** does not select an alternate port number (that is, it communicates with nothing) 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.  
**NOTE** 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

---

**-server**

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: **-search\_port** or **-port**. To stop **tvdsvr** from listening and accepting network connections, you must terminate it by pressing Ctrl+C in the terminal window from which it was started or by using the **kill** command.

**-set\_pw** *hexnumhigh:hexnumlow*

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

```
pw = hexnumhigh:hexnumlow
```

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

**NOTE** If necessary, you can disable password checking by specifying the “**-set\_pw 0:0**” option with the **tvdsvr** command. Disabling password checking is dangerous; it allows anyone to connect to your server and start programs, including shell commands, using your UID. Therefore, we do not recommend disabling password checking.

**-set\_pws** *password-list*

Sets 64-bit passwords. TotalView must supply these passwords when **tvdsvr** establishes the connection with it. The argument to this command is a comma-separated list of passwords that TotalView automatically generates. This option is most often used with a bulk launch.

For more information, see Chapter 4, "Setting Up Remote Debugging Sessions" in the TOTALVIEW USERS GUIDE.

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

*Default: info*

**-working\_directory** *directory*

Makes *directory* the directory to which TotalView will be connected.

Note that the command assumes that the host machine and the target machine mount identical file systems. That is, the path name of the directory to which TotalView is connected must be identical on both the host and target machines.

After performing this operation, the TotalView Debugger Server is started.

## Replacement Characters

When placing a **tvdsvr** command in a **Server Launch** or **Bulk Launch** string (see the **File > Preferences** command within the online Help for more information), you will need to use special replacement characters. When your program needs to launch a remote process, TotalView replaces these command characters with what they represent. Here are the replacement characters:

- %C** 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.
- %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

	<p>in a regular server launch and within a <b>tvdsvr</b> command contained within a temporary file.)</p>
%L	<p>If TotalView is launching one process, this is replaced by the host name and TCP/IP port number (<i>hostname:port</i>) on which TotalView is listening for connections from <b>tvdsvr</b>.</p> <p>If a bulk launch is being performed, TotalView replaces this with a comma-separated list of the host names and TCP/IP port numbers (<i>hostname:port,hostname:port...</i>) on which TotalView is listening for connections from <b>tvdsvr</b>.</p> <p>For more information, see Chapter 4, "Setting Up Remote Debugging Sessions" in the TOTALVIEW USERS GUIDE.</p>
%N	<p>Is replaced by the number of servers that TotalView will launch. This is only used in a bulk server launch command.</p>
%P	<p>If TotalView is launching one process, this is replaced by the password that TotalView automatically generated.</p> <p>If a bulk launch is being performed, TotalView replaces this with a comma-separated list of 64-bit passwords.</p>
%R	<p>Is replaced by the host name of the remote machine specified in the <b>File &gt; New Program</b> command.</p>
%S	<p>If TotalView is launching one process, it replaces this symbol with the port number on the machine upon which the debugger is running.</p> <p>If a bulk server launch is being performed, TotalView replaces this with a comma-separated list of port numbers.</p>
%t1 and %t2	<p>Is replaced by files that TotalView creates containing information it generates. This is only available in a bulk launch.</p> <p>These temporary files have the following structure:</p> <ol style="list-style-type: none"><li>(1) An optional header line containing initialization commands required by your system.</li><li>(2) One line for each host being connected to, containing host-specific information.</li><li>(3) An optional trailer line containing information needed by your system to terminate the temporary file.</li></ol>

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.

The templates for temporary files can also be set using X resources.

**%V**

Is replaced by the current TotalView verbosity setting.





## 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 8: Compilers and Platforms

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 9: 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 different from platform to platform.

### Chapter 10: Architectures

When debugging assembly-level functions, you will need to know how TotalView refers to your machine's registers.



## Chapter 8

# Compilers and Platforms

This chapter describes the compilers and parallel runtime environments used on platforms supported by TotalView. You must refer to the TotalView Release Notes included in the TotalView distribution for information on the specific compiler and runtime environment supported by TotalView.

For information on supported operating systems, please refer to Chapter 9, “*Operating Systems*” on page 237.

Topics in this chapter are:

- Compiling with Debugging Symbols
- Using Exception Data on HP 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.

**NOTE** Please refer to the release notes in your TotalView distribution for the latest information about supported versions of the compilers and parallel runtime environments listed here.

## HP Alpha Running Linux

Table 3 lists the procedures to compile programs on HP Alpha running Linux.

TABLE 3: Compiling with Debugging Symbols on HP Alpha Linux

Compiler	Compiler Command Line
HP Alpha Linux C	<code>ccc -g program.c</code>
HP Alpha Linux Fortran	<code>cfal -g program.f</code>
GCC EGCS C	<code>gcc -g program.c</code>
GCC EGCS C++	<code>g++ -g program.cxx</code>
GCC EGCS Fortran	<code>g77 -g program.f</code>

## HP Tru64 UNIX

Table 4 lists the procedures to compile programs on HP Tru64 UNIX.

TABLE 4: Compiling with Debugging Symbols on HP Tru64 UNIX

Compiler	Compiler Command Line
HP Tru64 UNIX C	<code>cc -g program.c</code>
HP Tru64 UNIX C++	<code>cxx -g program.cxx</code>
HP Tru64 UNIX Fortran 77	<code>f77 -g program.f</code>
HP Tru64 UNIX Fortran 90	<code>f90 -g program.f90</code>
HP Tru64 UPC compiler	<code>upc -g [-fthreads n] program.upc</code>
GCC EGCS C	<code>gcc -g program.c</code>
GCC EGCS C++	<code>g++ -g program.cxx</code>
KAI C	<code>KCC +K0 program.c</code>
KAI C++	<code>KCC +K0 program.cxx</code>
KAI Guide C (OpenMP)	<code>guidec -g +K0 program.c</code>
KAI Guide C++ (OpenMP)	<code>guidec -g +K0 program.cxx</code>
KAI Guide F77 (OpenMP)	<code>guidef77 -g -WG,-cmpto=i program.f</code>

When compiling with KCC for debugging, we recommend that you use the `+K0` option and not the `-g` option. Also, the `-WG,-cmpto=i` option to the `guidef77` command may not be required on all versions because `-g` can imply these options.

## HP-UX

Table 5 lists the procedures to compile programs on HP-UX.

TABLE 5: Compiling with Debugging Symbols on HP-UX

Compiler	Compiler Command Line
HP ANSI C	<code>cc -g program.c</code>
HP C++	<code>aCC -g program.cxx</code>
HP Fortran 90	<code>f90 -g program.f90</code>
KAI C	<code>KCC +K0 program.c</code>
KAI C++	<code>KCC +K0 program.cxx</code>
KAI Guide C (OpenMP)	<code>guidec -g +K0 program.c</code>
KAI Guide C++ (OpenMP)	<code>guidec -g +K0 program.cxx</code>
KAI Guide F77 (OpenMP)	<code>guidef77 -g -WG,-cmpo=i program.f</code>

When compiling with KCC for debugging, we recommend that you use the `+K0` option and not the `-g` option. Also, the `-WG,-cmpo=i` option to the `guidef77` command may not be required on all versions because `-g` can imply these options.

## IBM AIX on RS/6000 Systems

Table 6 lists the procedures to compile programs on IBM RS/6000 systems running AIX.

TABLE 6: Compiling with Debugging Symbols on AIX

Compiler	Compiler Command Line
GCC EGCS C	<code>gcc -g program.c</code>
GCC EGCS C++	<code>g++ -g program.cxx</code>
IBM xlc C	<code>xlc -g program.c</code>
IBM xlc C++	<code>xlc -g program.cxx</code>
IBM xlf Fortran 77	<code>xlf -g program.f</code>
IBM xlf90 Fortran 90	<code>xlf90 -g program.f90</code>
KAI C	<code>KCC +K0 -qnofullpath program.c</code>
KAI C++	<code>KCC +K0 -qnofullpath program.cxx</code>
KAI Guide C (OpenMP)	<code>guidec -g +K0 program.c</code>

TABLE 6: Compiling with Debugging Symbols on AIX (cont.)

Compiler	Compiler Command Line
KAI Guide C++ (OpenMP)	<code>guidec -g +K0 program.cxx</code>
KAI Guide F77 (OpenMP)	<code>guidef77 -g -WG,-cmpo=i program.f</code>

You should not define any of the following variables when debugging threaded applications:

- AIXTHREAD\_DEBUG
- AIXTHREAD\_COND\_DEBUG
- AIXTHREAD\_MUTEX\_DEBUG
- AIXTHREAD\_RWLOCK\_DEBUG

When compiling with KCC, you must specify the `-qnofullpath` option; KCC is a pre-processor 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 -g program.F`

If you will be moving any program compiled with any of the IBM *xl* compilers from its creation directory, or you do not want to set the search directory path during debugging, use the `-qfullpath` compiler option. For example:

```
xlf -qfullpath -g -c program.f
```

## Linux Running on an x86 Platform

Table 7 lists the procedures to compile programs on Linux x86 platforms.

TABLE 7: Compiling with Debugging Symbols on Linux x86

Compiler	Compiler Command Line
GCC EGCS C	<code>gcc -g program.c</code>
GCC EGCS C++	<code>g++ -g program.cxx</code>
Intel C++ Compiler	<code>icc -g program.cxx</code>

TABLE 7: Compiling with Debugging Symbols on Linux x86 (cont.)

Compiler	Compiler Command Line
Intel Fortran Compiler	<code>ifc -g program.f</code>
KAI C	<code>KCC +K0 program.c</code>
KAI C++	<code>KCC +K0 program.cxx</code>
KAI Guide C (OpenMP)	<code>guidec -g +K0 program.c</code>
KAI Guide C++ (OpenMP)	<code>guidec -g +K0 program.cxx</code>
KAI Guide F77 (OpenMP)	<code>guidef77 -g -WG,-cmpo=i program.f</code>
Lahey/Fujitsu Fortran	<code>lf95 -g program.f</code>
PGI Fortran 90	<code>pgf90 -g program.f</code>

When compiling with KCC for debugging, we recommend that you use the `+K0` option and not the `-g` option. Also, the `-WG,-cmpo=i` option to the `guidef77` command may not be required on all versions because `-g` can imply these options.

## SGI IRIX-MIPS Systems

Table 8 lists the procedures to compile programs on SGI MIPS systems running IRIX.

TABLE 8: Compiling with Debugging Symbols on IRIX-MIPS

Compiler	Compiler Command Line
GCC EGCS C	<code>gcc -g program.c</code>
GCC EGCS C++	<code>g++ -g program.cxx</code>
Intrepid (GCC UPC)	<code>upc -g [-fthreads n] program.upc</code>
KAI C	<code>KCC +K0 program.c</code>
KAI C++	<code>KCC +K0 program.cxx</code>
KAI Guide C (OpenMP)	<code>guidec -g +K0 program.c</code>
KAI Guide C++ (OpenMP)	<code>guidec -g +K0 program.cxx</code>
KAI Guide F77 (OpenMP)	<code>guidef77 -g -WG,-cmpo=i program.f</code>
SGI MIPSpro 90	<code>f90 -n32 -g program.f90</code> <code>f90 -64 -g program.f90</code>
SGI MIPSpro C	<code>cc -n32 -g program.c</code> <code>cc -64 -g program.c</code>

TABLE 8: Compiling with Debugging Symbols on IRIX-MIPS (cont.)

Compiler	Compiler Command Line
SGI MIPSpro C++	CC -n32 -g <i>program.cxx</i> CC -64 -g <i>program.cxx</i>
SGI MIPSpro77	f77 -n32 -g <i>program.f</i> f77 -64 -g <i>program.f</i>

You cannot compile your program using either of the `-n32` or `-64` command line options. TotalView does not support compiling with `-32`, which is the default for some compilers. You must specify either `-n32` or `-64`.

When compiling with KCC for debugging, we recommend that you use the `+K0` option and not the `-g` option. Also, the `-WG,-cmpto=i` option to the `guidef77` command may not be required on all versions because `-g` can imply these options.

## SunOS 5 on SPARC

Table 9 lists the procedures to compile programs on SunOS 5 SPARC.

TABLE 9: Compiling with Debugging Symbols on SunOS 5

Compiler	Compiler Command Line
Apogee C	apcc -g <i>program.c</i>
Apogee C++	apcc -g <i>program.cxx</i>
GCC EGCS C	gcc -g <i>program.c</i>
GCC EGCS C++	g++ -g <i>program.cxx</i>
KAI C	KCC +K0 <i>program.c</i>
KAI C++	KCC +K0 <i>program.cxx</i>
KAI Guide C (OpenMP)	guidec -g +K0 <i>program.c</i>
KAI Guide C++ (OpenMP)	guidec -g +K0 <i>program.cxx</i>
KAI Guide F77 (OpenMP)	guidef77 -g -WG,-cmpto=i <i>program.f</i>
SunPro/WorkShop C	cc -g <i>program.c</i>
SunPro/WorkShop C++	CC -g <i>program.cxx</i>
SunPro/WorkShop Fortran 77	f77 -g <i>program.f</i>
WorkShop Fortran 90	f90 -g <i>program.f90</i>

When compiling with KCC for debugging, we recommend that you use the `+KO` option and not the `-g` option. Also, the `-WG,-cmpp=i` option to the `guidef77` command may not be required on all versions because `-g` can imply these options.

## Using Exception Data on HP 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 -WI,-u,_fpdata_size program.c
```

where:

- `-WI` Passes the arguments that follow to another compilation phase (`-W`), 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.

## Linking with the dbfork Library

If your program uses the `fork()` and `execve()` system calls, and you want to debug the child processes, you need to link programs with the `dbfork` library.

## Linking with dbfork and HP Tru64 UNIX

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 `-dbfork` 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/lib/rs6000/libdbfork.a`
- `-L/usr/totalview/lib -ldbfork -bkeepfile:/usr/totalview/lib/rs6000/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`
- `-L/usr/totalview/lib -ldbfork_64 \`  
`-bkeepfile:/usr/totalviewrs6000/lib/libdbfork.a`

For example:

```
cc -o program program.c \
    -L/usr/totalview/rs6000/lib -ldbfork \
    -bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
```

When you use `gcc` or `g++`, use the `-Wl,-bkeepfile` option instead of using the `-bkeepfile` option, which will pass the same option to the binder. For example:

```
gcc -o program program.c -L/usr/totalview/rs6000/lib \
    -ldbfork -Wl, \
    -bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
```

### Linking C++ Programs with dbfork

You cannot use the `-bkeepfile` binder option with the IBM xLC C++ compiler. The compiler passes all binder options to an additional pass called **munch**, which will not handle the `-bkeepfile` option.

To work around this problem, we have provided the C++ header file `libdbfork.h`. You must include this file somewhere in your C++ program. This forces the components of the **dbfork** library to be kept in your executable. The file `libdbfork.h` is included only with the RS/6000 version of TotalView. This means that if you are cre-

ating 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/rs6000/lib/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/rs6000/lib/libdbfork.a`
- `-L/usr/totalview/rs6000/lib -ldbfork`

## Linux

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`

where *platform* is either **linux-86** or **linux-alpha**.

For example:

```
cc -o program program.c -L/usr/totalview/linux-86/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 /usr/totalview/platform/lib
```

where *platform* is again either **linux-86** or **linux-alpha**.

## 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
```



## Chapter 9

# Operating Systems

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 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 document on our web site. You can locate this document by going to <http://www.etnus.com/Support/docs/>.

- HP Alpha workstations running HP Tru64 UNIX versions V4.0F, V5.1, and V5.1A. Many versions require patches. See "HP UNIX Patch Procedures" in the TOTALVIEW PLATFORMS document for instructions.
- HP PA-RISC 1.1 or 2.0 systems running HP-UX Version 11.00, 11.10, and 11.11i.
- IBM RS/6000 and SP systems running AIX versions 4.3.3 and 5.1L.
- Linux Red Hat 7.1, 7.2, 7.3, and 8.0.

- SGI IRIX 6.5.1.15f and 6.5.1.16f on any MIPS R4000, R4400, R4600, R5000, R8000, R10000, and R12000 processor-based systems.
- Sun Sparc Solaris 7, 8 and 9.

## 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 d_process object`

To determine whether the **/proc** file system is mounted, enter the appropriate command from the following table.

TABLE 10: Commands for Determining Whether /proc Is Mounted

Operating System	Command
HP Tru64 UNIX	<code>% /sbin/mount -t procfs /proc on /proc type procfs (rw)</code>
SunOS 5	<code>%/sbin/mount   grep /proc /proc on /proc read/write/setuid on ...</code>
IRIX	<code>%/sbin/mount   grep /proc /proc on /proc type proc (rw)</code>

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 11: Commands for Automatically Mounting the `/proc` File System

Operating System	Name of File	Line to add
HP Tru64 UNIX	<code>/etc/fstab</code>	<code>/proc /proc procfs rw 0 0</code>
SunOS 5	<code>/etc/vfstab</code>	<code>/proc - /proc proc - no -</code>

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 TotalView 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 **SZ** column shows the pages occupied by a program.

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

## Maximum Data Size

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

```
limit datasize
```

The following command displays the current *hard* limit:

```
limit -h datasize
```

If the current limit is lower than the hard limit, you can easily raise the current limit.

To change the current limit, enter:

```
limit datasize new_data_size
```

If the hard limit is too low, you must reconfigure and rebuild the kernel, and then reboot. This is most easily done using **SAM**.

To change **maxdsiz**, use the following path through the **SAM** menus:

```
Kernel Configuration > Configurable Parameters >
  maxdsiz > Actions > Modify Configurable Parameter >
  Specify New Formula/Value > Formula/Value
```

You can now enter the new maximum data segment size.

You may also need to change the value for **maxdsiz\_64**.

Here is the command that lets you rebuild the kernel with these changed values:

```
Configurable Parameter > Actions > Process New Kernel
```

Answer **yes** to process the kernel modifications, **yes** to install the new kernel, and **yes** again to reboot the machine with the new kernel.

When the machine reboots, the value you set for **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 **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:

```
totalview executable
```

Press Ctrl+Z to suspend TotalView.

- 2 Use the following command to see how much swap space TotalView is using:

```
ps u
```

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.

## Shared Libraries

TotalView supports dynamically linked executables, that is, executables that are linked with shared libraries.

When you start TotalView with a dynamically linked executable, TotalView loads an additional set of symbols for the shared libraries, as indicated in the shell from which you started TotalView. To accomplish this, TotalView:

- 1 Runs a sample process and discards it.
- 2 Reads information from the process.
- 3 Reads the symbol table for each library.

When you create a process without starting it, and the process does not include shared libraries, the PC points to the entry point of the process, usually the **start** routine. If the process does include shared libraries, however, TotalView takes the following actions:

- Runs the dynamic loader (SunOS 5: **ld.so**, HP Tru64 UNIX: **/sbin/loader**, Linux: **/lib/ld-linux.so.?**, IRIX: **rdl**).
- 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.

**NOTE** 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 6, “*TotalView Command Syntax*” on page 207 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's 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 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. (See "*File > Preferences Dialog Box: Dynamic Libraries Page*" on page 246.)

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.

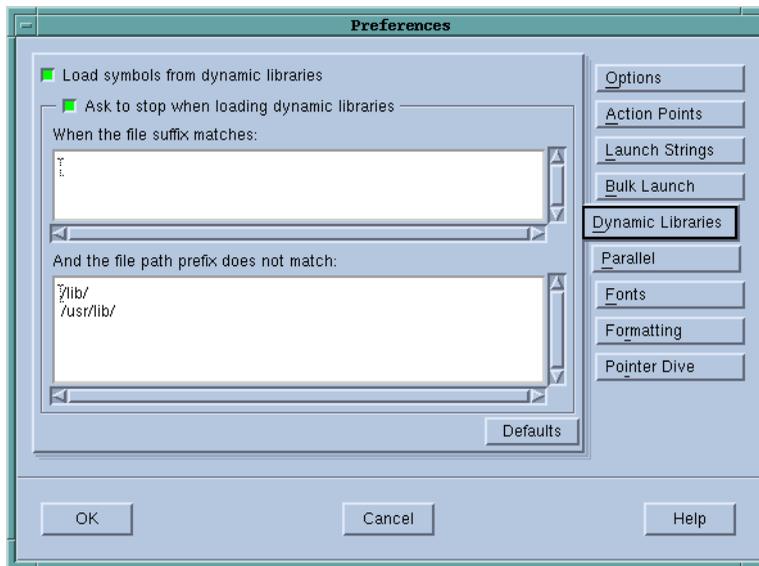


FIGURE 1: **File > Preferences Dialog Box: Dynamic Libraries Page**

- 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 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 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 2.)

To allow the process to continue executing, answer **No**. Stopping the process allows you to insert breakpoints in the newly loaded shared library.

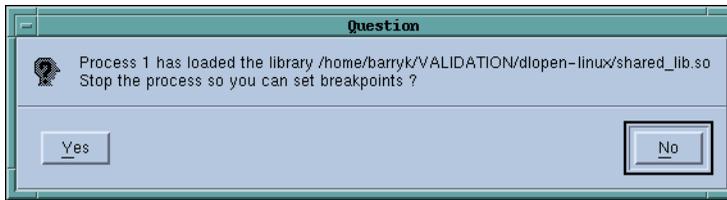


FIGURE 2: **Stop Process Question Dialog Box**

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 12: **Default “Don’t Ask” on Load List**

Platform	Value
HP Tru64 UNIX Alpha	<code>/usr/shlib/</code> <code>/usr/ccs/lib/</code> <code>/usr/lib/cmplrs/cc/</code> <code>/usr/lib/</code> <code>/usr/local/lib/</code> <code>/var/shlib/</code>
HP-UX	<code>/usr/lib/</code> <code>/usr/lib/pa20_64</code> <code>/opt/langtools/lib/</code> <code>/opt/langtools/lib/pa20_64/</code>
IBM AIX	<code>/lib/</code> <code>/usr/lib/</code> <code>/usr/lpp/</code> <code>/usr/ccs/lib/</code> <code>/usr/dt/lib/</code> <code>/tmp/</code>
SGI IRIX	<code>/lib/</code> <code>/usr/lib/</code> <code>/usr/local/lib/</code> <code>/lib32/</code> <code>/usr/lib32/</code> <code>/usr/local/lib32/</code> <code>/lib64/</code> <code>/usr/lib64/</code> <code>/usr/local/lib64</code>
SUN Solaris 2.x	<code>/lib/</code> <code>/usr/lib/</code> <code>/usr/ccs/lib/</code>
Linux x86	<code>/lib</code> <code>/usr/lib</code>
Linux Alpha	<code>/lib</code> <code>/usr/lib</code>

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'
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**:

```
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's 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 14 of the TOTALVIEW USERS GUIDE.



## Chapter 10

# Architectures

This chapter describes the architectures TotalView supports, including:

- HP Alpha
- HP PA-RISC
- IBM Power
- Intel-x86 (Intel 80386, 80486 and Pentium processors)
- SGI MIPS
- Sun SPARC

## HP Alpha

This section contains the following information:

- Alpha General Registers
- Alpha Floating-Point Registers
- Alpha FPCR Register

**NOTE** 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.

TABLE 13: Alpha General-Purpose Integer Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
V0	Function value register	<long>	yes	yes	\$v0
T0 – T7	Conventional scratch registers	<long>	yes	yes	\$t0 – \$t7
S0 – S5	Conventional saved registers	<long>	yes	yes	\$s0 – \$s5
S6	Stack frame base register	<long>	yes	yes	\$s6
A0 – A5	Argument registers	<long>	yes	yes	\$a0 – \$a5
T8 – T11	Conventional scratch registers	<long>	yes	yes	\$t8 – \$t11
RA	Return Address register	<long>	yes	yes	\$ra
T12	Procedure value register	<long>	yes	yes	\$t12
AT	Volatile scratch register	<long>	yes	yes	\$at
GP	Global pointer register	<long>	yes	yes	\$gp
SP	Stack pointer	<long>	yes	yes	\$sp
ZERO	ReadAsZero/Sink register	<long>	no	yes	\$zero
PC	Program counter	<code> []	no	yes	\$pc
FP	Frame pointer. The Frame Pointer is a software register that TotalView maintains; it is not an actual hardware register. TotalView computes the value of FP as part of the stack backtrace.	<long>	no	yes	\$fp

## 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 14: Alpha Floating-Point Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
F0 – F1	Floating-point registers (f registers), used singly	<double>	yes	yes	\$f0 – \$f1
F2 – F9	Conventional saved registers	<double>	yes	yes	\$f2 – \$f9
F10 – F15	Conventional scratch registers	<double>	yes	yes	\$f10 – \$f15
F16 – F21	Argument registers	<double>	yes	yes	\$f16 – \$f21
F22 – F30	Conventional scratch registers	<double>	yes	yes	\$f22 – \$f30
F31	ReadAsZero/Sink register	<double>	yes	yes	\$f31
FPCR	Floating-point control register	<long>	yes	no	\$fpcr

## 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 15: Alpha FPCR Register Bit Settings

Value	Bit Setting	Meaning
SUM	0x8000000000000000	Summary bit
DYN=CHOP	0x0000000000000000	Rounding mode — Chopped rounding mode
DYN=MINF	0x0400000000000000	Rounding mode — Negative infinity
DYN=NORM	0x0800000000000000	Rounding mode — Normal rounding

TABLE 15: Alpha FPCR Register Bit Settings (cont.)

Value	Bit Setting	Meaning
DYN=PINF	0x0c00000000000000	Rounding mode — Positive infinity
IOV	0x0200000000000000	Integer overflow
INE	0x0100000000000000	Inexact result
UNF	0x0080000000000000	Underflow
OVF	0x0040000000000000	Overflow
DZE	0x0020000000000000	Division by zero
INV	0x0010000000000000	Invalid operation

## 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 16: PA-RISC General Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
r0	Always contains zero	<long>	no	no	\$r0
r1-r31	General registers	<long>	yes	yes	\$r1-\$r31
pc	Current instruction pointer	<long>	yes	yes	\$pc
nxtpc	Next instruction pointer	<long>	yes	yes	\$nxtpc
pcs	Current instruction space	<long>	no	no	\$pcs

TABLE 16: PA-RISC General Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
nxtpcs	Next instruction space	<long>	no	no	\$nxtpcs
psw	Processor status word	<long>	yes	no	\$psw
sar	Shift amount register	<long>	yes	no	\$sar
sr0-sr7	Space registers	<long>	no	no	\$sr0-\$sr7
recov	Recovery counter	<long>	no	no	\$recov
pid1-pid8	Protection IDs	<long>	no	no	\$pid1-\$pid8
ccr	Coprocessor configuration	<long>	no	no	\$ccr
scr	SFU configuration register	<long>	no	no	\$scr
eiem	External interrupt enable mask	<long>	no	no	\$eiem
iir	Interrupt instruction	<long>	no	no	\$iir
isr	Interrupt space	<long>	no	no	\$isr
ior	Interrupt offset	<long>	no	no	\$ior
cr24-cr26	Temporary registers	<long>	no	no	\$cr24-\$cr26
tp	Thread pointer	<long>	yes	yes	\$tp

## 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 17: PA-RISC Processor Status Word

Value	Bit Setting	Meaning
W	0x0000000008000000	64-bit addressing enable
E	0x0000000004000000	Little-endian enable
S	0x0000000002000000	Secure interval timer
T	0x0000000001000000	Taken branch flag
H	0x0000000000800000	Higher-privilege transfer trap enable
L	0x0000000000400000	Lower-privilege transfer trap enable
N	0x0000000000200000	Nullify current instruction
X	0x0000000000100000	Data memory break disable

TABLE 17: PA-RISC Processor Status Word (cont.)

Value	Bit Setting	Meaning
B	0x0000000000008000	Taken branch flag
C	0x0000000000004000	Code address translation enable
V	0x0000000000002000	Divide step correction
M	0x0000000000001000	High-priority machine check mask
O	0x0000000000000080	Ordered references
F	0x0000000000000020	Performance monitor interrupt unmask
R	0x0000000000000010	Recovery counter enable
Q	0x0000000000000008	Interrupt state collection enable
P	0x0000000000000004	Protection identifier validation enable
D	0x0000000000000002	Data address translation enable
I	0x0000000000000001	External interrupt unmask
C/B	0x000000FF0000FF00	Carry/borrow bits

## 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 18: PA-RISC Floating-Point Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
status	Status register	<int>	no	no	\$status
er1-er7	Exception registers	<int>	no	no	\$er1-\$er7
fr4-fr31	Double floating-point registers	<double>	yes	yes	\$fr4-\$fr31
fr4l-fr31l	Left half floating-point registers	<float>	yes	yes	\$fr4l-\$fr31l
fr4r-fr31r	Right half floating-point registers	<float>	yes	yes	\$fr4r-\$fr31r
fr4/fr5-fr30/fr31	Extended floating-point register pairs	<extended>	yes	yes	\$fr4_fr5-\$fr30_fr31

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 19: Floating-Point Status Word Use

Type	Value	Meaning
Rounding Mode	0	Round to nearest
	1	Round toward zero
	2	Round toward +infinity
	3	Round toward -infinity
Exception Enable and Exception Flag Bits	V	Invalid operation
	Z	Division by zero
	O	Overflow
	U	Underflow
	I	Inexact result
Comparison Fields	C	Compare bit; contains the result of the most recent queued compare instruction.
	CQ	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.
	CA	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.
Other Flags	T	Delayed trap
	D	Denormalized as zero

## PA-RISC Floating-Point Format

The PA-RISC processor supports the IEEE floating-point format.

## IBM Power

This section contains the following information:

- Power General Registers
- Power MSR Register
- Power Floating-Point Registers
- Power FPSCR Register
- Using the Power FPSCR Register

**NOTE** The Power architecture supports the IEEE floating-point format.

## Power General Registers

TotalView displays Power general registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each general register, and the actions you can take with each register.

TABLE 20: Power General-Purpose Integer Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
R0	General register 0	<int>	yes	yes	\$r0
SP	Stack pointer	<int>	yes	yes	\$sp
RTOC	TOC pointer	<int>	yes	yes	\$rtoc
R3 – R31	General registers 3 – 31	<int>	yes	yes	\$r3 – \$r31
INUM		<int>	yes	no	\$inum
PC	Program counter	<code>[]	no	yes	\$pc
SRR1	Machine status save/restore register	<int>	yes	no	\$srr1
LR	Link register	<int>	yes	no	\$lr
CTR	Counter register	<int>	yes	no	\$ctr
CR	Condition register	<int>	yes	no	\$cr
XER	Integer exception register	<int>	yes	no	\$xer
DAR	Data address register	<int>	yes	no	\$dar

TABLE 20: Power General-Purpose Integer Registers (cont.)

Register	Description	Data Type	Edit	Dive	Specify in Expression
MQ	MQ register	<int>	yes	no	\$mq
MSR	Machine state register	<int>	yes	no	\$msr
SEG0 – SEG9	Segment registers 0 – 9	<int>	yes	no	\$seg0 – \$seg9
SG10 – SG15	Segment registers 10 – 15	<int>	yes	no	\$sg10 – \$sg15
SCNT	SS_COUNT	<int>	yes	no	\$scnt
SAD1	SS_ADDR 1	<int>	yes	no	\$sad1
SAD2	SS_ADDR 2	<int>	yes	no	\$sad2
SCD1	SS_CODE 1	<int>	yes	no	\$scd1
SCD2	SS_CODE 2	<int>	yes	no	\$scd2
TID		<int>	yes	no	

## 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 21: Power MSR Register Bit Settings

Value	Bit Setting	Meaning
0x8000000000000000	SF	Sixty-four bit mode
0x0000000000004000	POW	Power management enable
0x0000000000002000	TGPR	Temporary GPR mapping
0x0000000000001000	ILE	Exception little-endian mode
0x0000000000000800	EE	External interrupt enable
0x0000000000000400	PR	Privilege level
0x0000000000000200	FP	Floating-point available
0x0000000000000100	ME	Machine check enable
0x0000000000000080	FE0	Floating-point exception mode 0
0x0000000000000040	SE	Single-step trace enable
0x0000000000000020	BE	Branch trace enable
0x0000000000000010	FE1	Floating-point exception mode 1
0x0000000000000004	IP	Exception prefix

TABLE 21: Power MSR Register Bit Settings (cont.)

Value	Bit Setting	Meaning
0x0000000000000020	IR	Instruction address translation
0x0000000000000010	DR	Data address translation
0x0000000000000002	RI	Recoverable exception
0x0000000000000001	LE	Little-endian mode enable

## 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 22: Power Floating-Point Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
F0 – F31	Floating-point registers 0 – 31	<double>	yes	yes	\$f0 – \$f31
FPSCR	Floating-point status register	<int>	yes	no	\$fpscr
FPSCR2	Floating-point status register 2	<int>	yes	no	\$fpscr2

## 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 23: Power PFSCR Register Bit Settings

Value	Bit Setting	Meaning
0x80000000	FX	Floating-point exception summary
0x40000000	FEX	Floating-point enabled exception summary
0x20000000	VX	Floating-point invalid operation exception summary
0x10000000	OX	Floating-point overflow exception
0x08000000	UX	Floating-point underflow exception

TABLE 23: Power PFSCR Register Bit Settings (cont.)

Value	Bit Setting	Meaning
0x04000000	ZX	Floating-point zero divide exception
0x02000000	XX	Floating-point inexact exception
0x01000000	VXSNAN	Floating-point invalid operation exception for SNaN
0x00800000	VXISI	Floating-point invalid operation exception: $\infty - \infty$
0x00400000	VXIDI	Floating-point invalid operation exception: $\infty / \infty$
0x00200000	VXZDZ	Floating-point invalid operation exception: $0 / 0$
0x00100000	VXIMZ	Floating-point invalid operation exception: $\infty * \infty$
0x00080000	VXVC	Floating-point invalid operation exception: invalid compare
0x00040000	FR	Floating-point fraction rounded
0x00020000	FI	Floating-point fraction inexact
0x00010000	FPRF=(C)	Floating-point result class descriptor
0x00008000	FPRF=(L)	Floating-point less than or negative
0x00004000	FPRF=(G)	Floating-point greater than or positive
0x00002000	FPRF=(E)	Floating-point equal or zero
0x00001000	FPRF=(U)	Floating-point unordered or NaN
0x00011000	FPRF=(QNaN)	Quiet NaN; alias for FPRF=(C+U)
0x00009000	FPRF=(-INF)	-Infinity; alias for FPRF=(L+U)
0x00008000	FPRF=(-NORM)	-Normalized number; alias for FPRF=(L)
0x00018000	FPRF=(-DENORM)	-Denormalized number; alias for FPRF=(C+L)
0x00012000	FPRF=(-ZERO)	-Zero; alias for FPRF=(C+E)
0x00002000	FPRF=(+ZERO)	+Zero; alias for FPRF=(E)
0x00014000	FPRF=(+DENORM)	+Denormalized number; alias for FPRF=(C+G)
0x00004000	FPRF=(+NORM)	+Normalized number; alias for FPRF=(G)
0x00005000	FPRF=(+INF)	+Infinity; alias for FPRF=(G+U)
0x00000400	VXSOFT	Floating-point invalid operation exception: software request
0x00000200	VXSQRT	Floating-point invalid operation exception: square root
0x00000100	VXCVI	Floating-point invalid operation exception: invalid integer convert

TABLE 23: Power PFSCR Register Bit Settings (cont.)

Value	Bit Setting	Meaning
0x00000080	VE	Floating-point invalid operation exception enable
0x00000040	OE	Floating-point overflow exception enable
0x00000020	UE	Floating-point underflow exception enable
0x00000010	ZE	Floating-point zero divide exception enable
0x00000008	XE	Floating-point inexact exception enable
0x00000004	NI	Floating-point non-IEEE mode enable
0x00000000	RN=NEAR	Round to nearest
0x00000001	RN=ZERO	Round toward zero
0x00000002	RN=PINF	Round toward +infinity
0x00000003	RN=NINF	Round toward -infinity

## Using the Power FPSCR Register

On AIX, if you compile your program to catch floating-point exceptions (IBM compiler `-qfltrap` 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** (as shown in Table 23) so that TotalView traps the “divide by zero” exception. The string displayed next to the FPSR register should now include **ZE**. Now, when your program divides by zero, it receives a **SIGTRAP** signal, which will be caught by TotalView. See “*Handling Signals*” in Chapter 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-x86

This section contains the following information:

- Intel-x86 General Registers
- Intel-x86 Floating-Point Registers
- Intel-x86 FPCR Register

- Using the Intel-x86 FPCR Register
- Intel-x86 FPSR Register

**NOTE** The Intel-x86 processor supports the IEEE floating-point format.

## Intel-x86 General Registers

TotalView displays the Intel-x86 general registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each general register, and the actions you can take with each register.

TABLE 24: Intel-x86 General Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression	
EAX	General registers	<void>	yes	yes	\$eax	
ECX		<void>	yes	yes	\$ecx	
EDX		<void>	yes	yes	\$edx	
EBX		<void>	yes	yes	\$ebx	
EBP		<void>	yes	yes	\$ebp	
ESP		<void>	yes	yes	\$esp	
ESI		<void>	yes	yes	\$esi	
EDI		<void>	yes	yes	\$edi	
CS		Selector registers	<void>	no	no	\$cs
SS			<void>	no	no	\$ss
DS			<void>	no	no	\$ds
ES			<void>	no	no	\$es
FS			<void>	no	no	\$fs
GS			<void>	no	no	\$gs
EFLAGS	Instruction pointer	<void>	no	no	\$eflags	
EIP		<code>[]	no	yes	\$eip	
FAULT		<void>	no	no	\$fault	
TEMP		<void>	no	no	\$temp	
INUM		<void>	no	no	\$inum	
ECODE		<void>	no	no	\$ecode	

TABLE 24: Intel-x86 General Registers (cont.)

Register	Description	Data Type	Edit	Dive	Specify in Expression
XMM0_L	Streaming SIMD	<long long>	yes	yes	\$xmm0_l
...	Extension: left half				...
XMM7_L					\$xmm7_l
XMM0_H	Streaming SIMD	<long long>	yes	yes	\$xmm0_h
...	Extension: right half				...
XMM7_H					\$xmm7_h

**NOTE** The Pentium III and 4 have 8 128-bit registers that are used by SSE and SSE2 instructions. TotalView displays these as 16 64-bit registers. These registers can be used in the following ways: 16 bytes, 8 words, 2 long longs, 4 floating point, 2 double, or a single 128-bit value. TotalView shows each of these hardware registers as two <long long> registers. To change the type, dive and then edit the type in the data window to be an array of the type you wish. For example, cast it to "<char>[16]", "<float>[4]", and so on.

## Intel-x86 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 25: Intel-x86 Floating-Point Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
ST0	ST(0)	<extended>	yes	yes	\$st0
ST1	ST(1)	<extended>	yes	yes	\$st1
ST2	ST(2)	<extended>	yes	yes	\$st2
ST3	ST(3)	<extended>	yes	yes	\$st3
ST4	ST(4)	<extended>	yes	yes	\$st4
ST5	ST(5)	<extended>	yes	yes	\$st5
ST6	ST(6)	<extended>	yes	yes	\$st6
ST7	ST(7)	<extended>	yes	yes	\$st7
FPCR	Floating-point control register	<void>	yes	no	\$fpcr
FPSR	Floating-point status register	<void>	no	no	\$fpsr
FPTAG	Tag word	<void>	no	no	\$fptag
FPIOFF	Instruction offset	<void>	no	no	\$fpioff

TABLE 25: Intel-x86 Floating-Point Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
FPISEL	Instruction selector	<void>	no	no	\$fpisel
FPDOFF	Data offset	<void>	no	no	\$fpdoff
FPDSEL	Data selector	<void>	no	no	\$fpdsel

## Intel-x86 FPCR Register

For your convenience, TotalView interprets the bit settings of the FPCR and FPSR registers.

You can edit the value of the FPCR and set it to any of the bit settings outlined in the next table.

TABLE 26: Intel-x86 FPCR Register Bit Settings

Value	Bit Setting	Meaning
RC=RN	0x0000	To nearest rounding mode
RC=R-	0x2000	Toward negative infinity rounding mode
RC=R+	0x4000	Toward positive infinity rounding mode
RC=RZ	0x6000	Toward zero rounding mode
PC=SGL	0x0000	Single-precision rounding
PC=DBL	0x0080	Double-precision rounding
PC=EXT	0x00c0	Extended-precision rounding
EM=PM	0x0020	Precision exception enable
EM=UM	0x0010	Underflow exception enable
EM=OM	0x0008	Overflow exception enable
EM=ZM	0x0004	Zero-divide exception enable
EM=DM	0x0002	Denormalized operand exception enable
EM=IM	0x0001	Invalid operation exception enable

## 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** (as shown in Table 26) 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 27: Intel-x86 FPSR Register Bit Settings

Value	Bit Setting	Meaning
TOP= < <i>i</i> >	0x3800	Register < <i>i</i> > is top of FPU stack
B	0x8000	FPU busy
C0	0x0100	Condition bit 0
C1	0x0200	Condition bit 1
C2	0x0400	Condition bit 2
C3	0x4000	Condition bit 3
ES	0x0080	Exception summary status
SF	0x0040	Stack fault
EF=PE	0x0020	Precision exception
EF=UE	0x0010	Underflow exception
EF=OE	0x0008	Overflow exception
EF=ZE	0x0004	Zero divide exception
EF=DE	0x0002	Denormalized operand exception
EF=IE	0x0001	Invalid operation exception

## Intel-x86 MXSCR 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 28: Intel-x86 MXCSR Register Bit Settings**

Value	Bit Setting	Meaning
FZ	0x8000	Flush to zero
RC=RN	0x0000	To nearest rounding mode
RC=R-	0x2000	Toward negative infinity rounding mode
RC=R+	0x4000	Toward positive infinity rounding mode
RC=RZ	0x6000	Toward zero rounding mode
EM=PM	0x1000	Precision mask
EM=UM	0x0800	Underflow mask
EM=OM	0x0400	Overflow mask
EM=ZM	0x0200	Divide-by-zero mask
EM=DM	0x0100	Denormal mask
EM=IM	0x0080	Invalid operation mask
DAZ	0x0040	Denormals are zeros
EF=PE	0x0020	Precision flag
EF=UE	0x0010	Underflow flag
EF=OE	0x0008	Overflow flag
EF=ZE	0x0004	Divide-by-zero flag
EF=DE	0x0002	Denormal flag
EF=IE	0x0001	Invalid operation flag

## SGI MIPS

This section contains the following information:

- MIPS General Registers
- MIPS SR Register
- MIPS Floating-Point Registers
- MIPS FCSR Register
- Using the MIPS FCSR Register
- MIPS Delay Slot Instructions

**NOTE** 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 either `-64` or `-n32` have 64-bit registers. TotalView uses `<long>` for `-64` compiled programs and `<long long>` for `-n32` compiled programs.

TABLE 29: MIPS General (Integer) Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
ZERO	Always has the value 0	<code>&lt;long&gt;</code>	no	no	<code>\$zero</code>
AT	Reserved for the assembler	<code>&lt;long&gt;</code>	yes	yes	<code>\$at</code>
V0 – V1	Function value registers	<code>&lt;long&gt;</code>	yes	yes	<code>\$v0 – \$v1</code>
A0 – A7	Argument registers	<code>&lt;long&gt;</code>	yes	yes	<code>\$a0 – \$a7</code>
T0 – T3	Temporary registers	<code>&lt;long&gt;</code>	yes	yes	<code>\$t0 – \$t3</code>
S0 – S7	Saved registers	<code>&lt;long&gt;</code>	yes	yes	<code>\$s0 – \$s7</code>
T8 – T9	Temporary registers	<code>&lt;long&gt;</code>	yes	yes	<code>\$t8 – \$t9</code>
K0 – K1	Reserved for the operating system	<code>&lt;long&gt;</code>	yes	yes	<code>\$k1 – \$k2</code>
GP	Global pointer	<code>&lt;long&gt;</code>	yes	yes	<code>\$gp</code>
SP	Stack pointer	<code>&lt;long&gt;</code>	yes	yes	<code>\$sp</code>
S8	Hardware frame pointer	<code>&lt;long&gt;</code>	yes	yes	<code>\$s8</code>
RA	Return address register	<code>&lt;code&gt;[]</code>	no	yes	<code>\$ra</code>
MDLO	Multiply/Divide special register, holds least-significant bits of multiply, quotient of divide	<code>&lt;long&gt;</code>	yes	yes	<code>\$mdlo</code>
MDHI	Multiply/Divide special register, holds most-significant bits of multiply, remainder of divide	<code>&lt;long&gt;</code>	yes	yes	<code>\$mdhi</code>
CAUSE	Cause register	<code>&lt;long&gt;</code>	yes	yes	<code>\$cause</code>
EPC	Program counter	<code>&lt;code&gt;[]</code>	no	yes	<code>\$epc</code>

TABLE 29: MIPS General (Integer) Registers (cont.)

Register	Description	Data Type	Edit	Dive	Specify in Expression
SR	Status register	<long>	no	no	\$sr
VFP	Virtual frame pointer 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.	<long>	no	no	\$vfp

## MIPS SR Register

For your convenience, TotalView interprets the bit settings of the SR register as outlined in the next table.

TABLE 30: MIPS SR Register Bit Settings

Value	Bit Setting	Meaning
0x00000001	IE	Interrupt enable
0x00000002	EXL	Exception level
0x00000004	ERL	Error level
0x00000008	S	Supervisor mode
0x00000010	U	User mode
0x00000018	U	Undefined (implemented as User mode)
0x00000000	K	Kernel mode
0x00000020	UX	User mode 64-bit addressing
0x00000040	SX	Supervisor mode 64-bit addressing
0x00000080	KX	Kernel mode 64-bit addressing
0x0000FF00	IM= <i>i</i>	Interrupt Mask value is <i>i</i>
0x00010000	DE	Disable cache parity/ECC
0x00020000	CE	Reserved
0x00040000	CH	Cache hit
0x00080000	NMI	Non-maskable interrupt has occurred
0x00100000	SR	Soft reset or NMI exception
0x00200000	TS	TLB shutdown has occurred
0x00400000	BEV	Bootstrap vectors
0x02000000	RE	Reverse-Endian bit

TABLE 30: MIPS SR Register Bit Settings (cont.)

Value	Bit Setting	Meaning
0x04000000	FR	Additional floating-point registers enabled
0x08000000	RP	Reduced power mode
0x10000000	CU0	Coprocessor 0 usable
0x20000000	CU1	Coprocessor 1 usable
0x40000000	CU2	Coprocessor 2 usable
0x80000000	XX	MIPS IV instructions usable

## 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 31: MIPS Floating-Point Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
F0, F2	Hold results of floating-point type function; \$f0 has the real part, \$f2 has the imaginary part	<double>	yes	yes	\$f0, \$f2
F1 – F3, F4 – F11	Temporary registers	<double>	yes	yes	\$f1 – \$f3, \$f4 – \$f11
F12 – F19	Pass single- or double-precision actual arguments	<double>	yes	yes	\$f12 – \$f19
F20 – F23	Temporary registers	<double>	yes	yes	\$f20 – \$f23
F24 – F31	Saved registers	<double>	yes	yes	\$f24 – \$f31
FCSR	FPU control and status register	<int>	yes	no	\$fcsr

## MIPS FCSR Register

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 32: MIPS FCSR Register Bit Settings

Value	Bit Setting	Meaning
RM=RN	0x00000000	Round to nearest
RM=RZ	0x00000001	Round toward zero
RM=RP	0x00000002	Round toward positive infinity
RM=RM	0x00000003	Round toward negative infinity
flags=(I)	0x00000004	Flag=inexact result
flags=(U)	0x00000008	Flag=underflow
flags=(O)	0x00000010	Flag=overflow
flags=(Z)	0x00000020	Flag=divide by zero
flags=(V)	0x00000040	Flag=invalid operation
enables=(I)	0x00000080	Enables=inexact result
enables=(U)	0x00000100	Enables=underflow
enables=(O)	0x00000200	Enables=overflow
enables=(Z)	0x00000400	Enables=divide by zero
enables=(V)	0x00000800	Enables=invalid operation
cause=(I)	0x00001000	Cause=inexact result
cause=(U)	0x00002000	Cause=underflow
cause=(O)	0x00004000	Cause=overflow
cause=(Z)	0x00008000	Cause=divide by zero
cause=(V)	0x00010000	Cause=invalid operation
cause=(E)	0x00020000	Cause=unimplemented
FCC=(0/c)	0x00800000	FCC=Floating-Point Condition Code 0; c=Condition bit
FS	0x01000000	Flush to zero
FCC=(1)	0x02000000	FCC=Floating-Point Condition Code 1
FCC=(2)	0x04000000	FCC=Floating-Point Condition Code 2
FCC=(3)	0x08000000	FCC=Floating-Point Condition Code 3
FCC=(4)	0x10000000	FCC=Floating-Point Condition Code 4
FCC=(5)	0x20000000	FCC=Floating-Point Condition Code 5

TABLE 32: MIPS FCSR Register Bit Settings (cont.)

Value	Bit Setting	Meaning
FCC=(6)	0x40000000	FCC=Floating-Point Condition Code 6
FCC=(7)	0x80000000	FCC=Floating-Point Condition Code 7

## 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** (as shown in Table 31). 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.

## Sun SPARC

This section has the following information:

- SPARC General Registers
- SPARC PSR Register
- SPARC Floating-Point Registers
- SPARC FPSR Register
- Using the SPARC FPSR Register

**NOTE** The SPARC processor supports the IEEE floating-point format.

## SPARC General Registers

TotalView displays the SPARC general registers in the Stack Frame Pane of the Process Window. The following table describes how TotalView treats each general register, and the actions you can take with each register.

TABLE 33: SPARC General Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
G0	Global zero register	<int>	no	no	\$g0
G1 – G7	Global registers	<int>	yes	yes	\$g1 – \$g7
O0 – O5	Outgoing parameter registers	<int>	yes	yes	\$o0 – \$o5
SP	Stack pointer	<int>	yes	yes	\$sp

TABLE 33: SPARC General Registers (cont.)

Register	Description	Data Type	Edit	Dive	Specify in Expression
O7	Temporary register	<int>	yes	yes	\$o7
L0 – L7	Local registers	<int>	yes	yes	\$l0 – \$l7
I0 – I5	Incoming parameter registers	<int>	yes	yes	\$i0 – \$i5
FP	Frame pointer	<int>	yes	yes	\$fp
I7	Return address	<int>	yes	yes	\$i7
PSR	Processor status register	<int>	yes	no	\$psr
Y	Y register	<int>	yes	yes	\$y
WIM	WIM register	<int>	no	no	
TBR	TBR register	<int>	no	no	
PC	Program counter	<code>[]	no	yes	\$pc
nPC	Next program counter	<code>[]	no	yes	\$npc

## 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 34: SPARC PSR Register Bit Settings

Value	Bit Setting	Meaning
ET	0x00000020	Traps enabled
PS	0x00000040	Previous supervisor
S	0x00000080	Supervisor mode
EF	0x00001000	Floating-point unit enabled
EC	0x00002000	Coprocessor enabled
C	0x00100000	Carry condition code
V	0x00200000	Overflow condition code
Z	0x00400000	Zero condition code
N	0x00800000	Negative condition code

## 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 35: SPARC Floating-Point Registers

Register	Description	Data Type	Edit	Dive	Specify in Expression
F0, F1, F0_F1	Floating-point registers ( <i>f</i> registers), used singly	<float>	no	yes	\$f0, \$f1, \$f0_f1
F2 – F31	Floating-point registers ( <i>f</i> registers), used singly	<float>	yes	yes	\$f2– \$f31
F0, F1, F0_F1	Floating-point registers ( <i>f</i> registers), used as pairs	<double>	no	yes	\$f0, \$f1, \$f0_f1
F0/F1 – F30/F31	Floating-point registers ( <i>f</i> registers), used as pairs	<double>	yes	yes	\$2 – \$f30_f31
FPCR	Floating-point control register	<int>	no	no	\$fpcr
FPSR	Floating-point status register	<int>	yes	no	\$fpsr

TotalView allows you to use these registers singly or in pairs, depending on how they are used by your program. For example, if you use F1 by itself, its type is <float>, but if you use the F0/F1 pair, its type is <double>.

## SPARC FPSR Register

For your convenience, TotalView interprets the bit settings of the SPARC FPSR register. You can edit the value of the FPSR and set it to any of the bit settings outlined in the following table.

TABLE 36: SPARC FPSR Register Bit Settings

Value	Bit Setting	Meaning
CEXC=NX	0x00000001	Current inexact exception
CEXC=DZ	0x00000002	Current divide by zero exception
CEXC=UF	0x00000004	Current underflow exception
CEXC=OF	0x00000008	Current overflow exception
CEXC=NV	0x00000010	Current invalid exception
AEXC=NX	0x00000020	Accrued inexact exception

TABLE 36: SPARC FPSR Register Bit Settings (cont.)

Value	Bit Setting	Meaning
AEXC=DZ	0x00000040	Accrued divide by zero exception
AEXC=UF	0x00000080	Accrued underflow exception
AEXC=OF	0x00000100	Accrued overflow exception
AEXC=NV	0x00000200	Accrued invalid exception
EQ	0x00000000	Floating-point condition =
LT	0x00000400	Floating-point condition <
GT	0x00000800	Floating-point condition >
UN	0x00000c00	Floating-point condition unordered
QNE	0x00002000	Queue not empty
NONE	0x00000000	Floating-point trap type None
IEEE	0x00004000	Floating-point trap type IEEE Exception
UFIN	0x00008000	Floating-point trap type Unfinished FPop
UIMP	0x0000c000	Floating-point trap type Unimplemented FPop
SEQE	0x00010000	Floating-point trap type Sequence Error
NS	0x00400000	Nonstandard floating-point FAST mode
TEM=NX	0x00800000	Trap enable mask – Inexact Trap Mask
TEM=DZ	0x01000000	Trap enable mask – Divide by Zero Trap Mask
TEM=UF	0x02000000	Trap enable mask – Underflow Trap Mask
TEM=OF	0x04000000	Trap enable mask – Overflow Trap Mask
TEM=NV	0x08000000	Trap enable mask – Invalid Operation Trap Mask
EXT	0x00000000	Extended rounding precision – Extended precision
SGL	0x10000000	Extended rounding precision – Single precision
DBL	0x20000000	Extended rounding precision – Double precision
NEAR	0x00000000	Rounding direction – Round to nearest (tie-even)
ZERO	0x40000000	Rounding direction – Round to 0
PINF	0x80000000	Rounding direction – Round to +Infinity
NINF	0xc0000000	Rounding direction – Round to –Infinity

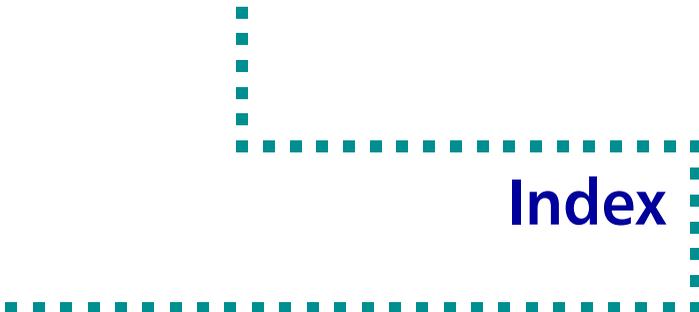
## Using the SPARC FPSR Register

The SPARC processor does not catch floating-point errors by default. You can change the value of the FPSR within TotalView to customize the exception handling for your program.

For example, if your program inadvertently divides by zero, you can edit the bit setting of the FPSR register in the Stack Frame Pane. In this case, you would change

the bit setting for the FPSR to include **0x01000000** (as shown in Table 35) so that TotalView traps the “divide by zero” bit. The string displayed next to the FPSR register should now include **TEM=(DZ)**. Now, when your program divides by zero, it receives a **SIGFPE** signal, which you can catch with TotalView. See “*Handling Signals*” in Chapter 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 **AEXC=(DZ)** bit.





# Index

## Symbols

- # scoping separator character  
31, 36, 68
- \$newval variable in watchpoints  
113
- \$oldval variable in watchpoints  
113
- \$stop built-in function 81
- %B server launch replacement  
character 219
- %C server launch replacement  
character 219
- %D path name replacement char-  
acter 219
- %H hostname replacement char-  
acter 219
- %L host and port replacement  
character 220
- %N line number replacement  
character 220
- %P password replacement char-  
acter 220
- %S source file replacement char-  
acter 220
- %t1 file replacement character  
220
- %t2 file replacement character  
220
- %V verbosity setting replacement  
character 221
- .totalview/lib\_cache subdirectory  
39
- .tvd files 152
- /proc file system 238

- = symbol for PC of current buried  
stack frame 69
- > symbol for PC 69
- @ symbol for action point 69

## A

- a option to totalview command  
208
- ac, *see* dactions command
- acquiring processes 180
- Action Point > Save All com-  
mand 171
- action point identifiers 23, 50
- action points
  - default for newly created 165
  - deleting 45, 131, 141, 147,  
156
  - disabling 23, 24, 47
  - displaying 23
  - enabling 23, 24
  - identifiers 23
  - information about 23
  - loading 23
  - loading automatically 211
  - loading saved information 24
  - reenabling 50
  - saving 23
  - saving information about 24
  - scope of what is stopped 166
  - sharing 165
  - stopping when reached 190
- actionpoint command 131
- actionpoint properties 131
- actions, *see* dactions command

- add verb, image command 144
- adding group members 59
- adding groups 58
- address property 131
- advancing by steps 100
- after\_checkpointing options 41
- AIX
  - compiling on 227
  - linking C++ to dbfork library  
233
  - linking to dbfork library 233
  - swap space 241
- alias command 19
- aliases
  - default 19
  - removing 128
- Alpha
  - architecture 251
  - floating-point registers 253
  - FPCR register 253
  - general registers 252
- append, *see* dlappend command
- appending to CLI variable lists 67
- architectures 167, 251
  - Alpha 251
  - HP PA-RISC 254
  - Intel-x86 262
  - MIPS 267
  - PowerPC 258
  - SPARC 273
- arenas 54, 77
- ARGS variable 89, 94, 161
- ARGS\_DEFAULT variable 89, 94,  
161

arguments  
 command line 94  
 default 161  
 for totalview command 207  
 for tvdsrv command 216

arrays  
 automatic dereferencing 168  
 number of elements displayed 168

arriving at barrier 32

as, *see* dassign command

ask on dlopen option 245

ask\_on\_dlopen option 247

ask\_on\_dlopen variable 168

assemble, displaying symbolically 180

assembler instructions, stepping 103

assign, *see* dassign command

assigning string values 26

assigning values 26

asynchronous execution 51

at, *see* dattach command

attach, *see* dattach command

attaching to parallel processes 28

attaching to processes 28

attaching, using PIDs 29

auto\_array\_cast\_bounds variable 168

auto\_array\_cast\_enabled variable 168

auto\_deref\_in\_all\_c variable 168

auto\_deref\_in\_all\_fortran variable 169

auto\_deref\_initial\_c variable 169

auto\_deref\_initial\_fortran variable 170

auto\_deref\_nested\_c variables 170

auto\_deref\_nested\_fortran variables 170

auto\_load\_breakpoints variables 171

auto\_save\_breakpoints variables 171

automatic dereferencing of arrays 168

automatically attaching to processes 186

## B

b, *see* dbreak command

ba, *see* dbarrier command

–background option 208

back-tick analogy 21

barrier breakpoint 33

barrier is satisfied 162, 172

barrier, *see* dbarrier command

BARRIER\_STOP\_ALL variable 31, 33, 161

barrier\_stop\_all variables 171

BARRIER\_STOP\_WHEN\_DONE variable 32, 162

barrier\_stop\_when\_done variables 172

barriers 31, 33  
 arriving 32  
 creating 32  
 scope of what is stopped 161  
 what else is stopped 31

baud rate, specifying 217

baw, *see* dbarrier command

–bg option 208

–bkeepfile option 233

blocking command input 112

blocking input 112

break, *see* dbreak command

breakpoints  
 automatically loading 171  
 barrier 31  
 default file in which set 37  
 defined 37  
 popping Process Window 198  
 setting at functions 37  
 stopping all processes at 36  
 temporary 107  
 triggering 37

breakpoints file 24, 171

bt, *see* dbreak command

bulk launch 220

bulk\_launch\_base\_timeout variables 172

bulk\_launch\_enabled variables 172

bulk\_launch\_incr\_timeout variables 172

bulk\_launch\_string variables 173

bulk\_launch\_tmpfile1\_header\_line variables 173

bulk\_launch\_tmpfile1\_host\_lines variables 173

bulk\_launch\_tmpfile1\_trailer\_line variables 173

bulk\_launch\_tmpfile2\_header\_line variables 173

bulk\_launch\_tmpfile2\_host\_lines variables 174

bulk\_launch\_tmpfile2\_trailer\_line variables 174

buried stack frame 68

## C

C language escape characters 26

C shell 239

C++  
 demangler 208  
 including libdbfork.h 233

C++ demangler 176

c\_type\_strings variables 174

cache, flushing 39

cache, *see* dcache command

call stack 110  
 displaying 121

call tree saved position 195

–callback option 215, 216

–callback\_host 216

–callback\_ports 216

callbacks 183  
 after loading a program 186  
 when opening the CLI 185

capture command 21, 125

case sensitive searching 187

Cast dereferenced C pointers to array string checkbox 168

CGROUP variable 162

changing CLI variables 96

changing dynamic context 110

changing focus 54

changing value of program variable 26, 41, 61, 65, 79, 91, 105, 107

- chase\_mouse variables 193
- checkpoint, *see* dcheckpoint command
- checkpointing
  - preserving IDs 41
  - process state 40
  - reattaching to parallel 40
  - restarting 91
  - scope 40
  - socket issue 41
- CLI commands
  - action points 17
  - alias 19
  - aliases 161, 201
  - capture 21, 125
  - dactions 23
  - dassign 26
  - dattach 28
  - dbarrier 31
  - dbreak 36
  - dcache 39
  - dcheckpoint 40
  - dcont 43
  - ddelete 45
  - ddetach 46
  - ddisable 47
  - ddown 48
  - dec2hex 133
  - denable 50
  - dflush 51, 81
  - dfocus 54
  - dgo 57
  - dgroups 58
  - dhalt 64
  - dhold 65
  - dkill 66
  - dlappend 67
  - dlist 68
  - dlist command 165
  - dload 71
  - dmstat 73
  - dnext 75
  - dnexti 77
  - dout 79
  - dprint 81
  - dptsets 86
  - drerun 89
  - drestart 91
  - drun 66, 93
  - drun, reissuing 94
  - dset 96
  - dstatus 98
  - dstep 100
  - dstepi 103
  - dunhold 105
  - dunset 106
  - duntil 107
  - dup 110
  - dwait 112
  - dwatc 113
  - dwhat 117
  - dwhere 121
  - dworker 123
  - environment 15
  - executing immediately 210
  - execution control 17
  - exit 124
  - focus of 161, 201
  - help 125
  - initialization 16
  - overview 15, 129
  - program information 16
  - quit 126
  - responding to 150
  - stty 127
  - summary 3
  - termination 16
  - TV::actionpoint 131
  - TV::errorCodes 134
  - TV::expr 81, 136
  - TV::focus\_groups 138
  - TV::focus\_processes 139
  - TV::focus\_threads 140
  - TV::group 141
  - TV::hex2dec 143
  - TV::image 144
  - TV::process 147
  - TV::respond 150
  - TV::source\_process\_startup 152
  - TV::thread 154
  - TV::type 156
  - unalias 128
- CLI variables
  - ARGS 89, 94, 161
  - ARGS\_DEFAULT 89, 94, 161
  - ask\_on\_dlopen 168
  - auto\_array\_cast\_bounds 168
  - auto\_array\_cast\_enabled 168
  - auto\_deref\_in\_all\_c 168
  - auto\_deref\_in\_all\_fortran 169
  - auto\_deref\_initial\_fortran 170
  - auto\_deref\_intial\_c 169
  - auto\_deref\_nested\_c 170
  - auto\_deref\_nested\_fortran 170
  - auto\_load\_breakpoints 171
  - auto\_save\_breakpoints 171
  - BARRIER\_STOP\_ALL 31, 33, 161
  - barrier\_stop\_all 171
  - BARRIER\_STOP\_WHEN\_DONE 32
  - BARRIER\_STOP\_WHEN\_DONE E 162
  - barrier\_stop\_when\_done 172
  - bulk\_launch\_base\_timeout 172
  - bulk\_launch\_enabled 172
  - bulk\_launch\_incr\_timeout 172
  - bulk\_launch\_string 173
  - bulk\_launch\_tmpfile1\_header\_line 173
  - bulk\_launch\_tmpfile1\_host\_lines 173
  - bulk\_launch\_tmpfile1\_trailer\_line 173
  - bulk\_launch\_tmpfile2\_header\_line 173
  - bulk\_launch\_tmpfile2\_host\_lines 174
  - bulk\_launch\_tmpfile2\_trailer\_line 174
  - c\_type\_strings 174
  - CGROUP 162
  - changing 96
  - chase\_mouse 193
  - comline\_patch\_area\_base 174

- comline\_path\_area\_length 174
- COMMAND\_EDITING 163
- command\_editing 174
- compile\_expressions 175
- compiler\_vars 175
- copyright\_string 176
- current\_cplus\_demangler 176
- current\_fortran\_demangler 176
- data\_format\_double 177
- data\_format\_ext 178
- data\_format\_int16 179
- data\_format\_int32 179
- data\_format\_int64 179
- data\_format\_int8 179
- data\_format\_single 179
- data\_format\_singlen 180
- dbfork 180
- default value for 96
- deleting 96
- display\_assembler\_symbolically 180
- display\_font\_dpi 193
- dll\_ignore\_prefix 180
- dll\_stop\_suffix 180
- dpvm 181
- dump\_core 181
- dynamic 181
- editor\_launch\_string 181
- enabled 193
- errorCodes 81
- EXECUTABLE\_PATH 29, 69, 163
- fixed\_font 193
- fixed\_font\_family 194
- fixed\_font\_size 194
- font 194
- force\_default\_cplus\_demangler 182
- force\_default\_f9x\_demangler 182
- force\_window\_position 194
- geometry\_call\_tree 195
- geometry\_cli 195
- geometry\_globals 195
- geometry\_help 195
- geometry\_memory\_stats 196
- geometry\_message\_queue 196
- geometry\_message\_queue\_graph 196
- geometry\_modules 196
- geometry\_process 196
- geometry\_ptset 197
- geometry\_pvm 197
- geometry\_root 197
- geometry\_thread\_objects 197
- geometry\_variable 197
- geometry\_variable\_stats 198
- global\_typenames 182
- GROUP 163
- GROUPS 28, 71, 164
- ignore\_control\_c 183
- image\_load\_callbacks 183
- in\_setup 183
- kcc\_classes 183
- keep\_search\_dialog 198
- kernel\_launch\_string 184
- library\_cache\_directory 184
- LINES\_PER\_SCREEN 164
- local\_interface 184
- local\_server 184
- local\_server\_launch\_string 185
- MAX\_LIST 68, 165
- message\_queue 185
- parallel 185
- parallel\_attach 186
- parallel\_stop 186
- platform 186
- pop\_at\_breakpoint 198
- pop\_on\_error 198
- process\_load\_callbacks 186
- PROMPT 165
- PTSET 165
- pvm 187
- save\_window\_pipe\_or\_filename 187
- search\_case\_sensitive 187
- server\_launch\_enabled 187
- server\_launch\_string 187
- server\_launch\_timeout 188
- SGROUP 165
- SHARE\_ACTION\_POINT 165
- share\_action\_point 188
- signal\_handling\_mode 188
- single\_click\_dive\_enabled 198
- source\_pane\_tab\_width 190
- spell\_correction 190
- STOP\_ALL 36, 115, 166
- stop\_all 190
- stop\_relatives\_on\_proc\_error 190
- suffix 191
- TAB\_WIDTH 69, 166
- THREADS 166
- TOTAL\_VERSION 167
- TOTALVIEW\_ROOT\_PATH 166
- TOTALVIEW\_TCLLIB\_PATH 167
- ttf 191
- ui\_font 198
- ui\_font\_family 199
- ui\_font\_size 199
- user\_threads 191
- using\_color 199
- using\_text\_color 199
- using\_title\_color 199
- VERBOSE 167
- version 191, 199
- viewing 96
- visualizer\_launch\_enabled 192
- visualizer\_launch\_string 192
- visualizer\_max\_rank 192
- warn\_step\_throw 192
- WGROUP 167
- wrap\_on\_search 192
- CLI, activated from GUI flag 193
- clusterid property 147
- co, *see* dcont command
- code, displaying 68
- color
  - foreground 210
- comline\_patch\_area\_base variables 174

- comline\_path\_area\_length variables 174
  - command aliases 161, 201
  - command arguments 161
  - command focus 54, 161, 201
  - command input, blocking 112
  - command line arguments 94
  - command output 21
  - command prompt 165
  - command summary 3
  - COMMAND\_EDITING variable 163
  - command\_editing variables 174
  - commands
    - totalview 207
    - tvdsvr
      - syntax and use 215
    - user-defined 19
  - commands verb
    - actionpoint command 131
    - expr command 136
    - group command 141
    - image command 144
    - process command 147
    - thread command 154
    - type command 156
  - commands, responding to 150
  - compile\_expressions variables 175
  - compiler\_vars option 208
  - compiler\_vars variables 175
  - compilers, KCC 183
  - compiling
    - debugging symbols 225
      - g compiler option 225
      - on HP Tru64 UNIX 226
      - on HP-UX 227, 228
      - on IRIX 229
      - on SunOS 230
    - options 225
  - conditional watchpoints 113
  - connection directory 219
  - console output for tvdsvr 217
  - console output redirection 208
  - cont, *see* dcont command
  - continuation\_sig property 154
  - control group variable 162
  - control group, stopping 190
  - control list element 163
  - copyright\_string variables 176
  - core
    - dumping for TotalView 210
      - when needing to debug TotalView itself 181
  - core files, loading 28
  - count property 141
  - creating a group 58, 60
  - creating barrier breakpoints 32
  - creating commands 19
  - creating new process objects 71
  - creating threads 57
  - Ctrl+C, ignoring 183
  - Ctrl+D to exit CLI 124, 126
  - current data size limit 240
  - current list location 48
  - current\_cplus\_demangler variables 176
  - current\_fortran\_demangler variables 176
- ## D
- d, *see* ddown command
  - d\_process object 238
  - dactions command 23
  - dassign command 26
  - data format, presentation styles 177
  - data size 73
  - data size limit in C shell 239
  - data\_format\_double variables 177
  - data\_format\_ext variables 178
  - data\_format\_int16 variables 179
  - data\_format\_int32 variables 179
  - data\_format\_int64 variables 179
  - data\_format\_int8 variables 179
  - data\_format\_single variables 179
  - data\_format\_stringlen variables 180
  - data\_size property 145
  - datatype incompatibilities 26
  - dattach command 28
  - dbarrier command 31
  - dbfork library
    - linking with 231
    - syntax 208
  - dbfork option 208
  - dbfork variables 180
  - dbreak command 36
  - dcache command 39
  - dcheckpoint command 40
    - preserving IDs 41
    - process 40
    - reattaching to parallel 40
    - scope 40
    - socket issue 41
  - dcont command 43
  - ddelete command 45
  - ddetach command 46
  - ddisable command 47
  - ddown command 48
  - de, *see* ddelete command
  - deactivating action points 47
  - deadlocks at barriers 33
  - debug\_file option 208, 217
  - debugger server 187, 215
  - debugging remote systems 39
  - debugging session, ending 124
  - dec2hex command 133
  - default aliases 19
  - default arguments 94, 161
    - modifying 94
  - default focus 54
  - default value of variables, restoring 106
  - defining the current focus 165
  - delay slot instructions for MIPS 272
  - delete verb, expr command 136
  - delete, *see* ddelete command
  - deleting action points 45, 131, 141, 147, 156
  - deleting cache 39
  - deleting CLI variables 96
  - deleting groups 58, 60
  - deleting variables 106
  - demangler
    - C++ 176
    - forcing use 182
    - Fortran 176
    - overriding 208, 210

- demangler option 208
- denable command 50
- dereferencing
  - C pointers automatically 169
  - C structure pointers automatically 170
  - Fortran pointers automatically 170
  - values automatically 168
- dereferencing values automatically 169
- det, *see* ddetach command
- detach, *see* ddetach command
- detaching from processes 46
- dflush command 51, 81
- dfocus command 54
- dgo command 57
- dgroups command 58
  - add 59
  - delete 60
  - intersect 60
  - list 60
  - new 60
  - remove 61
- dhalt command 64
- dhold command 65
- di, *see* ddisable command
- directory search paths 163
- disable, *see* ddisable command
- disabling action points 23, 24, 47
- disabling PVM support 212
- display call stack 121
  - display option 209
- display\_assembler\_symbolically variables 180
- display\_font\_dpi variables 193
- displaying
  - code 68
  - current execution location 121
  - error message information 167
  - help information 125
  - information on a name 117
  - lines 165
  - values 81
- displaying expressions 81
  - diving, single click 198
  - dkill command 66
  - dlappend command 67
  - dlist command 68, 165
  - dlist, number of lines displayed 165
  - DLL Do Query on Load list 246
  - DLL Don't Query on Load list 246
  - dll\_ignore\_prefix variables 180
  - dll\_stop\_suffix variables 180
  - dload command 71
  - dlopen 245
    - ask when loading 168
  - dmstat command 73
  - dnext command 75
  - dnexti command 77
  - done property 136
  - double-precision data format 177
  - dout command 79
  - down, *see* ddown command
  - dpid 162
  - dpid property 154
  - dpids property 145
  - dprint command 81
  - dptsets command 86
    - dpvm option 209, 217
  - dpvm variables 181
  - drerun command 89
  - drestart command 91
    - attaching automatically 91
    - attaching to processes 91
    - process state 91
  - drun command 66, 93
    - poe issues 95
    - reissuing 94
  - dset command 96
  - dstatus command 98
  - dstep command 100
    - iterating over focus 100
  - dstepli command 103
  - duhttp, *see* dunhold command
  - duid property 147, 154
    - dump\_core option 210
  - dump\_core variables 181
  - dunhold command 105
  - dunset command 106
  - duntil command 107
    - group operations 107
  - dup command 110
  - dwait command 112
  - dwatch command 113
  - dwhat command 117
  - dwhere command 121
    - levels 165
  - dworker command 123
  - dynamic library support limitations 248
  - dynamic variables 181
  - dynamically loaded libraries 245

## E

  - editor\_launch\_string variables 181
    - eliminating tab processing 69
  - en, *see* denable command
  - enable, *see* denable command
  - enabled property 131
  - enabled variables 193
  - enabling action points 23, 24, 50
  - enabling PVM support 212
  - ending debugging session 124
  - enum\_values property 156
  - environment variables
    - LD\_LIBRARY\_PATH 232, 234, 235
  - error message information 167
  - ERROR state 167
  - errorCodes command 81, 134
  - errorCodes variable 134
  - errors, raising 134
  - escape characters 26
  - evaluating functions 81
  - evaluation points, *see* dbreak
  - evaluations, suspended, flushing 51
  - exception data on HP Tru64 231
  - exception subcodes 81
  - exception, warning when thrown 192
  - executable property 147
  - EXECUTABLE\_PATH variable 29, 69, 163
  - executing as one instruction 77
  - executing as one statement 75

executing assembler instructions 103  
 executing source lines 100  
 execution  
   halting 64  
   resuming 43  
 execution location, displaying 121  
 execve() 231  
   calling 208  
   catching 180  
 exit command 124  
 expr command 81, 136  
 expression property 131, 136  
 expression system  
   AIX 249  
   Alpha 248  
   IRIX 249  
 expression values, printing 81  
 expressions, compiling 175  
 extensions for file names 191

## F

f, *see* dfocus command  
 fatal errors 239  
 -fg option 210  
 file name extensions 191  
 files, libdbfork.h 233  
 fixed\_font variables 193  
 fixed\_font\_family variables 194  
 fixed\_font\_size variables 194  
 floating point data format  
   double-precision 177  
   extended floating point 178  
   single-precision 179  
   SPARC 258  
 flush, *see* dflush command  
 flushing cache 39  
 flushing suspended evaluations 51  
 focus  
   *see also* dfocus command  
   commands 161, 201  
   default 54  
   defining 165  
 focus\_groups command 138  
 focus\_processes command 139

focus\_threads command 140  
 focus\_threads property 136  
 font variables 194  
 fonts 193  
   fixed 193, 194  
   ui 194, 198  
   ui font family 199  
   ui font size 199  
 force\_default\_cplus\_demangler variables 182  
 force\_default\_f9x\_demangler variables 182  
 force\_window\_position variables 194  
 -foreground option 210  
 fork() 231  
   calling 208  
   catching 180  
 Fortran demangler 176  
 functions  
   evaluating 81  
   setting breakpoints at 37

## G

g, *see* dgo command  
 geometry\_call\_tree variables 195  
 geometry\_cli position 195  
 geometry\_cli variables 195  
 geometry\_globals variables 195  
 geometry\_help variables 195  
 geometry\_memory\_stats variables 196  
 geometry\_message\_queue variables 196  
 geometry\_message\_queue\_graph variables 196  
 geometry\_modules variables 196  
 geometry\_process variables 196  
 geometry\_ptset variables 197  
 geometry\_pvm variables 197  
 geometry\_root variables 197  
 geometry\_thread\_objects variables 197  
 geometry\_variable variables 197  
 geometry\_variable\_stats variables 198  
 get verb

actionpoint command 131  
 expr command 136  
 group command 141  
 image command 144  
 process command 147  
 thread command 154  
 type command 156  
 global\_typenames variables 182  
 -global\_types option 210  
 go, *see* dgo command  
 goal breakpoint 101  
 gr, *see* dgroups command  
 group command 141  
 group members, stopping flag 166  
 group of interest 101  
 GROUP variable 163  
 group width stepping behavior 101  
 groups  
   accessing properties 141  
   adding 58  
   adding members 59  
   creating 58, 60  
   deleting 58, 60  
   intersecting 58, 60  
   listing 58, 60  
   naming 59  
   placing processes in 29  
   removing 61  
   removing members 58  
   returning list of 138  
   setting properties 141  
 GROUPS variable 28, 71, 164  
 groups, *see* dgroups command

## H

h, *see* dhalt command  
 halt, *see* dhalt command  
 halting execution 64  
 handling signals 213  
 handling user-level (M:N) thread packages 191  
 heap size 73  
 heap\_size property 147  
 held property 148, 154  
 help command 125

- help window position 195
  - hex2dec command 143
  - hexadecimal conversion 133
  - hold, *see* dhold command
  - holding processes 65
  - holding threads 32, 65
  - host ports 216
  - hostname
    - expansion 219
    - for tvdsvr 216
    - replacement 220
  - hostname property 148
  - HP Tru64 UNIX
    - /proc file system 238
    - linking to dbfork library 232
    - swap space 240
  - hp, *see* dhold command
  - HP-UX
    - architecture 254
    - shared libraries 245
    - swap space 240
  - ht, *see* dhold command
  - htp, *see* dhold command
- I**
- I/O redirection 89, 93
  - id property 131, 137, 141, 145, 148, 154, 156
  - ignore\_control\_c variables 183
  - ignoring libraries by prefix 209
  - image browser window position 195
  - image command 144
  - image\_id property 156
  - image\_ids property 148
  - image\_load\_callbacks variables 183
  - images
    - getting properties 145
    - setting properties 145
  - in\_setup variables 183
  - inet interface name 184
  - INFO state 167
  - information on a name 117
  - initialization file 128
  - initially\_suspended\_process property 137
  - input, blocking 112
  - inserting working threads 123
  - instructions, stepping 103
  - integer (64-bit) data format 179
  - integer data format
    - 32-bit 179
    - 8-bit 179
  - integer data formats
    - 16-bit 179
  - Intel-x86
    - architecture 262
    - floating-point registers 264
    - FPCR register 265
      - using 265
    - FPSR register 266
    - general registers 263
  - interface name for server 184
  - intersecting groups 58, 60
  - IRIX
    - /proc file system 238
    - linking to dbfork library 234
    - swap space 242
  - is\_dll property 145
- J**
- job\_t::launch 238
- K**
- k, *see* dkill command
  - kcc\_classes option 211
  - kcc\_classes variables 183
  - keep\_search\_dialog variables 198
  - kernel\_launch\_string variables 184
  - keys, remapping 248
  - keysym 248
  - kill, *see* dkill command
  - killing processes 66
- L**
- l, *see* dlist command
  - language property 131, 156
  - lappend, *see* dlappend command
  - launch string
    - for editor 181
    - for server (Sun only) 185
    - for Visualizer 192
  - Launch Strings page 192
  - launching
    - local server 184
    - processes 93
    - single process sever launch string 187
    - tvdsvr 215
    - Visualizer 192
  - lb option 211
  - length property 131, 156
  - levels for dwhere 165
  - levels, moving down 48
  - libdbfork.a 231
  - libdbfork.h file 233
  - libraries
    - dbfork 208
    - ignoring by prefix 209
    - loading by suffix 180
    - loading symbols from 181
    - not loading based on prefix 180
    - shared 243
  - library cache, flushing 39
  - library\_cache\_directory variables 184
  - line property 131
  - LINES\_PER\_SCREEN variable 164
  - linking to dbfork library 231
    - AIX 233
    - C++ and dbfork 233
    - HP Tru64 UNIX 232
    - IRIX 234
    - SunOS 5 235
  - Linux swap space 242
  - list location 48
  - list, *see* dlist command
  - listing groups 58, 60
    - using a regular expression 60
  - listing lines 165
  - lo, *see* dload command
  - load and loadbind 245
  - load, *see* dload command
  - loading
    - action points 23, 211
    - programs 71
    - symbols from shared libraries 181
    - tvd files 152

loading action point information  
24

local\_interface variables 184

local\_server variables 184

local\_server\_launch\_string variables 185

lockstep list element 163

lookup verb, image command  
144

lookup\_keys verb, image command 144

## M

machine instructions, stepping  
103

manager property 155

manager threads, running 100

mangler, overriding 208, 210

MAX\_LIST variable 68, 165

maxdsiz\_64 241

maximum data segment size 241

Maximum permissible rank field  
192

member\_type property 141

member\_type\_values property  
141

members property 141

memory

- data size 73
- heap 73
- stack 73
- text size 73

memory statistics window position 196

memory use 73

message queue graph window position 196

message queue window position 196

message verbosity variable 167

–message\_queue option 211

message\_queue variables 185

MIPS

- architecture 267
- delay slot instructions 272
- FCSR register 271
- using 272

floating-point registers 270

general registers 268

SR register 269

mkswap command 242

modules window position 196

more processing 81

more prompt 125, 164

mounting /proc file system 238

–mqd option 211

multiprocess programs, attaching to processes 29

## N

n, *see* dnext command

name property 145, 156

name, information about 117

namespaces 96

- TV::  
96
- TV::GUI::  
97
- using wildcards 96

naming the host 216

nested subroutines, stepping out of 79

new groups 60

newval variable in watchpoints  
113

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 option 211

–no\_ask\_on\_dlopen option 247

–no\_compiler\_vars option 208

–no\_dbfork option 208

–no\_dpvm option 210

–no\_dump\_core option 210

–no\_dynamic option 244

–no\_global\_types option 211

–no\_kcc\_classes option 211

–no\_message\_queue option 211

–no\_mqd option 211

–no\_parallel option 211

–no\_pvm option 212

–no\_user\_threads option 214

nodeid property 148

nw, *see* dnext command

## O

oldval variable in watchpoints  
113

Open (or raise) process window  
at breakpoint checkbox 198

Open process window on error  
signal check box 198

options

- tvdsvr
  - callback 215
  - serial 215
  - server 215
  - set\_pw 216
  - user\_threads 214

ou, *see* dout command

oul, *see* dout command

out, *see* dout command

ouw, *see* dout command

## P

p, *see* dprint command

p/t expressions 86

p/t set browser position 197

panes, width 190

–parallel option 211

parallel processes, attaching to  
28

parallel runtime libraries 185

parallel variables 185

Parallel Virtual Machine 181, 187,  
209

parallel\_attach variables 186

parallel\_stop variables 186

password checking 218

passwords 218

- generated by tvdsvr 216

–patch\_area\_base option 211

–patch\_area\_length option 212

PATH environment variable  
for tvdsvr 215

pc property 155

Plant in share group checkbox  
166, 188

platform variables 186

pop\_at\_breakpoint variables 198

pop\_on\_error variables 198

- popping Process Window on error variable 198
- port 4142 218
- port number 217
  - for tvdsvr 216
  - replacement 220
  - searching 217
- port option 217
- ports on host 216
- PowerPC
  - architecture 258
  - floating-point registers 260
  - FPSCR register 260
    - using the 262
  - FPSCR register, using 262
  - general registers 258
  - MSR register 259
- preserving IDs in checkpoint 41
- print, *see* dprint command
- printing expression values 81
- printing information about current state 98
- printing registers 83
- printing slices 82
- printing variable values 81
- proc file system problems 238
- Process > Startup command 57
- process barrier breakpoint, *see* barrier breakpoint
- process command 147
- process groups, *see* groups
- process information, saving 41
- process list element 163
- process objects, creating new 71
- process statistics 73
- process width stepping behavior 100
- process window position 196
- process/thread sets
  - changing 54
- process\_load\_callbacks variable 186
- process\_set checkpoint options 41
- processes
  - attaching to 28, 71
  - automatically acquiring 180
  - automatically attaching to 186
  - current status 98
  - detaching when exiting CLI 124, 126
  - detaching from 46
  - holding 65
  - killing 66
  - properties 147
  - releasing 105
  - releasing control 46
  - restarting 89, 93
  - returning list of 139
  - starting 89, 93
  - terminating 66
- program control groups, placing processes in 29
- program stepping 100
- program variable, changing value 26, 41, 61, 65, 79, 91, 105, 107
- programs, loading 71
- PROMPT variable 165
- prompting when screen is full 81
- properties
  - address 131
  - clusterid 147
  - continuation\_sig 154
  - count 141
  - data\_size 145
  - done 136
  - dpid 154
  - dpids 145
  - duid 147, 154
  - enabled 131
  - enum\_values 156
  - executable 147
  - expression 131, 136
  - focus\_threads 136
  - heap\_size 147
  - held 148, 154
  - hostname 148
  - id 131, 137, 141, 145, 148, 154, 156
  - image\_id 156
  - image\_ids 148
  - initially\_suspended\_process 137
  - is\_dll 145
  - language 131, 156
  - length 131, 156
  - line 131
  - manager 155
  - member\_type 141
  - member\_type\_values 141
  - members 141
  - name 145, 156
  - nodeid 148
  - pc 155
  - prototype 157
  - rank 157
  - result 137
  - satisfaction\_group 132
  - share 132
  - sp 155
  - stack\_size 148
  - stack\_vm\_size 148
  - state 148, 155
  - state\_values 148, 155
  - status 137
  - stop\_when\_done 132
  - stop\_when\_hit 132
  - struct\_fields 157
  - syspid 148
  - systid 155
  - text\_size 145, 149
  - threadcount 149
  - threads 149
  - type 141
  - type\_transformations 145
  - type\_values 132, 141
  - vm\_size 149
- properties verb
  - actionpoint command 131
  - expr command 136
  - group command 141
  - image command 145
  - process command 147
  - thread command 154
  - type command 156
- prototype property 157
- PTSET variable 165
- ptsets, *see* dptsets

- PVM 217
  - pvm option 212, 217
  - pvm variables 187
  - pvm window position 197
  - pxdb command 245
  - pxdb64 command 245
- Q**
- qualifying symbol names 68
  - quit command 126
  - quotation marks 26
- R**
- r option 212
  - r, *see* drun command
  - raising errors 134
  - rank property 157
  - reading action points file 23
  - reenabling action points 50
  - registers
    - Alpha FPCR 253
    - floating-point
      - Alpha 253
      - Intel-x86 264
      - MIPS 270
      - PowerPC 260
      - SPARC 275
    - general
      - Alpha 252
      - Intel-x86 263
      - MIPS 268
      - PowerPC 258
      - SPARC 273
    - Intel-x86 FPCR 265
      - using the 265
    - Intel-x86 FPSR 266
    - MIPS FCSR 271
      - using the 272
    - MIPS SR 269
    - Power FPSCR 260
    - Power MSR 259
    - PowerPC FPSCR 260
      - using 262
    - PowerPC FPSCR,
      - using 262
    - PowerPC MSR 259
    - printing 83
    - SPARC FPSR 275
      - SPARC FPSR, using 276
      - SPARC PSR 274
  - registers, using in evaluations 37
  - release 162
  - releasing control 46
  - releasing processes and threads
    - 31, 105
  - remapping keys 248
  - remote debugging, tvdsrv command syntax 215
  - remote option 212
  - remote systems
    - debugging 39
  - removing
    - aliases 128
    - group member 58
    - groups 61
    - variables 106
    - worker threads 123
  - remsh command 219
  - replacement characters 219
  - replacing tabs with spaces 166
  - rerun, *see* rerun command
  - respond 150
  - restart, *see* drestart command
  - restarting processes 89, 93
  - restoring variables to default values 106
  - result property 137
  - resuming execution 37, 43, 57, 66
  - returning error information 134
  - root path 166
    - of TotalView 166
  - Root Window position 197
  - routines, stepping out of 79
  - rr, *see* drerun command
  - rsh command, with tvdsrv 188
  - run, *see* drun command
  - running to an address 107
- S**
- s, *see* dstep command
  - satisfaction set 32, 162, 172
  - satisfaction\_group property 132
  - save\_window\_pipe\_or\_filename
    - variables 187
  - saved position
    - Call Tree Window 195
    - CLI Window 195
    - Help Window 195
    - Image Browser Window 195
    - Memory Statistics Window 196
    - Message Queue Graph Window 196
    - Message Queue window 196
    - Modules Window 196
    - P/T Set Browser Window 197
    - Process Window 196
    - PVM Window 197
    - Root Window 197
    - Thread Objects Window 197
    - Variable Window 197
  - saving action point information
    - 24
  - saving action points 23
  - saving process information 41
  - screen size 164
  - search dialog, remaining displayed 198
  - search paths 163
  - search\_case\_sensitive variables 187
  - search\_port option 217
  - searching
    - case sensitive 187
    - wrapping 192
  - serial line connection 217
  - serial option 212, 215, 217
  - server launch command 219
  - server option 215, 218
  - server\_launch\_enabled variables 187
  - server\_launch\_string variables 187
  - server\_launch\_timeout variables 188
  - servers, number of 220
  - set verb
    - actionpoint command 131
    - group command 141
    - image command 145
    - process command 147
    - thread command 154

- type command 156
- set, *see* dset command
- set\_pw option 216, 218
- set\_pws option 218
- setting lines between more prompts 164
- setting terminal properties 127
- SGROUP variable 165
- share groups
  - share group variable 165
- share list element 164
- share property 132
- SHARE\_ACTION\_POINT variable 165
- share\_action\_point variables 188
- share\_in\_group flag 165
- shared libraries 243
  - HP-UX 245
- shared libraries, loading symbols from 181
- shm option 213, 214
- showing current status 98
- showing Fortran compiler variables 175
- si, *see* dstepi command
- SIGINT 183
- signal\_handling\_mode option 213
- signal\_handling\_mode variable 188
- signal\_handling\_mode option 213
- signals, handling in TotalView 213
- sil, *see* dstepi command
- SILENT state 167
- single process server launch 187
- single\_click\_dive\_enabled variables 198
- siw, *see* dstepi command
- sl, *see* dstep command
- slices, printing 82
- source code, displaying 68
- source\_pane\_tab\_width variables 190
- source\_process\_startup command 152
- sourcing tvd files 152
- sp property 155
- SPARC
  - architecture 273
  - floating-point format 258
  - floating-point registers 275
  - FPSR register 275
    - using 276
  - general registers 273
  - PSR register 274
- spell\_correction variable 190
- st, *see* dstatus command
- stack frame 68
  - moving down through 48
- stack memory 73
- stack movements 110
- stack, unwinding 51
- stack\_size property 148
- stack\_vm\_size property 148
- starting a process 89, 93
- Startup command 57
- start-up file, tvdinit.tvd 19
- state property 148, 155
- state\_values property 148, 155
- status property 137
- status, *see* dstatus command
- stderr redirection 89, 93
- stdin redirection 89, 93
- stdout redirection 89, 93
- step, *see* dstep command
- stepi, *see* dstepi command
- stepping
  - group width behavior 101
  - machine instructions 77, 103
  - process width behavior 100
  - see also* dnxt command, dn-exti command, dstep command, and dstepi command
  - thread width behavior 100
  - warning when exception thrown 192
- stop group breakpoint 37
- STOP\_ALL variable 36, 115, 161, 166
- stop\_all variable 190
- stop\_group flag 166
- stop\_relives\_on\_proc\_error variables 190
- stop\_when\_done command-line option 162
- stop\_when\_done property 132
- stop\_when\_hit property 132
- stopped process, responding to resume commands 32
- stopping execution 64
- stopping group members flag 166
- stopping the control group 190
- string length format 180
- strings, assigning values to 26
- struct\_fields property 157
- structure definitions in KCC 183
- stty command 127
- suffixes variable 191
- SunOS 5
  - /proc file system 238
  - key remapping 248
  - linking to dbfork library 235
  - swap space 242
- sw, *see* dstep command
- swap command 242
- swap space 239, 242
  - AIX 241
  - HP Tru64 240
  - HP-UX 240
  - IRIX 242
  - Linux 242
  - SunOS 242
- swapon command 242
- symbol name qualification 68
- symbols, interpreting 26
- syspid property 148
- system variables, *see* CLI variables
- sysuid property 155

## T

- tab processing 69
- TAB\_WIDTH variable 69, 166
- tabs, replacing with spaces 166
- target processes 64
  - terminating 66
- target property 157
- terminal properties, setting 127

- terminating debugging session 124
  - terminating processes 66
  - text size 73
  - text\_size property 145, 149
  - thread barrier breakpoint, *see* barrier breakpoint
  - thread command 154
  - thread groups, *see* groups
  - thread list element 164
  - thread objects window position 197
  - thread of interest 100, 107
  - thread width stepping behavior 100
  - threadcount property 149
  - threads
    - barriers 33
    - creating 57
    - current status 98
    - destroyed when exiting CLI 124, 126
    - getting properties 154
    - holding 32, 65
    - list variable 166
    - releasing 105
    - returning list of 140
    - setting properties 154
  - threads property 149
  - THREADS variable 166
  - totalview command 207
    - description 207
    - options 207
    - synopsis 207
    - syntax and use 207
  - TotalView Debugger Server 28, 41
  - TotalView executable 166
  - TotalView GUI version 199
  - TotalView version 191
  - totalview/lib\_cache subdirectory 39
  - TOTALVIEW\_ROOT\_PATH variable 166
  - TOTALVIEW\_TCLLIB\_PATH variable 167
  - TOTALVIEW\_VERSION variable 167
  - triggering breakpoints 37
  - troubleshooting xiv
  - ttf variable 191
  - TV::namespace 96
  - TV::actionpoint command 131
  - TV::errorCodes command 134
  - TV::expr command 81, 136
  - TV::focus\_groups command 138
  - TV::focus\_processes command 139
  - TV::focus\_threads command 140
  - TV::group command 141
  - TV::GUI::namespace 97
  - TV::hex2dec command 143
  - TV::image command 144
  - TV::process command 147
  - TV::respond command 150
  - TV::source\_process\_startup command 152
  - TV::thread command 154
  - TV::type command 156
  - tvd files 152
  - TVD.breakpoints file 24, 171
  - tvdinit.tvd start-up file 19, 128
  - tvdsvr command 215, 216, 219
    - description 215
    - options 216
    - password 216
    - PATH environment variable 215
    - synopsis 215
    - use with DPVM applications 217
    - use with PVM applications 217
  - tvdsvr.conf 218
  - TVDSVRLAUNCHCMD environment variable 219
  - type command 156
  - type names 182
  - type property 132, 141, 157
  - type transformation variable 191
  - type transformations
    - applied to image 145
  - type\_transformations property 145
  - type\_values property 132, 141, 157
- ## U
- u, *see* dup command
  - uhp, *see* dunhold command
  - uht, *see* dunhold command
  - ui\_font variables 198
  - ui\_font\_family variables 199
  - ui\_font\_size variables 199
  - un, *see* duntill command
  - unalias command 128
  - unconditional watchpoints 113
  - unhold, *see* dunhold command
  - unl, *see* duntill command
  - unset, *see* dunset command
  - until, *see* duntill command
  - unw, *see* duntill command
  - unwinding the stack 51
  - up, *see* dup command
  - user\_threads option 214
  - user\_threads variables 191
  - user-defined commands 19
  - user-level (M:N) thread packages 191
  - using quotation marks 26
  - using\_color variables 199
  - using\_text\_color variables 199
  - using\_title\_color variables 199
- ## V
- value for newly created action points 165
  - values, printing 81
  - Variable Window position 197
  - variables
    - assigning command output to 21
    - changing values 26, 41, 61, 65, 79, 91, 105, 107
    - default value for 96
    - printing 81
    - removing 106
    - watched 114
    - watching 113
  - VERBOSE variable 167
  - verbosity option 214, 219

- verbosity setting replacement
  - character 221
- version variables 191, 199
- version, TotalView 167
- vfork()
  - calling 208
  - catching 180
- viewing CLI variables 96
- visualizer\_launch\_enabled variables 192
- visualizer\_launch\_string variables 192
- visualizer\_max\_rank variables 192
- vm\_size property 149
- W**
- w, *see* `dwhere` command
- wa, *see* `dwatch` command
- wait, *see* `dwait` command
- warn\_step\_throw variables 192
- WARNING state 167
- watch, *see* `dwatch` command
- watchpoints 113
  - `$newval` 113
  - `$oldval` 113
  - conditional 113
  - information not saved 24
  - length of 114
  - supported systems 114
- WGROUP variable 167
- wh, *see* `dwhat` command
- what, *see* `dwhat` command
- When barrier done, stop value 162
- When barrier hit, stop value 161
- where, *see* `dwhere` command
- window position, forcing 194
- worker group list variable 167
- worker threads 167
  - inserting 123
  - removing 123
- worker, *see* `dworker` command
- workers list element 164
- `-working_directory` option 219
- wot, *see* `dworker` command
- wrap\_on\_search variables 192