TotalView Graphic User Interface Reference Guide

version 8.8

TotalView Technologies

Book Overview

1 Root Window 1

2 Process Window 49

3 Variable Window 125

4 Visualizer Window 155

5 Fortran Modules Window 161

6 Program Browser Window 169

7 Message Queue Window 175

8 PVM Tasks Window 183

9 Thread Objects Window 191

10 Expression List Window 205

11 Other Topics 217

Contents

1 Root Window

Root Window Pages 1
File Menu Commands 4
    File > New Program 4
        Program Tab 4
        Arguments Tab 8
        Standard I/O Tab 9
        Parallel Tab 11
    File > Search Path 12
    File > Preferences 13
        Options Page 13
        Action Points Page 16
        Launch Strings Page 18
        Single Debug Server Launch 18
        Visualizer Launch 21
        Source Code Editor 22
        Bulk Launch Page 23
        Dynamic Libraries Page 26
        Parallel Page 29
        Fonts Page 31
        Formatting Page 32
        Pointer Dive Page 35
        ReplayEngine Page 36
    File > Save Pane 37
    File > Exit 38
Edit Menu Commands 38
    Edit > Undo 39
    Edit > Cut 39
    Edit > Copy 39
    Edit > Paste 39
    Edit > Delete 39
    Edit > Find 40
    Edit > Find Again 40
View Menu Commands 41

View > Dive 41
View > Dive in New Window 41
View > Expand All 41
View > Collapse All 41
View > Display Managers 42
View > Display Exited Threads 42
Tools Menu Commands 42
Tools > Restart Checkpoint 42
Tools > Open MemoryScape 44
Tools > Event Log 44
Tools > Warnings 45
Tools > PVM Tasks 45
Tools > Command Line 45
Window Menu Commands 46
Window > Update 46
Window > Update All 46
Window > Memorize 46
Window > Memorize all 47

2 **Process Window**

Process Window Panes 49
Stack Trace Pane 50
Stack Frame Pane 50
Source Pane 50
Threads Tab 51
Processes/Ranks Tab 51
Action Points Tab 52
File Menu Commands 53
File > New Program 53
File > Search Path 53
File > Signals 54
File > Preferences 55
File > Open Source 56
File > Edit Source 56
File > Save Pane 56
File > Rescan Libraries 57
File > Close Relatives 57
File > Close 57
File > Exit 58
Edit Menu Commands 58
Edit > Undo 58
Edit > Cut 58
Edit > Copy 59
Edit > Paste 59
Edit > Delete 59
Edit > Find 59
Edit > Find Again 60
View Menu Commands 60
View > Dive 61
View > Dive in New Window 61
View > Undive 62

View > Redive 62
View > Reset 62
View > Lookup Function 62
View > Lookup Variable 63
View > Next Process 64
View > Previous Process 64
View > Source As > Source 65
View > Source As > Assembler 65
View > Source As > Both 65
View > Assembler > Symbolically 65
View > Assembler > By Address 66
View > Display Managers 66
Group Menu Commands 67
Group > Go 67
Group > Halt 68
Group > Next 68
Group > Step 68
Group > Out 69
Group > Run To 69
Group > Next Instruction 70
Group > Step Instruction 71
Group > Hold 71
Group > Release 71
Group > Attach Subset 71
Group > Custom Groups 73
Group > Detach 74
Group > Restart 74
Group > Kill 74
Process Menu Commands 75
Process > Go 75
Process > Halt 76
Process > Next 76
Process > Step 76
Process > Out 77
Process > Run To 77
Process > Next Instruction 78
Process > Step Instruction 78
Process > Hold 79
Process > Hold Threads 79
Process > Release Threads 80
Process > Create 80
Process > Detach 80
Process > Startup Parameters 80
Debugging Options Page 81
Arguments Page 81
Standard I/O Page 83
Parallel Page 84
Thread Menu Commands 86
Thread > Go 86
Thread > Halt 86
Thread > Next 87

Thread > Step 87
Thread > Out 87
Thread > Run To 88
Thread > Next Instruction 89
Thread > Step Instruction 89
Thread > Set PC 89
Thread > Hold 89
Thread > Continuation Signal 90
Action Point Menu 91
Action Point > Set Breakpoint 91
Action Point > Set Barrier 91
Action Point > At Location 91
Action Point > Create Watchpoint 93
Action Point > Enable 93
Action Point > Disable 93
Action Point > Delete 93
Action Point > Properties 93
Action Point > Suppress All 99
Action Point > Delete All 99
Action Point > Load All 99
Action Point > Save All 100
Action Point > Save As 100
Debug Menu Commands 100
Debug Enable ReplayEngine 101
Debug > Previous 101
Debug > Unstep 101
Debug > Caller 101
Debug > Back To 101
Debug > Live 101
Debug > Enable Memory Debugging 102
Debug > Stop on Memory Errors 102
Debug > Open MemoryScape 102
Debug > Heap Baseline >Set Heap Baseline (in Process) 102
Debug > Heap Baseline> Set Heap Baseline (in Group) 102
Debug > Heap Baseline >Heap Change Summary 103
Debug > Memory Block Properties 105
Debug > Memory Event Details 106
Tools Menu Commands 108
Tools > Evaluate 108
Tools > Expression List 109
Tools > Program Browser 109
Tools > Fortran Modules 109
Tools > Call Graph 109
Tools > Debugger Loaded Libraries 110
Tools > Event Log 111
Tools > Warnings 112
Tools > Thread Objects 112
Tools > Message Queue 113
Tools > Message Queue Graph 113
Options Dialog Box 115
Tools > Create Checkpoint 117

Tools > Restart Checkpoint 120
Tools > PVM Tasks 122
Tools > Global Arrays 122
Tools > Command Line 122
Window Menu Commands 123
Window > Update 123
Window > Update All 123
Window > Duplicate 123
Window > Memorize 124
Window > Memorize all 124
Window > Root 124

3 Variable Window

Variable Window Overview 125
File Menu Commands 131
File > Save Pane 132
File > Close 132
File > Exit 132
Edit Menu Commands 133
Edit > Undo 133
Edit > Reset Defaults 133
Edit > Cut 134
Edit > Copy 134
Edit > Paste 134
Edit > Delete 134
Edit > Find 134
Edit > Find Again 135
View Menu Commands 135
View > Dive 136
View > Dive in New Window 136
View > Dive In All 136
View > Expand All 137
View > Collapse All 137
View > Undive 137
View > Undive All 137
View > Redive 137
View > Redive All 138
View > Freeze 138
View > Lock Address 138
View > Show Across > None 139
View > Show Across > Process 139
View > Show Across > Thread 139
View > Compilation Scope > Fixed 139
View > Compilation Scope > Floating 139
View > Loader Symbols 140
View > Padding 140
View > Break At Newlines 140
View > Examine Format > None 140
View > Examine Format > Structured 140
View > Examine Format > Raw 141
View > Block Status 142

Tools Menu Commands 142
Tools > Create Watchpoint 142
Tools > Add to Expression List 146
Tools > Add to Block Properties 146
Tools > Visualize 148
Tools > Visualize Distribution 148
Tools > Statistics 149
Tools > Attach Subset (Array of Ranks) 150
Window Menu Commands 152
Window > Update 152
Window > Update All 152
Window > Duplicate 152
Window > Memorize 152
Window > Memorize all 153
Window > Root 153

4 Visualizer Window

Dataset Window 156
File Menu Commands 156
View Menu Commands 156
Options Menu Command 156
View Window 157
File Menu Commands 158

5 Fortran Modules Window

File Menu Commands 162
File > Close Similar 162
File > Close 162
Edit Menu Commands 163
Edit > Undo 163
Edit > Cut 163
Edit > Copy 163
Edit > Paste 163
Edit > Delete 164
Edit > Find 164
Edit > Find Again 165
View Menu Commands 165
View > Dive 165
View > Dive in New Window 165
Window Menu Commands 166
Window > Update 166
Window > Update All 166
Window > Memorize 166
Window > Memorize all 166
Window > Root 167

6 Program Browser Window

File Menu Commands 169
File > Close Similar 169
File > Close 169
File > Exit 170

Edit Menu Commands 171
Edit > Undo 171
Edit > Cut 171
Edit > Copy 171
Edit > Paste 171
Edit > Delete 172
Edit > Find 172

Edit > Find Again 173
View Menu Commands 173
View > Dive 173
View > Dive in New Window 173
Window Menu Commands 173
Window > Update 174
Window > Update All 174
Window > Memorize 174
Window > Memorize all 174
Window > Root 174

7 Message Queue Window

Message Queue Window Overview........ 175
Message Operations 177
File Menu Commands 178
File > Close Similar 178
File > Close 178
File > Exit 178
Edit Menu Commands 178
Edit > Undo 179
Edit > Cut 179
Edit > Copy 179
Edit > Paste 179
Edit > Delete 179
Edit > Find 180
Edit > Find Again 181
View Menu Commands 181
View > Dive 181
View > Dive in New Window 181
Window Menu Commands 181
Window > Update 182
Window > Update All 182
Window > Memorize 182
Window > Memorize all 182
Window > Root 182

8 PVM Tasks Window

File Menu Commands 184
File > Close 184
File > Exit 184
Edit Menu Commands 185
Edit > Undo 185
Edit > Cut 185

Edit > Copy 185
Edit > Paste 186
Edit > Delete 186
Edit > Find 186
Edit > Find Again 187
View Menu Commands 187
View > Dive 187
View > Dive in New Window 187
Window Menu Commands 188
Window > Update 188
Window > Update All 188
Window > Memorize 188
Window > Memorize all 188
Window > Root 189

9 Thread Objects Window

HP Tru64 UNIX 191
Mutexes Page 191
Condition Variables 193
IBM AIX 194
Mutexes Page 194
Condition Variables Page 196
R/W Locks Page 196
Data Keys Page 198
File Menu Commands 198
File > Close Similar 199
File > Close 199
File > Exit 199
Edit Menu Commands 199
Edit > Undo 199
Edit > Cut 200
Edit > Copy 200
Edit > Paste 200
Edit > Delete 200
Edit > Find 201
Edit > Find Again 201
View Menu Commands 202
View > Dive 202
View > Dive ... in New Window 202
Window Menu Commands 202
Window > Update 203
Window > Update All 203
Window > Memorize 203
Window > Memorize all 203
Window > Root 203

10 Expression List Window

Expression List Window Overview 205
Entering Variables and Expression into the Expression List Window
205
Opening and Closing the Expression List Window 207

What Can You Enter in the Expression Column 207
Manipulating the Expression List Window 208
Multiprocess/Multithreaded Behavior 209
File Menu Commands 210
File > Preferences 210
File > Save Pane 210
File > Close Similar 211
File > Close 211
File > Exit 211
Edit Menu Commands 211
Edit > Undo 212
Edit > Reset Default 212
Edit > Cut 212
Edit > Copy 212
Edit > Paste 212
Edit > Delete 213
Edit > Delete Expression 213
Edit > Delete All Expressions 213
Edit > Duplicate Expression 213
Edit > Find 213
Edit > Find Again 214
View Menu Commands 214
View > Dive 214
Window Menu Commands 215
Window > Update 215
Window > Update All 215
Window > Duplicate 215
Window > Memorize 215
Window > Memorize all 215
Window > Root 216

11 Other Topics

Other Dialog Boxes 217
Ambiguous Line Dialog Box 217
Ambiguous Function Dialog Box 218
Process Load Libraries 218
Other Topics 218
MPICH 218
IBM PE 219
SGI MPI 220
RMS MPI 220
Linking Your Application With the Agent 220
Installing tvheap_mr.a 223
LIBPATH and Linking 223
Setting Search Paths Using TotalView Variables 225
Search Path Variables That You Can Set 225
TOTALVIEW_SRC 226
PATH 226
EXECUTABLE_PATH 226
Search Path Variables That TotalView Sets 226
COMPILATION_DIRECTORY_COMPONENT 227

COMPILATION_WORKING_DIRECTORY 227
COMPILATION_DIRECTORY 227
EXECUTABLE_DIRECTORY_COMPONENT 227
EXECUTABLE_WORKING_DIRECTORY 228
EXECUTABLE_DIRECTORY 228
TotalView Built-in Functions 228
Using the $tree() Function 229
Using the File > Search Path Dialog Box 229

chapter

1

Root Window

Root Window Pages

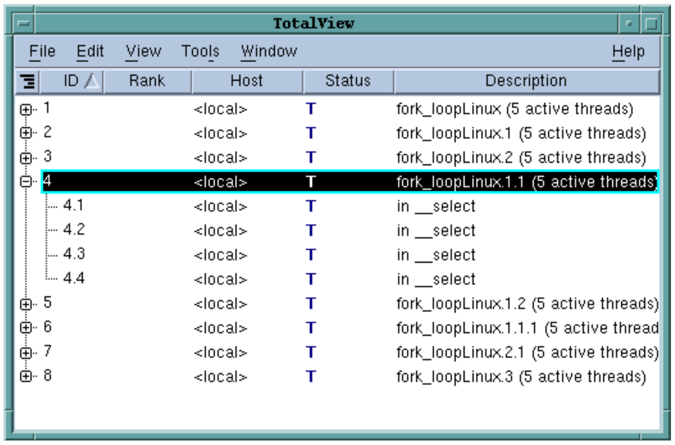
The Root Window contains a list of all the processes and threads you are currently debugging. Diving into an process or thread by double-clicking or using the **View** > **Dive** command opens a window for the selected process. If a window already exists for that process, TotalView brings it to the front. Each process is shown with its PID (Process ID) and a short summary of its state.

TotalView can display the Attached Page in two ways. In the preceding figure, information is shown as a "tree". Selecting the hierarchy toggle button (☰) shifts the view from a structured to linear view. The difference between the views is that you cannot sort and aggregate the information in linear view. Sorting and aggregating is discussed later in this section.

When displaying data hierarchically, you can perform two operations that can't be performed when data is displayed linearly. You can:

- Selectively display information using the + or – indicators.
- Sort a column by clicking on a column header.

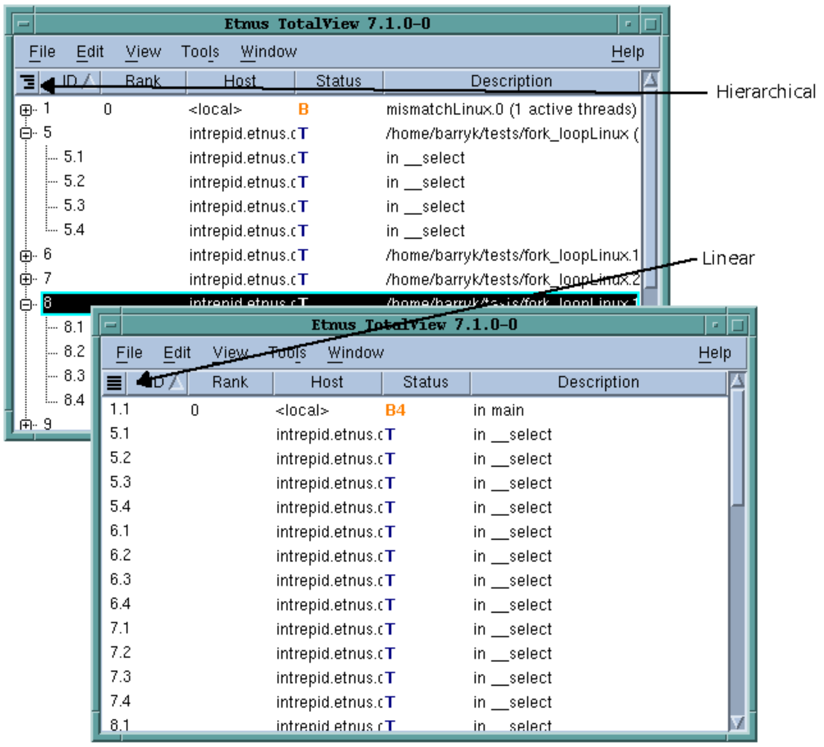
Figure 1: Root Window



ID	Rank	Host	Status	Description
1		<local>	T	fork_loopLinux (5 active threads)
2		<local>	T	fork_loopLinux.1 (5 active threads)
3		<local>	T	fork_loopLinux.2 (5 active threads)
4		<local>	T	fork_loopLinux.1.1 (5 active threads)
4.1		<local>	T	in __select
4.2		<local>	T	in __select
4.3		<local>	T	in __select
4.4		<local>	T	in __select
5		<local>	T	fork_loopLinux.1.2 (5 active threads)
6		<local>	T	fork_loopLinux.1.1.1 (5 active thread
7		<local>	T	fork_loopLinux.2.1 (5 active threads)
8		<local>	T	fork_loopLinux.3 (5 active threads)

Combining these two operations provides an easy way to see information about your program. For example, if you sort the information by clicking on the **Status** header, TotalView will group all attached progress by its status such as if it is being held, at a breakpoint, and so on. When they are aggregated like this, you can also display information selectively. See Figure 2.

Figure 2: Root Window: Showing the Attached Page



ID	Rank	Host	Status	Description
1	0	<local>	B	mismatchLinux.0 (1 active threads)
5		intrepid.etnus.c	T	/home/barryk/tests/fork_loopLinux (
5.1		intrepid.etnus.c	T	in __select
5.2		intrepid.etnus.c	T	in __select
5.3		intrepid.etnus.c	T	in __select
5.4		intrepid.etnus.c	T	in __select
6		intrepid.etnus.c	T	/home/barryk/tests/fork_loopLinux.1
7		intrepid.etnus.c	T	/home/barryk/tests/fork_loopLinux.2
8		intrepid.etnus.c	T	/home/barryk/tests/fork_loopLinux.1
8.1				
8.2				
8.3				
8.4				
9				

ID	Rank	Host	Status	Description
1.1	0	<local>	B4	in main
5.1		intrepid.etnus.c	T	in __select
5.2		intrepid.etnus.c	T	in __select
5.3		intrepid.etnus.c	T	in __select
5.4		intrepid.etnus.c	T	in __select
6.1		intrepid.etnus.c	T	in __select
6.2		intrepid.etnus.c	T	in __select
6.3		intrepid.etnus.c	T	in __select
6.4		intrepid.etnus.c	T	in __select
7.1		intrepid.etnus.c	T	in __select
7.2		intrepid.etnus.c	T	in __select
7.3		intrepid.etnus.c	T	in __select
7.4		intrepid.etnus.c	T	in __select
8.1		intrepid.etnus.c	T	in __select

You can control which columns TotalView displays by right-clicking anywhere within the header line. TotalView responds by displaying a context menu listing all possible choices. If a checkbox is selected within this list, TotalView displays the column. Click on the column name within this menu to change its display status.

The default Attached Page has the columns of information:

- **Expand/collapse indicator**

  This control lets you display or hide information. Clicking on a + indicator displays hidden information. Clicking on – hides the information by collapsing the display.

- **ID**

  A TotalView-created identifier. When displaying a thread, TotalView shows it using the thread index that it created. This number is arbitrary, so don't read anything into it.

  If you dive into a process, TotalView opens (or brings to the front) a Process Window focused on an arbitrary thread. If there is no open window associated with the process, TotalView opens a window containing the first thread in the process. If you dive into a thread, TotalView displays that thread in a Process Window

TotalView does not reuse these numbers when you restart your program or when processes and threads are deleted.

- **Rank**

  If you are debugging an MPI program, this column shows a process's rank within the program.
- **Host**

  The name of the computer upon which the process is executing.
- **Status**

  A one-letter state indicator as follows:

Character and Meaning	Definition
blank (Not begun)	*Process only*: the process has not begun running.
Bnn (Breakpoint)	*nn* is the ID of the breakpoint if it is a thread. *Process*: one or more threads are stopped at a breakpoint. *Thread*: the thread is stopped at a breakpoint. If only one thread is at a breakpoint, or all threads are at the same breakpoint, TotalView displays the ID of that breakpoint for the process as well as the thread. If the process's threads are at different breakpoints, TotalView displays a "*" for the process's breakpoint ID.
E (Error)	The **Error** state usually indicates that your program received a fatal signal from the operating system. Signals such as **SIGSEGV**, **SIGBUS**, and **SIGFPE** can indicate an error in your program. *Process*: one or more threads are in the **Error** state. *Thread*: stopped because of an error.
H (Held)	Either you or TotalView is holding the process of a thread. *Holding* means that the process or thread cannot run until it is released. You can explicitly release it, or TotalView can release it if the condition that caused it to be held is satisfied.
K (Kernel)	*Thread only*: the thread is executing inside the kernel (that is, it made a system call); when a thread is in the kernel, the operating system does not allow TotalView to view the thread's full state.
M (Mixed)	*Process only*: either some threads in the process are running or the process is expecting some of its threads to stop.
R (Running)	*Process*: all threads in the process are running or can run. *Thread*: running or can run.
T (Stopped)	*Process*: one or more threads are stopped, but none are in the **At Breakpoint** state. *Thread*: while the thread is stopped, it is not at a breakpoint and no error occurred.
W (Watchpoint)	*Process*: one or more threads are stopped at a watchpoint. *Thread*: stopped at a watchpoint.

- **Description**
  Information about the process or thread. If this is a process description, TotalView displays summary and status information. If this is a thread description, it indicates the last place in which the PC was seen. This is the place where the thread was last stopped.

File Menu Commands

The commands on the **File** pulldown are:

- **File > New Program** on page 4
- **File > Search Path** on page 12
- **File > Preferences** on page 13
- **File > Save Pane** on page 37
- **File > Exit** on page 38

File > New Program

Use this dialog box to specify the name of a new executable file, to attach to an existing running process or core file, or to specify the location of the process.

The **New Program** dialog box allows you to load another program. It also lets you manage startup information.

This dialog box has four tabs:

- "*Program Tab*" on page 4
- "*Arguments Tab*" on page 8
- "*Standard I/O Tab*" on page 9
- "*Parallel Tab*" on page 11

Program Tab

The actions you can perform using the Program tab are:

- Start a new process
- Attach to an existing process
- Open a core file

Depending upon the kind of process, you can supply other information:

- You can define startup arguments and environment variables in the Arguments tab.
- You can define how your program interacts with standard I/O.
- You can describe how your MPI program starts.

If the process has children that called **execve()**, TotalView tries to determine each child's executable. If TotalView cannot determine the executables for the children, you need to delete (kill) the parent process and start it under TotalView control.

If the executable is a multiprocess program, TotalView asks if you want to attach to all relatives of the process. To examine all processes, select **Yes**.

This is the default behavior. You can change this behavior by using commands within the **File > Preference***'s* **Parallel Tab**.

Start a new process: To load a program, type the program's name in the **Program** field. If you had previously loaded the program into TotalView, its name is remembered and you may find it on the pull-down list contained within this field. The name you enter can either be a full or relative path name.

If instead of typing the program's name, you want to use a file selection dialog box to locate it, press the **Browse** button.

If you enter just a file name, TotalView searches for the file in the directories you specified with the **File > Search Path** command and in all the directories named in your **PATH** environment variable.

If the program is to run on a different machine, you can select the computer's name from the **On host** field. If this machine isn't on the list, select the **Add host** button. After TotalView displays a dialog box, enter the machine's name or an IP address.

By selecting the Enable memory debugging program, you tell TotalView to set up your program for Memory Debugging. Enabling it here is the same as enabling it within the Memory Debugger or using the Process Window's **Debug> Enable Memory Debugging** command.

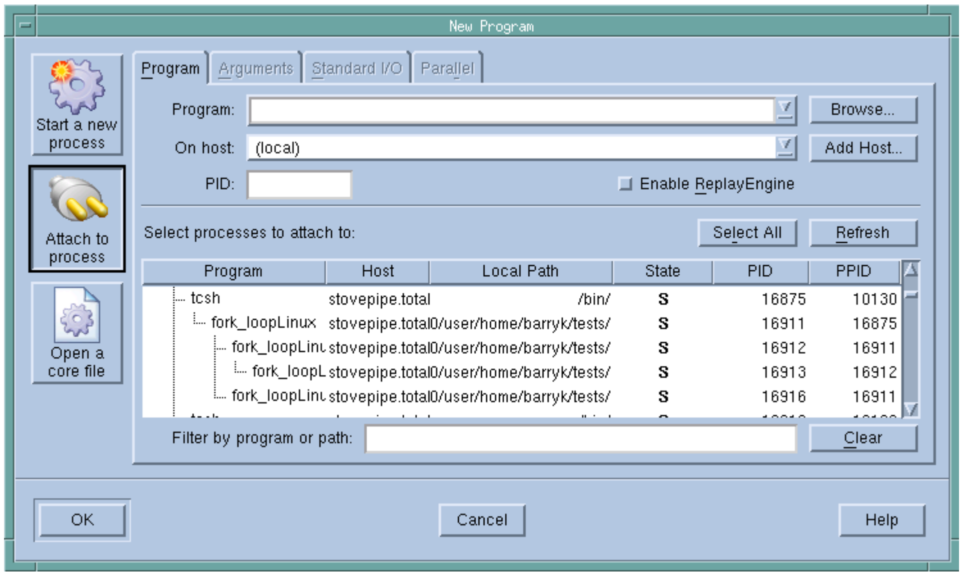
The **Enable memory debugging** and **Halt on memory errors** check boxes perform the same functions as the **Enable memory debugging** and **On memory event, halt execution** checkboxes do within the Advanced Options on MemoryScape's Memory Debugging Options page. This is the equivalent of the basic Low setting.

The **Enable ReplayEngine** check box is visible only on Linux-x86 and Linux-86-64 platforms. If you do not have a license for ReplayEngine, enabling the check box has no effect and TotalView displays an error message when your program begins executing. Selecting this check box tells TotalView that it should instrument your code so that you can move back to previously executed lines.

Attach to process: After selecting **Attach to process**, TotalView shows a list of all your processes on the local host. If TotalView is connected to multiple hosts, the **On host** field contains an additional **all hosts** line. Select **all hosts** to see the processes from all of the hosts, useful for attaching to a multiprocess program running on several different hosts..

In the displayed list, processes to which TotalView is already attached are shown in gray. The processes displayed in black are not currently running

Figure 3: Attach to an Existing Process



Program:

On host: (local)

PID:

Enable ReplayEngine

Select processes to attach to:

Program	Host	Local Path	State	PID	PPID
tcsh	stovepipe.total	/bin/	S	16875	10130
fork_loopLinux	stovepipe.total	0/user/home/barryk/tests/	S	16911	16875
fork_loopLinu	stovepipe.total	0/user/home/barryk/tests/	S	16912	16911
fork_loopL	stovepipe.total	0/user/home/barryk/tests/	S	16913	16912
fork_loopLinu	stovepipe.total	0/user/home/barryk/tests/	S	16916	16911

Filter by program or path:

under TotalView control. To attach TotalView to any of these processes, select the process or processes, then press the **OK** button.

Use the **Filter by program or path** field to view only the processes with a program or local path name matching the name you enter, helpful for finding a process or processes in which you are interested.

In some cases, the name provided to TotalView by your operating system may not be the actual name of the program. In this case, you will not be able to simply select the name and press the **OK** button. Instead, you should

- Determine what its actual name is by using a command such as **ls** in a shell window.
- Select the name as this will fill in much of the program's name.
- Move to the **Program** control, and type its actual name, then press the **OK** button.

If you wish to attach to a multiprocess program, you can either pick up the processes one at a time or you can restart the program under TotalView control so that the processes are automatically picked up as they are forked. In most cases, this requires you to link your program with the **dbfork** library. This is discussed in the *TotalView Reference Guide*.

If the process you are attaching to is one member of a collection of related processes created with **fork()** calls, TotalView asks if you want to also attach to all of its relatives. If you answer **yes**, TotalView attaches to all the process's ancestors, descendants, and cousins.

You can control how TotalView attaches to processes by using the commands in the Parallel Tab within the **File > Preferences** dialog box.

If some of the processes in the collection have called exec(), TotalView tries to determine the new executable file for the process. If TotalView appears to read the wrong file, you should start over, compile the program using the dbfork *library, and start the program under TotalView control.*

The information in this area has the following columns:

- **Program**: The name of the executing program. Notice that TotalView indents some names. This indentation indicates the parent/child relationship within the UNIX process hierarchy.
- **Host**: The name of the machine upon which the program is executing.
- **Local Path**: The program's path on the local machine, that is, the machine where TotalView is running. A process running on a remote host may be executing a program from a directory path valid only on the remote host. TotalView maps the remote path name of the program to a local path name and searches for the program using the File > Search Path > Programs dialog. The remote path name is tried first, but if that search fails, it is retried using only the program name. TotalView makes sure that the local path of the program is compatible with the architecture of the host on which the process is running. For example, if the program name is "/bin/bash," the local host is Linux-x86_64, and the remote host is Linux-Power, TotalView will not use the local Linux-x86_64 "/bin/bash" because it is not compatible with the remote Linux-Power host. If TotalView cannot find an architecture-compatible program, local path is left empty.
- **State**: A letter indicating the program's state, as follows:

Character and Meaning	Definition
I (Idle)	Process has been idle or sleeping for *more* than 20 seconds.
R (Running)	Process is running or can run.
S (Sleeping)	Process has been idle or sleeping for *less* than 20 seconds.
T (Stopped)	Process is stopped.
Z (Zombie)	Process is a "zombie"; that is, it is a child process that has terminated and is waiting for its parent process to gather its status.

- **PID**: The operating system program ID.
- **PPID**: The parent program's ID.

If you attach to multiple processes, TotalView places all of them into the same control group, allowing you to stop and start them as a group.

For information on entering information in the Program and On host areas, see "*Start a new process*" on page 5.

Open a core file: The information you specify when opening a core file is nearly identical to that you enter a regular process. It differs in that you also have to enter the name of the core file. You must enter a program name in the Program field in addition to the Core file field because TotalView cannot know if this program is actually associated with the process.

You can use the **Browse** button to search the file system for the core file.

When naming a core file, you can use glob-style wild cards.

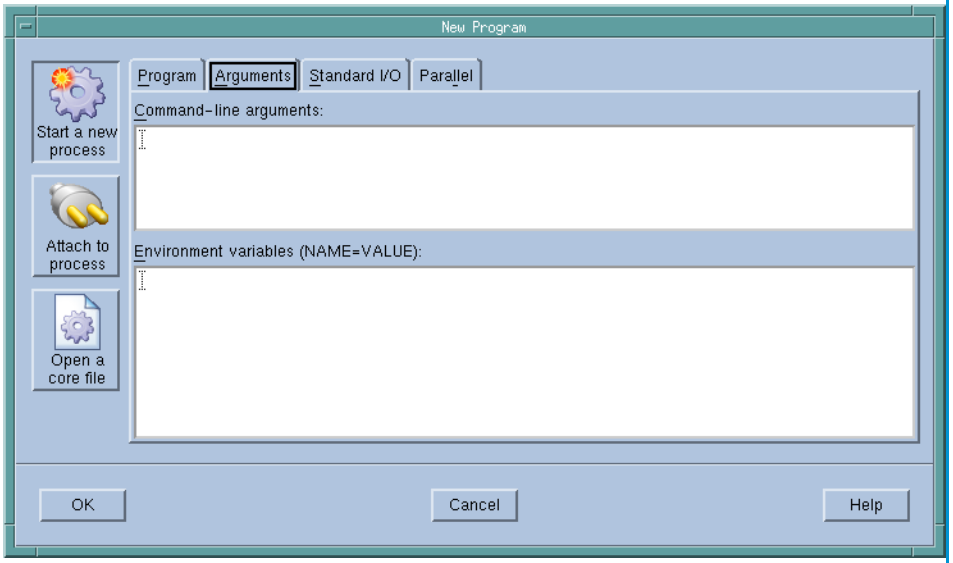
For information on the Program and host fields, see "*Start a new process*" on page 5.

Arguments Tab

use this tab to define both the arguments and environment variables that TotalView passes to a process when it is next launched.

Command-line arguments: The arguments typed in this area are those that you would have entered if you were starting the program from a shell. If you were directly starting your program under TotalView control, these arguments are those you would enter using the TotalView **–a** command-line option.

Figure 4: File > New Program Dialog Box: Showing Arguments Tab



This tab is identical to the Arguments tab that TotalView displays when you select the **Process > Startup Parameters** *command.*

TotalView uses these arguments whenever it starts your program. In contrast, if you need to pass arguments to a starter process such as **mpirun** or **poe**, enter those arguments in the Parallel tab.

You can enter arguments in two ways:

- Place them on separate lines.
- Separate them with blanks.

If either case, an argument must be entered on one line. TotalView will rewrap what you type, so do not be concerned with how it looks in this window.

Here are some special cases:

- If an argument contains embedded blanks, enclose the argument in quotation marks (").
- If an argument contains a quotation mark, precede it with a backslash.
- If an argument contains a backslash character (\), precede it with a second backslash.
- TotalView interprets **\n** as an embedded newline.

As the information in this page is just text, use standard dialog box editing commands to remove arguments you no longer need. If you delete these arguments before execution begins, TotalView does not use them.

Environment variables: Use this area to define additional environment variables that TotalView passes to a process when it is launched.

By default, a new process inherits TotalView environment variables, and a remote process inherits **tvdsvr**'s environment variables. Using this window, you can add new variables, change the value of existing variables, or delete an existing variable.

An environment variable is specified as *name=value*. For example, the following definition creates an environment variable named **DISPLAY** whose value is **unix:0.0**:

```
DISPLAY=unix:0.0
```

Place each environment variable on a separate line.

Standard I/O Tab

The controls within this tab let you change how TotalView handles standard input (**stdin**), standard output (**stdout**), and standard error (**stderr**). Each is handled separately. (See Figure 5 on page 10.)

This tab is identical to the Standard I/O tab that TotalView displays when you select the **Process > Startup Parameters** *command.*

If you want to use the default **stdio**, **stdout**, or **stderr**, you can clear the button that precedes the area.

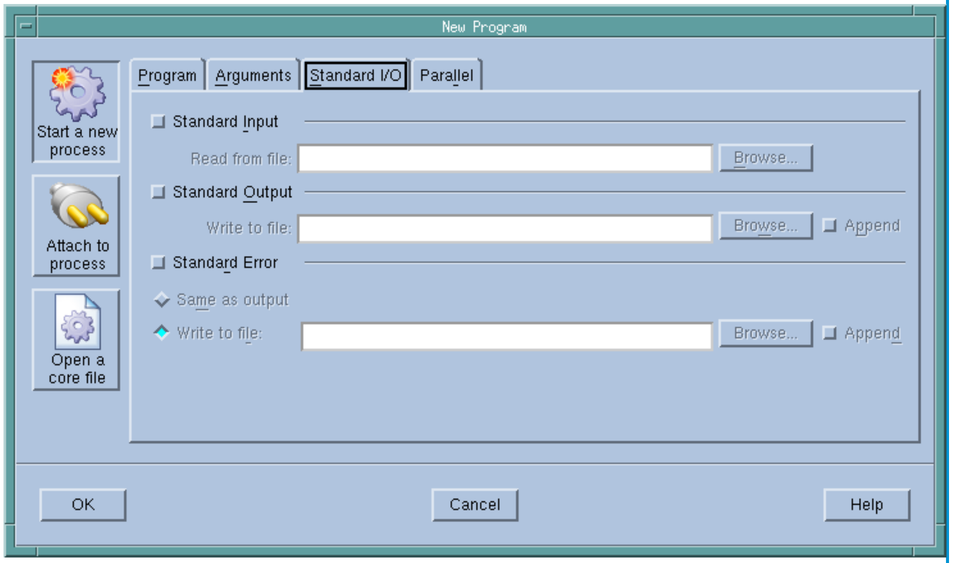
Standard Input Lets you name the file that will be connected to the process's standard input (**stdin**) when it is next launched.

Processes running under TotalView control inherit standard input from TotalView. This field lets you set the target process's standard input to be a file. You must do this before the process is created.

Read from file

If your program should receive input from a file, you can either type the file name directly or use the **Browse** button to locate the file.

Figure 5: File > New Program Dialog Box: Showing Standard I/O Tab



Standard Output

Lets you name the file that will be connected to the process's standard output (**stdout**) when it is next launched.

Processes run under TotalView inherit their standard output from TotalView. This field lets you set the target process's standard output to a file. You must do this before the process is created.

Write to file

If you want your program to send output to a file, you can either type the file name directly or use the **Browse** button to locate the file.

stdout is buffered. If it is pointed to a file, the last few lines of the program's output are not actually written to the file until the buffer is flushed. If the target process terminates abnormally or if TotalView deletes it, the last few lines of output may never be written to the file.

Standard Error

Lets you name the file that will be connected to the process's standard error (**stderr**) when it is next launched.

Processes run under TotalView inherit **stderr** from TotalView. This field lets you set the target process's **stderr** to a file. You must do this before the process is created.

stderr is buffered. If it is pointed to a file, the last few lines of the program's output are not actually written to the file until the buffer is flushed. If the target process terminates abnormally or if TotalView deletes it, the last few lines of output may never be written to the file.

Write to file

If you want your program to send error information to a file, you can either type the file name directly or use the **Browse** button to locate the file.

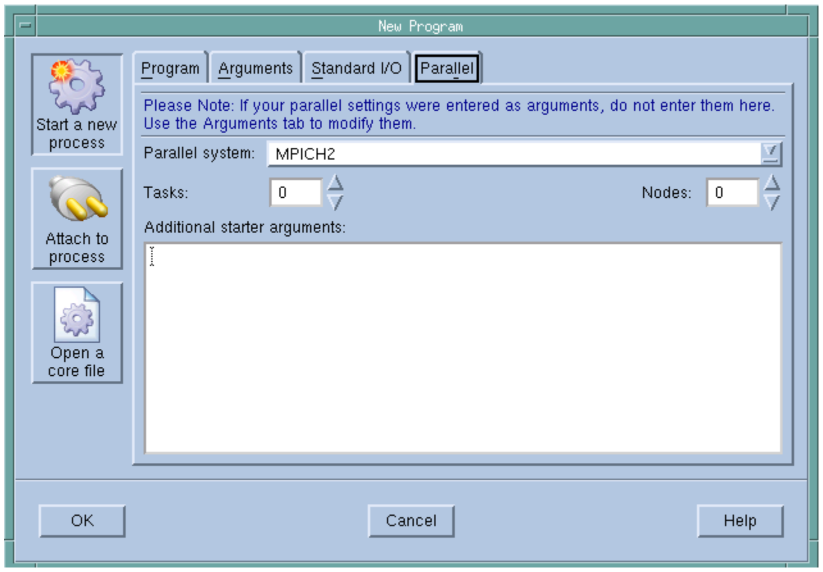
Same as output

If you would like **stderr** to go to the same file as **stdout**, select this check box.

Parallel Tab

The Parallel tab lets you tell TotalView how it should start your parallel job. You can, of course, also directly start your job directly from a shell.

Figure 6: File > New Program Dialog Box: Parallel Tab



Please Note: If your parallel settings were entered as arguments, do not enter them here. Use the Arguments tab to modify them.

Parallel system: MPICH2

Tasks: 0

Nodes: 0

Additional starter arguments:

This tab is identical to the Parallel tab that TotalView displays when you select the Process > Startup Parameters *command.*

Parallel system

Select which parallel system profile TotalView should use when it starts your program. This profile can be one that TotalView Technologies provides, one created for your site, or one that you create. For information, see http://www.totalviewtech.com/Documentation/.

Tasks

Enter a number indicating how many tasks your program should create. Entering a value of 0 (zero) indicates that your system's default value should be used.

Nodes

Enter a number indicating how many nodes your program should use when running your program. Not all systems use this value. Entering a value of 0 (zero) indicates that your system's default value should be used.

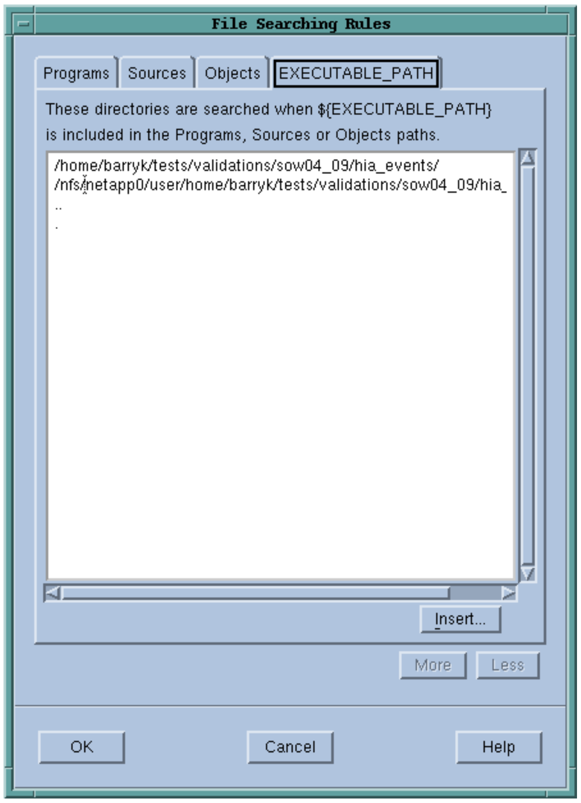
Additional starter arguments

If your program's execution requires that you use arguments to send information to the starter process such as **mpirun** or **poe**, enter them in this area. In contrast, if you need to use arguments to send information to your program, enter those arguments in the Arguments tab.

File > Search Path

Use this dialog box to set the directories in which TotalView will search for executable and source files. You can type a directory name within the EXECUTABLE_PATH tab and you can use the **Insert** button to graphically move through your system's file system to select a directory to be inserted.

Figure 7: File > Search Path Dialog Box, EXECUTABLE_PATH



These directories are searched when ${EXECUTABLE_PATH} is included in the Programs, Sources or Objects paths.

```
/home/barryk/tests/validations/sow04_09/hia_events/
/nfs/netapp0/user/home/barryk/tests/validations/sow04_09/hia_
..
.
```

The search path system allows TotalView to find source, object, and executable files throughout your file system. This help topic only discusses basic usage. For more information, see "Setting Search Paths Using TotalView Variables" on page 225.

TotalView searches for source files, in the following order:

1 The current working directory (.).

2 The directories you specify by using the **File > Search Path** command in the exact order you enter them.

3 If you entered a full path name for the executable when you started TotalView, TotalView searches this directory.

4 If your executable is a symbolic link, TotalView will look in the directory in which your executable actually resides for the new file.

   As you can have multiple levels of symbolic links, TotalView keeps on following links until it finds the actual file. After it has found the current executable, it will look in its directory for your file. If it isn't there, it'll back up the chain of links until either it finds the file or determines that the file can't be found.

5 The directories specified in your **PATH** environment variable.

The search path is local to the machine upon which TotalView is running. TotalView does not search for files on remote hosts.

File > Preferences

Use this dialog box to set preferences for how TotalView will behave, as well as define some general characteristics. The pages in this dialog box are:

- "*Options Page*" on page 13
- "*Action Points Page*" on page 16
- "*Launch Strings Page*" on page 18
- "*Bulk Launch Page*" on page 23
- "*Dynamic Libraries Page*" on page 26
- "*Parallel Page*" on page 29
- "*Fonts Page*" on page 31
- "*Formatting Page*" on page 32
- "*Pointer Dive Page*" on page 35
- "*ReplayEngine Page*" on page 36

When you save your preferences, TotalView writes them to a file named **preferences6.tvd** in your **.totalview** directory. If this file exists, TotalView reads it and sets the preferences indicated within it before it begins executing.

Some preferences can differ from platform to platform. For example, the bulk launch parameters on SGI and IBM differ. Consequently, if a parameter can differ, TotalView stores a unique version for each platform. This could be the reason that a preference you set on one platform does not appear when viewing preferences on another. In general, this applies to the server launch strings and dynamic library paths.

Options Page

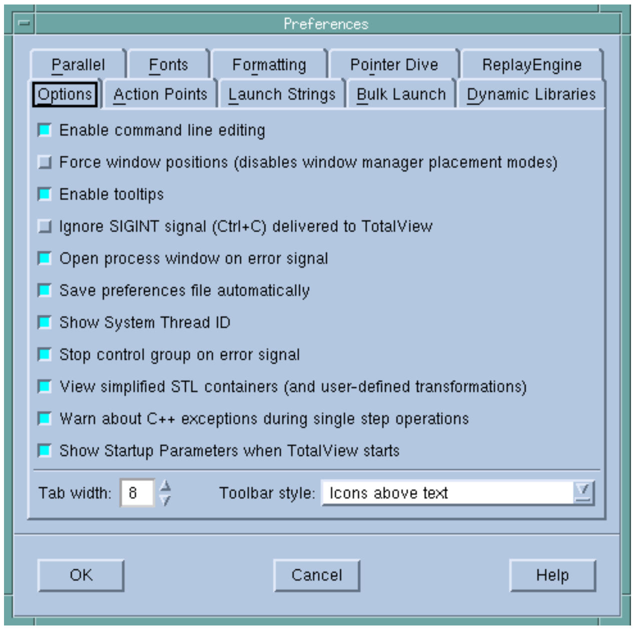
This page contains preferences that are basically unrelated to one another. (See Figure 8 on page 14.)

You can set the following preferences:

Enable command line editing

When set, TotalView enables some EMACS-like commands within the CLI. These commands are listed within the **dset** discussion with the CLI help.

Figure 8: Options Page



- [✓] Enable command line editing
- [ ] Force window positions (disables window manager placement modes)
- [✓] Enable tooltips
- [ ] Ignore SIGINT signal (Ctrl+C) delivered to TotalView
- [✓] Open process window on error signal
- [✓] Save preferences file automatically
- [✓] Show System Thread ID
- [✓] Stop control group on error signal
- [✓] View simplified STL containers (and user-defined transformations)
- [✓] Warn about C++ exceptions during single step operations
- [✓] Show Startup Parameters when TotalView starts

Tab width: 8 Toolbar style: Icons above text

For more information, see the **COMMAND_EDITING** variable.

Force window positions (disables window manager placement modes)

Setting this preference 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. This is usually better.

This preference interacts with the **Window > Memorize** command. If selected, TotalView remembers the window's size and position. If it isn't selected, only the size is remembered.

Dialog boxes still chase the pointer as needed and are unaffected by this setting.

For more information, see the **TV::force_window_position** variable.

Enable tooltips Setting this preference tells TotalView that when you place the cursor over a variable or an expression, it should display the value in a small window next to the cursor.

Ignore SIGINT signal (Ctrl+C) delivered to TotalView

When set, TotalView ignores Ctrl+C. This prevents you from inadvertently terminating the TotalView process. Set this checkbox if your program catches **SIGINT** signals and you do not want TotalView to terminate when Ctrl+C is typed on the terminal.

For more information, see the **TV::ignore_control_c** variable.

Open process window on error

If selected, TotalView opens or raises the Process Window when your program encounters an error signal. (This is the default.) Deselecting this checkbox tells TotalView that it should not open or raise the window.

If processes in a multiprocess program encounter an error, TotalView only opens a Process Window for the *first* process that encounters an error. This prevents the screen from filling up with Process Windows.

For more information, see the **TV::pop_on_error** variable.

Save preferences file on exit

If selected, TotalView will write changed preferences to your preferences file. This file is stored in a **.totalview** subdirectory within your home directory.

Show System Thread ID

If selected, TotalView displays the system thread ID in the information area just about the thread in the Process Window.

Stop control group on error signal

If selected, TotalView stops the execution of all processes in the control group when it processes a signal as an error. See **File > Signals** for more information.

For more information, see the **TV::stop_relatives_on_proc_error** variable.

View simplified STL containers (and user-defined transformations)

If selected (and the default is selected), TotalView displays STL vector, list, and maps in a logical format rather than in their actual format. In addition, if you've added your own transformations for data types, these transforms will also be used.

For more information, see the **TV::ttf** variable.

Warn about C++ exceptions during single step operations

When set, TotalView asks if it should stop a single-step operation if your program throws a C++ exception while stepping. TotalView will stop the process at the C++ run time library's throw routine.

If this is not selected, TotalView does not catch C++ exceptions thrown during single-step operations. It is possible that TotalView could lose control of the process, and let it run away.

For more information, see the **TV::warn_step_throw** variable.

Tab width Tells TotalView what tab interval it should use when it displays tabs embedded within your source code.

For more information, see the **TV::source_pane_tab_width** variable.

Show Startup Parameters when TotalView starts
If you name the file you are debugging as an argument to the **totalview** (or similar) command, TotalView displays its Startup Parameters dialog box. If you do not want TotalView to display it, uncheck this check box.

Toolbar style Tells TotalView how it should display the icons contained within the toolbar. Your choices are:

Icons above text
Icons besides text
Icons
Text

In some cases, you may need to restart TotalView before you will see the change.

For more information, see the **TV::GUI:toolbar_style** variable.

For information on other Preference pages, see:

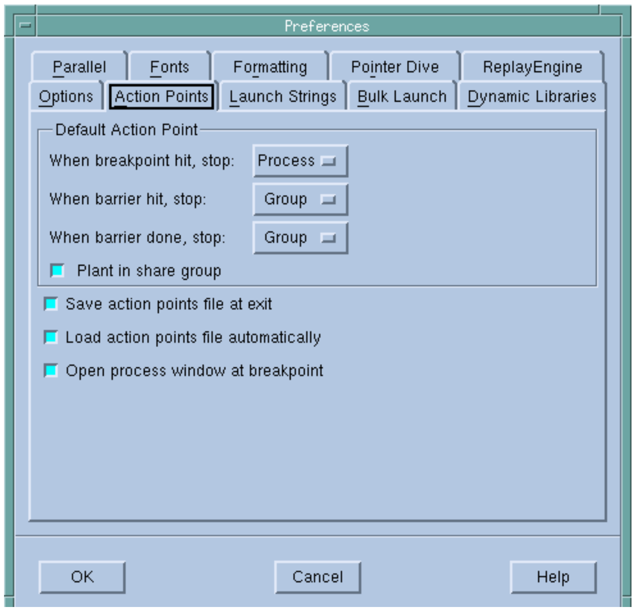
- "*Action Points Page*" on page 16
- "*Launch Strings Page*" on page 18
- "*Bulk Launch Page*" on page 23
- "*Dynamic Libraries Page*" on page 26
- "*Parallel Page*" on page 29
- "*Fonts Page*" on page 31
- "*Formatting Page*" on page 32
- "*Pointer Dive Page*" on page 35

Action Points Page

The commands within this page set the default values for the properties assigned when you create an action point. Some of these commands define what TotalView will do when it encounters an action point. Others tell TotalView that it should automatically save information about action point to a file so it can reload them at a later time. In this way, you do not have to reset action points every time you start TotalView. (See Figure 9)

For additional information, see **Action Point > Properties**.

Figure 9: Action Points Page



Default Action Point

When breakpoint hit, stop: Process

When barrier hit, stop: Group

When barrier done, stop: Group

☑ Plant in share group

☑ Save action points file at exit

☑ Load action points file automatically

☑ Open process window at breakpoint

Default Action Point

The four controls in this area define the properties that TotalView assigns to action points when you create them.

When breakpoint hit, stop

Indicates what else is stopped when TotalView encounters a breakpoint. Your options are:

group: When one thread reaches the breakpoint, TotalView stops all processes in its program group.

process: Just stop the process that hit the breakpoint.

thread: Just stop the thread that hit the breakpoint.

For more information, see the TV::stop_all variable.

When barrier hit, stop

Indicates what else is stopped when TotalView encounters a barrier breakpoint. Your options are:

group: When one thread reaches the barrier, TotalView stops all processes in its program group.

process: Just stop the process that hit the barrier.

thread: Just stop the thread that hit the barrier.

For more information, see the TV::barrier_stop_all variable.

When barrier done, stop

Indicates what occurs when all threads or processes are stopped at the barrier breakpoint. Note that the set of threads and processes that are stopped at a barrier breakpoint is called the *satisfaction* set.

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

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

thread: Only the threads in the satisfaction set are stopped; other threads are not affected. For process barriers, there is no difference between process and thread.

In all cases, the satisfaction set is released when the barrier is satisfied.

For more information, see the TV::barrier_stop_when_done variable.

Plant in share group

If set, enabling and disabling an action point alters it in all members of the share group. If this button is not selected, you must enable and disable the action point in each share group member individually.

For more information, see the TV::share_action_point variable.

Save action points file at exit

When set, TotalView automatically saves action points to an action points file when you exit. For more information, see the Action Point > Save All command and the TV::auto_save_breakpoints variable.

Load action points file automatically

When set, TotalView automatically loads action points when it loads a file. For more information, see the Action Point > Load All command and the TV::auto_load_actionpoints variable.

Open process window at breakpoint

If selected, TotalView opens or raises the Process Window when your program reaches a breakpoint. Unlike other preferences on this page, this preference changes how existing action points behave.

For more information, see the TV::pop_at_breakpoint variable.

For information on other Preference pages, see:

- "*Options Page*" on page 13
- "*Launch Strings Page*" on page 18
- "*Bulk Launch Page*" on page 23
- "*Dynamic Libraries Page*" on page 26
- "*Parallel Page*" on page 29
- "*Fonts Page*" on page 31
- "*Formatting Page*" on page 32
- "*Pointer Dive Page*" on page 35

Launch Strings Page

You can set the launch strings for the following:

- "*Single Debug Server Launch*" on page 18

- "*Visualizer Launch*" on page 21
- "*Source Code Editor*" on page 22

Single Debug Server Launch

You can modify the TotalView Debugger Server (**tvdsvr**) auto launch feature by changing the structure of the command that TotalView uses to start this server on a remote host. By default, TotalView uses the **rsh** command (**remsh** on HP-UX). Chapter 4 of the *TotalView User's Guide* contains a detailed description of these operations, along with instructions for starting the server manually if that becomes necessary.

TotalView lets you start MPI *jobs in two ways. You can directly invoke TotalView upon your program, which is identical to entering arguments into the File > New Program dialog box, or by directly or indirectly involving a starter program such as poe or mpirun. TotalView uses this feature when it is directly invoked on a starter program. This is discussed in the TotalView Users Guide.*

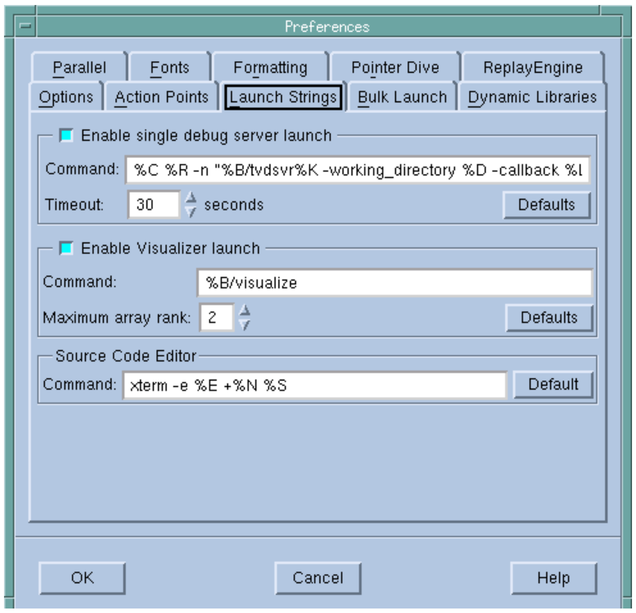


Figure 10: Launch Strings Page

[✓] Enable single debug server launch
Command: %C %R -n "%B/tvdsvr%K -working_directory %D -callback %l
Timeout: 30 seconds

[✓] Enable Visualizer launch
Command: %B/visualize
Maximum array rank: 2

Source Code Editor
Command: xterm -e %E +%N %S

Here is a a brief summary of the automatic feature:

Enable single debug server launch

When selected, TotalView automatically starts a server process when you ask it to debug a process on a remote host.

For more information, see the **TV::server_launch_enabled** variable.

Even if you enable bulk server launch, you should also enable this optional. TotalView uses this launch string when you start it upon a file and you name a host within the **File > New Program** *dialog box or use the* **–remote** *command line option. Only disable single server launch when it can't work.*

Command	The command TotalView uses when it starts the remote server. You must include the **–callback** and **–set_pw** arguments. For more information, see the **TV::server_launch_string** variable.
Timeout	Time in seconds that TotalView will wait before giving up trying to establish a connection. For more information, see the **TV::server_launch_timeout** variable.
Defaults	Changes the values defined within this area to their default values. This action overrides changes you have made using this preference, or to values set using options or X resources.

The expansion strings and options that you can use in the launch command string are:

%B	Expands to the bin directory where **tvdsvr** is installed.
%C	Expands to the default name of the command used to start a remote process. If defined, the value of the environment variable **TVDSVRLAUNCHCMD** is used. Otherwise, the default name is **rsh** (**remsh** on HP-UX).
%K	(Red Storm and BlueGene architectures) If TotalView must use an alternative name for tvdsvr, specify its name here. For example, on BlueGene, the server name is **tvdsvr_bg1**. On Red Storm, it is **tvdsvr_rs**.
%N	Is replaced by the number of servers that TotalView will launch. This is only used in a bulk server launch command.
%R	Expands to the hostname of the remote machine as specified in the **File > New Program** dialog box.
–n	Tells the remote shell to read standard input from **/dev/null**.
–working_directory %D	Expands to the full path of the current working directory in which TotalView is running. The default command string tells **tvdsvr** to first try to **cd** to this directory. This directory name may be inappropriate if the target system's file system is not organized the same way as the host's file system.
–callback	Tells the server to call back to TotalView. This must be followed by the hostname and TCP/IP port number to call back to.

%L	Expands to the hostname and TCP/IP port number on which TotalView is listening for connections from **tvdsvr**.
%H	Expands to the hostname on which TotalView is running.
%S	Expands to the TCP/IP port number on which TotalView is listening for connections from **tvdsvr**.
–set_pw	Sets a 64-bit password for security. TotalView must supply this password when **tvdsvr** establishes the connection with it.
%P	Expands to the password that TotalView automatically generated.
%V	Expands to the TotalView verbosity setting. This launches the TotalView Debugger Server with the same verbosity level as TotalView.
%F	Contains the "tracer configuration flags" that need to be sent to **tvdsvr** processes. These are system-specific startup options that the **tvdsvr** process needs.
%t1 and **%t2**	Is replaced by files that TotalView creates containing information it generates. This is only available in a bulk launch.

These temporary files have the following structure:

(1) An optional header line containing initialization commands required by your system.

(2) One line for each host being connected to, containing host-specific information.

(3) An optional trailer line containing information needed by your system to terminate the temporary file.

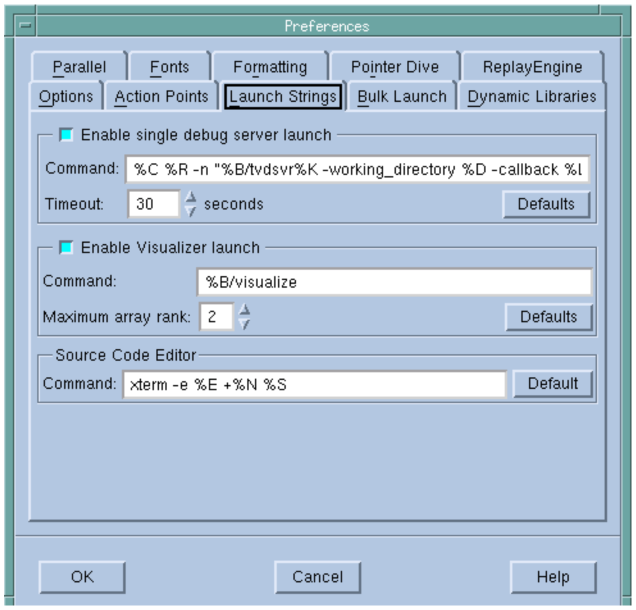
The **File > Preferences Bulk Server** Page allows you to define templates for the contents of temporary files. These files may use these replacement characters. The **%N**, **%t1**, and **%t2** replacement characters can only be used within header and trailer lines of temporary files. All other characters can be used in header or trailer lines or within a host line defining the command that initiates a single-process server launch. In header or trailer lines, they behave as defined for a bulk launch within the host line. Otherwise, they behave as defined for a single-server launch

Visualizer Launch

The launch string defined within this area indicates how TotalView will launch a visualizer.

Commands within this area are:

Figure 11: Launch Strings Page



☑ Enable single debug server launch

Command: %C %R -n "%B/tvdsvr%K -working_directory %D -callback %l

Timeout: 30 seconds [Defaults]

☑ Enable Visualizer launch

Command: %B/visualize

Maximum array rank: 2 [Defaults]

Source Code Editor

Command: xterm -e %E +%N %S [Default]

Enable Visualizer launch
: When checked, TotalView will automatically attempt to start a visualizer process when it encounters a visualization command. If this is not checked, TotalView will not launch a visualizer even if you select the **Tools > Visualize** command or have used a **$visualize** intrinsic.

  For more information, see the **TV::visualizer_launch_enabled** variable.

Command
: The command TotalView uses when it starts a visualizer. If you are using your own visualizer, you would enter its startup command here.

  For more information, see the **TV::visualizer_launch_string** variable.

Maximum array rank
: Sets the maximum rank. Edit this value if you plan to save the data exported from TotalView or display it in a different visualizer.

  The maximum value you can enter is 16 and the default value is 2.

  For more information, see the **TV::visualizer_max_rank** variable.

Defaults
: Changes the values defined within this area to their default values. This action overrides changes you have made using this preference, or values set using command-line options.

Source Code Editor

The source code editor launch string area allows you to specify the command string TotalView will use to start an editor when you use the Process Window's **File > Edit Source** command. TotalView expands this string into a command that is executed by the **sh** shell.

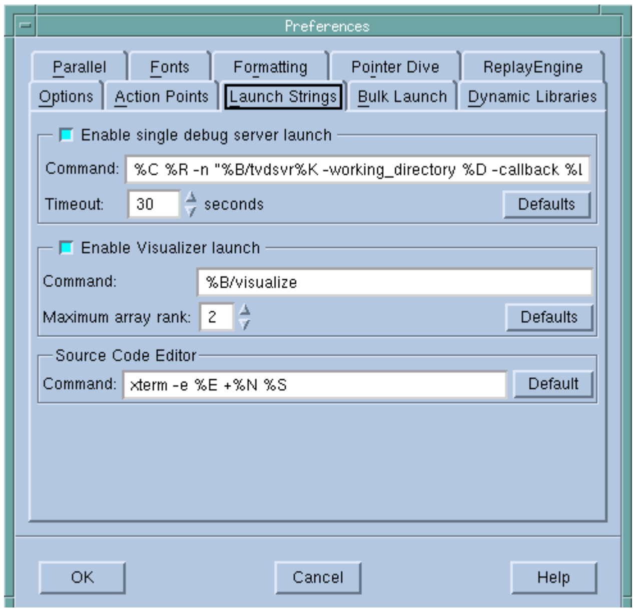


Figure 12: Launch Strings Page

Enable single debug server launch [✓]
Command: %C %R -n "%B/tvdsvr%K -working_directory %D -callback %l
Timeout: 30 seconds
Defaults

Enable Visualizer launch [✓]
Command: %B/visualize
Maximum array rank: 2
Defaults

Source Code Editor
Command: xterm -e %E +%N %S
Default

Items recognized in the launch command string are:

%E	Expands to the value of the **EDITOR** environment variable, or to **vi** if **EDITOR** if not set.
%N	Expands to the line number in the Process Window's Source Pane.
%S	Expands to the file name of the source file displayed in the Process Window's Source Pane.
%F	Expands to the font name with which you started TotalView.
Default	Changes the values defined within this area to their default values. This action overrides changes you have made using this preference, or values set using options or X Resources. The default editor launch string is **xterm -e %E +%N %S**.

For more information, see the **TV::editor_launch_string** variable. For information on other Preference pages, see:

- "*Options Page*" on page 13
- "*Action Points Page*" on page 16
- "*Bulk Launch Page*" on page 23
- "*Dynamic Libraries Page*" on page 26
- "*Parallel Page*" on page 29
- "*Fonts Page*" on page 31

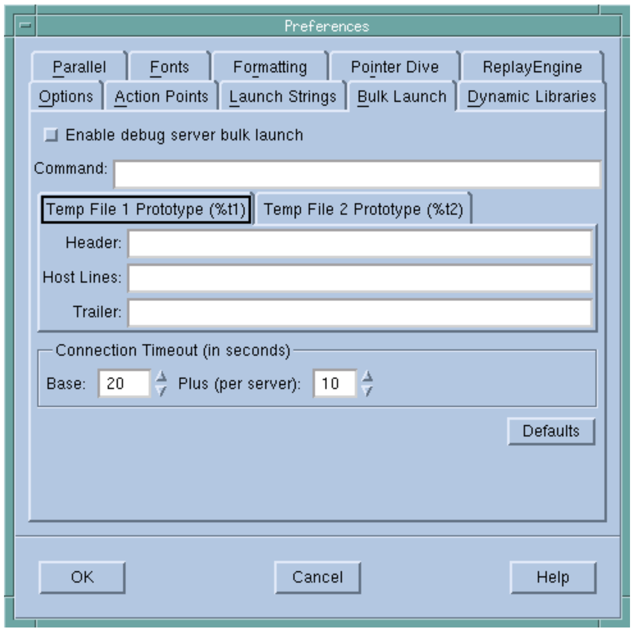
- "*Formatting Page*" on page 32
- "*Pointer Dive Page*" on page 35

Bulk Launch Page

The Bulk Launch Page allows you to specify the parameters TotalView will use when it does a bulk launch of TotalView Debugger Servers on remote hosts.

TotalView lets you start MPI jobs in two ways. You can directly invoke TotalView upon your program, which is identical to entering arguments into the File > New Program dialog box, or by directly or indirectly involving a starter program such as poe or mpirun. TotalView uses this feature when it is directly invoked on a starter program. This is discussed in the TotalView Users Guide.

Figure 13: Bulk Launch Page



Enable debug server bulk launch

Command:

Temp File 1 Prototype (%t1) | Temp File 2 Prototype (%t2)

Header:

Host Lines:

Trailer:

Connection Timeout (in seconds)

Base: 20 Plus (per server): 10

Information on bulk launching is contained within Chapter 4 of the *TotalView Users Guide*.

Enable debug server bulk launch

When this box is checked, TotalView will start multiple server processes using a bulk launch command.

For more information, see the **TV::bulk_launch_enabled** variable.

When you enable bulk server launch, you probably do not want to disable single server launch. TotalView uses the single server launch string when you start TotalView upon a file when you have named a host within the **File > New Program** *dialog box or have used the* **–remote** *command line option. You only want to disable single server launch when it can't work.*

Command	The command TotalView uses when it attempts to start the (remote) servers. This information is stored within the **TV::bulk_launch_string** variable. Items that you can use in the bulk launch command string are:
%C	Expands to the default name of the command used to start a remote process. If defined, the value of the environment variable **TVDSVRLAUNCHCMD** is used. Otherwise, the default name is **rsh** (**remsh** on HP-UX).
%D	Expands to the full path of the directory to which TotalView is connected. The default command string tells **tvdsvr** to first try to **cd** to this directory. This directory name may be inappropriate if the target system's file system is not organized the same way as the host's file system.
%F	Contains the "tracer configuration flags" that need to be sent to **tvdsvr** processes. These are system-specific startup options that **tvdsvr** processes need.
%N	Expands to the number of servers that will be launched.
%H	Expands to the hostname on which TotalView is running.
%L	Expands to a comma-separated list of remote *host:port* for all remote hosts.
%R	Expands to a comma-separated list containing the entire remote host list.
%S	Expands to the comma-separated list containing the entire port list.
%P	Expands to the comma-separated list containing the entire password list.
%V	Expands to the TotalView verbosity setting. Setting this value allows the TotalView Server (**tvdsvr**) to be launched with the same verbosity level as TotalView.
%t1	Expands to the file name of temporary file number 1 (see Temporary Files later in this section).
%t2	Expands to the file name of temporary file number 2 (see Temporary Files later in this section).
Temp File Prototypes	Discussed later in this section. This information is stored in variables beginning with **TV::bulk_launch_tmp**.
Connection timeout (in seconds)	Time to wait before giving up trying to establish the connections. The total timeout is calculated as a **Base** value, **Plus** an amount for each server launched. For more information, see the **TV::bulk_launch_incr_timeout** variable.

Temporary Files

The bulk server launch facility allows you to create temporary files whose names are passed in the bulk server launch command. Each of these file has a *Header* line, followed by one line for each remote *Host*, followed by a *Trailer* line. Each tab within this page defines one set of these three files. The substitutions available in the header and trailer lines are those available in the bulk launch command just described.

Items that you can use in the host lines of a temporary file are:

%C	Expands to the default name of the command used to start a remote process. If defined, the value of the environment variable **TVDSVRLAUNCHCMD** is used. Otherwise, the default name is **rsh** (**remsh** on HP-UX).
%R	Expands to the hostname of the remote machine as specified in the **File** > **New Program** command.
–working_directory %D	Expands to the full path of the current working directory on which TotalView is running. The default command string tells **tvdsvr** to first try to **cd** to this directory. This directory name may be inappropriate if the target system's file system is not organized the same way as the host's file system.
–callback	Tells the server to call back to TotalView. This must be followed by the hostname and TCP/IP port number to call back to.
%L	Expands to the hostname and TCP/IP port number on which TotalView is listening for connections from **tvdsvr**.
%H	Expands to the hostname on which TotalView is running.
%S	Expands to the TCP/IP port number on which TotalView is listening for connections from **tvdsvr**.
–set_pw	Sets a 64-bit password for security. TotalView must supply this password when **tvdsvr** establishes the connection with it.
%P	Expands to the password that TotalView automatically generated.
–verbosity %V	Expands to the TotalView verbosity setting. This allows the TotalView Server to be launched with the same verbosity level as TotalView.

For more information, see "*TotalView Command Syntax*" in the *TotalView Reference Guide*.

For information on other Preference pages, see:

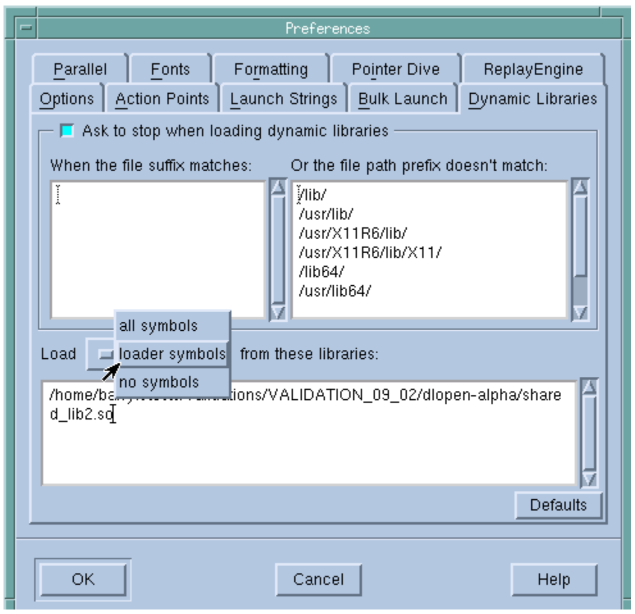
- "*Options Page*" on page 13
- "*Action Points Page*" on page 16
- "*Launch Strings Page*" on page 18
- "*Dynamic Libraries Page*" on page 26
- "*Parallel Page*" on page 29

- "*Fonts Page*" on page 31
- "*Formatting Page*" on page 32
- "*Pointer Dive Page*" on page 35

Dynamic Libraries Page

The controls within this page manage two different library behaviors:

- The top controls allow you to control if TotalView should stop execution when your program and TotalView loads a named shared library. In most cases, you would do this so that you can set a breakpoint.
- The bottom controls tell TotalView how much information is should read when it loads a shared library. (See "*Symbol Loading*" on page 28.)



```
/lib/
/usr/lib/
/usr/X11R6/lib/
/usr/X11R6/lib/X11/
/lib64/
/usr/lib64/
```

```
/home/ba          tions/VALIDATION_09_02/dlopen-alpha/share
d_lib2.so
```

Figure 14: Dynamic Libraries Page

The **Default** *button at the bottom of this page sets all fields on this page to their initial values.*

Stopping Before Execution Begins

The controls in the top area tell TotalView if it should ask you if it is alright to load dynamic libraries. If it is OK, you can indicate which libraries TotalView should ask questions about before loading and which it should just load.

Ask to stop when loading dynamic libraries

If selected, TotalView uses the shared library path and file suffix to determine if it should ask if it should stop processes that load a shared library. The decision it makes is based on what you type in the two text areas.

For more information, see the **TV::ask_on_dlopen** variable.

When the file suffix matches

Enter the suffixes that TotalView uses when it decides whether it will ask if it should stop the process when it loads a dynamic library. If the library being opened has a suffix that is on this list, TotalView asks if it should stop the process.

Each suffix must reside on its own line. By default, this list is empty. This list is global and applies to all processes in this debugging session.

For more information, see the **TV::dll_stop_prefix** variable.

And the file path prefix doesn't match

Enter prefixes that TotalView uses when it decides whether it will ask if it should stop the process when it loads a dynamic library. If the shared library being opened has a prefix that is on this list, TotalView does not ask if should stop the process.

Each prefix must be on its own line. By default, this list is empty. The list is global and it applies to all processes you examine in this debugging session.

For more information, see the **TV::dll_ignore_prefix** variable.

Symbol Loading

The three items on the **Load from these libraries list** control whether TotalView reads loader and debugging symbols when it opens a library. Here's what placing entries into these areas means:

all symbols

(default) TotalView reads all symbols. Only enter a library name if it is excluded by a wildcard in the **loader symbols** and **no symbols** areas.

For more information, see the **TV::dll_read_all_symbols** variable.

loader symbols

TotalView only reads a library's loader symbols. If your program uses a number of large shared libraries that you will not be debugging, you might set this to *. You would then enter the names of DLLs that you need to debug in the **all symbols** area.

For more information, see the **TV::dll_read_loader_symbols_only** variable.

no symbols

Only name libraries on this list if you really need to increase performance. If TotalView doesn't load a library's symbols, it may not be able to create a backtrace through this library.

For more information, see the **TV::dll_read_no_symbols** variable.

When reading a library, TotalView looks at these lists in the following order:

1. all symbols
2. loader symbols

3 no symbols

That is, TotalView processes these lists in order. This means that if you name a library in more than one list, TotalView ignores the second (or third) references to the library.

When entering library names, you can use the * and ? wildcard characters. For example:

mystuff	Matches **./lib/libmystuff.so** as well as anything else that contains the **mystuff** string in its filename.
/home/myname/dev/*	Matches any library in the /**home/myname/dev** directory.
*	Matches every library that TotalView would read.

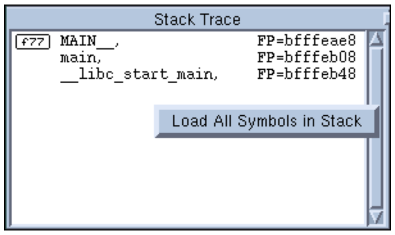
If your program stops in a library that has not had its symbols read, TotalView reads its symbols before reporting the error that caused execution to stop.

You can tell TotalView that it should automatically read a library's symbols when it stops by setting the **TV::auto_read_symbol_at_stop** variable.

Load All Symbols in Stack Context Menu Command

If you place the cursor in the Stack Trace Pane and click your right mouse button, TotalView displays the **Load All Symbols in Stack** command. Selecting this command tells TotalView to examine the stack trace for the current thread and finds any frames where the thread was executing in a library that has not had all its symbols read. If TotalView locates any libraries, it reads in their debugging symbols.

Figure 15: Load All Symbols in Stack Context Menu



```
MAIN__,                   FP=bfffeae8
main,                     FP=bfffeb08
__libc_start_main,        FP=bfffeb48
```

If, while reading in these libraries, it discovers other libraries that must be read in, it will also read in these additional symbols.

For information on other Preference pages, see:

- "*Options Page*" on page 13
- "*Action Points Page*" on page 16
- "*Launch Strings Page*" on page 18
- "*Bulk Launch Page*" on page 23
- "*Parallel Page*" on page 29
- "*Fonts Page*" on page 31
- "*Formatting Page*" on page 32
- "*Pointer Dive Page*" on page 35

Parallel Page

The preferences on this page tell TotalView that it should enable the use of **dbfork**. It also indicates what will happen when your program goes parallel.

TotalView lets you start MPI jobs in two ways. You can directly invoke TotalView upon your program, which is identical to entering arguments into the File > New Program dialog box, or by directly or indirectly involving a starter program such as poe or mpirun. TotalView only refers to these settings when it is directly invoked on a starter program. For programs directly started by TotalView, the program executes in actually the same way as a non-parallel program. That is, all processes created are part of the same control group and TotalView allows this control group to run freely. This is discussed in the TotalView Users Guide.

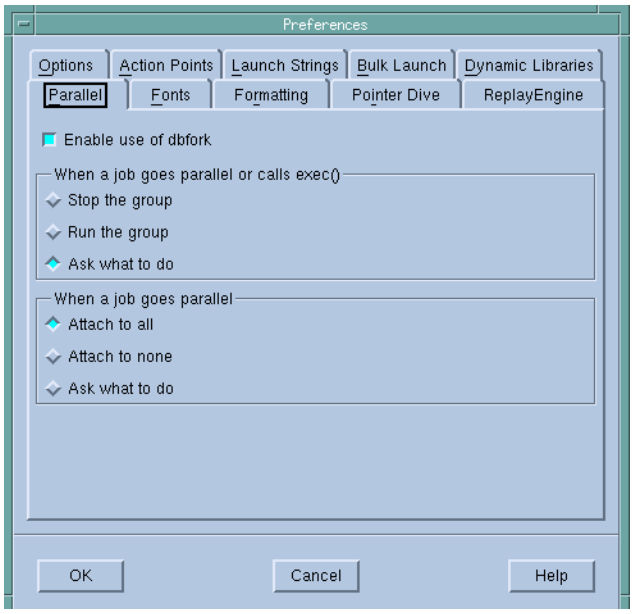


Figure 16: Parallel Page

Preferences

Options | Action Points | Launch Strings | Bulk Launch | Dynamic Libraries
Parallel | Fonts | Formatting | Pointer Dive | ReplayEngine

[✓] Enable use of dbfork

When a job goes parallel or calls exec()
- Stop the group
- Run the group
- [✓ Ask what to do]

When a job goes parallel
- [✓ Attach to all]
- Attach to none
- Ask what to do

OK | Cancel | Help

Enable use of dbfork

When set, TotalView catches the **fork()**, **vfork()**, and **exexcve()** system calls if your executable is linked with the **dbfork** library. This is discussed in Chapter 8 of the *TotalView Reference Guide*.

For more information, see the **TV::dbfork** variable.

When a job goes parallel or calls exec()

The buttons in this area have the following meaning:

Stop the group

Stop the control group immediately after your processes creates the group's processes.

Run the group

Allows all newly created processes in the control group to run freely.

Ask what to do

If set, TotalView asks if it should start the created processes. If you check this box, TotalView display its **Subset Attach** dialog box when the job goes parallel.

For more information, see the **TV::parallel_stop** variable and "*Group > Attach Subset*" on page 71.

When a job goes parallel

The buttons in this area have the following meaning:

Attach to all

TotalView automatically attaches to all processes when they begin executing.

Attach to none

TotalView will not attach to any created process when it begins executing.

Ask what to do

If set, TotalView opens the same dialog box that it displays when you select **Group > Attach Subset** command. Using this dialog box, you tell TotalView to which processes it should attach. Note that TotalView does not display this dialog box when you set the preference. Instead, this preference tells TotalView that it should display the dialog box when it is about to created processes.

For more information, see the **TV::parallel_attach** variable.

For information on other Preference pages, see:

- "*Options Page*" on page 13
- "*Action Points Page*" on page 16
- "*Launch Strings Page*" on page 18
- "*Bulk Launch Page*" on page 23
- "*Dynamic Libraries Page*" on page 26
- "*Fonts Page*" on page 31
- "*Formatting Page*" on page 32
- "*Pointer Dive Page*" on page 35

Fonts Page

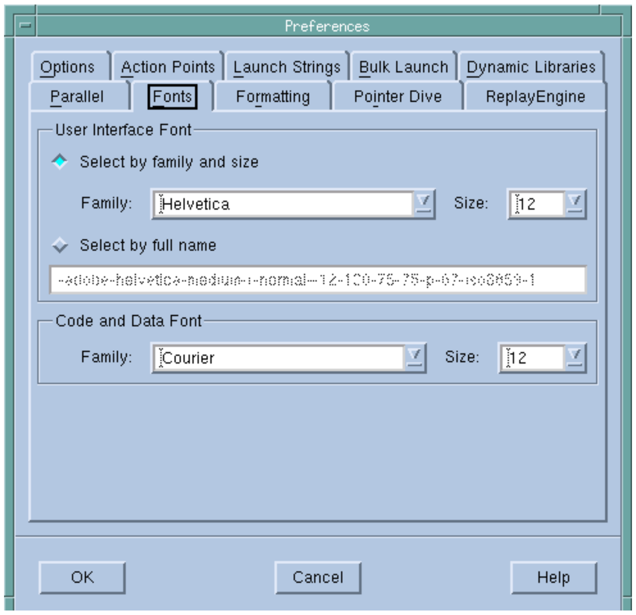
TotalView uses two fonts, a fixed width and a variable width font. Program data is displayed in a fixed width font. User Interface menus, buttons, labels, and dialog boxes use a variable width font.

The fonts you can select are those already installed with your X server.

You can select a variable width font by either selecting the font family and size or by entering the exact font name. In the first case, TotalView will attempt to select a compatible font. In the second, TotalView uses the name you select.

The following controls set the user interface font. This is the font TotalView uses when it want to display information using a variable width font. For the most part, this is the information that is not part of your code. (See Figure 17 on page 32.)

Figure 17: Fonts Page



User Interface Font

◆ Select by family and size

Family: Helvetica Size: 12

◇ Select by full name

-adobe-helvetica-medium-r-normal--12-120-75-75-p-67-iso8859-1

Code and Data Font

Family: Courier Size: 12

Select by family and size

Use the controls in this area to indicate the **Family** and the **Size** of the variable width font. The font **Family** indicates the kind of font that will be used; for example, **Helvetica** or **Times Roman**. The **Size** indicates the point size at which TotalView displays characters in the **Family** are displayed.

Select by full name

When you select a font name, you must supply the complete font name. The **xlsfonts** program supplied with your X server lists the fonts you can use.

The remaining settings are used when TotalView displays code and data. Using these controls, you can also select a font family and a font size.

For information on other Preference pages, see:

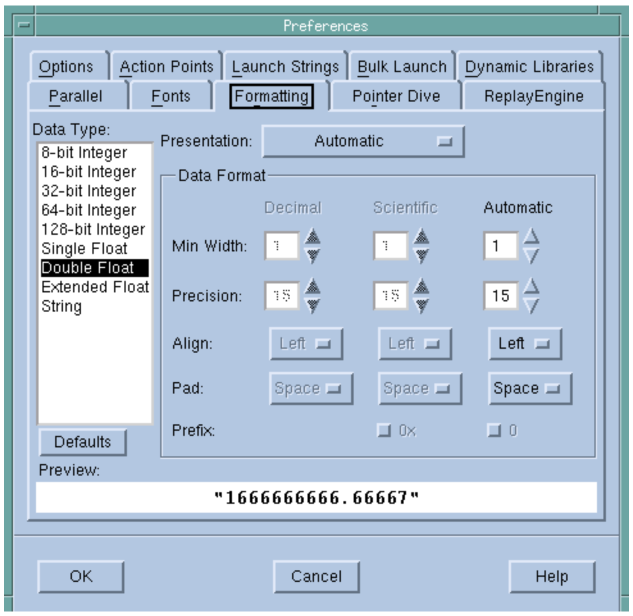
- "*Options Page*" on page 13
- "*Action Points Page*" on page 16
- "*Launch Strings Page*" on page 18
- "*Bulk Launch Page*" on page 23
- "*Dynamic Libraries Page*" on page 26
- "*Parallel Page*" on page 29
- "*Formatting Page*" on page 32
- "*Pointer Dive Page*" on page 35

Formatting Page

The controls within this page specify the precision at which TotalView should display a variable's value. You can define the precision for the following data types:

- 8-bit integers
- 16-bit integers
- 64-bit integers
- Single precision floating point numbers
- Double precision floating point numbers
- Extended precision floating point numbers
- Strings

Figure 18: Formatting Page



Data Type:
8-bit Integer
16-bit Integer
32-bit Integer
64-bit Integer
128-bit Integer
Single Float
Double Float
Extended Float
String

Presentation: Automatic

Data Format

	Decimal	Scientific	Automatic
Min Width:	1	1	1
Precision:	15	15	15
Align:	Left	Left	Left
Pad:	Space	Space	Space
Prefix:		0x	0

Preview:

```
"1666666666.66667"
```

The variables set by this preference begin with **TV::data_format**.

If you have selected a numeric data type in the left hand list, the presentations you can use are:

Selection	Tells TotalView To Display Values:
Automatic	As hex(dec) in C and C++ and dec(hex) in Fortran.
Hexadecimal (Decimal)	In both hexadecimal and decimal. The decimal value is displayed within parentheses.
Decimal (Hexadecimal)	In both decimal and hexadecimal. The hexadecimal value is displayed within parentheses.
Decimal	As decimal numbers.
Hexadecimal	As hexadecimal numbers.
Maximum Length	If the data type selected is String, the display changes to a single box and up and down controls that let you specify the maximum number of characters that will be in the displayed string.

Selection	Tells TotalView To Display Values:
Octal	As octal numbers.
Scientific	Using scientific notation.

After you have selected a data type, you can specify the precision using the following controls:

Format	Meaning
Min Width	The total number of positions used to display a number. This value includes decimal points as well as any other characters contained within the display. If this width is too small to display a value, TotalView will use more characters.
Precision	The number of characters to the right of the decimal point.
Align	Selecting **Left** or **Right** tells TotalView how it should display information within the specified width. For example, if the **Min Width** is 20 and TotalView needs only 12 characters to display a value, the value can be placed to the right with 8 preceding spaces or to the left with 8 trailing spaces.
Pad	If the **Align** value is **Right** and TotalView needs fewer positions to display the value than indicated in the **Min Width** control, it can print the leading spaces as either **Spaces** or **Zeroes**.
Prefix	If TotalView is displaying **Hexadecimal** numbers, it can include the **0x** hexadecimal indicator with the number. Similarly, when it is displaying **Octal** numbers, it can include the **0** octal indicator.

As you change values, the **Preview** area shows the effect of your changes.

Before making changes, it may be helpful to set a large minimum width, then play with the other controls to see what happens as you alter how TotalView will display values.

Use the following two controls for setting strings:

Maximum Display Length
: Defines how much of the string is displayed before diving on the string.

Long String Maximum Display Length
: Defines the maximum size for a string being displayed in a scrolling text window.

For example, if you dive on **argv** you will see an array of strings. The maximum length of each string is defined by **Maximum Display Length**. If you dive on **argv[0]**, the text appears in a scrollable control. Here, its maximum size is defined by the value set in **Long String Maximum Display Length**.

If you are viewing a string variable (not a pointer to a string), TotalView displays the first 8000 characters. This can be adjusted to a maximum of 60,000 characters using the **Long String Maximum Display Length** preference. When viewing pointers to strings or strings contained within aggregates, the **Maximum Display Length** preference applies with a default of 100 characters and a maximum of 2,000.

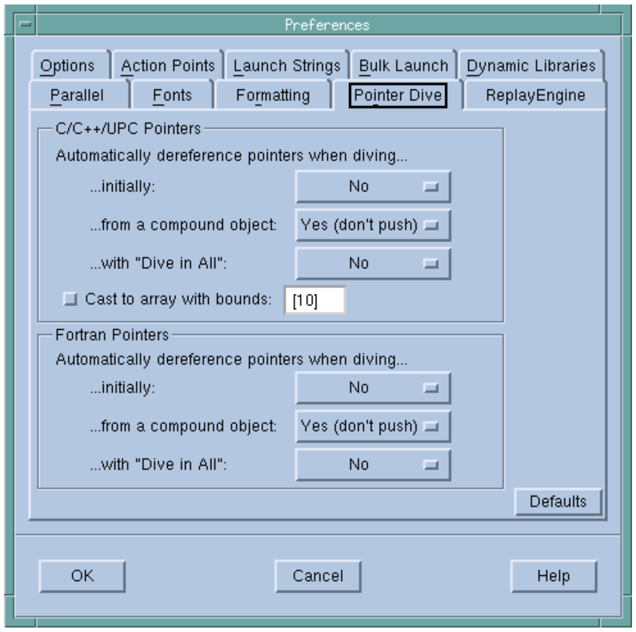
For information on other Preference pages, see:

- *"Options Page"* on page 13
- *"Action Points Page"* on page 16
- *"Launch Strings Page"* on page 18
- *"Bulk Launch Page"* on page 23
- *"Dynamic Libraries Page"* on page 26
- *"Parallel Page"* on page 29
- *"Fonts Page"* on page 31
- *"Pointer Dive Page"* on page 35

Pointer Dive Page

The controls within this page tell TotalView if it should automatically dereference pointers when diving. One set of controls is used for C, C++, and UPC. The other set of controls is for Fortran. (See Figure 19 on page 35.)

Figure 19: Pointer Dive Page



C/C++/UPC Pointers
Automatically dereference pointers when diving...
...initially: No
...from a compound object: Yes (don't push)
...with "Dive in All": No
Cast to array with bounds: [10]

Fortran Pointers
Automatically dereference pointers when diving...
...initially: No
...from a compound object: Yes (don't push)
...with "Dive in All": No

The shared controls let you automatically dereference pointers when diving as follows:

...initially
Tells TotalView what it should do when you dive on a variable.

...from a compound object
Tells TotalView what it should do when you dive on an element in a compound data element such as a structure.

...with "Dive in All"	Tells TotalView what it should when you dive on a variable using the **Dive in All** command.

You can specify one of the following for each of these controls:

No	Do not automatically dereference variables when diving.
Yes	Automatically dereference variables. In addition, place the variable on the "variable" stack so that you can use the **View > Undive** command to see the pointer's value.
Yes (don't push)	Automatically dereference variables. Do not place the variable on the "variable" stack. This means that you cannot use the **View > Undive** command to see the pointer's value.

The **Cast to array with bounds** control tells TotalView that it should assume that the pointer to is pointing to an array of values when it dereferences a C or C++ pointer. The text box lets you state how many items exist in the array.

For information on other Preference pages, see:

- "*Options Page*" on page 13
- "*Action Points Page*" on page 16
- "*Launch Strings Page*" on page 18
- "*Bulk Launch Page*" on page 23
- "*Dynamic Libraries Page*" on page 26
- "*Parallel Page*" on page 29
- "*Fonts Page*" on page 31
- "*Formatting Page*" on page 32

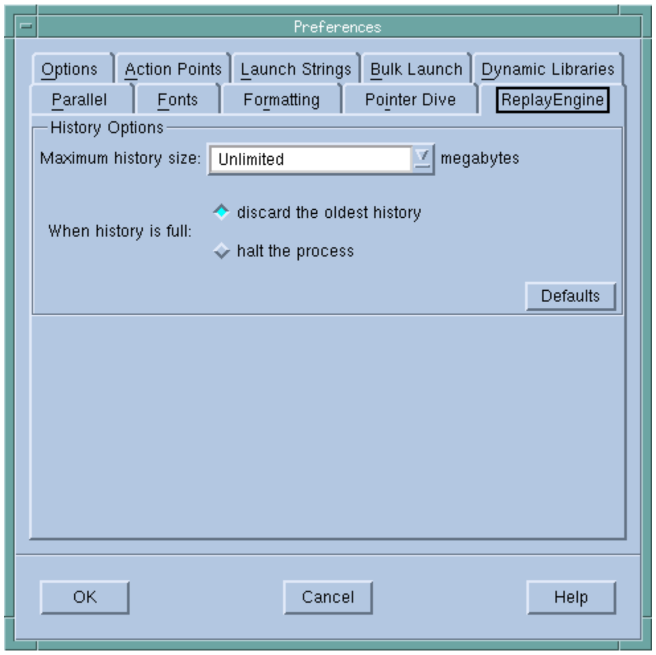
ReplayEngine Page

The options on this page control how ReplayEngine handles recorded history.

The **Maximum history size** option sets the size in megabytes for ReplayEngine's history buffer. The default value, Unlimited, means ReplayEngine will use as much memory as is available to save recorded history. You can enter a new value into the text field or select from a drop down list. (See Figure 21 on page 37.)..

The **When history is full** option defines the tool's behavior when the history buffer is full. By default, the oldest history will be discarded so that recording can continue. You can change that so that the recording process will simply stop when the buffer is full.

Figure 20: ReplayEngine Page

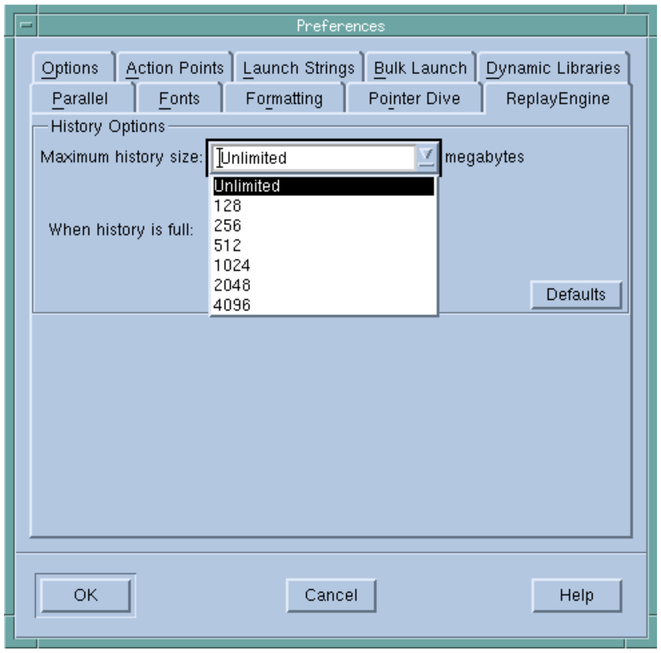


History Options

Maximum history size: Unlimited megabytes

When history is full: [✓ discard the oldest history] halt the process

Figure 21: ReplayEngine History Options



History Options

Maximum history size: Unlimited megabytes

Unlimited
128
256
512
1024
2048
4096

When history is full:

File > Save Pane

Use this dialog box to write the contents of the selected page, pane, or window.

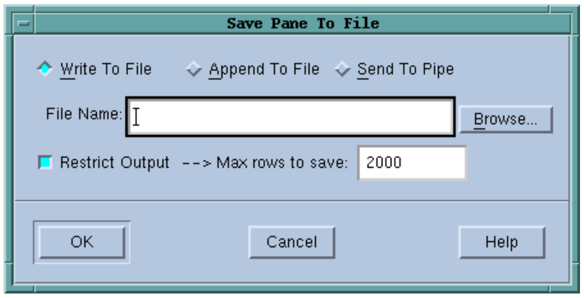


Figure 22: File > Save Pane Dialog Box

Write to File
Tells TotalView to write information to a file. You can either enter the name of the file in the **File Name** field or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView overwrites it. If the file does not exist, TotalView creates the file before writing this information.

Append To File
Tells TotalView to add information to a file. You can either enter the name of the file in the **File Name** edit box or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView adds this information to the end of the file. If the file does not exist, TotalView creates the file before writing this information.

Send To Pipe
Sends the data to the program or script named in the **File Name** field.

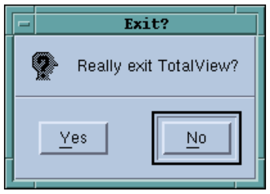
Restrict Output --> Max rows to save
If checked, TotalView limits how much information it should send. If the default value of 2000 rows is not what you want, you can specify how many rows TotalView should write.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.)

As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

Figure 23: File > Exit Command



If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 39
- **Edit > Cut** on page 39
- **Edit > Copy** on page 39
- **Edit > Paste** on page 39
- **Edit > Delete** on page 39
- **Edit > Find** on page 40
- **Edit > Find Again** on page 40

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After you make the editing change, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

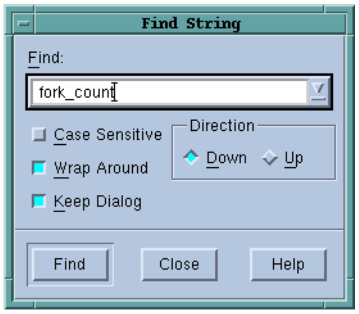
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 24: Edit > Find Dialog Box



Find: fork_count
Case Sensitive: off
Wrap Around: on
Keep Dialog: on
Direction: Down

The controls in this dialog box are:

Find	Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered.
Case Sensitive	If selected, TotalView only locates text having the same capitalization as the text entered in the **Find** field.
Wrap On Search	If selected, TotalView will continue the search from either the beginning (if **Down** is also selected) or the end (if **Up** is also selected.) For example, you search for

	"foo" and the **Down** button is selected. If it isn't found 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 had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the **Find** button. If you select this option, you will need to select the **Close** button to dismiss this dialog box.
Direction	Sets the direction in which TotalView searches. **Up** means "search from the current position to the beginning of the file." **Down** means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the **Find** box.
Close	Closes the **Find** dialog box.

After you find a string, you can locate another instance of the string by using the **Edit** > **Find Again** command.

Edit > Find Again

Executes a search defined by the **Find** command. TotalView begins searching at the current text cursor position. The direction in which TotalView searches is the same as the last search you defined.

View Menu Commands

The commands on the **View** pulldown are:

- **View > Dive** on page 41
- **View > Dive in New Window** on page 41
- **View > Expand All** on page 41
- **View > Collapse All** on page 41
- **View > Display Managers** on page 42
- **View > Display Exited Threads** on page 42

View > Dive

Opens the selected process or thread in a Process Window. If the process already owns a Process Window, that window is moved to the front of the display. If a Process Window does not exist, TotalView creates a window for it or replaces the contents in an existing window. Note that TotalView tries very hard to reuse Process Window.

If a Process Window already exists, the window is focused on an arbitrary thread. If the process has no open windows, TotalView opens a window containing the first thread in the process.

If you want the dive operation to create a new window, use the **View > Dive in New Window** command

View > Dive in New Window

Opens the selected process or thread in a Process Window. Unlike a **View > Dive in New Window** command, this will create a window for a new process.

If a Process Window already exists, the window is focused on an arbitrary thread. If the process has no open windows, TotalView opens a window containing the first thread in the process.

View > Expand All

Expands all trees that are not displaying all of their information. That is, this is equivalent to selecting every + icon within the Root Window.

View > Collapse All

Collapses all trees. That is, this is the equivalent to selecting every – icon within the Root Window.

View > Display Managers

When selected (which is the default), TotalView displays manager processes and threads.

Manager processes are processes such as **mpirun** that are used to launch and perhaps control an HPC job. While they are important to the program, they are not what you'll be debugging (unless, of course, you're creating your own starter process).

Manager threads are threads created by the operating system that support your program's activities.

In most cases, you are not interested in these processes and threads, so clearing this command is usually what you want to do.

If you are running TotalView Team Debugger, a manager process uses a token in exactly the same way a user process. For example, if you are running a 32 process MPI *job that is invoked using* **mpirun***, you will need 33 tokens.*

View > Display Exited Threads

When selected (which is the default), TotalView displays exited threads.

When debugging a multithreaded application, tracking threads as your program creates and deletes them can be difficult. When this command is selected, TotalView doesn't remove them from the display.

Tools Menu Commands

The commands on the **Tools** pulldown are:

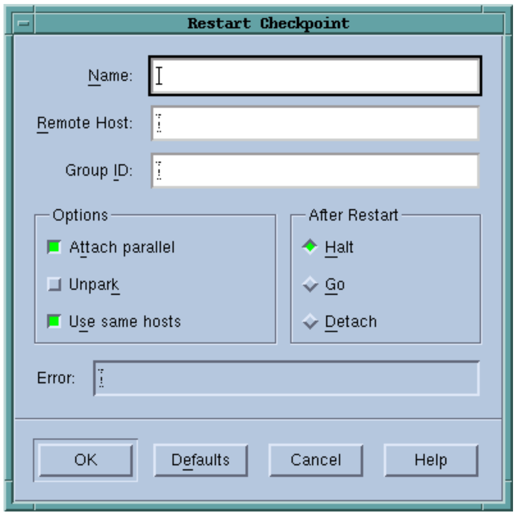
- **Tools > Restart Checkpoint** on page 42
- **Tools > Event Log on page 44**
- **Tools > Open MemoryScape** on page 44
- **Tools > Warnings** on page 45
- **Tools > PVM Tasks** on page 45
- **Tools > Command Line** on page 45

Tools > Restart Checkpoint

Use this dialog box to restore and restart all of the checkpointed processes. By default, TotalView attaches to the base process. If parallel processes are related to this base process, TotalView attaches to them. If you do not want TotalView to automatically attach to them, deselect the **Attach parallel** option.

If an error occurs while attempting to restart the program from the checkpoint, information is displayed in the **Error** area.

Figure 25: Tools > Restart Checkpoint



Restart Checkpoint

Name:

Remote Host:

Group ID:

Options: [✓ Attach parallel] Unpark [✓ Use same hosts]

After Restart: [✓ Halt] Go Detach

Error:

The CLI's **drestart** command performs the same operations as this command.

Name	Names a previously saved checkpoint file.
Remote Host	Names the remote host upon which the restart will occur.
Group ID	Names the control group into which TotalView places all created processes

Options Indicates control options that you may find useful. If this is an RS/6000 checkpoint, **Attach parallel** is automatically checked and it cannot be unchecked. These options have the following meaning:

- **Attach parallel** If selected, TotalView attaches to parallel processes as they are being created. If this item is not selected, TotalView only attaches to the base process.
- **Unpark** (SGI only) Select this checkbox if the checkpoint was created outside of TotalView or if you did not select the **Park** checkbox within the **Tools** > **Restart Checkpoint** dialog box when you created the checkpoint file.
- **Use Same Hosts** (IBM only) If selected, the restart operation tries to use the same hosts as were used when the checkpoint was created. If TotalView cannot use the same hosts, the checkpoint operation fails.

After Restart Defines the state of the process both before and after the checkpoint. You can use one of the following options:

- **Halt** Parallel processes are held immediately after the place where the checkpoint occurred. TotalView attaches to these created parallel processes. (This is the default.)
- **Go** (SGI only) Checkpointed parallel processes are started and TotalView attaches to the created processes.
- **Detach** (SGI only) Checkpointed process are started. TotalView does not attempt to attach to them.

Restarting on AIX using LoadLeveler: On the RS/6000, if you wish to debug a **LoadLever poe** job from the point at which the checkpoint was made, you must resubmit the program as a **LoadLeveler** job to restart the checkpoint. You will also need to set the **MP_POE_RESTART_SLEEP** environment variable to an appropriate number of seconds. After you restart **poe**, start TotalView and attach to **poe**.

When attaching to **poe**, *parallel tasks will not yet be created, so do not try to attach to any of them. Also, you'll need to set the* **Attach to none** *option with the* **Parallel** *Page of the* **File** > **Preferences** *Dialog Box.*

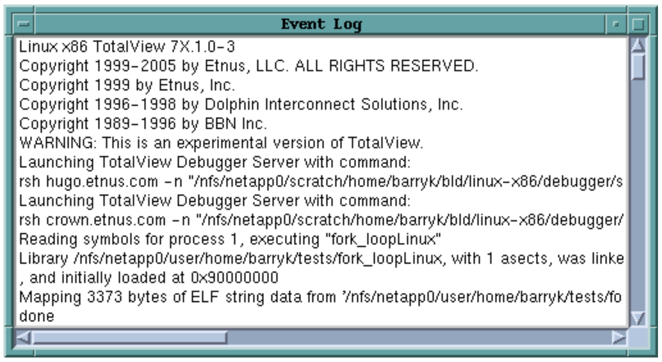
When doing this, you cannot use the restart the checkpoint using this command. **poe** will tell TotalView when it is time to attach to the parallel task so that it can complete the restart.

Tools > Open MemoryScape

For information, see MemoryScape Help.

Tools > Event Log

When events occur in the life of a process (for example, an error occurs or the process hits a breakpoint), TotalView places a line of text in the Event Log window indicating what occurred. (See Figure 26 on page 44.)



```
Linux x86 TotalView 7X.1.0-3
Copyright 1999-2005 by Etnus, LLC. ALL RIGHTS RESERVED.
Copyright 1999 by Etnus, Inc.
Copyright 1996-1998 by Dolphin Interconnect Solutions, Inc.
Copyright 1989-1996 by BBN Inc.
WARNING: This is an experimental version of TotalView.
Launching TotalView Debugger Server with command:
rsh hugo.etnus.com -n "/nfs/netapp0/scratch/home/barryk/bld/linux-x86/debugger/s
Launching TotalView Debugger Server with command:
rsh crown.etnus.com -n "/nfs/netapp0/scratch/home/barryk/bld/linux-x86/debugger/
Reading symbols for process 1, executing "fork_loopLinux"
Library /nfs/netapp0/user/home/barryk/tests/fork_loopLinux, with 1 asects, was linke
, and initially loaded at 0x90000000
Mapping 3373 bytes of ELF string data from '/nfs/netapp0/user/home/barryk/tests/fo
done
```

Figure 26: Tools > Event Log Page

The amount of information that TotalView writes into this window depends upon the value set for the **VERBOSE** variable. The values that can be set are:

info	Prints errors, warnings, and informational messages. Informational messages include data on dynamic libraries and symbols. This is the default.
warning	Only print errors and warnings.
error	Only print error messages.
silent	Does not print error, warning, and informational messages. This also shuts off the printing of results from CLI commands. This should only be used when the CLI is run in batch mode.

Tools > Warnings

Displays warning message.

If you reinvoke this command, the contents remain. TotalView only changes them when a new warning occurs.

At this time, not all warning message display in this box.

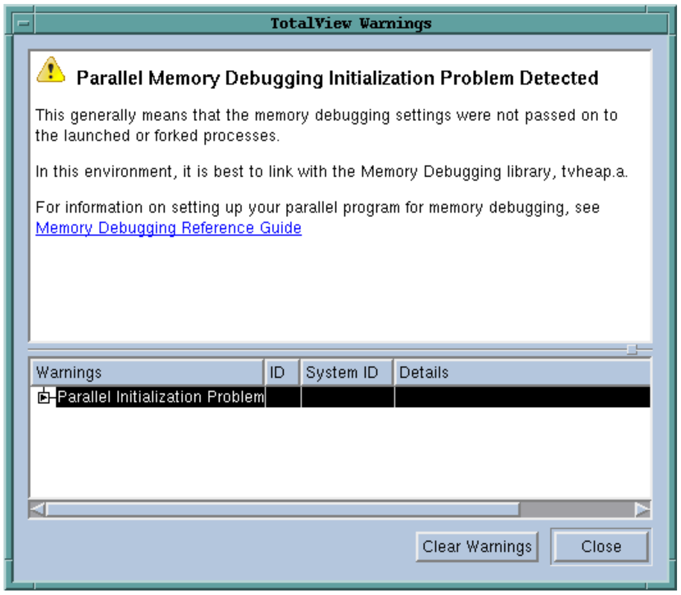
Tools > PVM Tasks

See "PVM *Tasks Window*" on page 183 for information.

Tools > Command Line

Opens the CLI window. This window is an **xterm** window into which you enter CLI commands.

Information on using the CLI is located in the *TotalView Users Guide*. This book can also be accessed by using the **Help** command.



Parallel Memory Debugging Initialization Problem Detected

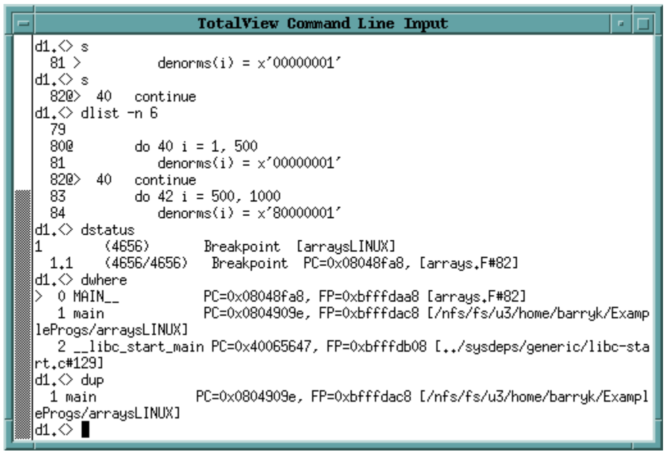
This generally means that the memory debugging settings were not passed on to the launched or forked processes.

In this environment, it is best to link with the Memory Debugging library, tvheap.a.

For information on setting up your parallel program for memory debugging, see Memory Debugging Reference Guide

Warnings	ID	System ID	Details
Parallel Initialization Problem			

Figure 27: Tools > Warnings Dialog Box



```
d1.<> s
  81 >          denorms(i) = x'00000001'
d1.<> s
  82@>  40   continue
d1.<> dlist -n 6
  79
  80@        do 40 i = 1, 500
  81            denorms(i) = x'00000001'
  82@>  40   continue
  83        do 42 i = 500, 1000
  84            denorms(i) = x'80000001'
d1.<> dstatus
1        (4656)       Breakpoint  [arraysLINUX]
  1.1    (4656/4656)   Breakpoint  PC=0x08048fa8, [arrays.F#82]
d1.<> dwhere
>  0 MAIN__           PC=0x08048fa8, FP=0xbfffdaa8 [arrays.F#82]
   1 main             PC=0x0804909e, FP=0xbfffdac8 [/nfs/fs/u3/home/barryk/Examp
leProgs/arraysLINUX]
   2 __libc_start_main PC=0x40065647, FP=0xbfffdb08 [../sysdeps/generic/libc-sta
rt.c#129]
d1.<> dup
  1 main             PC=0x0804909e, FP=0xbfffdac8 [/nfs/fs/u3/home/barryk/Exampl
eProgs/arraysLINUX]
d1.<> █
```

Figure 28: Tools > Command Line (CLI) Window

Window Menu Commands

The commands on the **Window** pulldown are:

- **Window > Update** on page 46
- **Window > Update All** on page 46
- **Window > Memorize** on page 46
- **Window > Memorize all** on page 47

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.

TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the File > Preference*'s Options Page.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.

TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

chapter 2

Process Window

Process Window Panes

The Process Window contains general information about the state of the process, with a summary of its current threads and their states. It also displays detailed information for one of the process's threads. This thread is called the "current thread."

The Process Window is divided into four areas:

- **Stack Trace Pane**, which displays the call stack.
- **Stack Frame Pane**, which displays information about the current thread's variables.
- **Source Pane**, which contains the source code or assembly instructions for your program.
- **Tabs Pane: Action Points Tab**, which displays a list of the thread's current action points.
- **Tabs Pane: Processes/Ranks Tab**, which displays a grid containing the processes or ranks within the current control group.
- **Tabs Pane: Threads Tab**, which contains a list of all active threads in the process.

Immediately above some panes is information about the process and thread being displayed:

- The *process bar* displays process status information. This information includes the process ID (PID), a process name, and a status indicator. If the process is running on a remote machine, this information is also displayed.
- The *thread bar* displays a thread ID, which is a combination of the PID and TID (thread ID) generated by TotalView and a status indicator. If the process is running on a remote machine, this information is also displayed.

Stack Trace Pane

Shows the call stack of routines that the selected thread is executing. You can move up and down the call stack by selecting the routine. When you select a different stack frame, TotalView updates the Stack Frame and Source Code panes to show the information about the routine that you just selected.

The information in this pane is as follows:

- The beginning of each line indicates the language in which that routine is written.
- The second column is the name of the routine.
- The final column indicates the location of the routine's frame pointer.

Stack Frame Pane

Displays all the function parameters, local variables, and registers for the selected stack frame. This frame does not include information on your program's global variables; use the **Tools > Program Browser** command to obtain this information.

This frame can be set in two ways:

- It is selected implicitly when TotalView hits a breakpoint or when it loads a program.
- You select a routine in the Stack Trace Pane.

To change the value of any item in this pane, just click on the value you wish to change and then edit its value. To see more information about a variable or to dereference a chain of pointer variables, dive on the line containing the variable.

If you are debugging OpenMP code and the current thread is a slave thread in a parallel region, TotalView shows a special stack frame in the Stack Frame Pane.

Source Pane

Contains the source for the routine associated with the selected stack frame. The arrow in the left margin of the Source Pane indicates the location of the PC for that stack frame.

To set breakpoints in the process, click on the line number. A "stop sign" icon appears under the cursor. To clear a breakpoint, place your mouse over the stop sign and click on it again. To set or alter an action point's settings, select the line and then select the **Action Point > Properties** command. (Right-clicking the line brings up a popup menu. You can select **Properties** from this menu.) Note that breakpoints apply to all threads in the process.

To view the source for a function or the contents for a variable whose name appears in the Source Pane, double-click on it or, after selecting it, use the **View > Dive** command. If you click on a function, TotalView shows the function in the source pane by replacing the information that was being shown. You can return the display to how it was previously by selecting the < indicator located to the right of the source pane.

If you click on a variable, information for the variable appears in a separate window.

Threads Tab

The *Threads Tab* shows the threads that currently exist in this process. You can change to a different thread by selecting a thread in this list, TotalView updates other panes to show the information for the selected thread.

This tab has three columns, as follows:

- The first contains the thread ID, a slash, and the system ID.
- The second is the thread's status, as follows:

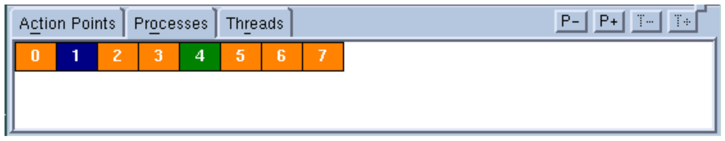
Character and Meaning	Definition
Bnn (Breakpoint)	Stopped at a breakpoint. *nn* is the ID of the breakpoint if it is a thread.
E (Error)	The **Error** state usually indicates that your program received a fatal signal from the operating system. Signals such as **SIGSEGV**, **SIGBUS**, and **SIGFPE** can indicate an error in your program.
H (Held)	Either you or TotalView is holding the thread. *Holding* means that the process or the thread cannot run until it is released. You can explicitly release it or TotalView will release it when the condition that caused it to be held is satisfied.
K (Kernel)	The thread is executing inside the kernel (that is, something made a system call). When a thread is in the kernel, the operating system does not allow TotalView to view the full state of the thread.
R (Running)	The thread is running or can run.
T (Stopped)	Stopped; however, the thread is not stopped at a breakpoint and because of an error.
W (Watchpoint)	Stopped at a watchpoint.

- The third column names the routine containing the PC.

Processes/Ranks Tab

Displays a grid containing all of the processes in the current control group. If you are debugging an MPI program, TotalView displays ranks instead of processes.

Figure 29: Processes Tab



The block's color represents the state, as follows:

Color	Meaning
Blue	Stopped; usually due to another process or thread hitting a breakpoint
Orange	At breakpoint.
Green	All threads in the process are running or can run.
Red	The **Error** state usually indicates that your program received a fatal signal from the operating system. Signals such as **SIGSEGV**, **SIGBUS**, and **SIGFPE** can indicate an error in your program.
Gray	The process has not begun running.

Diving (clicking) on a block switches the context within the Process Window so that the Process is contained within the Process Window. If the process contains more than one thread, TotalView switches to the first worker thread. That is, it does not switch context to a manager thread.

If you select a group within the scope pulldown, which at the upper-left corner of the Process Window, TotalView dims blocks representing processes that are not within the group.

Action Points Tab

The Action Points Tab shows the list of breakpoints, evaluation points, and watchpoints for the process. This tab has three columns, as follows:

- The first indicates the kind of breakpoint. Here you will see an icon indicating if you are at a breakpoint, evaluation point, barrier point, or watchpoint. TotalView displays the icon in gray if you had disabled or suppressed the action point. See **Action Point > Enable** on page 93 and **Action Point > Suppress All** on page 99 for more information.
- A TotalView action point identifier. These identifiers are never reused within a session. This identifier is more often used within the CLI than within the TotalView GUI.
- Text indicating where the breakpoint resides. This information includes a line number within a file, a program name, and the offset at which the breakpoint is set.

TotalView orders this list so that breakpoints are sorted by module name, routine name, line number, and address.

When TotalView is stopped at an action point, it places a yellow arrow over the action point's icon. This lets you know exactly where TotalView has stopped execution within a stack frame. (Each stack frame has its own stack pointer.)

If you are working with templated code, you will see ellipses (...) after the address. These ellipses indicate that there are additional addresses associated with the breakpoint.

File Menu Commands

The commands on the **File** pulldown are:

- **File > New Program** on page 53
- **File > Search Path** on page 53
- **File > Signals** on page 54
- **File > Preferences** on page 55
- **File > Open Source** on page 56
- **File > Edit Source** on page 56
- **File > Save Pane** on page 56
- **File > Rescan Libraries** on page 57
- **File > Close Relatives** on page 57
- **File > Close** on page 57
- **File > Exit** on page 58

File > New Program

For information, see **File > New Program** on page 4 in the Root Window section

File > Search Path

Use this dialog box to set the directories in which TotalView will search for executable and source files. You can type a directory name within the EXECUTABLE_PATH tab and you can use the **Insert** button to graphically move through your system's file system to select a directory to be inserted.

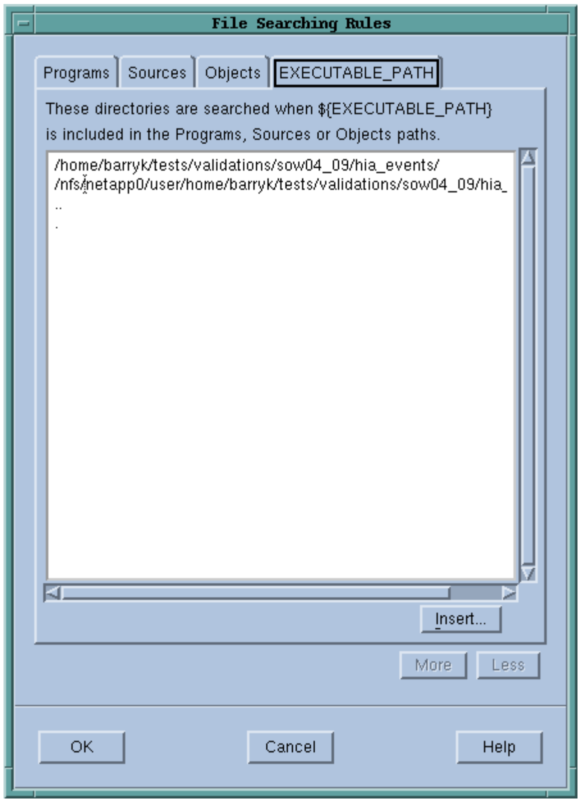
*The search path system allows TotalView to find source, object, and executable files throughout your file system. This help topic only discusses basic usage. For more information, see "*Setting Search Paths Using TotalView Variables*" on page* 225.

TotalView searches for source files, in the following order:

1. The current working directory (.).
2. The directories you specify by using the **File > Search Path** command in the exact order you enter them.
3. If you entered a full path name for the executable when you started TotalView, TotalView searches this directory.
4. If your executable is a symbolic link, TotalView will look in the directory in which your executable actually resides for the new file.

   As you can have multiple levels of symbolic links, TotalView keeps on following links until it finds the actual file. After it has found the current executable, it will look in its directory for your file. If it isn't there, it'll back up the chain of links until either it finds the file or determines that the file can't be found.
5. The directories specified in your **PATH** environment variable.

Figure 30: File > Search Path Dialog Box, EXECUTABLE_PATH



These directories are searched when ${EXECUTABLE_PATH} is included in the Programs, Sources or Objects paths.

```
/home/barryk/tests/validations/sow04_09/hia_events/
/nfs/netapp0/user/home/barryk/tests/validations/sow04_09/hia_
..
.
```

The search path is local to the machine upon which TotalView is running. TotalView does not search for files on remote hosts.

File > Signals

Use this dialog box to control the way TotalView responds to UNIX signals.

Here are some special considerations:

- In most cases, you should not select **Ignore** for signals such as **SIGSEGV** or **SIGBUS** that indicate that an error occurred. Modifying the behavior of these signals is unlikely to do what you want and may cause TotalView to get caught in a fault loop with your program.
- You cannot alter **SIGTRAP** and **SIGSTOP** because these signals are used internally by TotalView.

If several processes encounter errors simultaneously, TotalView only opens a window for the *first* error. Thus, if 64 processes in a parallel program try to divide by zero at the same time, TotalView will not open 64 process windows simultaneously; instead, it only raises one window.

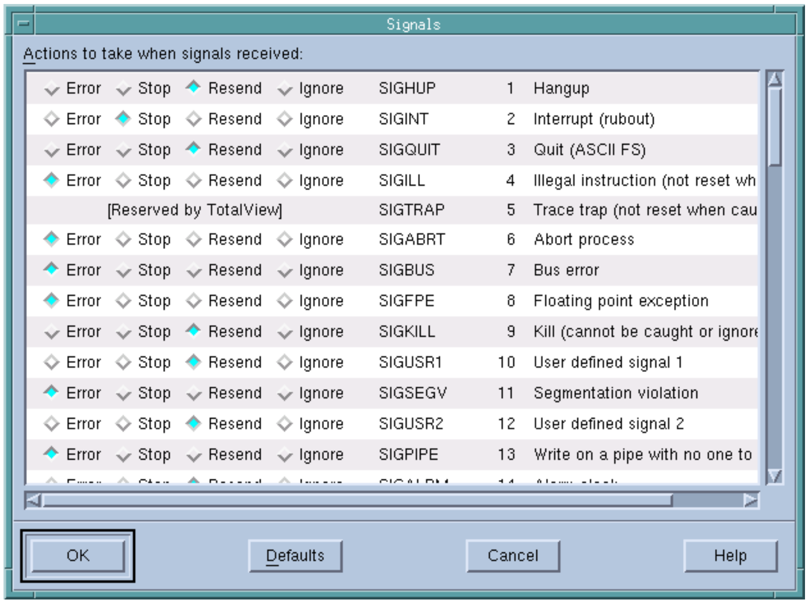


Figure 31: File > Signals Dialog Box

Signals

Actions to take when signals received:

Error	Stop	[✓ Resend]	Ignore	SIGHUP	1	Hangup
Error	[✓ Stop]	Resend	Ignore	SIGINT	2	Interrupt (rubout)
Error	Stop	[✓ Resend]	Ignore	SIGQUIT	3	Quit (ASCII FS)
[✓ Error]	Stop	Resend	Ignore	SIGILL	4	Illegal instruction (not reset wh
[Reserved by TotalView]				SIGTRAP	5	Trace trap (not reset when cau
[✓ Error]	Stop	Resend	Ignore	SIGABRT	6	Abort process
[✓ Error]	Stop	Resend	Ignore	SIGBUS	7	Bus error
[✓ Error]	Stop	Resend	Ignore	SIGFPE	8	Floating point exception
Error	Stop	[✓ Resend]	Ignore	SIGKILL	9	Kill (cannot be caught or ignore
Error	Stop	[✓ Resend]	Ignore	SIGUSR1	10	User defined signal 1
[✓ Error]	Stop	Resend	Ignore	SIGSEGV	11	Segmentation violation
Error	Stop	[✓ Resend]	Ignore	SIGUSR2	12	User defined signal 2
[✓ Error]	Stop	Resend	Ignore	SIGPIPE	13	Write on a pipe with no one to

OK Defaults Cancel Help

The buttons indicate what TotalView should do when a signal is raised. The actions TotalView can perform are:

Error	Stop a process, place it in the error state, and display an error in the title bar of the Process Window. If the **Stop control group on error** check box within the Preference's Option Page is selected, TotalView also stops all related processes. Select this mode for severe error conditions such as **SIGSEGV** and **SIGBUS** signals.
Stop	Stop a process and place it in the stopped state; that is, stop the process and take no further action. Its status will be shown as **T** in the Root Window. Select this mode if you want TotalView to handle this signal as if it were a **SIGSTOP** signal.
Resend	Immediately forward the signal to the process. From your program's point of view, the only difference between TotalView handling this signal and how it is handled otherwise is that your process receives the signal a little slower than it normally would. By default, the common signals for terminating a process (**SIGKILL** and **SIGHUP**) use this mode.
Ignore	Discard the signal and continue the process as if it had not occurred.

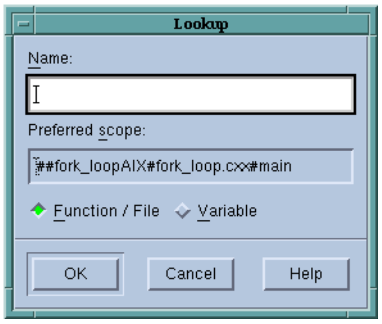
File > Preferences

Use this dialog box to set preferences for how TotalView will behave situations, as well as define some general characteristics. For more information, see "**File > Preferences**" on page 13, which is within the Root Windows help.

File > Open Source

After entering the name of one of your program’s source files, TotalView will display this file within it’s Source Pane. If a header file contains executable code, you can enter a header file name.

Figure 32: File > Open Source Dialog Box



Notice that this is the same dialog box that TotalView displays when you select the **View > Lookup Function** command.

File > Edit Source

Tells TotalView to open the file associated with the contents of the Source Pane in a text editor.

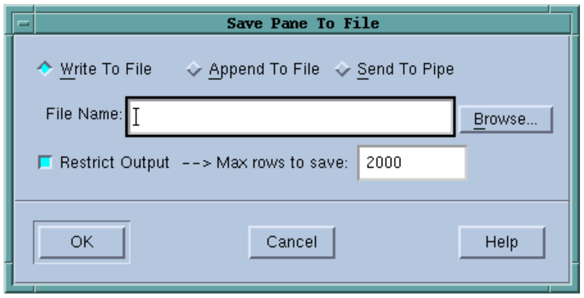
TotalView uses an editor launch string to determine how to start your editor. TotalView expands the editor launch string into a command that is then executed by the **sh** shell.

You can set the command that TotalView uses when it launches a text editor by setting the **Source Code Editor** launch string. For more information, see **File > Preferences** on page 55.

File > Save Pane

Use this dialog box to write the contents of the selected page, pane, or window.

Figure 33: File > Save Pane Dialog Box



Write to File Tells TotalView to write information to a file. You can either enter the name of the file in the **File Name** field or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView overwrites it. If the file does not exist, TotalView creates the file before writing this information.

Append To File Tells TotalView to add information to a file. You can either enter the name of the file in the **File Name** edit box or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView adds this information to the end of the file. If the file does not exist, TotalView creates the file before writing this information.

Send To Pipe Sends the data to the program or script named in the **File Name** field.

Restrict Output --> Max rows to save
If checked, TotalView limits how much information it should send. If the default value of 2000 rows is not what you want, you can specify how many rows TotalView should write.

File > Rescan Libraries

Scans the shared library (dynamic link) information looking for new information. If new information is found, it is reloaded. If, however, the information is up-to-date, no updating occurs.

You would use this command when you have recompiled or moved a shared library. This command was sometimes needed in previous versions of TotalView. It is seldom, if ever, needed currently. However, it does exist if an unforeseen problem occurs.

File > Close Relatives

Closes windows that were created using controls on this window and also closes similar windows. If any of these windows had also created windows (for example, creating a Variable Window from a Image List Window), TotalView also closes these secondary windows.

While TotalView does not close the current Process Window, Process Windows related to this window are closed.

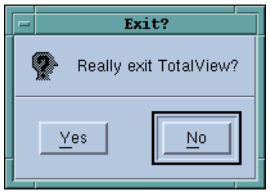
File > Close

Closes the Process Window. TotalView does not close other windows that were spawned from it. For example, if you had created one or more Variable Windows from within a Process Window, TotalView does not close these windows.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.)

Figure 34: File > Exit Command



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The following commands are on the **Edit** pulldown:

- **Edit > Undo** on page 58
- **Edit > Cut** on page 58
- **Edit > Copy** on page 59
- **Edit > Paste** on page 59
- **Edit > Delete** on page 59
- **Edit > Find** on page 59
- **Edit > Find Again** on page 60

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After you make the editing change, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

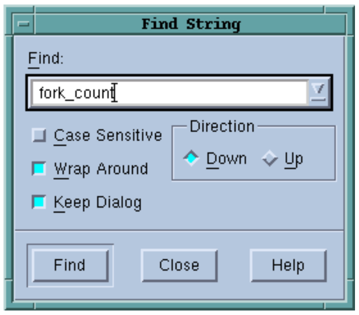
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 35: Edit > Find Dialog Box



fork_count

The controls in this dialog box are:

Find	Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered.
Case Sensitive	If selected, TotalView only locates text having the same capitalization as the text entered in the **Find** field.
Wrap On Search	If selected, TotalView will continue the search from either the beginning (if **Down** is also selected) or the end (if **Up** is also selected.) For example, you search for "foo" and the **Down** button is selected. If it isn't found 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 had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the **Find** button. If you select this option, you will need to select the **Close** button to dismiss this dialog box.
Direction	Sets the direction in which TotalView searches. **Up** means "search from the current position to the beginning of the file." **Down** means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the **Find** box.
Close	Closes the **Find** dialog box.

After you find a string, you can locate another instance of the string by using the **Edit** > **Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The following commands are on the **View** pulldown:

- **View > Dive** on page 61
- **View > Dive in New Window** on page 61
- **View > Undive** on page 62
- **View > Redive** on page 62
- **View > Reset** on page 62

- View > Lookup Function on page 62
- View > Lookup Variable on page 63
- View > Next Process on page 64
- View > Previous Process on page 64
- View > Source As > Source on page 65
- View > Source As > Assembler on page 65
- View > Source As > Both on page 65
- View > Assembler > Symbolically on page 65
- View > Assembler > By Address on page 66
- View > Display Managers on page 66

View > Dive

Dives on the selected item. (*Diving* means either displaying the selected information in a window or change the display in this window.) The action that occurs depends upon which item is selected, as follows:

Stack Trace Pane
: Selecting a routine performs a recursive dive operation; that is, it replaces the current contents with the source lines from the selected routine

Stack Frame Pane
: Opens a Variable Window that contains information about the variable and its contents.

Source Pane
: If a variable is selected, opens a Variable Window containing information about the variable and its contents. If the line contains a function or subroutine call, TotalView updates the Source Pane so that the line containing the routine is visible.

: When you dive on a routine, a > indicator appears in the Source Pane's title, indicating that TotalView has performed a nested dive operation. If you again dive, a second > indicator appears. Selecting the < icon to the right of the Source Pane's title *unwinds* TotalView so that it displays a position it previously displayed.

Threads Tab
: Selecting an entry performs a dive operation. That is, TotalView replaces the contents of the existing Process Window with information for the selected thread.

Processes/Ranks Tab
: Clicking on a block changes the contents of the Source Pane to the first worker thread in the process.

Action Points Tab
: Updates the Source Pane so that the line containing the action point is displayed.

In all cases, if the window already exists, TotalView just raises it to the top of the screen.

View > Dive in New Window

The action that occurs depends upon which item is selected, as follows:

Stack Trace Pane	Disabled.
Stack Frame Pane	Opens a new Variable Window that contains information about the variable and its contents.
Source Pane	Same as **Dive New** if you are diving on a routine name. If you are diving on a variable name, TotalView creates a new Variable Window.

View > Undive

Pops the Source Pane's *dive stack* so that you return to the place you were previously at in the Source Pane. The dive stack is a history of the source locations you have visited while examining information. This command is analogous to the "Back" button in a browser in that it returns you to a previous position. Each time you "undive", you pop one live off the dive stack.

As an alternative, you can select the "<" icon above and to the right of the Source Pane's title.

For additional information, see **View > Redive** on page 62.

View > Redive

Pushes the Source Pane's dive stack so that you return to places you "undove" from. The dive stack is a history of the source locations you have visited while examining information. This command is analogous to the "Forward" button in a browser in that it returns you to a position you previously returned from. Each time you "redive", you push one level back onto the dive stack.

As an alternative, you can select the ">" icon above and to the right of the Source Pane's title.

For additional information, see **View > Undive** on page 62.

View > Reset

Resets the source view to the "home" position; that is, it undives the stack and displays the PC.

This command lets you undo the effect of command's such as **Edit > Find**, **View > Lookup Variable**,or **View > Lookup Function** or any other command that changes the stack frame. It can also be useful if TotalView does not update the Source Pane.

View > Lookup Function

Use this dialog box to search for a function, file, or variable in your program.

- If you are searching for a function or file, TotalView will display the found information in the Source Pane. If it is not found, TotalView uses a simple spelling correction procedure to search for a function with a similar name.

- If you are searching for a variable, TotalView displays the variable's information in a Variable Window.

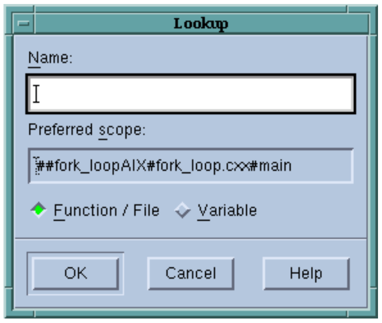


Figure 36: View > Lookup Function Dialog Box

The fields in this dialog box are:

Find	Enter the name of the function or file that TotalView will search for.
Function/File	Searches for a function or file. TotalView assumes that you are typing a function name. If TotalView cannot find the function, it assumes that you are typing a file's name and will search for it. The source for the function is placed into the Source Pane using a dive operation. You can return to the previous contents of the Source Pane by clicking on the < Undive icon in the Source Pane title bar.
Variable	Searches for this variable in your program's symbol table.

The **Preferred scope** field shows the place from which TotalView begins looking.

View > Lookup Variable

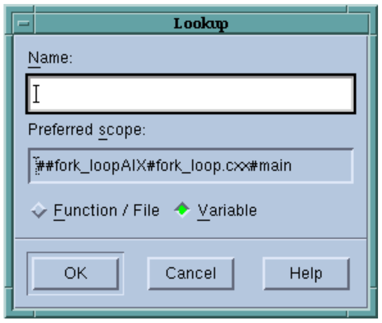
Use this dialog box to search for a variable, function, or file in your program.

- If you are searching for a variable, TotalView displays the variable's information in a Variable Window.
- If you are searching for a function or file, TotalView will display it in the Source Pane. If it is not found, TotalView uses a simple spelling correction procedure to search for a function with a similar name. (See Figure 37.)

The fields in this dialog box are:

Find	Enter the name of the function or file that TotalView will search for.
Function/File	Searches for a function or file. TotalView assumes that you are typing a function name. If it cannot find the function, it assumes that what you typed was a file's name and will search for it.

Figure 37: View > Lookup Variable Dialog Box



The source for the function is placed into the Source Pane using a Dive operation. Consequently, you can return to the previous contents of the Source Pane by clicking on the < Undive icon in the Source Pane title bar.

Variable	Searches for a local, static, or global variable in your program's symbol table. If you specify a local variable, it must be in the current stack frame. If you specify a pair of addresses instead of a name, TotalView displays the data from the first address to the second (in hex).

The **Preferred scope** field shows the place from which TotalView begins looking.

If a local and global variable have the same name, TotalView displays the local variable. If TotalView cannot find the local variable, it next checks for a global or static variable.

You cannot tell TotalView which instance of a global or static variable to display. This means that this command cannot locate and display more than one variable with the same name.

If you enter a number, TotalView will display the value at that address as a **$void**. If you enter two numbers separated by a comma, TotalView displays all locations from the first value to the second as an array of type **$void**.

After a Variable Window appears, you can edit the type field to show the data in a different way. If you enter an expression, the data type is based on the type of the expression. Similarly, if you enter a cast, the value shown is the result of the cast. Casting is discussed in the *TotalView Users Guide*.

View > Next Process

Replaces the current display with the information for the *next* process.

The *next* process that TotalView displays is the one following this process's entry in the Attached Page of the Root Window.

View > Previous Process

Replaces the current display with the information for the *previous* process.

The *previous* process that TotalView displays is the one preceding this process's entry in the Attached Page of the Root Window.

View > Source As > Source

Tells TotalView that the information displayed within the Source Pane is displayed in the programming language in which it was written.

If this information does not exist, TotalView displays the source as assembler code. If TotalView cannot find the source file, use the **File > Search Path** command to include the directory containing the source file.

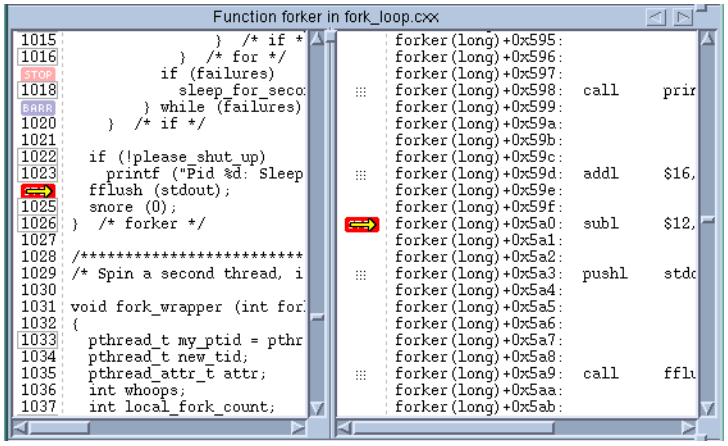
View > Source As > Assembler

Tells TotalView to display the assembler code that the compiler created from your source code.

View > Source As > Both

Tells TotalView to split the Source Pane into two parts and display your source code in the left pane and the assembler code in the right. As you perform actions in one pane, the action is reflected in the other. For example, setting a breakpoint in one pane sets it in both. Similarly, the two move in unison when you step your program.

Figure 38: Source Panes: Showing Both

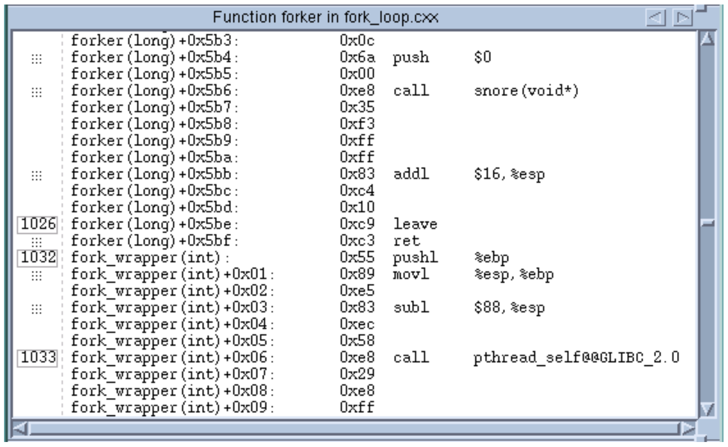


```
                    }  /* if *
                }  /* for */
            if (failures)
              sleep_for_seco
          } while (failures)
    }  /* if */

  if (!please_shut_up)
    printf ("Pid %d: Sleep
  fflush (stdout);
  snore (0);
}  /* forker */

/*************************
/* Spin a second thread, i

void fork_wrapper (int for
{
  pthread_t my_ptid = pthr
  pthread_t new_tid;
  pthread_attr_t attr;
  int whoops;
  int local_fork_count;
```

```
forker(long)+0x595:
forker(long)+0x596:
forker(long)+0x597:
forker(long)+0x598:   call     prir
forker(long)+0x599:
forker(long)+0x59a:
forker(long)+0x59b:
forker(long)+0x59c:
forker(long)+0x59d:   addl     $16,
forker(long)+0x59e:
forker(long)+0x59f:
forker(long)+0x5a0:   subl     $12,
forker(long)+0x5a1:
forker(long)+0x5a2:
forker(long)+0x5a3:   pushl    stdc
forker(long)+0x5a4:
forker(long)+0x5a5:
forker(long)+0x5a6:
forker(long)+0x5a7:
forker(long)+0x5a8:
forker(long)+0x5a9:   call     fflu
forker(long)+0x5aa:
forker(long)+0x5ab:
```

View > Assembler > Symbolically

Tells TotalView that it should display assembler code symbolically. This means that TotalView shows an instruction label and branch target symbolically as a function name plus an offset. (See Figure 39.)

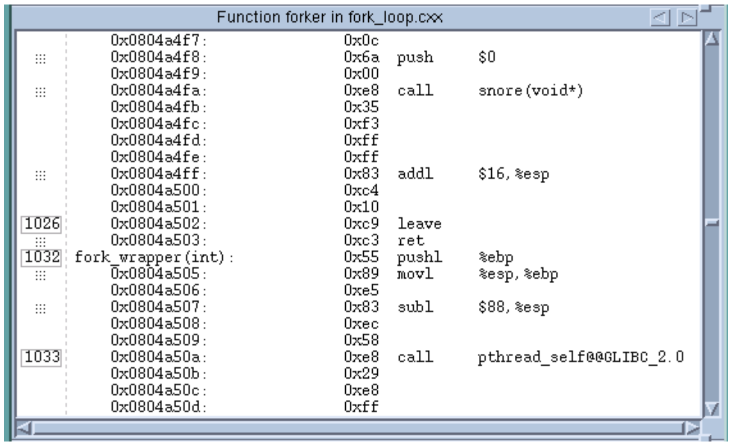


```
     forker(long)+0x5b3:       0x0c
     forker(long)+0x5b4:       0x6a  push   $0
     forker(long)+0x5b5:       0x00
     forker(long)+0x5b6:       0xe8  call   snore(void*)
     forker(long)+0x5b7:       0x35
     forker(long)+0x5b8:       0xf3
     forker(long)+0x5b9:       0xff
     forker(long)+0x5ba:       0xff
     forker(long)+0x5bb:       0x83  addl   $16,%esp
     forker(long)+0x5bc:       0xc4
     forker(long)+0x5bd:       0x10
1026 forker(long)+0x5be:       0xc9  leave
     forker(long)+0x5bf:       0xc3  ret
1032 fork_wrapper(int):        0x55  pushl  %ebp
     fork_wrapper(int)+0x01:   0x89  movl   %esp,%ebp
     fork_wrapper(int)+0x02:   0xe5
     fork_wrapper(int)+0x03:   0x83  subl   $88,%esp
     fork_wrapper(int)+0x04:   0xec
     fork_wrapper(int)+0x05:   0x58
1033 fork_wrapper(int)+0x06:   0xe8  call   pthread_self@@GLIBC_2.0
     fork_wrapper(int)+0x07:   0x29
     fork_wrapper(int)+0x08:   0xe8
     fork_wrapper(int)+0x09:   0xff
```

Figure 39: Source Pane: Showing Assembler Symbolically

View > Assembler > By Address

Tells TotalView that it should display assembler code by address. This means that TotalView shows the instruction labels and branch targets as hexadecimal addresses. TotalView will, however, always show the target address of branch-to-subroutine instructions symbolically. (See Figure 40.)



```
          0x0804a4f7:          0x0c
          0x0804a4f8:          0x6a  push   $0
          0x0804a4f9:          0x00
          0x0804a4fa:          0xe8  call   snore(void*)
          0x0804a4fb:          0x35
          0x0804a4fc:          0xf3
          0x0804a4fd:          0xff
          0x0804a4fe:          0xff
          0x0804a4ff:          0x83  addl   $16,%esp
          0x0804a500:          0xc4
          0x0804a501:          0x10
1026      0x0804a502:          0xc9  leave
          0x0804a503:          0xc3  ret
1032 fork_wrapper(int):        0x55  pushl  %ebp
          0x0804a505:          0x89  movl   %esp,%ebp
          0x0804a506:          0xe5
          0x0804a507:          0x83  subl   $88,%esp
          0x0804a508:          0xec
          0x0804a509:          0x58
1033      0x0804a50a:          0xe8  call   pthread_self@@GLIBC_2.0
          0x0804a50b:          0x29
          0x0804a50c:          0xe8
          0x0804a50d:          0xff
```

Figure 40: Source Pane: Showing Assembler By Address

View > Display Managers

When selected (which is the default), TotalView displays manager threads.

Manager threads are threads created by the operating system that support your program’s activities. In most cases, you are not interested in these threads, so clearing this command is usually what you want to do.

Group Menu Commands

The following commands are on the Group pulldown:

- **Group > Go** on page 67
- **Group > Halt** on page 68
- **Group > Next** on page 68
- **Group > Step** on page 68
- **Group > Out** on page 69
- **Group > Run To** on page 69
- **Group > Next Instruction** on page 70
- **Group > Step Instruction** on page 71
- **Group > Hold** on page 71
- **Group > Release** on page 71
- **Group > Detach** on page 74
- **Group > Attach Subset** on page 71
- **Group > Custom Groups** on page 73
- **Group > Restart** on page 74
- **Group > Kill** on page 74

All commands in this group operate at group width. If the command causes stepping to occur, the stepping is focused on the control group of the thread of interest.

Within the CLI, this is equivalent to executing a command having a focus of **gC**.

Group > Go

Starts or continues all processes in all control groups of the thread of interest.

Contrast the action performed by this command with the process and thread versions:

- The process version starts or continues all threads in the process of interest.
- The thread version continues only the current thread; the other threads in the process remain in their current state.

Typing **Group > Go** or **Process > Go** creates the process if you have not yet started executing your program.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Group > Halt

Halts all processes in the control group of the thread of interest. After this occurs, TotalView updates all windows associated with threads in this group.

Note the difference between this command and **Process > Halt**. In a multi-process program, this command stops all the processes in the current control group. **Process > Halt** stops all of the threads in the process of interest.

Group > Next

"Next steps" all processes in the control group that are in the same lockstep group as the thread of interest over a source line. That is, TotalView lets these lockstep threads execute one source line. If the line contains function calls, however, TotalView executes the entire function as if it were a single statement.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to run freely. It then waits until the thread of interest and each matching thread arrive at the next source statement or some thread hits a breakpoint or encounter an error.

If more than one statement exists on the line, all are executed.

Contrast this command with the process and thread versions:

- The process version steps all threads in the process of interest.
- The threads version only steps the current thread; the other threads in the process remain stopped during the step.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Group > Step

Steps all processes in the control group that are in the same lockstep group as the thread of interest. That is, TotalView lets these lockstep threads execute one source line. If the current line contains a function call, TotalView steps into this function.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to run freely. It then waits until the thread of interest and each matching thread arrive at the next source statement.

If more than one statement exists on the line, all are executed.

Contrast this command with the process and thread versions:

- The process version steps all threads in the process of interest.
- The threads version only steps the current thread; the other threads in the process remain stopped during the step.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Group > Out

Continues all processes in the control group that are in the same lockstep group as the thread of interest until execution returns from the current function.

You can tell TotalView to return out of more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can to return to routine A by selecting routine A in the Stack Trace Pane. In this case, the selected routine name is the routine you want to return *to*.

Be careful to distinguish between this command and "Run To". The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to run freely. It then waits until the thread of interest and each matching thread return from the function or some thread hits a breakpoint.

Contrast the action performed by this command with the thread and group version:

- The process version runs all threads in the process.
- The thread version runs only the current thread; the other threads in the process remain stopped.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Group > Run To

Continues all processes in the thread of interest's control group until one thread in each process in the thread of interest's lockstep group reaches the selected source line or instruction. While these threads are executing, other threads in all other control groups run freely.

If some process never reaches the selection, TotalView keeps waiting; if this happens, select **Group > Halt** *to interrupt the operation.*

Before using this command, you must select a line in the Source Pane. You can also select and run to a line in a another stack frame, in which case the operation does not complete until the thread reaches the selected line. If you are running to a line in another stack frame, TotalView will change the stack display because execution has moved to a different.

Contrast the action performed by this command with the thread and group version:

- The process version runs all threads in the process.
- The thread version runs only the current thread; the other threads in the process remain stopped.

Be careful to distinguish between this command and "Out"*. The difference is that* "Run To" *requires you to select a target line.* "Out" *either takes you out of the current routine or to a selected target routine.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Group > Next Instruction

"Next steps" all processes in the control group that are in the same lock-step group as the thread of interest over an assembler instruction. That is, TotalView lets these lockstep threads execute one source line. If the line contains a function call, TotalView executes the entire function as if it were a single statement.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to run freely. It then waits until the thread of interest and each matching thread reach this instruction.

Contrast the action performed by this command with the thread and group version:

- The process version runs all threads in the process of interest.
- The thread version runs only the current thread; the other threads in the process remain stopped.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the *TotalView Users Guide*.

Group > Step Instruction

Steps all processes in the control group that are in the same lockstep group as the thread of interest over one assembler instruction. That is, TotalView lets these lockstep threads execute one source line. If the line contains a function call, TotalView steps into this function.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to run freely. It then waits until the thread of interest and each matching thread reach this instruction.

Contrast the action performed by this command with the thread and group version:

- The process version runs all threads in the process of interest.
- The thread version runs only the current thread; the other threads in the process remain stopped.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Group > Hold

Holds all processes in the control group of the thread of interest. After using this command, you will need to explicitly release the processes before they can again execute.

If all of the processes in the control group are not currently held, this command stops and holds them.

For more information, see the "**Group > Release**" on page 71.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Group > Release

Releases all processes. "Releasing" means that TotalView will allow the process to execute if a command tell it to. That is, this command does not continue the group—you must use a separate "Go" command for that.

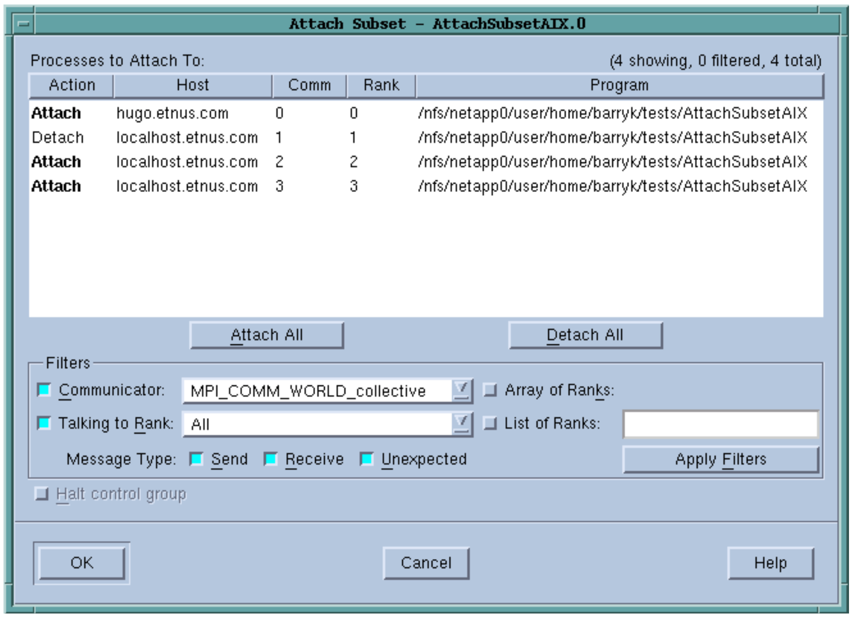
For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Group > Attach Subset

Lets you indicate which processes TotalView should attach to when these processes begin executing. Limiting the processes to which TotalView attaches is beneficial as TotalView does not have to be concerned with unattached processes. That is, because you know that you will not be interested in a what goes on in within a process, you can cut down on the time that TotalView uses to attach to all or most of your processes.

TotalView lets you start MPI jobs in two ways. One requires that the starter process be under TotalView control and have special instrumentation for TotalView, and the other does not. In the first case, you will enter the name of the starter program on the command line. The other requires that you enter information into the File > New Program or the Process > Startup Parameters dialog boxes. The Attach Subset command is only available if you directly name a starter program on the command line. This is discussed in the TotalView Users Guide.

Figure 41: Attach Subset Dialog Box



Attach Subset - AttachSubsetAIX.0

Processes to Attach To: (4 showing, 0 filtered, 4 total)

Action	Host	Comm	Rank	Program
Attach	hugo.etnus.com	0	0	/nfs/netapp0/user/home/barryk/tests/AttachSubsetAIX
Detach	localhost.etnus.com	1	1	/nfs/netapp0/user/home/barryk/tests/AttachSubsetAIX
Attach	localhost.etnus.com	2	2	/nfs/netapp0/user/home/barryk/tests/AttachSubsetAIX
Attach	localhost.etnus.com	3	3	/nfs/netapp0/user/home/barryk/tests/AttachSubsetAIX

Filters

Communicator: MPI_COMM_WORLD_collective

Talking to Rank: All

Array of Ranks:

List of Ranks:

Message Type: Send Receive Unexpected

Processes to Attach To

Use the controls in this area to specify the processes to which TotalView should attach when they are created. You have three choices:

Selection Area

Individually select or deselect processes

Attach All Attach to all of the listed processes.

Detach All Do not attach to any of these processes.

After selecting **All** or **None**, you can individually select or unselect processes. That is, if you only want to select a couple of processes, begin by clicking **None**, then select the few to which TotalView should attach.

Filters You can restrict the list by selecting the controls in this area.

Communicator

The communicators within this list tell TotalView which processes it should display. Selecting one of the communicators contained within this list tells TotalView that it should only display processes using this communicator. You can then select or clear these values in one of the three ways just discussed.

Talking to Rank

TotalView will limit the graph to communicators that receive messages from the indicated ranks. In addition to your rank numbers, TotalView includes two special variables: **All** and **MPI_ANY_SOURCE**.

Message Type

TotalView will only show **Send**, **Receive**, or **Unrestricted** messages.

Array of Ranks

This checkbox is automatically selected by TotalView if you have invoked this command from the **Tools > Attach Subset (Array of Ranks)** command within the Variable Window. If this window is displaying an array, this option specifies that the array's elements indicate ranks.

List of Ranks

After selecting this checkbox, you can now directly enter rank numbers. You can you a dash ("-") to indicate a range of ranks. For example, 3, 10-16, 24.

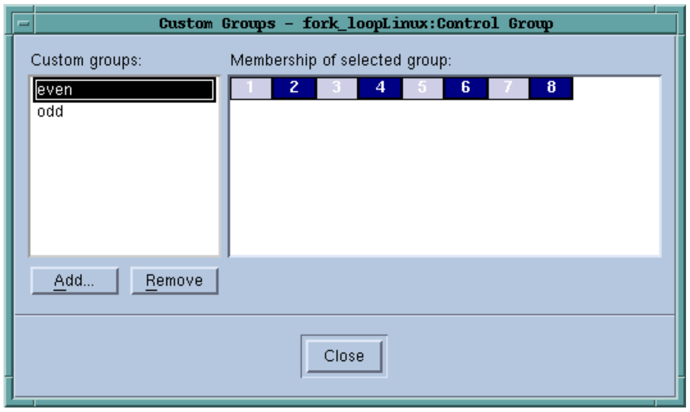
Halt control group

Selecting this button tells to stop all of the processes in the current process's control group after it attaches to a process. If it isn't selected. TotalView will immediately execute the control group after it attaches to them.

Group > Custom Groups

Allows you to create, delete, and edit process groups. (See Figure 42.)

Figure 42: Group > Custom Groups Dialog Box


Custom Groups - fork_loopLinux:Control Group
Custom groups:
even
odd
Membership of selected group:
1 2 3 4 5 6 7 8
Add...
Remove
Close

The first time you select this command, TotalView also displays a dialog box into which you can enter the name of the first group you will be creating. If you have already created a group, you will need to select the **Add** button before naming the group.

The way in which you create a group is simple: just select the processes (or ranks if you are debugging an MPI program). Select blocks in the normal way:

- **Clicking** selects the block. All other selections are removed.
- **Ctrl-clicking** adds to the selection.
- **Shift-clicking** lets you select a contiguous set of blocks.

If you have changed a group's membership and then press the **Add** button, TotalView asks if it should save the group. Similarly, if you select the **Close** button while a group's membership has changed, you are asked if the group's membership should change.

When you select the **Add** button, TotalView bases the newly formed created group and the group that you've selected. If this selection isn't what you want, just click on the first process or rank you want included. After clicking, only the selected block is selected.

To delete a group, select the group, then press the **Remove** button.

Group > Detach

Detaches all processes contained in the current control group. The Process Window from which you invoked this command is also removed.

Tis command releases all control over the processes, eliminates all debugger state information related to it (including action points). If you had attached to these processes, they continue executing in their normal runtime environment.

Group > Restart

Kill (deletes) all processes in the current control group, refocuses the Process Window on the master process, and then creates and starts this process. This command is equivalent to **Group > Kill** followed by a **Process > Go**.

Group > Kill

Kills (deletes) all processes in the current control group. TotalView focuses the Process Window on the master process.

The next time you start the program with the **Process > Go** command, TotalView creates and starts a new master process.

Process Menu Commands

The following commands appear on the **Process** pulldown:

- **Process > Go** on page 75
- **Process > Halt** on page 76
- **Process > Next** on page 76
- **Process > Step** on page 76
- **Process > Out** on page 77
- **Process > Run To** on page 77
- **Process > Next Instruction** on page 78
- **Process > Step Instruction** on page 78
- **Process > Hold** on page 79
- **Process > Hold Threads** on page 79
- **Process > Release Threads** on page 80
- **Process > Create** on page 80
- **Process > Detach** on page 80
- **Process > Startup Parameters** on page 80

All commands in this group operate on the current process. If the command causes stepping to occur, the stepping is focused on the control group of the thread of interest. A control group includes children that were forked (processes that share the same source code as the parent) and children that were forked but which subsequently called **execve()**. That is, a control group includes the children of the created processes that do not share the same source code as the parent.

Within the CLI, this is equivalent to executing a command having a focus of **pC**.

Process > Go

Starts or continues the current process. This command starts all threads in the process, not just the current thread.

Contrast the action performed by this command with the thread and group version:

- The group version runs all processes in the control group containing the thread of interest.
- The thread version runs only the thread of interest; the other threads in the process remain stopped during the step.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Halt

Halts the process. This command stops all executing threads in the process of interest. After they are stopped, TotalView updates the windows in which information about the process or its threads appears.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Next

"Next-steps" all threads in the lockstep group associated with the thread of interest that are within the current process over a source line. That is, TotalView lets these threads execute one source line. If the line contains function calls, TotalView executes the entire function as if it were a single statement.

If more than one statement exists on the line, all are executed.

If the process exists, TotalView runs the selected thread until the PC reaches the next address for which it has source line information. If the process does not yet exist, this command launches the process and runs the first thread until the PC reaches the first address for which TotalView has source line information.

While the lockstep group is being run to the next source line, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping, or another thread hits a breakpoint.

For related information, see "**Process > Go**" on page 75.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Step

Steps all threads in the lockstep group associated with the thread of interest that are within the process. That is, TotalView lets these lockstep threads execute one source line. If the current line contains a function call, TotalView steps into it.

If more than one statement exists on the line, all are executed.

If the process exists, TotalView runs the selected thread until the PC reaches the next address for which it has source line information. If the process does not yet exist, this command launches the process and runs the first thread until the PC reaches the first address for which TotalView has source line information.

While the lockstep group is being run to the next source line, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping, or another thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Out

Continues all threads in the lockstep group associated with the thread of interest that are within the process until the current thread returns from the function in which it is executing.

You can tell TotalView to return from more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can to return to routine A by selecting routine B in the Stack Trace Pane. (The selected routine name is the routine you want to return *from*.)

While the lockstep group is being run to the next source line, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group return from the routine or another thread hits a breakpoint.

For related information, see "**Process > Go**" on page 75.

Be careful to distinguish between this command and Run To*. The difference is that "*Run To*" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Run To

Continues all threads in the lockstep group associated with the thread of interest that are within the process. The processes continue running until threads in the lockstep group reach the selected source line or instruction in the Source Pane.

Before using this command, you must select a line in the Source Pane. You can also select and run to a line in a another stack frame, in which case the operation does not complete until the thread reaches the selected line. If you are running to a line in another stack frame, TotalView will change the stack display because execution has moved to a different.

While the lockstep group is being run to the next source line or instruction, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping or another thread hits a breakpoint.

Be careful to distinguish between this command and Out. The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

For related information, see "**Process > Go**" on page 75.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Next Instruction

"Next-steps" all threads in the lockstep group associated with the thread of interest that are within the current process over an assembler instruction. That is, this command tells TotalView to allow execution of the next instruction. If the current instruction is a function call, the call is executed as if it were a single instruction.

If more than one statement exists on the line, all are executed.

If the process exists, TotalView runs the selected thread until the PC reaches the next address for which it has information. If the process does not yet exist, this command launches the process and runs the first thread until the PC reaches the first address for which TotalView has information.

While the lockstep group is being run to the next instruction, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping or another thread hits a breakpoint.

For related information, see "**Process > Go**" on page 75.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Step Instruction

Steps all threads in the lockstep group associated with the thread of interest that are within the thread of interest's process over an assembler instruction. That is, TotalView lets one assembler instruction be executed. If the current instruction is a function call, TotalView steps into that function.

While the lockstep group is being run to the next instruction, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping or another thread hits a breakpoint.

Contrast this command with the thread and group equivalents:

- The thread version only steps the current thread.
- The group version steps all matching processes in the group.

If more than one statement exists on the line, all are executed.

If the process exists, TotalView runs the selected thread until the PC reaches the next address for which it has information. If the process does not yet exist, this command launches the process and runs the first thread until the PC reaches the first address for which TotalView has information.

For related information, see "**Process > Go**" on page 75.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Hold

Tells to TotalView to hold or release the current process.

- *Hold* means that you are telling TotalView that when you use a command that would run the threads in this process (such as a **Group > Go**), TotalView should ignore the command.
- *Release* means that you are telling TotalView that the thread can execute when you use a run command such as **Group > Go**.

The **Process > Hold Threads** on page 79 command can also hold. The description for that command describes the differences.

If the process is not currently held, this command stops and holds it. If it is currently held, this command releases it (but it does not continue the process—you must use a separate **Go** command for that).

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Hold Threads

Tells to TotalView to hold all threads in the current process. *Hold* means that you are telling TotalView that when you use a command that would run the threads in this process (such as a **Group > Go** command), it is ignored. Note that TotalView does not allow you to hold manager threads.

If a thread is already held when you enter this command, this state will be remembered when you enter a **Process > Release Threads** on page 80 command.

Compare this command with the **Process > Hold** on page 79 command. If you hold a process, none of the threads will ever run; that is, a *go* has no effect. When the process is held, the hold state of any of the process's threads is ignored.

If you release the process and hold all of the threads, the effect is essentially the same: none of the non-manager threads will run; and a *go* command will only run manger threads.

Here's one example illustrating why you need both commands. Suppose a thread barrier is not yet satisfied: some threads are held at the barrier and some are expected to get there later. Since a thread barrier can span processes, you can have multiple processes with multiple threads trying to get to the barrier. You may want to hold one or more processes while allowing threads in other processes to get to the barrier. In this case, holding at the process level means that you are not destroying the hold state of the individual threads. At a later time, you can release the processes to allow the remaining threads to get to the barrier.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Release Threads

Releases all threads that were held in the process. See **Process > Hold Threads** on page 79 for more information.

Process > Create

Create a process without starting it. If a program is linked with shared libraries, TotalView allows the dynamic loader to map into these libraries. Creating a process without starting it is useful if you need to:

- Create action points, watchpoints, or change global variables after a process is created, but before it runs.
- Debug C++ static constructor code.

Process > Detach

Detaches from the current process. Once detached, the process is no longer under TotalView control.

If you want to stop the process's execution, you should also use a **Thread > Continuation Signal** command to send a signal to the process. If you set the continuation signal to **SIGSTOP**, the process is stopped after you detach from it. Note that you can only send a signal to a thread; you cannot detach from an individual thread.

Use the **Group > Attach Subset** command to detach from one or more processes in a parallel program.

Process > Startup Parameters

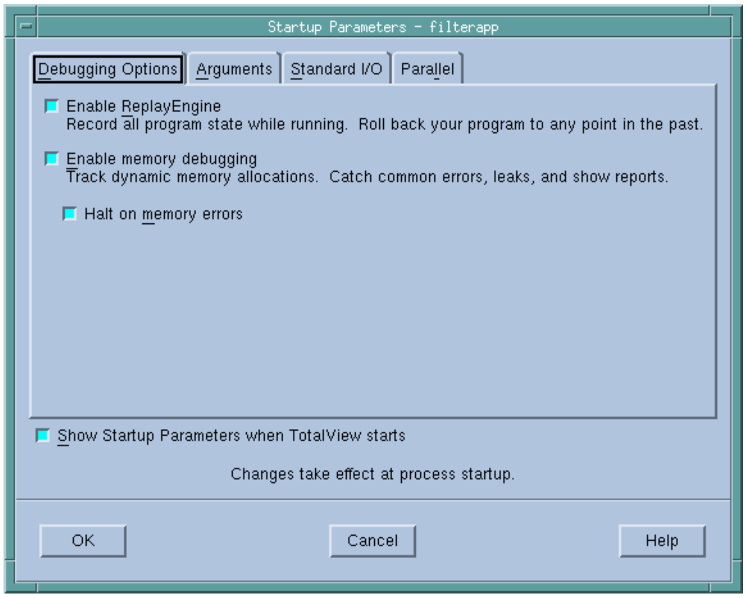
Use this dialog box to set:

- Debugging Options
- Commands and environments
- Standard I/O
- Parallel properties

In all cases, the information you enter or change in this dialog box is not used until TotalView starts a process.

Debugging Options Page

Use this page to enable the ReplayEngine and the Memory Debugger. Both must be enabled before your program begins execution. If you want to enable of disable these options, TotalView only makes the change when you restart your program.



Enable ReplayEngine
Record all program state while running. Roll back your program to any point in the past.

Enable memory debugging
Track dynamic memory allocations. Catch common errors, leaks, and show reports.

Halt on memory errors

Show Startup Parameters when TotalView starts

Changes take effect at process startup.

Figure 43: Process > Startup Parameters Dialog Box: Debugging Options Page

- When you select the **Enable ReplayEngine** checkbox, TotalView will record your program's statements as they are executed. This is only available on Linux-x86 and Linux-x86-64 machines. It always appears on these machines even if you do not have a license for it.
- When you select the **Enable memory debugging** checkbox, TotalView will record all interactions with the malloc library. When you select this check box, TotalView sets the **Halt on memory errors** check box. If you do not want he memory debugger to stop execution, uncheck it.
- This dialog box is displayed when you name the program being debugged as an argument to the **totalview** command.If you do not want it displayed, uncheck this check box.

Arguments Page

Use this page to define both the arguments that TotalView passes to a process when it is next launched and environment variables.

Command-line arguments: The arguments typed in this area are those that you would have entered if you were starting the program from a shell. If you were directly starting your program under TotalView control, these arguments are those you would enter using the TotalView –**a** command-line option.

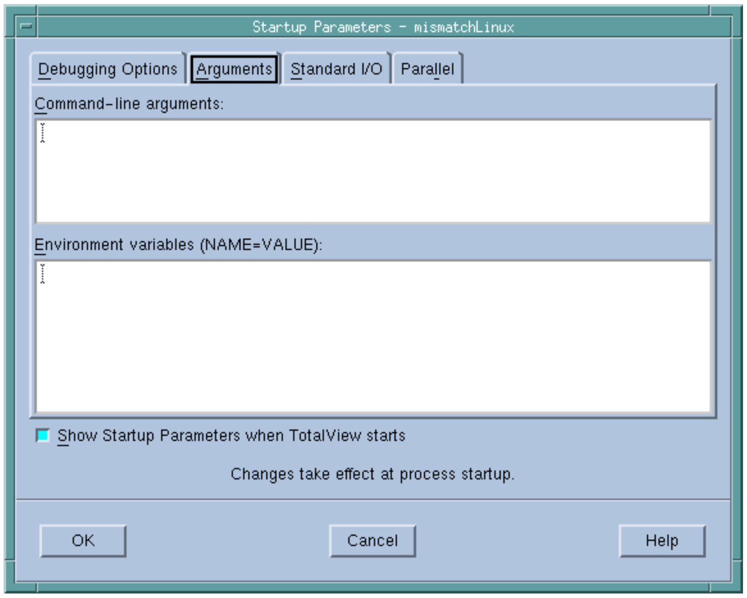

Startup Parameters - mismatchLinux
Debugging Options
Arguments
Standard I/O
Parallel
Command-line arguments:
Environment variables (NAME=VALUE):
Show Startup Parameters when TotalView starts
Changes take effect at process startup.
OK
Cancel
Help

Figure 44: Process > Startup Parameters Dialog Box: Arguments Page

This tab is identical to the Arguments tab that TotalView displays when you select the File > New Program *command.*

TotalView uses them arguments whenever it starts your program. In contrast, if you need to use arguments to send information to a starter process such as **mpirun** or **poe**, enter those arguments in the Parallel tab.

You can enter arguments in two ways:

- Place them on separate lines.
- Separate them with blanks.

If either case, an argument must be entered on one line. TotalView will rewrap what you type, so do not be concerned with how it looks in this window.

Here are some special cases:

- If an argument contains embedded blanks, enclose the argument in quotation marks (").
- If an argument contains a quotation mark, precede it with a backslash.
- If an argument contains a backslash character (\), precede it with a second backslash.
- TotalView interprets **\n** as an embedded newline.

As the information in this page is just text, use standard dialog box editing commands to remove arguments you no longer need. If you delete these arguments before execution begins, TotalView does not use them.

Environment variables: Use this area to define additional environment variables that TotalView passes to a process when it is launched.

By default, a new process inherits TotalView environment variables, and a remote process inherits **tvdsvr**'s environment variables. Using this window, you can add new variables, change the value of existing variables, or delete an existing variable.

An environment variable is specified as *name=value*. For example, the following definition creates an environment variable named **DISPLAY** whose value is **unix:0.0**:

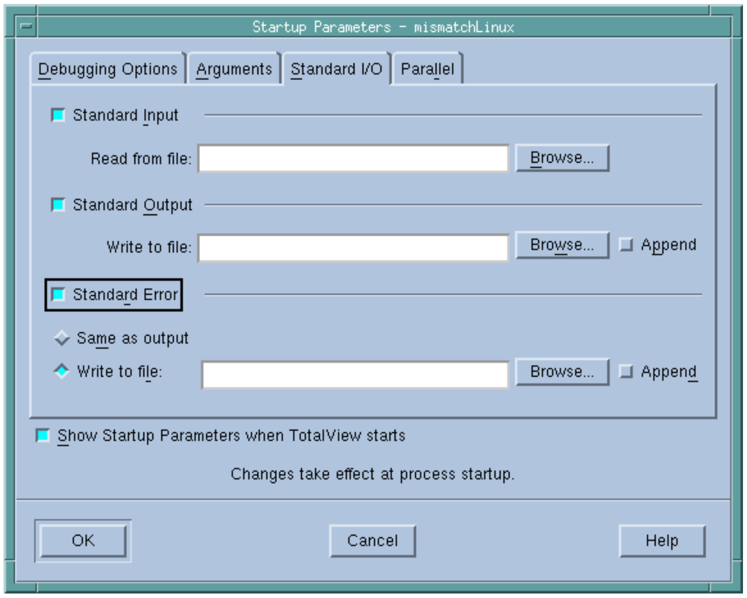
```
DISPLAY=unix:0.0
```

Place each environment variable on a separate line.

Standard I/O Page

The controls within this page let you change how TotalView handles standard input (**stdin**), standard output (**stdout**), and standard error (**stderr**). Each is handled separately. (See Figure 45 on page 83.)

Figure 45: Process > Startup Parameters Dialog Box: Standard I/O Page



Startup Parameters - mismatchLinux

Debugging Options | Arguments | Standard I/O | Parallel

☑ Standard Input
Read from file: [ ] Browse...

☑ Standard Output
Write to file: [ ] Browse... ☐ Append

☑ Standard Error
◇ Same as output
◆ Write to file: [ ] Browse... ☐ Append

☑ Show Startup Parameters when TotalView starts

Changes take effect at process startup.

OK Cancel Help

This tab is identical to the Standard I/O tab that TotalView displays when you select the **Process > Startup Parameters** *command.*

If you want to use the default **stdio**, **stdout**, or **stderr**, you can clear the button that precedes the area.

Standard Input Lets you name the file that will be connected to the process's standard input (**stdin**) when it is next launched.

Processes running under TotalView control inherit standard input from TotalView. This field lets you set the target process's standard input to be a file. You must do this before the process is created.

Read from file

If your program should receive input from a file, you can either type the file name directly or use the **Browse** button to locate the file.

Standard Output

Lets you name the file that will be connected to the process's standard output (**stdout**) when it is next launched.

Processes run under TotalView control inherit their standard output from TotalView. This field lets you set the target process's standard output to a file. You must do this before the process is created.

Write to file

If you want your program to send output to a file, you can either type the file name directly or use the **Browse** button to locate the file.

stdout is buffered. If it is pointed to a file, the last few lines of the program's output are not actually written to the file until the buffer is flushed. If the target process terminates abnormally or if TotalView deletes it, the last few lines of output may never be written to the file.

Standard Error

Lets you name the file that will be connected to the process's standard error (**stderr**) when it is next launched.

Processes run under TotalView control inherit **stderr** from TotalView. This field lets you set the target process's **stderr** to a file. You must do this before the process is created.

stderr is buffered. If it is pointed to a file, the last few lines of the program's output are not actually written to the file until the buffer is flushed. If the target process terminates abnormally or if TotalView deletes it, the last few lines of output may never be written to the file.

Write to file

If you want your program to send error information to a file, you can either type the file name directly or use the **Browse** button to locate the file.

Same as output

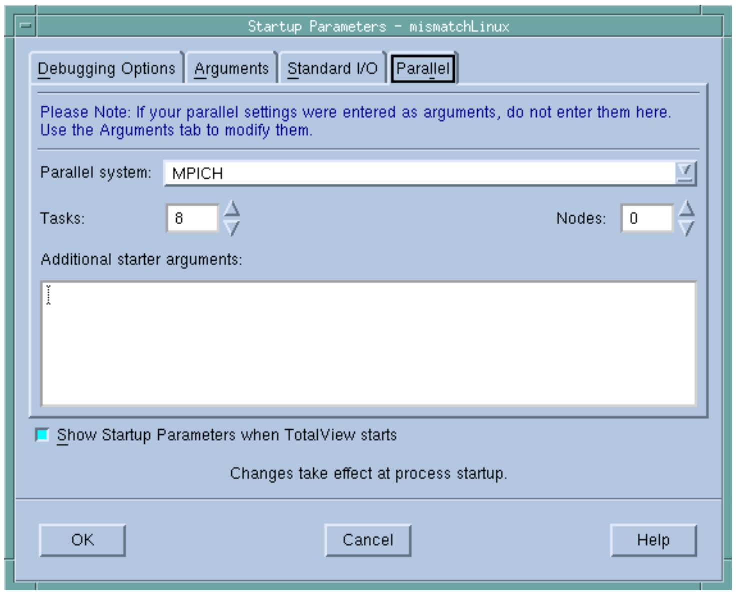
If you would like **stderr** to go to the same file as **stdout**, select this check box.

Parallel Page

The Parallel page lets you tell TotalView how it should start your parallel job. You can, of course, also directly start your job directly from a shell.

This tab is identical to the Parallel tab that TotalView displays when you select the **Process > Startup Parameters** *command.*

Figure 46: Process > Startup Parameters Dialog Box: Parallel Page



Please Note: If your parallel settings were entered as arguments, do not enter them here. Use the Arguments tab to modify them.

Parallel system: MPICH

Tasks: 8

Nodes: 0

Additional starter arguments:

Show Startup Parameters when TotalView starts

Changes take effect at process startup.

Parallel system

Select which parallel system profile TotalView should use when it starts your program. This profile can be one that TotalView Technologies provides, one created for your site, or one that you create. For more information, see the appendix in the *TotalView Reference Guide*.

Tasks	Enter a number indicating how many tasks your program should create. Entering a value of 0 (zero) indicates that your system's default value should be used.
Nodes	Enter a number indicating how many nodes your program should use when running your program. Not all systems use this value. Entering a value of 0 (zero) indicates that your system's default value should be used.

Additional starter arguments

If your program's execution requires that you use arguments to send information to the starter process such as **mpirun** or **poe**, enter them in this area. In contrast, if you need to use arguments to send information to your program, enter those arguments in the Arguments tab.

Issues When Using Starter Programs

Starter programs such as **poe** or **aprun** and TotalView can interfere with one another because each believes that it owns **stdin**. Because the starter program is trying to manage **stdin** on behalf of your processes, it continually reads from **stdin**, acquiring all characters that it sees. This means that TotalView never sees these characters. If your target process does not use **stdin**, you can use the **–stdinmode none** option. Unfortunately, this option is incompatible with **poe –cmdfile** option that is used when specifying **–pgmmodel mpmd**.

Thread Menu Commands

The following commands appear on the Thread pulldown:

- **Thread > Go** on page 86
- **Thread > Halt** on page 86
- **Thread > Next** on page 87
- **Thread > Step** on page 87
- **Thread > Out** on page 87
- **Thread > Run To** on page 88
- **Thread > Next Instruction** on page 89
- **Thread > Step Instruction** on page 89
- **Thread > Set PC** on page 89
- **Thread > Hold** on page 89
- **Thread > Continuation Signal** on page 90

Thread > Go

Continues just the current thread (the thread of interest) without starting other threads in the process.

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.

Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX *pthread program. These features are supported when your compiler used sprocs; for example, the* MIPSpro OpenMP *compiler.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Thread > Halt

Stops execution of this thread without affecting other threads in the process. After the thread stops, TotalView update this window.

Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX *pthread program. These features are supported when your compiler used sprocs; for example, the* MIPSpro OpenMP *compiler.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Thread > Next

"Next-steps" the current thread over a source line. That is, TotalView executes one line in this thread. If the line contains a function call, TotalView executes the call as if it were a single statement. If more than one statement exists on the line, all are executed.

While this thread is being stepped, no other process executes.

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.

Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX *pthread program. These features are supported when your compiler used sprocs; for example, the* MIPSpro OpenMP *compiler.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Thread > Step

Steps the current thread over a source line. That is, TotalView executes one line in this thread. If the line contains a functions call, TotalView steps into the function. If more than one statement exists on the line, all are executed.

While this thread is being stepped, no other process executes.

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.

Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX *pthread program. These features are supported when your compiler used sprocs; for example, the* MIPSpro OpenMP *compiler.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Thread > Out

Continues the current thread until execution control returns from the function in which execution had stopped. Other threads do not run freely while this action is occurring.

You can tell TotalView to return from more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace

Pane. For example, if routine A calls routine B and routine B calls routine C, you can to return to routine A by selecting routine A in the Stack Trace Pane. In this case, the selected routine name is the routine you want to return *to*.

Be careful to distinguish between this command and Run To. *The difference is that "*Run To*" requires you to select a target line. "*Out*" either takes you out of the current routine or to a selected target routine.*

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.

Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX *pthread program. These features are supported when your compiler used sprocs; for example, the* MIPSpro OpenMP *compiler.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Thread > Run To

Tells TotalView that it should let the thread run until it reaches the selected source line or instruction in the Source Pane. Other threads do not run freely while this action is occurring.

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.

Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX *pthread program. These features are supported when your compiler used sprocs; for example, the* MIPSpro OpenMP *compiler.*

Be careful to distinguish between this command and Run To. *The difference is that "*Run To*" requires you to select a target line. "*Out*" either takes you out of the current routine or to a selected target routine.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Thread > Next Instruction

"Next-steps" the current thread over one assembler instruction. That is, TotalView executes one instruction in this thread. If the instruction contains a function call, TotalView executes the call as if it were a single statement.

While this thread is being stepped, no other process executes.

Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX *pthread program. These features are supported when your compiler used sprocs; for example, the* MIPSpro OpenMP *compiler.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Thread > Step Instruction

Steps the current thread over one assembler instruction. That is, TotalView executes one instruction in this thread. If the instruction contains a function call, TotalView steps into the function.

While this thread is being stepped, no other process executes.

Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX *pthread program. These features are supported when your compiler used sprocs; for example, the* MIPSpro OpenMP *compiler.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Thread > Set PC

Changes the program counter (PC) to the value of the selected line. That is, before you select this command, select a line in the Source Pane. Ater selecting this command, TotalView asks for confirmation before it procedes.

If the Source Pane is displaying the source for the selected stack frame and if the selected frame is not on the top of the stack, TotalView will attempt to unwind the stack and restore the registers before adjusting the PC. If it needs to perform these operations, it asks if it is OK to proceed.

Thread > Hold

When this command is checked, you are telling TotalView that it should hold the thread. When it is unchecked, the thread can run when it receives a *go* command.

Hold (checked): Holds the thread. This stops the thread and places a hold on it. *Hold* means that a *go* command cannot tell this thread that it should begin executing. Only after you release the thread is it eligible to run.

Release (unchecked): Releases the thread. *Releasing* means that the thread will run if it subsequently receives a *go* or other run-level command.

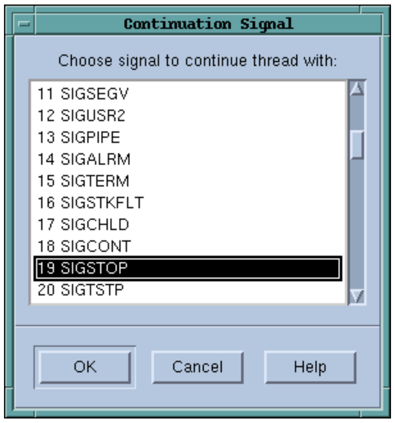
Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX *pthread program. These features are supported when your compiler used sprocs; for example, the* MIPSpro OpenMP *compiler.*

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Thread > Continuation Signal

Use this dialog box to tell TotalView to send a signal to the thread of interest when the thread resumes execution.

Figure 47: Thread > Continuation Signal Dialog Box



Select the name of the signal that TotalView will send to the selected thread the next time it is continued.

If you do not want a signal sent to the thread, select the first (**No Signal Pending**) item in the list.

Selecting a signal for a thread clears the continuation signals for all other threads for the same process. In other words, only one thread can have a continuation signal set at one time.

The signal is only sent the first time the thread is continued. If you need it sent a second time, you will need to reenter this command.

Action Point Menu

The following commands are on the **Action Point** pulldown:

- **Action Point > Set Breakpoint** on page 91
- **Action Point > Set Barrier** on page 91
- **Action Point > At Location** on page 91
- **Action Point > Create Watchpoint** on page 93
- **Action Point > Enable** on page 93
- **Action Point > Disable** on page 93
- **Action Point > Delete** on page 93
- **Action Point > Properties** on page 93
- **Action Point > Suppress All** on page 99
- **Action Point > Delete All** on page 99
- **Action Point > Load All** on page 99
- **Action Point > Save All** on page 100
- **Action Point > Save As** on page 100

Action Point > Set Breakpoint

Tells TotalView to set a breakpoint at the selected line or instruction. The properties of this breakpoint are:

- When Hit, Stop Group
- Enabled
- Plant in Share Group

For more information, see "**Action Point > Properties**" on page 93.

Action Point > Set Barrier

Tells TotalView to set a barrier breakpoint at the selected line or instruction. The properties of this breakpoint are:

- When Hit, Stop Group
- When Done, Stop Group
- Enabled
- Plant in Share Group

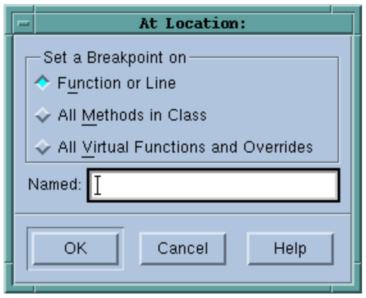
For more information, see "**Action Point > Properties**" on page 93.

Action Point > At Location

Use this dialog box to tell TotalView to toggle an action point. That is, if an action point is not at the location, TotalView creates one. If, however, an action point exists at this location, TotalView disables it.

After you click OK, TotalView either sets or disables a breakpoint. If you had entered a line number and the line does not contain an executable state-

Figure 48: Action Point > At Location Dialog Box



ment, TotalView either sets of disables a breakpoint on the next source line that has executable code.

You can also set action points by clicking on a line number in the Source Pane.

This dialog box has three options:

- **Function or Line**: Sets or disables a breakpoint at a function or line.
- **All Methods in Class**: Sets or disables one or more breakpoints on all methods contained within the named class. If you only want to set a breakpoint on some of these methods, first set breakpoints on all methods, then open the Properties dialog box, and then select the **Addresses** button. You will now be able to unselect the ones not needed.

  Although a set of breakpoints was created—and you will see a breakpoint when you display the method—each of the breakpoints has the same breakpoint ID. When you click on the breakpoint in the Action Points tab, TotalView displays a dialog box in which you can indicate which one you want to manipulate or see.
- **All Virtual Functions and Overrides**: Sets or disables one or more breakpoints on all named virtual functions and their overrides. Generally, handle the set of breakpoints that this command creates the same as those created when you select **All Methods in Class**.

If you type a function name that TotalView has no information about into the **Action Point > At Location** dialog box, it assumes that you have either mistyped the function name or that the library containing the function has not yet been loaded into memory. In this circumstance, you can tell it to set it anyway because the breakpoint could exist in a shared library or it could be loaded later. These kind of breakpoints are called *pending breakpoints*. When libraries are loaded, TotalView checks for the function's name. If the name is found, it sets the breakpoint. If it isn't in a newly library, TotalView just keeps on waiting for it to be loaded. You'll see information in the Action Points tab that tells you that the breakpoint is pending.

If the name you type is similar to the name of an existing function, TotalView displays its **Ambiguous Function** dialog box that lets you select which of these existing functions it should set a breakpoint on. If, however, the function will be loaded into memory later, you can set a pending breakpoint

If the name you entered was not ambiguous, TotalView just asks if it should set a pending breakpoint.

For more information on the information you can enter into this dialog box, consult the **dbreak** CLI command information.

TotalView can only place one action point on an address. Because the breakpoints you specify are actually expressions, the locations to which these expressions evaluate can overlap or even be the same. Sometimes, and this most often occurs with pending breakpoints in dynamically loaded libraries, TotalView cannot tell when action points overlap. If they do, TotalView only enables one of the action points and disables all others that evaluate to the same address. The action point that TotalView enables is the one with the lowest action point ID.

Action Point > Create Watchpoint

Creates a watchpoint. If you have selected information in the Source or Stack Frame Panes, TotalView immediately displays its **Watchpoint Properties** dialog box. If nothing is selected or it can't determine where it should set the watchpoint, TotalView displays a dialog box into which you can type an expression.

For more information, see "**Tools > Create Watchpoint**" on page 142.

Action Point > Enable

Enables a previously disabled action point. That is, when execution reaches the line or instruction containing this breakpoint, TotalView performs the action point's activity.

Action Point > Disable

Disables the selected action point. Disabling an action point leaves it set within TotalView but makes it inactive. That is, when execution reaches a disabled action point, TotalView ignores it. In contrast, if an action point is enabled, TotalView performs the action point's activity.

Action Point > Delete

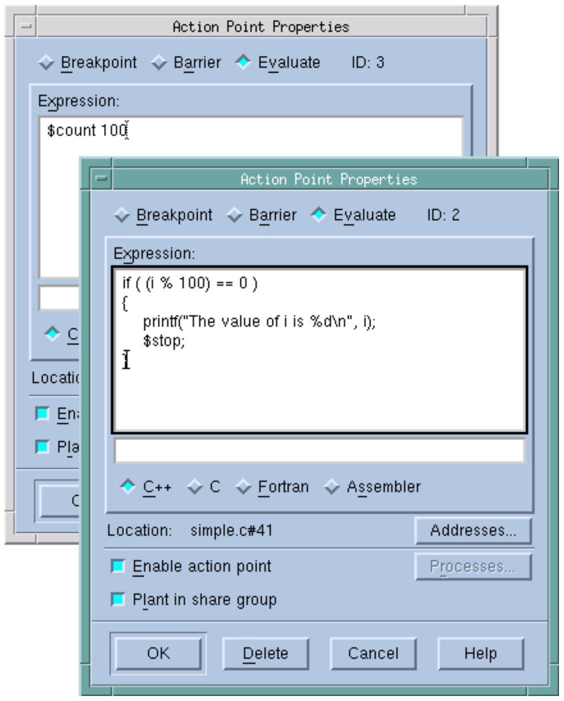
Deletes the action point associated with the current line or instruction.

Action Point > Properties

Use this dialog box to modify the properties of an existing action point. This dialog box lets you control attributes of your action point as well as change it from one kind of action point into another. If you are creating an evaluation point, you will use this window to enter your C, Fortran, or Assembler code.

The only kind of action point not controlled by this dialog box is the watchpoint. Use the **Tools > Create Watchpoint** *command within a Variable Window when you want to set or alter a watchpoint.*

Figure 49: Action Point > Properties Dialog Box



Action Point Properties

Breakpoint Barrier [✓ Evaluate] ID: 3

Expression:

```
$count 100
```

Action Point Properties

Breakpoint Barrier [✓ Evaluate] ID: 2

Expression:

```
if ( (i % 100) == 0 )
{
    printf("The value of i is %d\n", i);
    $stop;
```

[✓ C++] C Fortran Assembler

Location: simple.c#41

[✓] Enable action point

[✓] Plant in share group

The following three controls let you set or change what will happen when a program encounters an action point:

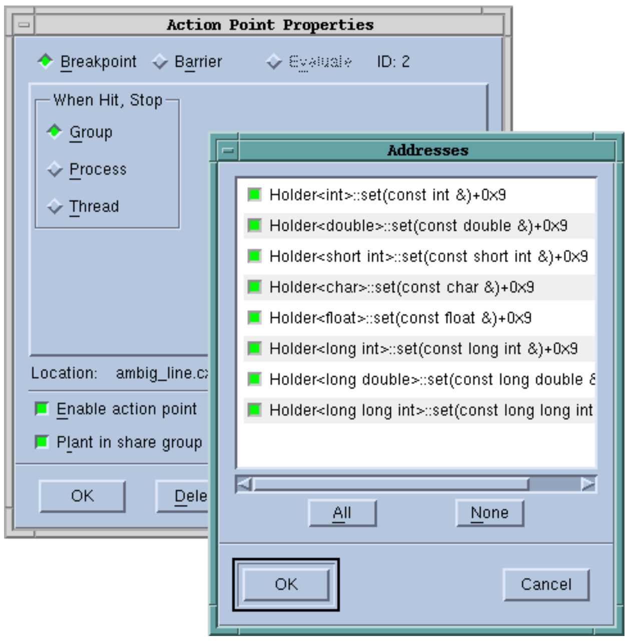
Breakpoint	When an executing thread encounters a breakpoint, it stops at the breakpoint. Other controls let you indicate if the thread's process or control group will also stop.
Barrier	Process barrier breakpoints are similar to simple breakpoints, differing in that they let you synchronize a group of processes in a multiprocess program. Other controls let you indicate if the thread's process or control group will also stop and what condition must be satisfied for TotalView to release threads held at the barrier.
Evaluate	An evaluation point is a breakpoint that has code associated with it. When a thread or process encounters an evaluation point, it executes this code. You can use evaluation points in several different ways, including as conditional breakpoints, thread-specific breakpoints, countdown breakpoints, and for patching code fragments into and out of your program.

General Controls

The following four controls are used by all three kinds of action points.

Addresses
If the selected line could be mapped to more than one place—for example, you selected a line in a template or an inline function—selecting this button tells TotalView to display a dialog box that lets you refine where TotalView places breakpoints. Do this by individually selecting or clearing the locations at which TotalView will set the breakpoint.

Figure 50: Addresses Dialog Box



Holder<int>::set(const int &)+0x9
Holder<double>::set(const double &)+0x9
Holder<short int>::set(const short int &)+0x9
Holder<char>::set(const char &)+0x9
Holder<float>::set(const float &)+0x9
Holder<long int>::set(const long int &)+0x9
Holder<long double>::set(const long double &
Holder<long long int>::set(const long long int

Process
Lets you indicate which process in a multi-process program will have enabled breakpoints. After selecting this button, TotalView displays a dialog box similar to the one it displays when you select the **Addresses** button. Note that if **Plant in share group** is selected, this button is not enabled because you've told TotalView to set the breakpoint in all processes.

Enable action point
When set, TotalView activates this action point. If this control is not selected, TotalView ignores the action point.

Plant in share group
When set, the action point is shared among all of the threads in the thread's share group. In all cases, TotalView places an action point in each member of the share group.

If you select this option, TotalView enables (makes active) all of these points.

If you do not select this option, TotalView only enables this action point; all others are disabled.

Breakpoint: Setting a breakpoint tells TotalView that when execution reaches this line, it should not allow execution to continue. (See Figure 51 on page 96.)

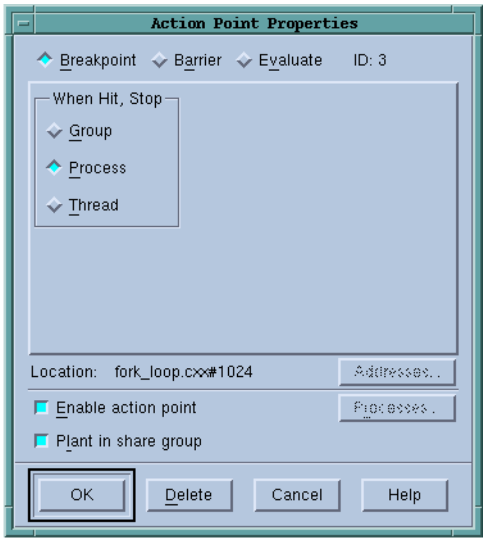


Figure 51: Action Point > Properties Dialog Box

The **When Hit, Stop** radio buttons indicate which other threads TotalView should stop when execution reaches the breakpoint, as follows:

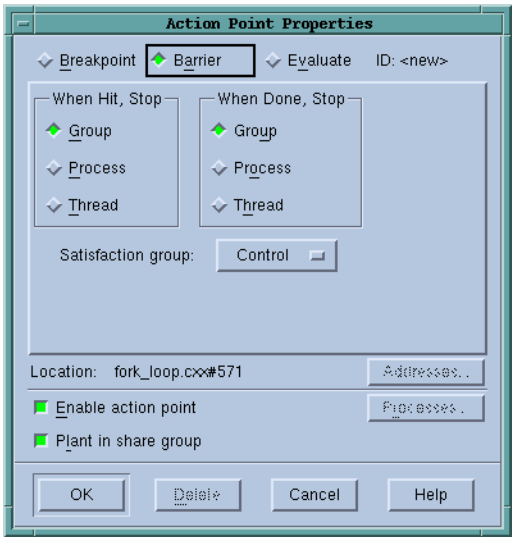
Scope	TotalView will:
Group	Stop all threads in the current thread's control group.
Process	Stop all threads in the current thread's process.
Thread	Only stop this thread.

You need to be careful when setting this attribute. For example, if you tell TotalView that it should stop the thread when this breakpoint is hit and you also tell it that the satisfaction set is, for example, the workers group, your thread will stop but the remainder of the workers group will continue executing. In most cases, this is not what you would want as most of the time you will be stopping an individual thread to examine all of the program's state at that time and TotalView cannot give you the entire state unless the other threads are also stopped.

Barrier: Select this radio button to set a process barrier breakpoint. Process barrier breakpoints are triggered when execution arrives at a line or instruction. (See Figure 52.)

Barrier breakpoints are most often used to synchronize a set of processes and threads. 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, *step*, *next*, or *go*) 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*

Figure 52: Action Point > Properties Dialog Box



continued. That is, they are left stopped at the barrier. If you now continue the process, those threads stopped at the barrier also run. This is in addition to any other threads that were not participating with the barrier.

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

When Hit, Stop Indicate which other threads TotalView should stop when execution reaches the breakpoint, as follows:

Group: Stop all threads in the current thread's control group.

Process: Stop all threads in the current thread's process.

Thread: Only stop this thread.

After all processes or threads reach the barrier, TotalView releases all held threads. (*Released* means that these threads and processes can now run.)

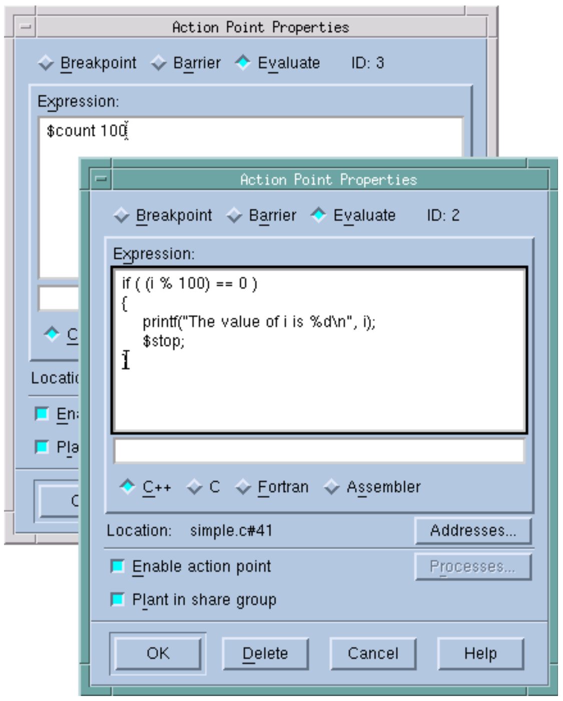
When Done, Stop

Tells TotalView what else it should stop. The meaning of these buttons is the same as the buttons in the **When Hit, Stop** area.

Satisfaction group

For even more control over what is stopped, you can indicate a *satisfaction set*. This set indicates which threads must be held before any of these held threads can be released. That is, the barrier is *satisfied* when all of the indicated threads are held. Your selection here tells TotalView that the satisfaction set consists of all threads in the current thread's **Share**, **Workers**, or **Lockstep** group.

Evaluate: When your program encounters an evaluation point, the code you enter is executed. TotalView can either compile or interpret the code you enter here. On some machines, it can only interpret your code. If TotalView can compile your code, it patches the compiled code into your process. If TotalView must interpret your code, TotalView executes this information after your program reaches the evaluation point. In either case, the code executes *before* the code within the source line executes.



```
$count 100
```

```
if ( (i % 100) == 0 )
{
    printf("The value of i is %d\n", i);
    $stop;
```

Location: simple.c#41

Figure 53: Action Point > Properties Dialog Box

When entering code, indicate which programming language you are using. You can specify **C++**, **C**, **Fortran**, or **Assembler**.

Chapter 16 of the *TotalView Users Guide* contains an extensive evaluation points discussion.

Here are some examples of ways to use evaluation points:

Countdown Breakpoints

The following is an example of a C language countdown breakpoint:

```
static int count = 100;
if (count-- == 0) {
   $stop;
   count = 100;
}
```

Conditional Breakpoints

The following is an example of a C language conditional breakpoint:

```
if (index < 0 || ptr == 0)
   $stopall;
```

Conditional Barriers

The following is an example of a C language conditional barrier point:

```
if (index < 0 || ptr == 0)
   $holdstopall;
```

In these examples, the **$stop**, **$stopall**, and **$holdstopall** special directives stop a process when a condition is true. For more information, see "**Built-In Statements**" in the *TotalView Users Guide*.

Action Point > Suppress All

Toggles the state of all action points between suppressed and unsuppressed, as follows:

Suppressed—command is selected

TotalView saves the current enabled/disabled state of each action point in the process, in addition to disabling the action points.

Unsuppressed—command is not selected

TotalView restores the saved enabled/disabled state of actions points.

Suppressing and *disabling* action points do similar things. If you suppress or disable an action point, TotalView ignores it when it is encountered. They differ in that *suppressing* disables all action points. In contrast, you can only disable action points individually. In other words, *suppress* means *disable all*.

When you suppress action points, you are also telling TotalView that it can not create additional points.

Action Point > Delete All

Removes all the action points in the current Process Window. After selecting this command, TotalView asks if this is really what you meant to do.

You can delete suppressed and disabled action points.

Action Point > Load All

Reads and sets the action point information previously written to a file. See **Action Point > Save All** on page 100 for more information.

TotalView can be told to automatically write this information to the file at the end of a session and to reload it when you start a new session. For more information, see the Action Points Page within the File > Preferences command.

When TotalView sets these action points, it will overwrite any changes you have made. For example, if you had changed a breakpoint to an evaluation point and entered code in this evaluation point, executing this command would change the evaluation point back to a breakpoint. Similarly, if you delete one of the action points saved in this file, it is restored. Reloading the saved action point does not, however, alter action points added to other lines.

Action Point > Save All

Writes information about all current action points (except watchpoints) to a file. These action points are placed in a file named **program.TVD.breakpoints**. You can restore these saved action points at a later time by using the **Action Point > Save As** command.

If a file with this name already exists, TotalView overwrites it.

TotalView can be told to automatically write this information to the file at the end of a session and to reload it when you start a new session. For more information, see the Action Points Page within the **File > Preferences** *command. Also, you can reload a previously saved file by using the Action Point > Save As command.*

Action Point > Save As

Saves all of your action point information to a file that you will name in the dialog box that will appear.

TotalView can be told to automatically write this information to the file at the end of a session and to reload it when you start a new session. For more information, see the Action Points Page within the File > Preferences command.

Use the **Action Point > Load All** command to load a saved action point file.

Debug Menu Commands

The following commands are on the Debug pulldown:

Replay

- **Debug Enable ReplayEngine** on page 101
- **Debug > Previous** on page 101
- **Debug > Unstep** on page 101
- **Debug > Caller** on page 101
- **Debug > Back To** on page 101
- **Debug > Live** on page 101

Memory Debugging

- **Debug > Enable Memory Debugging** on page 102
- **Debug > Stop on Memory Errors** on page 102
- **Debug > Open MemoryScape** on page 102
- **Debug > Heap Baseline >Set Heap Baseline (in Process)** on page 102

- **Debug > Heap Baseline> Set Heap Baseline (in Group)** on page 102
- **Debug > Heap Baseline >Heap Change Summary** on page 103
- **Debug > Memory Block Properties** on page 105
- **Debug > Memory Event Details** on page 106

Debug Enable ReplayEngine

Selecting the **Enable ReplayEngine** check box tells TotalView that it should instrument your code so that it saves execution information. This information will later be used when you decide to view the program's execution history

The **Enable ReplayEngine** check box is only visible on Linux-x86 and Linux-86-64 platforms. In you do not have a license for ReplayEngine, enabling the check box has no effect and TotalView displays an error message when your program begins executing.

Debug > Previous

(ReplayEngine only) Moves to the line that was previously executed. The state that existed before that line was executed is displayed. If that line had a function call, this command skips over the call.

Debug > Unstep

(ReplayEngine only) Moves to the line that was previously executed, The state that existed before that line was executed is displayed. If that line had a function call, this command moves back into that function, showing the state that existed before the last line of that routine was called.

Debug > Caller

(ReplayEngine only) Moves back to the routine that called the routine. This is the routine in which the yellow statement line is displayed. You will now see your program's state just before the routine was called.

Debug > Back To

(ReplayEngine only) After selecting a line that was previously executed, ReplayEngine moves back to that line and it will now be highlighted in yellow. It also displays the program's state just before that line was executed.

Debug > Live

(ReplayEngine only) Restores the program so that you can again start executing new statements under TotalView control. As your code executes, ReplayEngine resumes recording information.

Debug > Enable Memory Debugging

Use this command to enable memory debugging. Selecting this command is exactly the same as selecting **Enable memory debugging** within the MemoryScape's Memory Debugging Options page or in the **File > New Program** dialog box.

Debug > Stop on Memory Errors

Use this command to tell the Memory Debugger that it should stop program execution when a memory event or error occurs. Selecting this command is exactly the same as selecting **On memory event, halt execution** within the **Advanced Options** on MemoryScape's Memory Debugging Options page. This is the equivalent of the basic **Low** setting.

Debug > Open MemoryScape

Selecting this command tells TotalView to start MemoryScape.

When MemoryScape starts, it will try to interpret the state of TotalView and will open to the appropriate page, most likely the home page. You can select the Memory Debugging Options page to turn memory debugging on or off. However, if your program is running, you must kill it before the settings will take effect.

Debug > Heap Baseline >Set Heap Baseline (in Process)

Use the **Debug > Set Heap Baseline** command to set a baseline in each thread contained within a group. You can use this command any time an execution process is stopped or halted. After you select this command, the Memory Debugger will begin remembering all heap operations that occur within the thread. At a later time, you can select either the:

- Process Window's **Debug > Heap Baseline >Heap Change Summary** command. The window initially displays information about changes. Two buttons in this window lets you display detailed information about allocations made since you created a baseline or memory that was leaked since this baseline.
- Many of the views within the Memory Debugger have a **Relative to Baseline** check box. Selecting this check box alters the display so that view shows changes made since the baseline.

Before using this command, you must enable memory debugging either by selecting the **Debug > Enable Memory Debugging** command or selecting **Enable memory debugging** in MemoryScape's Memory Debugging Options page.

Debug > Heap Baseline> Set Heap Baseline (in Group)

Use the **Debug> Set Heap Baseline** command to set a baseline in each thread contained within a group. You can use this command any time an execution process is stopped or halted. After you select this command, the

Memory Debugger will begin remembering all heap operations that occur within the thread. At a later time, you can select the:

- **Debug > Heap Baseline >Heap Change Summary** command. The window initially displays information about changes. Two buttons in this window lets you display detailed information about allocations made since you created a baseline or memory that was leaked since this baseline.
- Many of the views within the Memory Debugger have a **Relative to Baseline** check box. Selecting this check box alters the display so that view shows changes made since the baseline.

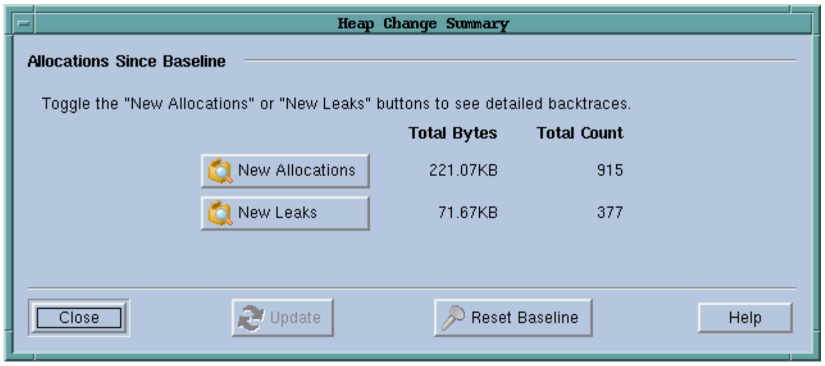
Before using this command, you must enable memory debugging either by selecting the **Debug > Enable Memory Debugging** command or selecting **Enable memory debugging** in MemoryScape's Memory Debugging Options page.

Debug > Heap Baseline >Heap Change Summary

After selecting this command, the Memory Debugger displays its Heap Change Summary window. Before using this command, you must have enabled memory debugging for the process using the **Debug > Enable Memory Debugging** command or selecting **Enable memory debugging** in MemoryScape's Memory Debugging Options page. In addition, you must have already set a baseline using either the **Debug > Heap Baseline >Set Heap Baseline (in Process)** or **Debug > Heap Baseline> Set Heap Baseline (in Group)** command.

Here is the window that the Memory Debugger displays:

Figure 54: Heap Change Summary Window



Heap Change Summary

Allocations Since Baseline

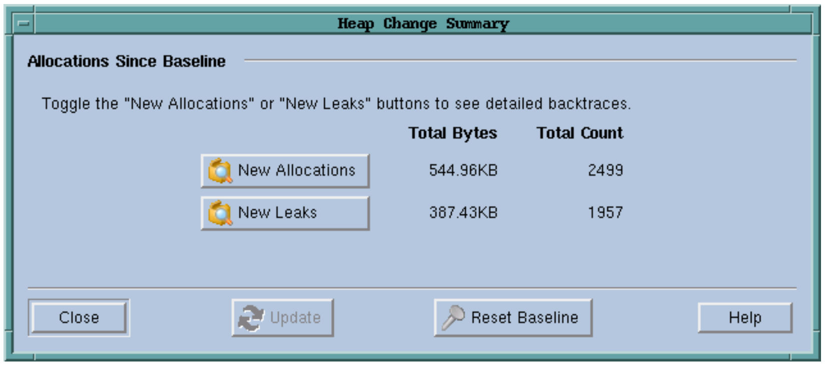
Toggle the "New Allocations" or "New Leaks" buttons to see detailed backtraces.

	Total Bytes	Total Count
New Allocations	221.07KB	915
New Leaks	71.67KB	377

This example shows that 221.07 KB was allocated since a baseline was created and this memory was spread over 915 allocations. Also, 5.51 KB of memory was leaked. This memory was created by 116 allocations.

A baseline does not show all allocations that have occurred. Instead, it only shows the allocations that still remain and any nw leaks that have occurred.

Compare the data here with the data captured at a later time:



Allocations Since Baseline

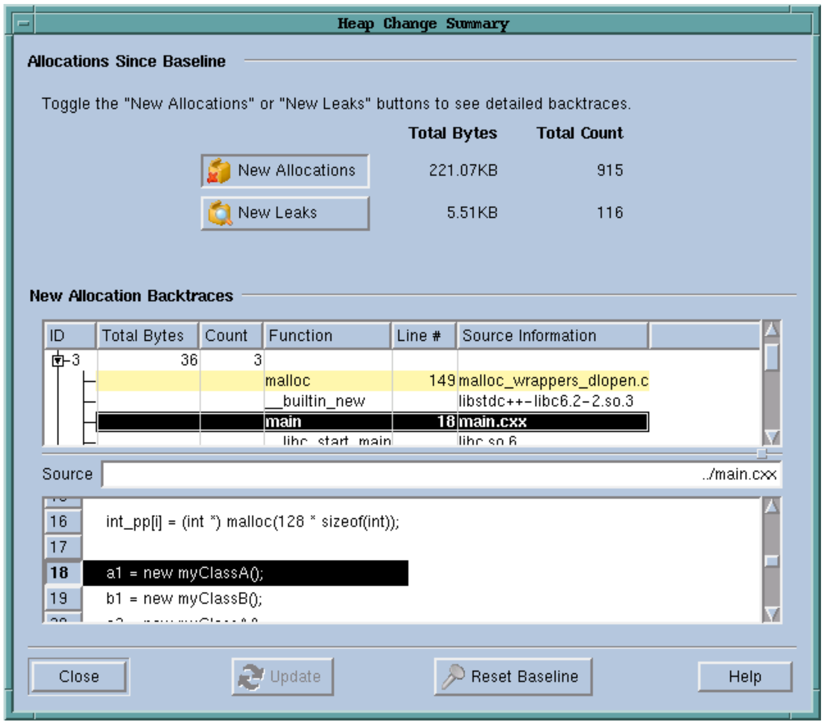
Toggle the "New Allocations" or "New Leaks" buttons to see detailed backtraces.

	Total Bytes	Total Count
New Allocations	544.96KB	2499
New Leaks	387.43KB	1957

Figure 55: Heap Change Summary Window Again

This window now shows that the program has allocated additional memory and, unfortunately, has continued to leak memory. The Memory Debugger can, of course, compare data to explicitly summarize this data.

The following figure shows how this window changes when you select the New Allocations button:



Allocations Since Baseline

Toggle the "New Allocations" or "New Leaks" buttons to see detailed backtraces.

	Total Bytes	Total Count
New Allocations	221.07KB	915
New Leaks	5.51KB	116

New Allocation Backtraces

ID	Total Bytes	Count	Function	Line #	Source Information
3	36	3			
			malloc	149	malloc_wrappers_dlopen.c
			__builtin_new		libstdc++-libc6.2-2.so.3
			main	18	main.cxx
			__libc_start_main		libc.so.6

Source ../main.cxx

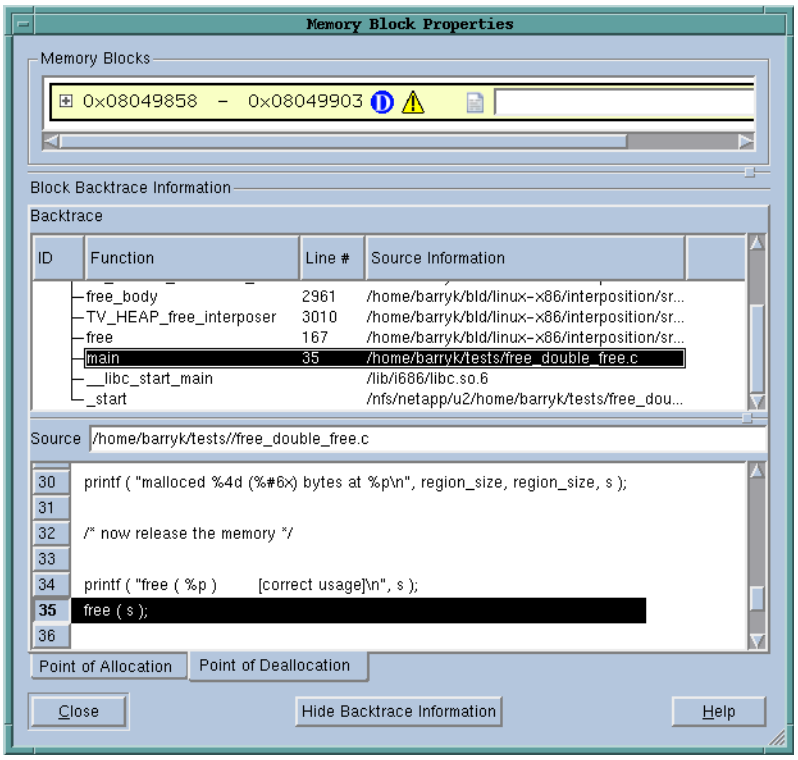
```
16    int_pp[i] = (int *) malloc(128 * sizeof(int));
17
18    a1 = new myClassA();
19    b1 = new myClassB();
```

Figure 56: Heap Change Summary Window, Showing New Allocations

The display is similar if you select the New Leaks button. Each display shows a backtrace that shows where the memory was allocated. After you select a backtrace, the Memory Debugger shows the source code associated with the allocation.

Debug > Memory Block Properties

The **Memory Block Properties** Window displays information about of the blocks that you asked the Memory Debugger to keep track of by using the **Tools > Add to Block Properties** command.



ID	Function	Line #	Source Information
	free_body	2961	/home/barryk/bld/linux-x86/interposition/sr...
	TV_HEAP_free_interposer	3010	/home/barryk/bld/linux-x86/interposition/sr...
	free	167	/home/barryk/bld/linux-x86/interposition/sr...
	main	35	/home/barryk/tests/free_double_free.c
	__libc_start_main		/lib/i686/libc.so.6
	_start		/nfs/netapp/u2/home/barryk/tests/free_dou...

Source /home/barryk/tests//free_double_free.c

```
30  printf ( "malloced %4d (%#6x) bytes at %p\n", region_size, region_size, s );
31
32  /* now release the memory */
33
34  printf ( "free ( %p )       [correct usage]\n", s );
35  free ( s );
36
```

Figure 57: Debug > Memory Block Properties Window

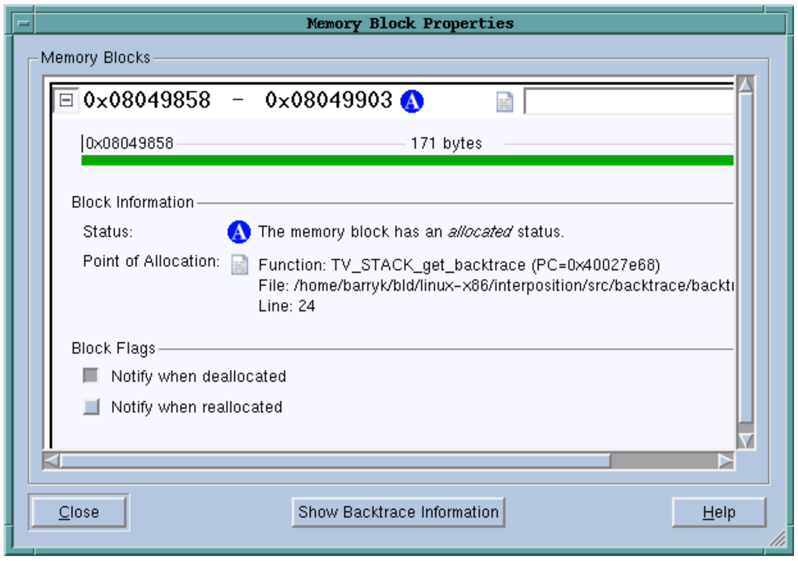
You can display this window in two ways. Pressing the **Hide Backtrace Information** conceals most of what is displayed in Figure 57 on page 105. Figure 58 on page 106 shows this reconfigured window. When this window is being displayed, pressing **Show Backtrace Information** shifts it back.

As an alternative, you can both make the window larger and use the splitter control to enlarge the top area.

The information within the bottom area is essentially the same as that which is displayed when you generate a Backtrace View in the Memory Debugger. You will find information on the contents of the Block Backtrace Information area within "**Heap Status Page**" on page 203

The first line contains a graphical summary of the blocks information. It contains:

- **Memory block extent**: the numbers in bold indicate the starting and ending address of the block. For example, the block in Figure 58 indicates the block starts at 0x08049858 and ends at 0x08049903.
- **Status**: an indicator indicating if the block is allocated, deallocated, leaked, or being hoarded. For example, the Ⓐ in Figure 58 on page 106 indicates that the block is allocated.



0x08049858 – 0x08049903

0x08049858 171 bytes

Status: The memory block has an *allocated* status.

Point of Allocation: Function: TV_STACK_get_backtrace (PC=0x40027e68)
File: /home/barryk/bld/linux-x86/interposition/src/backtrace/backtı
Line: 24

Notify when deallocated

Notify when reallocated

Figure 58: Debug > Memory Block Properties Window

- **Point of Allocation**: if you place your mouse over the icon, TotalView displays a tool tip that shows the function in which the block was allocated, the function's source file, and the line number within that file.
- **Comment**: if you have more than one or two memory blocks, you might want to enter a comment in the text box to remind yourself what the block is.

The next lines within the Memory Blocks area restate the information that was presented in the summary area. The color of the block's graphic is the same as that used in the Heap Status Page's Graphical View.

The Block Flags area contains two check boxes:

- **Notify when deallocated**
- **Notify when reallocated**

Selecting a box tells the Memory Debugger that it should stop execution when the block is deallocated or reallocated and display the Memory Event Details Window. This allows you to track when your program is managing memory and what is being managed. For example, if you have a double free problem, obtaining notification lets you know where the block is first freed.

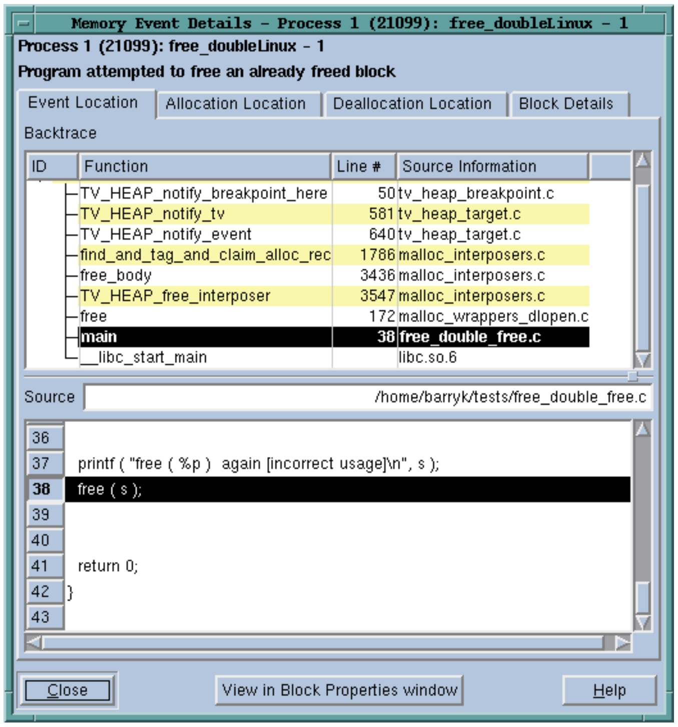
Debug > Memory Event Details

When a memory event occurs, the Memory Debugger automatically displays this window. After you dismiss it, you can redisplay it using this command.

This window has four areas, as follows:

- The top line tells you what type of error or event occurred.
- The Block Information area gives the memory location of the block and its status.
- The third area contains the function backtrace if the error or event is related to a block allocated on the heap. The Memory Debugger retains information about the backtrace that existed when the memory block was

Figure 59: Tools > Memory Event Details Window


Memory Event Details - Process 1 (21099): free_doubleLinux - 1
Event Location
Allocation Location
Deallocation Location
Block Details
Backtrace
Source
Close
View in Block Properties window
Help

Process 1 (21099): free_doubleLinux - 1

Program attempted to free an already freed block

ID	Function	Line #	Source Information
	TV_HEAP_notify_breakpoint_here	50	tv_heap_breakpoint.c
	TV_HEAP_notify_tv	581	tv_heap_target.c
	TV_HEAP_notify_event	640	tv_heap_target.c
	find_and_tag_and_claim_alloc_rec	1786	malloc_interposers.c
	free_body	3436	malloc_interposers.c
	TV_HEAP_free_interposer	3547	malloc_interposers.c
	free	172	malloc_wrappers_dlopen.c
	main	38	free_double_free.c
	__libc_start_main		libc.so.6

Source: /home/barryk/tests/free_double_free.c

```

   printf ( "free ( %p )  again [incorrect usage]\n", s );
   free ( s );


   return 0;
}

```

allocated and the backtrace when it was deallocated. You can tell the Memory Debugger which it should display by selecting either the **Point of Allocation** or **Point of Deallocation** tab.

- The bottom area shows you where the allocation or deallocation occurred in your program.

In some cases, the Memory Debugger does not display an allocation backtrace. For example, if you try to free memory allocated on the stack or in a data section, there's no backtrace because your program did not allocate the memory.

With a memory error, you can have up to three backtraces:

- The allocation backtrace, which indicates where your program allocated a memory block. This only exists for memory created within the heap.
- The deallocation backtrace, which indicates where your program freed a memory block. This only exists for memory created within the heap.
- The current backtrace, which is the place where TotalView stopped your program. This is the backtrace currently being displayed within the Process Window.

If you need to redisplay the this window after you dismiss it, select the **Debug > Memory Event Details** command.

For information on the Block Details tab, see **View > Examine Format > Raw** on page 141.

Tools Menu Commands

The following commands are on the **Tools** pulldown:

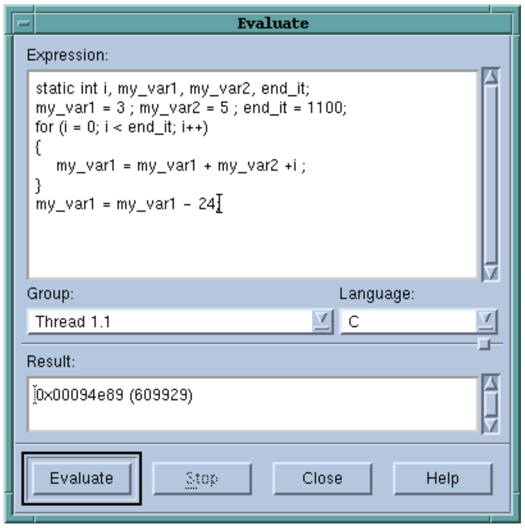
- **Tools > Evaluate** on page 108
- **Tools > Expression List** on page 109
- **Tools > Program Browser Window** on page 169
- **Tools > Fortran Modules Window** on page 161
- **Tools > Call Graph** on page 109
- **Tools > Debugger Loaded Libraries** on page 110
- **Tools > Event Log** on page 111
- **Tools > Warnings** on page 112
- **Tools > Thread Objects Window** on page 191
- **Tools > Message Queue Window** on page 175
- **Tools > Message Queue Graph** on page 113
- **Tools > Create Checkpoint** on page 117
- **Tools > Restart Checkpoint** on page 120
- **Tools > PVM Tasks Window** on page 183
- **Tools > Global Arrays** on page 122
- **Tools > Command Line** on page 122

Tools > Evaluate

Use this window to enter and evaluate small fragments of C++, C, Fortran, or assembler code. This code can contain local variable declarations. In C, the data types you can use are **char**, **short**, **int**, **float**, **double**, and pointers to these data types. In Fortran, the data types you can use are **INTEGER**, **REAL**, **DOUBLE PRECISION**, and **COMPLEX.** Your code fragment can also contain references to the target process's variables. The expression can reference local (stack) and global variables. TotalView evaluates stack variables in the context of the currently selected stack frame. (See Figure 60.)

The remaining controls let you enter the code that TotalView will evaluate.

Expression	Enter the code to be evaluated within this text area.
Group	Limits the debuggers's selection of what executing processes or threads it should chose.
Language	The programming language in which you are writing your code.
Result	The value of the last expression in the code being evaluated.
Evaluate	Tells TotalView to evaluate your code.
Stop	Stops the evaluation of an expression.



```
static int i, my_var1, my_var2, end_it;
my_var1 = 3 ; my_var2 = 5 ; end_it = 1100;
for (i = 0; i < end_it; i++)
{
   my_var1 = my_var1 + my_var2 +i ;
}
my_var1 = my_var1 - 24
```

Group: Thread 1.1

Language: C

Result:

```
0x00094e89 (609929)
```

Figure 60: Tools > Evaluate Window

Statements can affect variables in the target process. That is, if the expression you type alters the value of a variable, it is altered in your program's memory.

If TotalView encounters a breakpoint while executing a a function called from within your expression (or stops for any other reason), the Evaluate Window is *suspended*. You can now debug the called routine as though it had been encountered during normal program operation.

You cannot use the Evaluate Window while it is suspended. To evaluate a second expression while the first one is suspended, open a second Evaluate Window.

For more information, see the "*Evaluating Expressions*" section within the "Expressions" chapter of the *TotalView Users Guide*.

Tools > Expression List

For information, see "**Expression List Window**" on page 205.

Tools > Program Browser

For information, see **Program Browser Window** on page 169.

Tools > Fortran Modules

For information, see **Fortran Modules Window** on page 161.

Tools > Call Graph

Displays a window that will show a graphical representation of your program's stack. This representation, which is called a *call graph*, is a dynamic representation of your program's state.

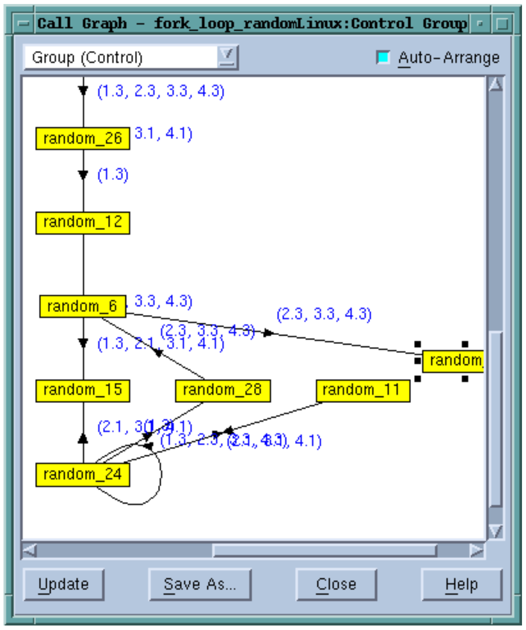


Figure 61: Tools > Call Graph Window

Functions and subroutines are displayed as boxes. The lines linking functions and subroutines to one another indicate that one routine was called by the other, with the arrow pointing to the called routine.

The numbers next to a link indicates the threads or ranks that called the routine. If there a great number of ranks or threads, diving on the arrow tells TotalView to display an information box listing them in a more legible format.

If you dive on the numbered list, TotalView displays a list containing all of the threads running into a routine.

Here's what the **Update** and **Save** As buttons do:

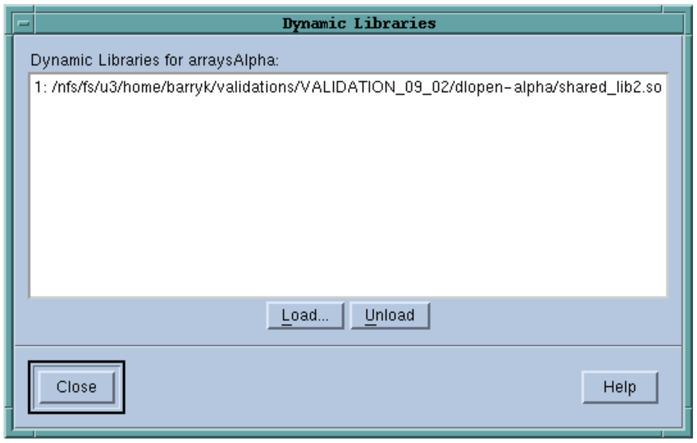
Update	Tells TotalView to update the display. You would do this if you have asked TotalView to display a call graph while your program is executing of if you select a different item within the Group pulldown.
Save As	Tells TotalView to write the call graph to disk using the **.dot** format used by such tools as Graphviz. For more information, see **http://www.graphviz.org/**.

Tools > Debugger Loaded Libraries

Displays a dialog box into that lets you tell TotalView that it should immediately open a shared library using **dlopen()**.

As your program executes, TotalView loads in shared libraries. In some cases, you may want to bring them into memory before they are needed. For example, you may need to refer to a function contained within the library when you are creating an evaluation point or using the **Tools > Evaluate** window.

Figure 62: Tools > Debugger Loaded Libraries Window



Dynamic Libraries for arraysAlpha:

```
1: /nfs/fs/u3/home/barryk/validations/VALIDATION_09_02/dlopen-alpha/shared_lib2.so
```

After a library is brought into memory, you can refer to any symbol contained within the library. After TotalView loads the library, it asks if you want to set breakpoints in the library.

The buttons that are unique to this dialog box are:

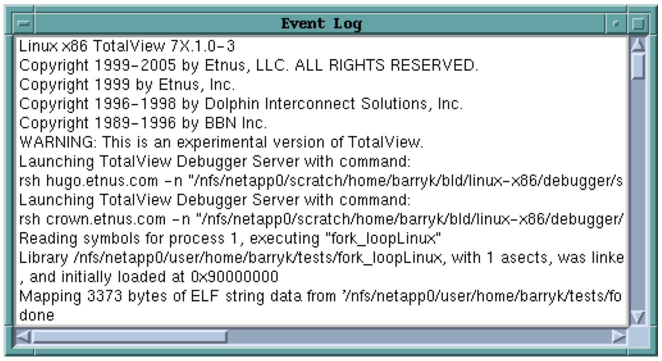
Load	Tells TotalView to display a dialog box that you can use to locate the library within your file system.
Unload	Deletes the selected library.

When you are done, select the **Close** button. As the libraries are already loaded, selecting this button just dismisses the dialog box.

Tools > Event Log

When events occur in the life of a process (for example, an error occurs or the process hits a breakpoint), TotalView places a line of text in the Event Log window indicating what occurred. (See Figure 63 on page 111.)

Figure 63: Tools > Event Log Page



```
Linux x86 TotalView 7X.1.0-3
Copyright 1999-2005 by Etnus, LLC. ALL RIGHTS RESERVED.
Copyright 1999 by Etnus, Inc.
Copyright 1996-1998 by Dolphin Interconnect Solutions, Inc.
Copyright 1989-1996 by BBN Inc.
WARNING: This is an experimental version of TotalView.
Launching TotalView Debugger Server with command:
rsh hugo.etnus.com -n "/nfs/netapp0/scratch/home/barryk/bld/linux-x86/debugger/s
Launching TotalView Debugger Server with command:
rsh crown.etnus.com -n "/nfs/netapp0/scratch/home/barryk/bld/linux-x86/debugger/
Reading symbols for process 1, executing "fork_loopLinux"
Library /nfs/netapp0/user/home/barryk/tests/fork_loopLinux, with 1 asects, was linke
, and initially loaded at 0x90000000
Mapping 3373 bytes of ELF string data from '/nfs/netapp0/user/home/barryk/tests/fo
done
```

The amount of information that TotalView writes into this window depends upon the value set for the VERBOSE variable. The values that can be set are:

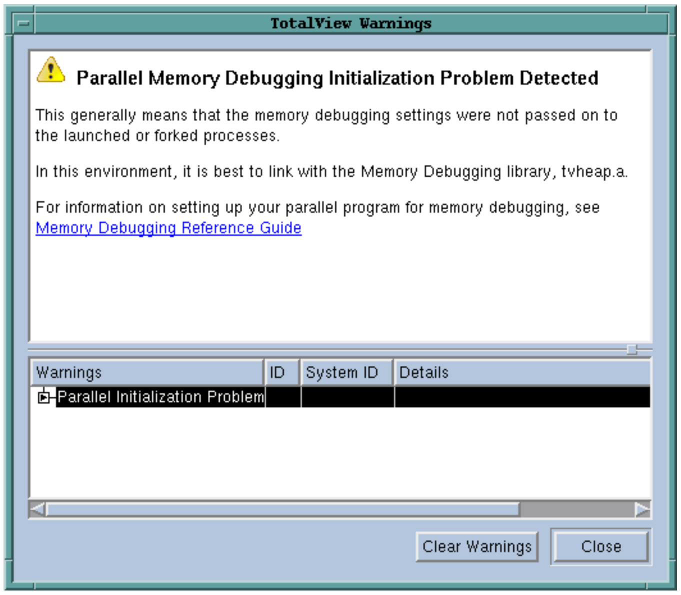
info	Prints errors, warnings, and informational messages. Informational messages include data on dynamic libraries and symbols. This is the default.
warning	Only print errors and warnings.
error	Only print error messages.
silent	Does not print error, warning, and informational messages. This also shuts off the printing of results from CLI commands. This should only be used when the CLI is run in batch mode.

The **Core File** and **Memory File** buttons only appear if you are running TotalView Team or TotalView TeamPlus. Pressing these buttons tells the Memory Debugger that it should either generate a core file or a lightweight memory debugging file. If you press core file, your operating system aborts your program in addition to writing the file. For information on lightweight memory debugging files, see "**Memory Event Actions Dialog Box**" on page 194.

Tools > Warnings

Displays warning message that TotalView displays. (See Figure 64 on page 112.)

Figure 64: Tools > Warnings Dialog Box



Parallel Memory Debugging Initialization Problem Detected

This generally means that the memory debugging settings were not passed on to the launched or forked processes.

In this environment, it is best to link with the Memory Debugging library, tvheap.a.

For information on setting up your parallel program for memory debugging, see Memory Debugging Reference Guide

Warnings	ID	System ID	Details
Parallel Initialization Problem			

If you reinvoke this command, the contents remain. TotalView only changes them when a new warning occurs.

At this time, not all warning message display in this box.

Tools > Thread Objects

For more information, see **Thread Objects Window** on page 191.

Tools > Message Queue

For information, see **Message Queue Window** on page 175.

Tools > Message Queue Graph

Displays a window that shows a graphic representation of the state of message queue information. (The **Tools > Message Queue** command tells TotalView to display this information in a non-graphical form.)

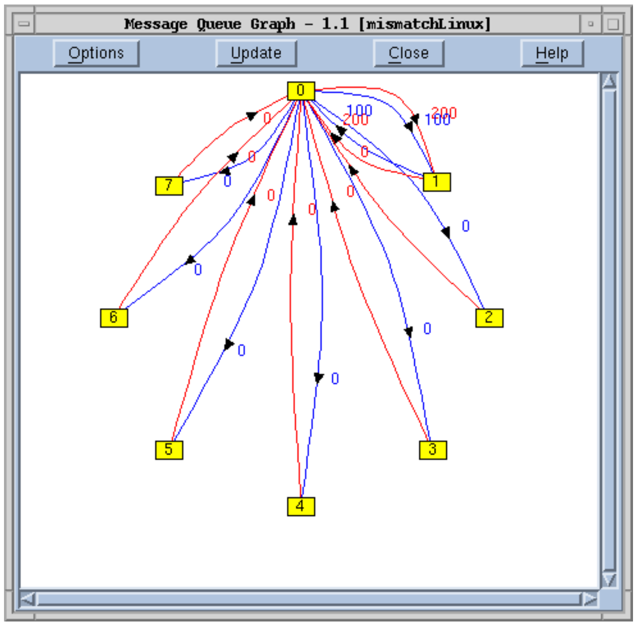


Figure 65: Tools > Message Queue Graph Window

If you right-click within the graph window, TotalView displays a context menu having the following items:

- **Dive**: dives into a process. See **Diving** on page 114 for more information.
- **Attach**: If you've detached from a rank, you can use this command to re-attach to it.
- **Detach**: Removes the selected rank from the display.
- **Subset Attach (talking to this rank)**: displays the same dialog box as is displayed when you select the **Group > Attach Subset** on page 71 dialog box. It differs in that it knows the current context.
- **Save as .DOT**: Tells TotalView to write the graph to disk using the **.dot** format used by such tools as Graphviz. For more information, see **http://www.graphviz.org/**.
- **Save as CSV**: Tells TotalView to write the graph's data to disk using CSV (Comma Separated Value) format.

This visual display of message queue information is often used to spot unexpected messages and to uncover deadlocks. For example, if you ask TotalView to search for and display cycles—this is in the Cycle Detection tab of the Options dialog box—the display shows this problem.

You can also locate other problems. here is a procedure for displaying this information:

1 Select one or more message types to be displayed. By default, TotalView displays **Unexpected** messages. However, you can display any combination of **Send**, **Receive**, and **Unexpected** messages.
2 Select the ranks for which you want information. The ranks are displayed in the pane in the lower left corner of the window. Ranks to which TotalView is attached are displayed as yellow numbered buttons. If a button isn't numbered, TotalView is not attached to the rank.

   Use your mouse to select processes. You can either select processes individually by left-clicking on the rank number or you can use your mouse to draw a rectangle around a group of rank numbers. If you want to remove an already selected rank, click on it while pressing the Control key.
3 Select the group containing the ranks to include in the display.
4 Press the **Update** button.

TotalView responds by displaying the message queue graph. Here, ranks are drawn as yellow boxes. The number within a box is the process's rank.

Wildcards are represented by a made-up process whose name is "ANY". For example, a request such as "receive message with tag 400 from any source" means that TotalView will create a process labeled "ANY".

The messages are indicated as curved lines with an arrow at the end. This arrow indicates the rank receiving the message. The message's tag (or number) is displayed near the line. The line's color represents the kind of message:

- **Red**: unexpected messages
- **Blue**: pending receives
- **Green**: pending sends; these messages seldom occur

TotalView only shows MPI message queue information for processes that are halted or at a breakpoint at the time when you selected the **Update** button. That is, if some of your MPI processes are running when you select **Update**, the message queue display will be inaccurate and incomplete.

Diving: You can dive into a process or a link by double-clicking on it.

- Double-clicking on a process tells TotalView to display a Process Window.
- Double-clicking on a link displays the window that is displayed when you select the **Tools > Message Queue Window** command.

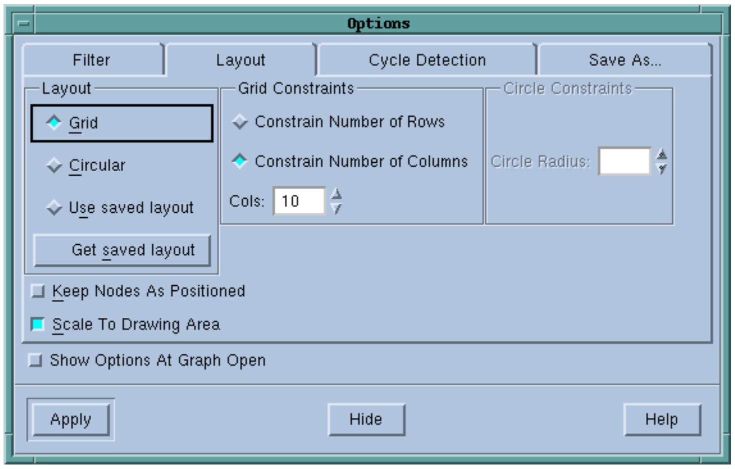
Performance Considerations: The procedure used to extract MPI information from your program can be slow. Consequently, TotalView may take a considerable time responding to your request. (The time TotalView takes is related to the number of processes being polled for information.)

Options Dialog Box

Layout Tab

Commands and controls on the **Layout Tab** control how TotalView displays information. Here is this tab:

Figure 66: Layout Tab



Layout area

TotalView can arrange nodes in a grid (rows and columns) or in a circle. In addition, if you've saved a layout using a button in the **Save As** tab, you can restore that layout and use it.

Grid Constraints area

When displaying nodes in a grid, you can constrain the number of rows or columns to the value indicated in the text box. The **Layout Tab** figure shows consume being selected.

Circle Constraints area

When displaying nodes in a circular pattern, you can constrain the size of the circle.

Other

Keep Nodes As Positioned
: When selected, TotalView will not rearrange nodes that you have moved.

Scale To Drawing Area
: When selected, TotalView arranges the information so that it fits within the displayed window

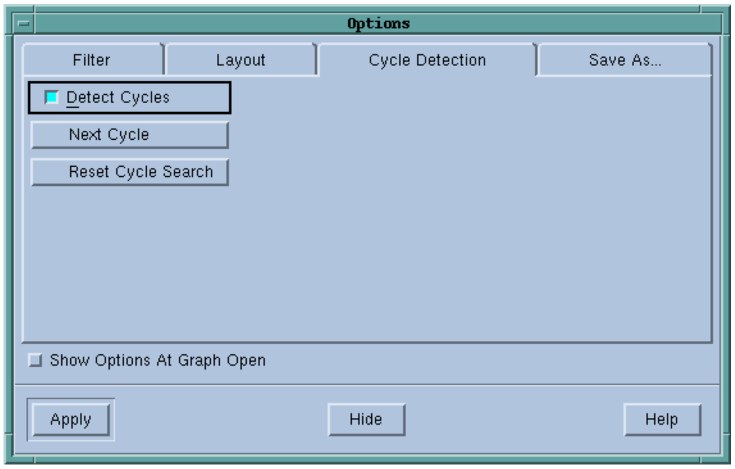
You can manually rearrange the placement of ranks and lines as follows:

- You can drag the yellow process rectangles to a new position by clicking on a yellow box and dragging it to a new position.
- You can drag the curve of the lines linking processes by clicking on the line and dragging the displayed rectangular dragging handles.

Cycle Detection Tab

Commands and controls on the **Cycle Detection** tab control how TotalView displays cycles within the message queue graph. Here is this tab:

Figure 67: Layout Tab



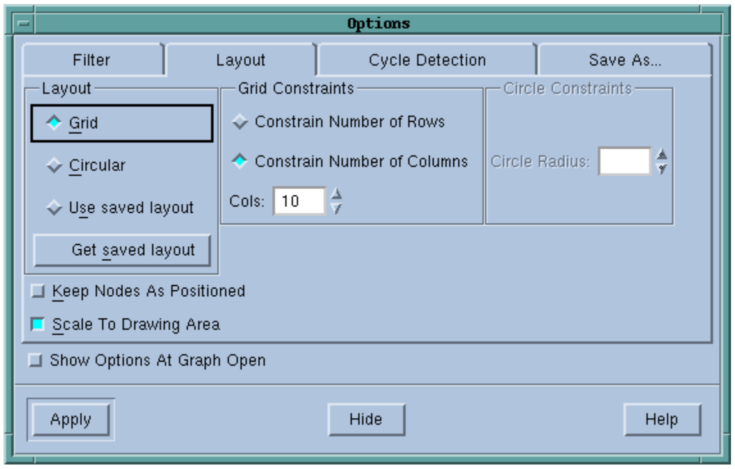
[✓ Detect Cycles]
Show Options At Graph Open

Detect Cycles	When selected, TotalView looks for cycles after it redisplays a message queue graph.
Next Cycle	If your display as more than one cycle, pressing this button moves you from one cycle to another.
Reset Cycle Search	If you've cycled through all of the cycles, you can reset the search using this button.

Filter Tab

Commands and controls on the **Filter** Tab determine which nodes TotalView displays. Here is this tab:

Figure 68: Layout Tab



Layout: [✓ Grid] Circular, Use saved layout
Grid Constraints: Constrain Number of Rows [✓ Constrain Number of Columns]
Cols: 10
Keep Nodes As Positioned
[✓ Scale To Drawing Area]
Show Options At Graph Open

The colored buttons have the following meanings:

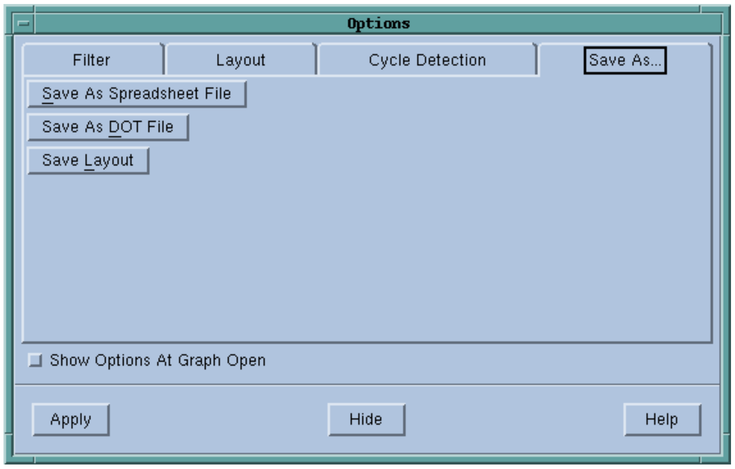
- **Green**: Pending Sends; these messages seldom occur
- **Blue**: Pending Receives
- **Red**: Unexpected Messages

Other controls in the tab let you:

- Select which ranks to display.
- Select the ranks contained within a named or TotalView default group.
- Only display ranks containing a tab that you specify.

Save As Tab Commands on the **Save As** Tab tell TotalView to save message queue information. Here is this tab:

Figure 69: Save As Tab



Save as Spreadsheet File	Pressing this button tells TotalView to save message queue graph information as a CSV file. This file can later be imported into a spreadsheet.
Save As DOT File	Pressing this button saves the graphic information as a DOT file. Third party graphic systems such as Graphviz
Save Layout	Pressing this button tells TotalView to save the layout in a proprietary format. You can restore this layout using the **Get Saved Layout** button on the **Layout** tab.

Tools > Create Checkpoint

This command is only available on SGI IRIX and IBM RS/6000.

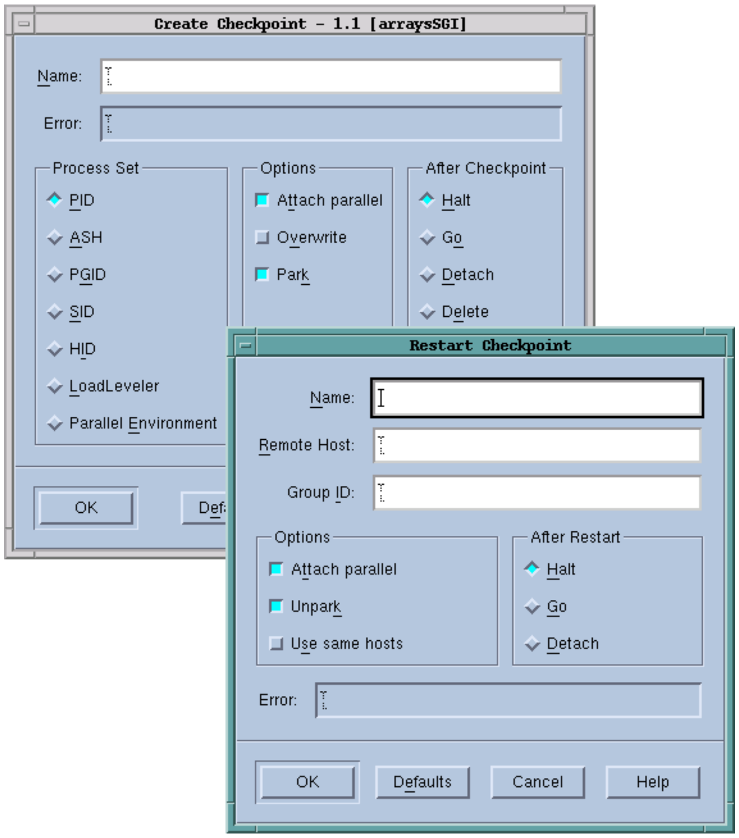
Saves a program and its process's information into the **Name** file.

You can then use the **Tools > Restart Checkpoint** command to restart the program at a later time. This saved information includes process and group IDs. While breakpoint information is not saved as part of the checkpoint, you can use the **Action Point > Save All** command to save the current breakpoints for later use.

You can checkpoint programs running on:

- RS/6000
- SGI IRIX

Figure 70: Checkpoint and Restart Dialog Boxes



Processes running remotely that communicate by using sockets can have difficulties when being checkpointed because IRIX will not checkpoint programs with open sockets. As the TotalView Server (**tvdsvr**) uses sockets to redirect **stdin**, **stdout**, and **stderr**, you will need to use the **Standard I/O Page** of the **Process > Startup Parameters** dialog box to modify the way your processes send information to a **tty** before creating a checkpoint.

If you are using SGI MPI, *you need to use the* **-cpr** *command-line option, which, if conditions are correct, you'll be able to create a checkpoint.* Use the ASH *option with* MPI *checkpoints*

This command is the same as the CLI **dcheckpoint** command.

RS/6000 If you are creating a checkpoint on an RS/6000 machine, press one of the following radio buttons to define the state of the process both before and after the checkpoint:

Halt	Processes stop executing after they are checkpointed. This is the default option.
Delete	Processes exit after being checkpointed.
Error	If a problem occurs, TotalView displays the reason in this area.

The name used for the checkpoint file is set using an environment variable. See your POE documentation for more information.

SGI IRIX The controls in this window are as follows:

Name	The name being assigned to the checkpoint. If this name already exists, TotalView overwrites it.
Error	If a problem occurs, TotalView displays the reason in this area.
Process Set	Indicates the set of processes that will be checkpointed. Your options are:
PID	Checkpoint the program indicated by a PID.
ASH	Checkpoint the array session.
PGID	Checkpoint the entire process group.
SID	Checkpoint the entire process session.
HID	Checkpoint the hierarchy rooted in the focus process.
Options	Indicates control options that you may find useful. Choose as many as needed to define your checkpoint.
Attach parallel	Tells TotalView that it should reattach to processes from which the checkpointing processes detached. (Some systems automatically detach you from processes being checkpointed.)
Overwrite	Lets TotalView assign new IDs when it restarts a checkpoint. If you do not use this option, the same IDs are used.
Park	Tells TotalView that it should hold all processes before it begins checkpointing them. If you will be restarting a checkpoint from the shell, you should make sure that this option is *not* selected. In addition, if this option is not selected and you will be restoring the checkpoint from within TotalView, you will need to select the **Tools > Restart Checkpoint Unpark** check box.
After Checkpoint	Defines the state of the process both before and after the checkpoint. Use one of the following options:
Halt	Processes stop executing after they are checkpointed. This is the default option.
Go	Processes continue running after being checkpointed.
Detach	Processes continue running after being checkpointed. In addition, TotalView detaches from them.
Delete	Processes exit after being checkpointed.
Defaults	Restores selections in this dialog box to their default values. These are as follows: **PID**, **Attach parallel**, and **Halt**.

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

The **After Checkpoint** options let you specify what happens after the checkpoint operation concludes. The default operation is that TotalView tells the checkpointed processes that they should **Halt**, which allows you to investigate a program's state at the checkpointed position. In contrast, **Go** tells TotalView that it should let the processes continue to run. The **Detach** and **Delete** options are less frequently used. **Detach** allows you to shut down TotalView and leave the processes running. The **Delete** option differs from **Detach** in that processes started by TotalView are also terminated.

Just before TotalView begins checkpointing your program, it temporarily stops (that is, *parks*) the processes being checkpointed. Parking ensures that the processes do not run freely after a **Tools > Create Checkpoint** or **Tools > Restart Checkpoint** 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 clear the **Park** check box.

TotalView detaches from processes before they are checkpointed. By default, TotalView automatically reattaches to them. If you want something different to occur, you can tell TotalView that it should never reattach by deselecting the **Attach parallel** check box.

The CLI's **dcheckpoint** command performs the same operations as this command.

Tools > Restart Checkpoint

This command is only available on SGI IRIX and IBM RS/6000.

Use this dialog box to restore and restart all of the checkpointed processes. By default, TotalView attaches to the base process. If parallel processes are related to this base process, TotalView attaches to them. If you do not want TotalView to automatically attach to them, deselect the **Attach parallel** option.

If an error occurs while attempting to restart the program from the checkpoint, information is displayed in the **Error** area.

The CLI's **drestart** command performs the same operations as this command.

Name	Names a previously saved checkpoint file.
Remote Host	Names the remote host upon which the restart will occur.
Group ID	Names the control group into which TotalView places all created processes
Options	Indicates control options that you may find useful. If this is an RS/6000 checkpoint, **Attach parallel** is automatically checked and it cannot be unchecked. These options have the following meaning:

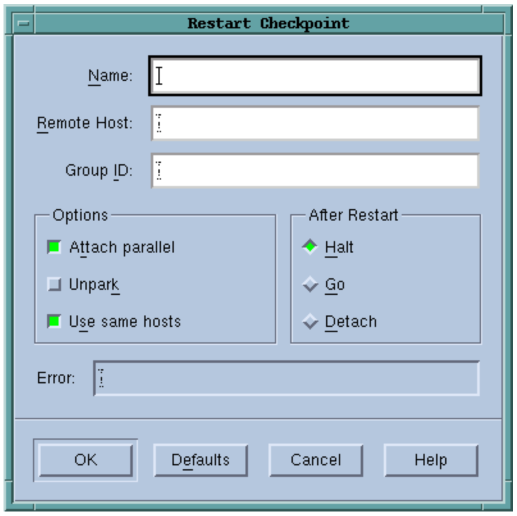


Figure 71: Tools > Restart Checkpoint

Attach parallel
If selected, TotalView attaches to parallel processes as they are being created. If this item is not selected, TotalView only attaches to the base process.

Unpark
(SGI only) Select this checkbox if the checkpoint was created outside of TotalView or if you did not select the **Park** checkbox within the **Tools** > **Restart Checkpoint** dialog box when you created the checkpoint file.

Use Same Hosts
(IBM only) If selected, the restart operation tries to use the same hosts as were used when the checkpoint was created. If TotalView cannot use the same hosts, the checkpoint operation fails.

After Restart Defines the state of the process both before and after the checkpoint. You can use one of the following options:

Halt Parallel processes are held immediately after the place where the checkpoint occurred. TotalView attaches to these created parallel processes. (This is the default.)

Go (SGI only) Checkpointed parallel processes are started and TotalView attaches to the created processes.

Detach (SGI only) Checkpointed process are started. TotalView does not attempt to attach to them.

Restarting on AIX using LoadLeveler: On the RS/6000, if you wish to debug a **LoadLever poe** job from the point at which the checkpoint was made, you must resubmit the program as a **LoadLeveler** job to restart the checkpoint. You will also need to set the **MP_POE_RESTART_SLEEP** environment variable to an appropriate number of seconds. After you restart **poe**, start TotalView and attach to **poe**.

When attaching to **poe***, parallel tasks will not yet be created, so do not try to attach to any of them. Also, you'll need to set the* **Attach to none** *option with the* **Parallel** *Page of the* **File > Preferences** *Dialog Box.*

When doing this, you cannot use the restart the checkpoint using this command. **poe** will tell TotalView when it is time to attach to the parallel task so that it can complete the restart.

Tools > PVM Tasks

For more information, see **PVM Tasks Window** on page 183.

Tools > Global Arrays

Opens a window containing a list of your program's global arrays. For each global array, TotalView displays four entries:

Handle	A value assigned to the array by the Global Arrays software.
Ghosts	Indicates if Global Arrays will write ghost cells from one process to another.
C type	Defines how the array is defined in the C programming language
Fortran Type	Defines how the array is defined in the Fortran programming language.

You can see an array's data by diving on either the Fortran or C type line. After you dive, TotalView displays a standard variable window containing this information. Because it is a standard window, you can manipulate its contents using standard TotalView commands. For example, you can filter and slice the array, obtain statistics, visualize its data, and so on.

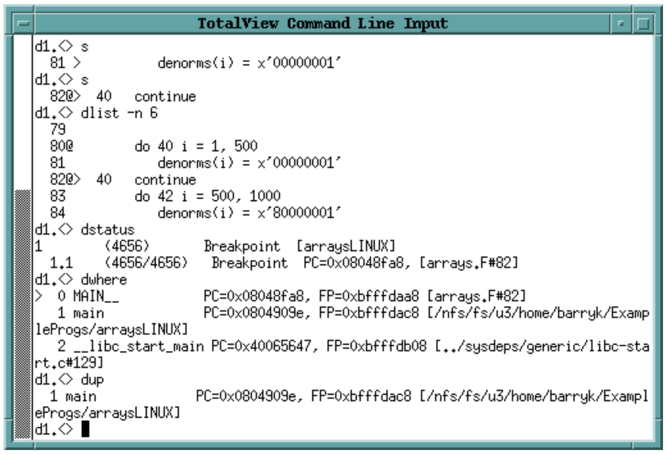
If you want to change the way TotalView displays the data from its C definition into its Fortran definition or vice versa, you can cast it, as follows:

$ga	Casts the array so it is showing data as it would be displayed within a C program.
$GA	Casts the array so it is showing data as it would be displayed within a Fortran program.
$Ga	Casts the language within the current context.

Tools > Command Line

Opens the CLI window. This window is an **xterm** window into which you enter CLI commands.

You can find information on the CLI in the *TotalView Users Guide* or by selecting **CLI** from the Help command on this window.



```
d1.<> s
  81 >          denorms(i) = x'00000001'
d1.<> s
  82@>  40   continue
d1.<> dlist -n 6
  79
  80@        do 40 i = 1, 500
  81            denorms(i) = x'00000001'
  82@>  40   continue
  83         do 42 i = 500, 1000
  84            denorms(i) = x'80000001'
d1.<> dstatus
1        (4656)       Breakpoint  [arraysLINUX]
  1.1    (4656/4656)   Breakpoint  PC=0x08048fa8, [arrays.F#82]
d1.<> dwhere
>  0 MAIN__            PC=0x08048fa8, FP=0xbfffdaa8 [arrays.F#82]
   1 main              PC=0x0804909e, FP=0xbfffdac8 [/nfs/fs/u3/home/barryk/Examp
leProgs/arraysLINUX]
   2 __libc_start_main PC=0x40065647, FP=0xbfffdb08 [../sysdeps/generic/libc-sta
rt.c#129]
d1.<> dup
  1 main              PC=0x0804909e, FP=0xbfffdac8 [/nfs/fs/u3/home/barryk/Exampl
eProgs/arraysLINUX]
d1.<> █
```

Figure 72: Tools > Command Line (CLI) Window

Window Menu Commands

The following commands are on the Window pulldown:

- Window > Update on page 123
- Window > Update All on page 123
- Window > Duplicate on page 123
- Window > Memorize on page 124
- Window > Memorize on page 124
- Window > Root on page 124

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Duplicate

Tells TotalView that it should create a second copy of the current Process Window.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.

TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference***'s* Options Page*.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.

TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Displays (uncovers) the Root Window.

chapter 3

Variable Window

Variable Window Overview

The Variable Window displays information about one of your program's objects.

This window is more than just a viewer. It lets you change a variable or element's value, as well as allowing you a variety of ways to alter what TotalView is displaying. For example, you can cast a variable to the way you want it to appear.

The type mapping procedure lets you control how TotalView displays information. For more information, see the TotalView Users Guide. The Creating Type Transformation chapter in the Reference Guide contains information on how you can create your own transformations.

Other examples of changes you can make are:

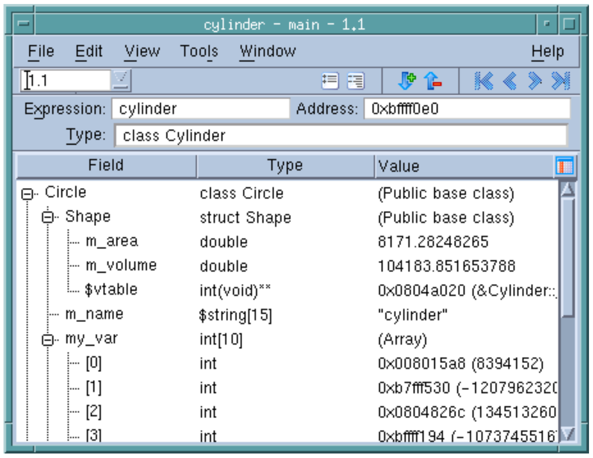
- If TotalView is displaying an array, you can display a limited section of the array, called a *slice*, instead of the whole array.
- You can also enter a *filter* that tells TotalView to make decisions about which array elements it should display.

Topics in this section are:

- *"Expression Field—Altering What is Being Displayed"* on page 126
- *"Address Field—Changing a Variable's Address"* on page 126
- *"Status Field—Seeing State and Error Messages"* on page 128
- *"Type Field—Changing a Variable or Element's Data Type"* on page 127
- *"Slice Field—Altering Subscripts"* on page 128
- *"FIlter Field—Filtering Array Values"* on page 128

Other operations you can perform are described in the following sections.

Figure 73: Variable Window



Expression: cylinder Address: 0xbffff0e0
Type: class Cylinder

Field	Type	Value
Circle	class Circle	(Public base class)
Shape	struct Shape	(Public base class)
m_area	double	8171.28248265
m_volume	double	104183.851653788
$vtable	int(void)**	0x0804a020 (&Cylinder::.
m_name	$string[15]	"cylinder"
my_var	int[10]	(Array)
[0]	int	0x008015a8 (8394152)
[1]	int	0xb7fff530 (-120796232
[2]	int	0x0804826c (134513260
[3]	int	0xbffff194 (-107374551

- "*Diving into Elements*" on page 130
- "*Sorting Columns*" on page 130
- "*Changing the Process and Thread*" on page 130

When you tell TotalView to open a Variable Window, it tries to find a window that is already displaying the element. If it finds one, TotalView moves this window to the front of the display.

If the variable or element being displayed contains substructures, only the name of the substructure is immediately visible. To display a substructure, click on the "+" icon. Clicking on the "–" icon closes the substructure, hiding the information. You can use the **View > Expand all** and **View > Collapse All** commands to expand and collapse all trees.

Expression Field—Altering What is Being Displayed: When TotalView first displays a Variable Window, the value of the **Expression** field contains a variable, element, or expression. The exact contents depend upon what you dove on or what was entered in a **View > Lookup Variable** command.

By changing this field's text, you can change what is displayed. For example, if the expression contained the variable **my_var**, you could change this to **my_var.struct1.substruct1[3]**.

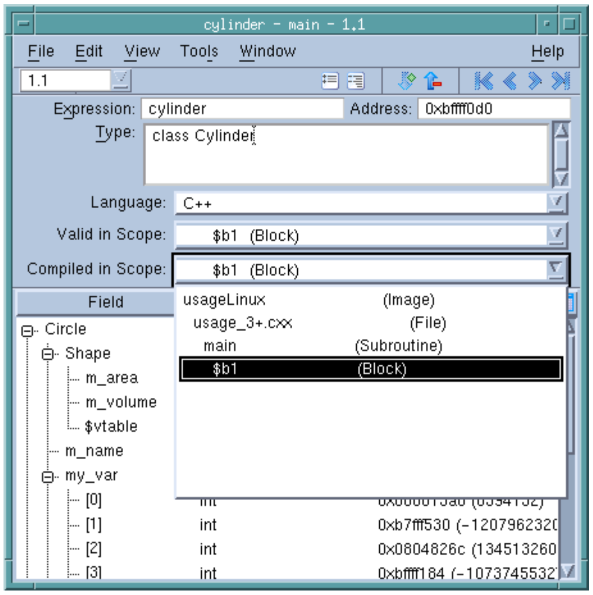
As the name of this field implies, you can type an expression in this field. For example, you could type either **i+3** or **my_var[i+3]**. You cannot, however, enter an expression that contains a function call or an expression that has a side-effect. For example, you could not enter **my_var[i++]**.

Address Field—Changing a Variable's Address: To view a different part of memory, edit the **Address** field in the upper left corner of the window. If TotalView needs to display information about an address, it displays this information in the unlabeled field to the right of the **Address** field. Note, however, that you cannot change the address of register variables.

Type Field—Changing a Variable or Element's Data Type: To change the data type that TotalView uses to format the variable, edit the text within the **Type** field.

While you cannot directly edit the type entry for a field in a substructure, you can dive into the substructure and then edit it.

More and Less Buttons: TotalView usually displays information about your variable in addition to the array's value. This information is sometimes called *meta-information*. You can control how much of this meta-information it displays by clicking on the **More** and **Less** buttons. For example:



```
cylinder - main - 1.1
File  Edit  View  Tools  Window                    Help
1.1
Expression: cylinder          Address: 0xbffff0d0
      Type: class Cylinder
  Language: C++
Valid in Scope: $b1  (Block)
Compiled in Scope: $b1  (Block)
    usageLinux              (Image)
      usage_3+.cxx              (File)
        main             (Subroutine)
        $b1              (Block)
Field
⊟ Circle
  ⊟ Shape
      m_area
      m_volume
      $vtable
    m_name
  ⊟ my_var
      [0]
      [1]      int      0xb7fff530 (-1207962320
      [2]      int      0x0804826c (134513260
      [3]      int      0xbffff184 (-1073745532
```

Figure 74: More and Less Buttons

As the button names indicate, clicking **More** displays more meta-information and clicking **Less** displays less of it. The following list describes the purpose of these fields.

Type	Contains the variable's data type. This field contains the same information that appears when TotalView is not showing you more information. However, if the name is very long, you'll be able to see its entire name here.
Actual Type	If the Variable Window is showing a Fortran assumed shape array, the **Type** field contains the current array definition. The **Actual Type** field shows how the array was declared.
Language	The programming language used for the currently scoped expression or variable.
Valid in Scope	Indicates the scope in which the variable resides. That is, the variable is "valid" anytime the PC is within this scope. If you display the pulldown list, you will see the complete scope specification.

Compiled in Scope

Indicates the scope in which the expression was compiled. If you alter the text within the **Expression** field of the Variable Window, this field lets you know the scope where the expression is defined. If you display the pull-down list, you will see the complete scope specification.

Status Field—Seeing State and Error Messages: If a problem occurs when you make a change to a field or if TotalView needs to tell you something about this window's contents, it will display information here.

For example, if a variable is no longer in scope—that is, your program is no longer executing in a scope in which the variable is defined—TotalView tells you that the variable is stale.

Slice Field—Altering Subscripts: To change an array's subscripts, edit the data that appears after the **Slice:** field at the top of the window. You can enter each array dimension by using one of the following formats, depending upon your programming language:

- `(slice descriptor,slice descriptor,...)`
- `[slice descriptor][slice descriptor]...`

The slice descriptor tells TotalView that it should display every *stride*th subscript from the lower bound to the upper bound, inclusive. If the stride is negative, TotalView shows every –*stride*th subscript from the upper bound to the lower bound, inclusive, in reverse order.

Each **Slice** field has a lower bound, an upper bound, and a stride. These elements are separated by a colon (:).

If you omit the stride value and its colon separator, the default stride value is 1. If you omit the upper bounds, the lower bounds value will be the upper bounds value. The lower bound must be less than or equal to the upper bound, and the stride cannot be 0.

FIlter Field—Filtering Array Values: You can tell TotalView that it should selectively display information from the array by entering a value in the **Filter** field.

While there are a number of ways to specify a filter, the general format is:

```
operator value
```

where *operator* is one or more of the following:

```
<, <=, >, >=, ==, !=
.lt. .le. .gt. .ge. .eq. .ne.
```

value can be a constant integer or real value. For example:

```
> 0
```

You can also enter a TotalView intrinsic, which is a built-in representation of IEEE floating-point NaN (Not a Number), INF (infinity), or denormalized value. These intrinsics are:

Intrinsic	Meaning
$nan	any NaN (Not a Number)
$nanq	quiet NaN (Not a Number)
$nans	signaling NaN (Not a Number)
$inf	any INF, either positive or negative
$ninf	negative INF
$pinf	positive INF
$denorm	denormalized number, either positive or negative
$pdenorm	positive denormalized number
$ndenorm	negative denormalized number

You can only use the == and != operators with these intrinsics. For example, != $denorm.

You can add a second component to the filter to indicate that TotalView should show elements contained within a range by using the following format:

[>]*low_value*:[<]*high_value*

Here are some points to consider:

- *low_value* and *high_value* are constant integers or real values.
- You can apply the < and > operators to the low and high values to allow for exclusive ranges. By default, the range is inclusive of the lower and upper values.
- *low_value* and *high_value* cannot be of different types. For example, the following range is invalid:

  ```
  1:2000u
  1.0:2000
  ```
- You can use the `$value` token to represent the current array element. For example:

  ```
  $value > 0 && $value < 100
  ```

  This filter expression tells TotalView to displays all array elements that are greater than 0 and less than 100.
- Your filter can also contain program variables. For example:

  ```
  $value != x && $value < y
  ```
- You cannot use function calls in a filter expression.

For more information, see Setting Action Points in the *TotalView Users Guide*.

Changing the Value of a Variable: To change the value of a simple variable or change the value of a field in a more complex variable (a structure or array), click the cursor within the variable or element's value, then enter the changed value.

Diving into Elements: To view the target of a pointer or to view one element in an array or structure, double-click on the entry. TotalView replaces the contents of the window with a view of the item you dived into (or the object it points to, in the case of pointers). This allows you to easily chase chains of linked structures.

After you dive into a field, you can edit its contents. (Diving into a substructure is the only way to edit the substructure's value.) You can also cast individual entries in arrays to different data types for display.

Sorting Columns: If you are viewing an array, you can sort the array's value by clicking on the Value heading.

- Clicking once on **Value** sorts the array into ascending order.
- Clicking again sorts the array into descending order.
- Clicking a third time returns the order to what it was before you clicked on the **Value** heading.

The hierarchy button ( ) within the heading allows you to change the Variable Window's display from a flat view to a hierarchical view. When displayed hierarchically, you can use the + and – minus buttons to expand and contract similar groups of information in the display. As you click on headings, you are telling TotalView to aggregate the information in a different way. This means you can create different information groupings.

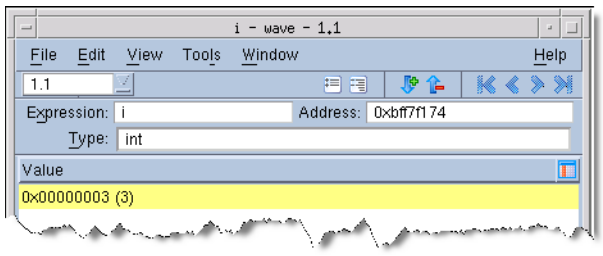
When you are displaying in formation in its flat view, aggregation does not occur when you sort information by clicking on a column heading.

Changing the Process and Thread: The process/thread box initially indicates the variable or element's context when you created the Variable Window. You can change this context be either typing in a new process/thread number (be sure to separate the process number from the thread number with a period) or selecting a value from the pulldown list.

Seeing Variable Value Changes: TotalView can tell you when a variable's value changes in several ways.

- When your program stops at a breakpoint, TotalView adds a yellow highlight to the variable's value if it has changed. This is shown in Figure 75 on page 130.

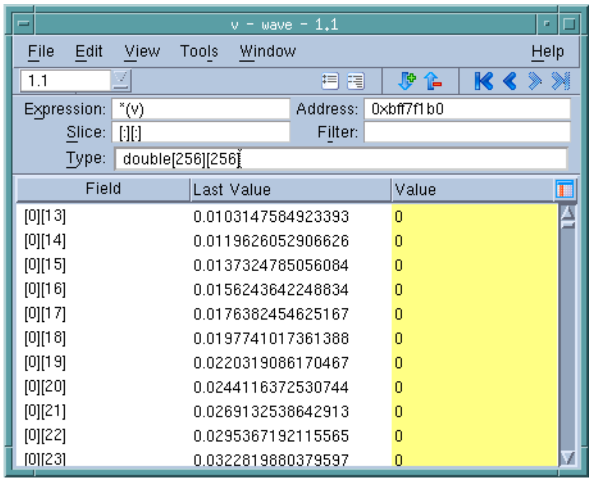
Figure 75: Variable Window With "Change" Highlighting



If the thread is stopped for another reason—for example, you've stepped the thread—and the value has changed, TotalView does not add yellow highlighting to the line.

- You can tell TotalView to display the **Last Value** column. Do this by selecting **Last Value** in the column menu, which is displayed after you click on the column menu ( ) icon.

Figure 76: Variable Window Showing Last Value Column



Expression: *(v) Address: 0xbff7f1b0
Slice: [:][:] Filter:
Type: double[256][256]

Field	Last Value	Value
[0][13]	0.0103147584923393	0
[0][14]	0.0119626052906626	0
[0][15]	0.0137324785056084	0
[0][16]	0.0156243642248834	0
[0][17]	0.0176382454625167	0
[0][18]	0.0197741017361388	0
[0][19]	0.0220319086170467	0
[0][20]	0.0244116372530744	0
[0][21]	0.0269132538642913	0
[0][22]	0.0295367192115565	0
[0][23]	0.0322819880379597	0

Notice that TotalView has highlighted all items that have changed within an array. In a similar fashion it can show the individual items that have changed within a structure.

When you scroll the Variable Window, TotalView discards the information it is tracking and fetches new information. So, while the values may have changed, TotalView does not have information about this change. That is, TotalView only tracks what it is visible. Similarly, when you scroll back to previously displayed values, TotalView needs to refetch this information. Because it is "new" information, no "last values" exist.

The Expression List window, also highlights data and can display a **Last Value** column.

File Menu Commands

The following commands are on the **File** pulldown:

- **File > Save Pane** on page 132
- **File > Close Similar** on page 132
- **File > Close** on page 132
- **File > Exit** on page 132

File > Save Pane

Use this dialog box to write the contents of the selected page, pane, or window.

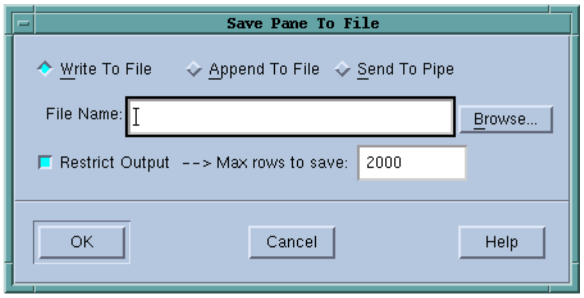


Figure 77: File > Save Pane Dialog Box

Write to File Tells TotalView to write information to a file. You can either enter the name of the file in the **File Name** field or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView overwrites it. If the file does not exist, TotalView creates the file before writing this information.

Append To File Tells TotalView to add information to a file. You can either enter the name of the file in the **File Name** edit box or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView adds this information to the end of the file. If the file does not exist, TotalView creates the file before writing this information.

Send To Pipe Sends the data to the program or script named in the **File Name** field.

Restrict Output --> Max rows to save
If checked, TotalView limits how much information it should send. If the default value of 2000 rows is not what you want, you can specify how many rows TotalView should write.

File > Close Similar

Closes all Variable Windows and windows derived from this window.

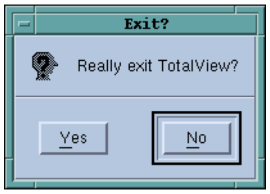
File > Close

Tells TotalView to close the current Variable Window.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.)

Figure 78: File > Exit Command



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The following commands are on the **Edit** pulldown:

- **Edit > Undo** on page 133
- **Edit > Reset Defaults** on page 133
- **Edit > Cut** on page 134
- **Edit > Copy** on page 134
- **Edit > Paste** on page 134
- **Edit > Delete** on page 134
- **Edit > Find** on page 134
- on page 135

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After you make the editing change, you cannot undo your edit.

Edit > Reset Defaults

Selecting this command tells TotalView to undo the changes you've made to the Variable Window that don't affect information stored in memory. For example, this command changes you made to the **Type**, **Expression**, **Slice**, and **Filter** fields. Changes you make to a variable or element's value are not reset.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

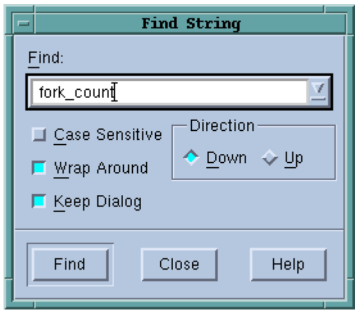
Edit > Find

Use this command to search for text within a page or a pane.

The controls in this dialog box are:

Find	Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered.
Case Sensitive	If selected, TotalView only locates text having the same capitalization as the text entered in the **Find** field.
Wrap On Search	If selected, TotalView will continue the search from either the beginning (if **Down** is also selected) or the end

Figure 79: Edit > Find Dialog Box



	(if **Up** is also selected.) For example, you search for "foo" and the **Down** button is selected. If it isn't found 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 had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the **Find** button. If you select this option, you will need to select the **Close** button to dismiss this dialog box.
Direction	Sets the direction in which TotalView searches. **Up** means "search from the current position to the beginning of the file." **Down** means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the **Find** box.
Close	Closes the **Find** dialog box.

After you find a string, you can locate another instance of the string by using the **Edit** > **Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The following commands are on the **View** pulldown:

- **View** > **Dive** on page 136
- **View** > **Dive in New Window** on page 136
- **View** > **Dive In All** on page 136

- **View > Expand All** on page 137
- **View > Collapse All** on page 137
- **View > Undive** on page 137
- **View > Undive All** on page 137
- **View > Redive** on page 137
- **View > Redive All** on page 138
- **View > Freeze** on page 138
- **View > Lock Address** on page 138
- **View > Show Across > None** on page 139
- **View > Show Across > Process** on page 139
- **View > Show Across > Thread** on page 139
- **View > Compilation Scope > Fixed** on page 139
- **View > Compilation Scope > Floating** on page 139
- **View > Loader Symbols** on page 140
- **View > Padding** on page 140
- **View > Break At Newlines** on page 140
- **View > Examine Format > None** on page 140
- **View > Examine Format > Structured** on page 140
- **View > Examine Format > Raw** on page 141
- **View > Block Status** on page 142

View > Dive

Replaces the contents of your Variable Window with information about the selected data element. TotalView lets you dive on array elements, pointers, structures, and substructure elements.

You can use the **View > Undive** command or the left-facing triangle in the window's upper right-hand corner to restore the window to what it was before you dived into an element.

View > Dive in New Window

Opens a new variable containing information about the selected element. TotalView lets you dive on array elements, pointers, structure, and sub-structure elements.

This command differs from a dive command in that TotalView displays information about the selected element in a new window.

View > Dive In All

The **View > Dive In All** command (which is also available when you right click on a field) allows you to display an element within an array of structures as if it were a simple array. After you select this command, TotalView replace the information in the current Variable Window with new information.

For example, suppose you have the following Fortran definition:

```
type embedded_array
   real r
   integer, pointer :: ia(:)
```

```
  end type embedded_array

  type(embedded_array) ea (3)
```

After you select the **View > Dive In All** command, TotalView displays all three **r** elements of the **ea** array as if it were a single array.

The **View > Dive in All** command can also display the elements of C array of structures as arrays.

As array manipulation commands work on what's displayed and not what is stored in memory, you can operate on an array created by this command in the same manner as any other array. For example, you can visualize the array, obtain statistics about it, filter elements within it, and so on.

View > Expand All

Expands all trees that are not displaying all of their information. That is, this is equivalent to clicking every + icon within the Variable Window.

View > Collapse All

Collapses all trees. That is, this is the equivalent to clicking every – icon within the Variable Window.

View > Undive

Moves up one level in the Variable Window's *dive list* so that you return to the data display previously shown in this window. This list is a history of the data you have displayed. This command is analogous to the "Back" button in a browser in that it returns you to a previous position. Each time you "undive", you move up one item in this list.

As an alternative, you can select the < icon on the right side above the data display.

For additional information, see **View > Redive** on page 137.

View > Undive All

Moves up to the top of the Variable Window's *dive list* so that you return to the data display that existed when you first opened the Variable Window. The dive list is a history of the data that TotalView has displayed in this window.

As an alternative, you can select the **|<**icon within the Variable Window's toolbar.

View > Redive

Moves down on level in the Variable Window's *dive list* so that you return to places you "undove" from. The dive list is a history of the data TotalView has displayed in this window. This command is analogous to the "Forward" button in a browser in that it returns you to a position you previously returned from. Each time you "redive", you move one level towards the bottom of the window's dive list.

As an alternative, you can select the > icon on the right above of the data display.

For additional information, see **View > Undive** on page 137.

View > Redive All

Restores the Variable Window's *dive list* so that you return to *bottom-most* places you "undove" from. The dive list is a history of the data TotalView has displayed in this window. That is, this command is equivalent to repeatedly selecting the Redive command until there are no longer any windows left to restore.

As an alternative, you can select the >| icon within the Variable Window's toolbar.

View > Freeze

Tells TotalView that it should not change the data displayed within this window. Whenever execution stops, TotalView updates the contents of Variable Windows. More precisely, TotalView reevaluates the data found with the Expression area. If you do not want this reevaluation to occur, use this command.

After you select this command, TotalView writes an icon into the window, letting you know that the data is frozen.

Select the **View > Freeze** command a second time to tell TotalView that it should evaluate this window's expression whenever execution stops.

In most cases, you'll want to compare this information with an unfrozen copy. Do this by selecting the **Window > Duplicate** command before you freeze the display. As these two windows are identical, it doesn't matter which one you freeze. If you use the Duplicate command after you freeze the display, just select **View > Freeze** in one of the windows to get that window to update normally.

View > Lock Address

Tells TotalView that it should keep the address at which is evaluating information the same. That is, this freezes the address being used, not the data at that address. (Compare this to **View > Freeze**.)

Here are two examples of using this command:

- If you need to look at a heap address access through a set of dive operations rooted in a stack frame that has become stale.
- If you dive on a ***this** pointer to see the actual value after ***this** goes stale.

Note put on stack

View > Show Across > None

Tells TotalView to remove the changes it made because of a previous view across command. This means that TotalView just shows you values residing in the current process; that is, it no longer shows you the variable's values in all of the program's processes. (*Viewing across* means that TotalView is showing you the value of a variable in more than one process or thread.)

View > Show Across > Process

Displays the value of a variable in the first thread in each process in the share group. When debugging a parallel program, seeing the value of a variable in all processes at the same time can be extremely useful.

View > Show Across > Thread

Displays the value of a variable in all threads in the process. When debugging a threaded program, seeing the value of a variable in all threads at the same time can be extremely useful. You can only turn on thread lamination if the process has more than one non-manager thread, and the variable being displayed depends on the stack or registers. (A static or global variable will have the same value in all threads, so there is no point seeing it use a view across command.)

When matching stack frames to construct a "view across" display, TotalView considers calls from different sites in the same function to be different.

View > Compilation Scope > Fixed

When selected, the scope in which TotalView looks for a variable is *fixed* as being the scope that existed when it displayed the Variable Window.

The Variable Window that TotalView creates contains information that allows TotalView to determine the scope in which a variable is valid. That is, the value that TotalView displays is locked to this scope. If a variable with the same name exists in another scope, TotalView will not display this second variable in the displayed Variable Window because its scope differs. For example, many programs reuse the variable **i** as the loop counter. A second example are the stack variables contained within recursive routines. In this case, each instance of the variable is locked to a different instance of the stack frame.

This behavior, which is the default, occurs when this command's button is selected. If you would like the scope to vary, select the **View > Compilation Scope > Floating** command.

View > Compilation Scope > Floating

When selected, TotalView looks in the current scope to determine if it contains a variable with the same name as the variable that was used when the Variable Window was first created. If TotalView finds one, it displays information about it in the Variable Window even though it scope is not the same as what originally existed.

For example, if you are use the variable **i** as a loop counter in many places, the Variable Window will always display the in-scope variable when this is set. Similarly, if you are displaying Variable Windows for stack variables in a recursive function, the Variable Window will show information for the current recursive routine.

If you want the scope to be fixed, select the **View > Compilation Scope > Fixed** command.

View > Loader Symbols

When you dive on a library or program name within the Program Browser Window, TotalView displays the data in a Variable Window. Normally, this information does not include loader symbols. Selecting this command tells TotalView to display these signals. Reselecting this command tells TotalView that it should no longer display this information.

For more information, see Chapter 6, "*Program Browser Window,*" on page 169.

View > Padding

When selected, TotalView displays padding data that the compiler has added to data structures. (The compiler adds padding bits to data to force the alignment on to word boundaries.)

By default, TotalView does not display this information as errors caused by padding seldom occur. However, you may see errors due to padding if you miscast your data.

View > Break At Newlines

When selected, TotalView inserts a line return instead of a "**\n**" character within long strings. Otherwise, TotalView displays the "\n" character.

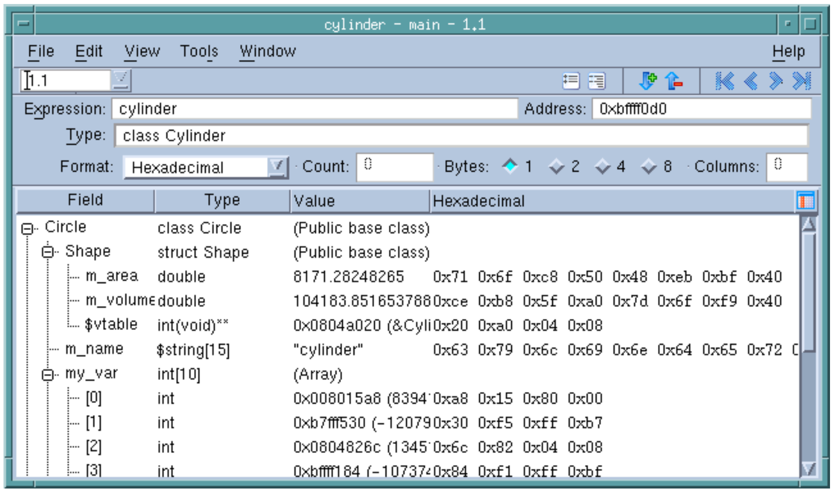
View > Examine Format > None

If the Variable Window is displaying data in either a structured or raw form, selecting this command tells TotalView to restore the Variable Window to its normal display.

View > Examine Format > Structured

When selected, the Variable Window displays data in a structured format. After you select this command, TotalView adds an additional line to the top portion and an additional column to the bottom. This command differs from a raw format (see **View > Examine Format > Raw** for more information) in that it retains the structure of your data.

The controls in the top portion let you display the data in a variety of formats such as hexadecimal, octal, character, etc.



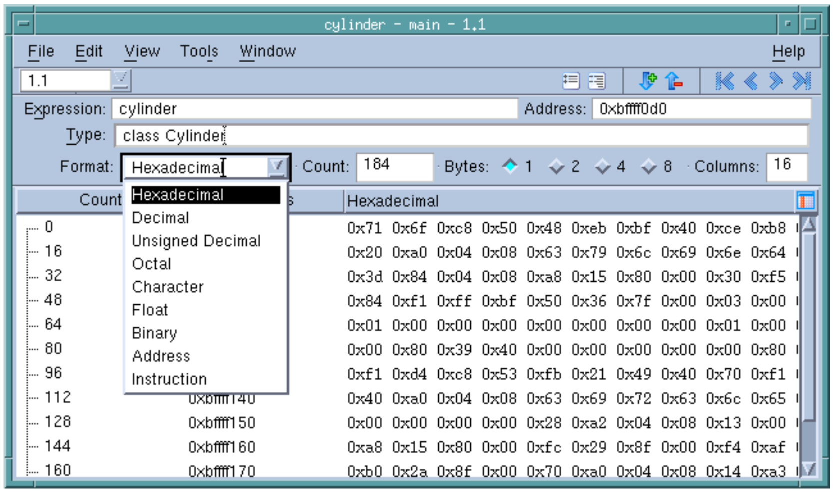
```
Expression: cylinder                Address: 0xbffff0d0
      Type: class Cylinder
    Format: Hexadecimal   Count: 0   Bytes: 1  2  4  8   Columns: 0

Field          Type          Value                Hexadecimal
Circle         class Circle  (Public base class)
  Shape        struct Shape  (Public base class)
    m_area     double        8171.28248265        0x71 0x6f 0xc8 0x50 0x48 0xeb 0xbf 0x40
    m_volume   double        104183.851653788     0xce 0xb8 0x5f 0xa0 0x7d 0x6f 0xf9 0x40
    $vtable    int(void)**   0x0804a020 (&Cyli    0x20 0xa0 0x04 0x08
  m_name       $string[15]   "cylinder"           0x63 0x79 0x6c 0x69 0x6e 0x64 0x65 0x72 (
  my_var       int[10]       (Array)
    [0]        int           0x008015a8 (8394     0xa8 0x15 0x80 0x00
    [1]        int           0xb7fff530 (-12079   0x30 0xf5 0xff 0xb7
    [2]        int           0x0804826c (1345     0x6c 0x82 0x04 0x08
    [3]        int           0xbffff184 (-10737   0x84 0xf1 0xff 0xbf
```

Figure 80: Variable Window: Displaying Structured Data

In addition, if you right-click on the header area of the table, a context menu lets you add a **Status** column. This column contains information such as "Allocated", "PostGuard", "Corrupted PreGuard", etc.

View > Examine Format > Raw

When selected, the Variable Window displays data in a raw format. After you select this command, TotalView adds an additional line to the top portion and changes the display in the bottom.The controls in the top portion let you display the data in a variety of formats such as hexadecimal, octal, character, etc. You can use the **Count** field to tell TotalView how much data to display and the **Bytes** radio buttons to control how the information is grouped.



```
Expression: cylinder                Address: 0xbffff0d0
      Type: class Cylinder
    Format: Hexadecimal   Count: 184   Bytes: 1  2  4  8   Columns: 16

Count                     Hexadecimal
  0                       0x71 0x6f 0xc8 0x50 0x48 0xeb 0xbf 0x40 0xce 0xb8
  16                      0x20 0xa0 0x04 0x08 0x63 0x79 0x6c 0x69 0x6e 0x64
  32                      0x3d 0x84 0x04 0x08 0xa8 0x15 0x80 0x00 0x30 0xf5
  48                      0x84 0xf1 0xff 0xbf 0x50 0x36 0x7f 0x00 0x03 0x00
  64                      0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00
  80                      0x00 0x80 0x39 0x40 0x00 0x00 0x00 0x00 0x00 0x80
  96                      0xf1 0xd4 0xc8 0x53 0xfb 0x21 0x49 0x40 0x70 0xf1
  112      0xbffff140     0x40 0xa0 0x04 0x08 0x63 0x69 0x72 0x63 0x6c 0x65
  128      0xbffff150     0x00 0x00 0x00 0x00 0x28 0xa2 0x04 0x08 0x13 0x00
  144      0xbffff160     0xa8 0x15 0x80 0x00 0xfc 0x29 0x8f 0x00 0xf4 0xaf
  160      0xbffff170     0xb0 0x2a 0x8f 0x00 0x70 0xa0 0x04 0x08 0x14 0xa3
```

Figure 81: Variable Window: Displaying Raw Data

When “counting” information, TotalView defines the size based on the element or field type. As this figure shows, a Raw display shows the contents of memory beginning at the address shown in the **Address** field. You can control the kind of display by selecting a different format.

In addition, if you right-click on the header area of the table, a context menu lets you add a **Status** column. This column contains information such as “Allocated”, “PostGuard”, “Corrupted PreGuard”, etc.

View > Block Status

If you select the **View > Block Status** command, TotalView will also give you additional information about memory when it is displaying structured or raw data. For example, you are told if the memory is in a **text**, **data**, or **bss** section. (If you see unknown, you are probably seeing a stack variable.)

If you have enabled the Memory Debugger, this additional information includes letting you know if memory is allocated or deallocated or being used by a guard block or hoarded.

Tools Menu Commands

The following are on the **Tools** Menu:

- **Tools > Create Watchpoint** on page 142
- **Tools > Add to Expression List** on page 146
- **Tools > Add to Block Properties** on page 146
- **Tools > Visualize** on page 148
- **Tools > Visualize Distribution** on page 148
- **Tools > Statistics** on page 149
- **Tools > Attach Subset (Array of Ranks)** on page 150

Tools > Create Watchpoint

Defines or modifies an unconditional data watchpoint, or a conditional data watchpoint.

TotalView only supports modify watchpoints. That is, TotalView only triggers the watchpoint if your program modifies the memory location. If the same value is written back to the location, the watchpoint does not trigger.

When a watchpoint triggers, the thread’s PC points to the instruction *after* the instruction that caused the watchpoint to trigger. In some cases, where a memory store instruction is the last instruction in the source line, the PC will point to the source line *after* the source line that contains the triggering instruction.

If TotalView displays a question asking if you really want to set a watchpoint on non-global memory, it is telling you that the memory is contained within memory that can change for reasons other than the program changing a variable's value. For example, you may be setting a watchpoint on stack memory. When the routine on the stack is popped, it will probably be over-written by a new stack frame, and this change will have nothing to do with your program changing a value.

This message isn't telling you not to do this. Instead, TotalView is making you aware that change could occur for other reasons.

Platform Restrictions

The number of watchpoints, their size, and alignment restrictions differ from platform to platform. (This is because TotalView relies on the operating system and its hardware to implement data watchpoints.)

Watchpoints are not available on Macintosh computers running OS X, IBM PowerPC *computers running* Linux Power*, and Hewlett Packard* (HP) *computers running either Alpha Linux or* HP-UX.

The following list describes constraints that are unique to each platform:

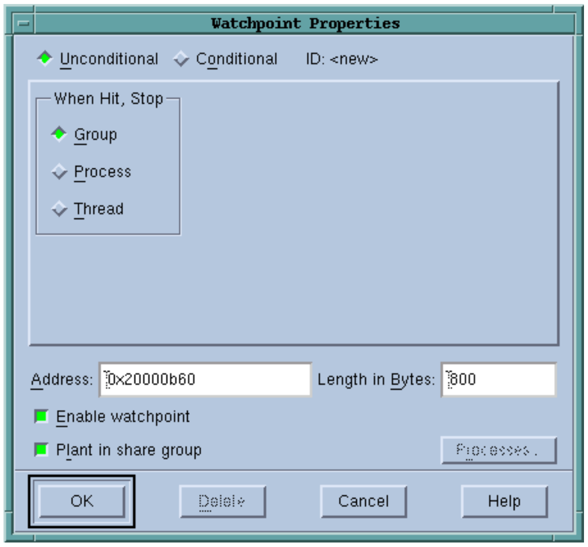
Computer	Constraints
HP Alpha Tru64	Tru64 places no limitations on the number of watchpoints that you can create, and no alignment or size constraints. However, watchpoints can't overlap, and you can't create a watchpoint on an already write-protected page. Watchpoints use a page-protection scheme. Because the page size is 8,192 bytes, watchpoints can degrade performance if your program frequently writes to pages that contains watchpoints
IBM AIX	You can create one watchpoint on AIX 4.3.3.0-2 (AIX 4.3R) or later systems running 64-bit chips. These are Power3 and Power4 systems. (AIX 4.3R is available as APAR IY06844.) A watchpoint cannot be longer than 8 bytes, and you must align it within an 8-byte boundary.
IRIX6 MIPS	Watchpoints are implemented on IRIX 6.2 and later operating systems. These systems let you create approximately 100 watchpoints. There are no alignment or size constraints. However, watchpoints can't overlap.
Linux x86,	You can create up to four watchpoints and each must be 1, 2, or 4 bytes in length, and a memory address must be aligned for the byte length. That is, you must align a 4-byte watchpoint on a 4-byte address boundary, and you must align 2-byte watchpoint on a 2-byte boundary, and so on.
Linux x-86-64 (AMD and Intel)	You can create up to four watchpoints and each must be 1, 2, 4, or 8 bytes in length, and a memory address must be aligned for the byte length. For example, you must align a 4-byte watchpoint on a 4-byte address boundary.

Computer	Constraints
HP-UX IA-64 Linux IA-64,	You can create up to four watchpoints. The length of the memory being watched must be a power of 2 and the address must be aligned to that power of 2; that is, **(address % length) == 0**.
Solaris SPARC	TotalView supports watchpoints on Solaris 2.6 or later operating systems. These operating system let you create hundreds of watchpoints, and there are no alignment or size constraints. However, watchpoints can't overlap.

Typically, a debugging session does not use many watchpoints. In most cases, only one memory location at a time is being monitored. So, restrictions on the number of values you can watch are seldom an issue.

Watchpoint Commands

Figure 82: Tools > Watchpoint Dialog Box



Address: 0x20000b60 Length in Bytes: 800

Watchpoint type

Lets you indicate if the watchpoint is conditional or unconditional.

Unconditional

Tells TotalView to trigger the watchpoint whenever the memory location's value is modified. You can tell TotalView that it should also stop the thread's process or control group when the thread is stopped. Unconditional watchpoints are discussed later in this section.

Conditional

Tells TotalView that, after a memory location is modified, it should evaluate the condition. Conditional watchpoints are discussed later in this section.

Address The first (or lowest) memory address to watch. Depending on the platform, this address may need to be aligned to a multiple of the **Byte Size** field. If you edit the address of an existing watchpoint, TotalView alters the watchpoint so it will watch this new memory location and reassigns the watchpoint's action point ID.

Byte Size	The number of bytes that TotalView should watch. Normally, this amount is the size of the variable. However, some architectures limit the amount of memory that can be watched. In other cases, you may want TotalView to monitor a few locations in an array. For information on architectural limitations, see "*Platform Restrictions*".
Enable watchpoint	If selected, TotalView makes this watchpoint active. (If a watchpoint is inactive, TotalView ignores changes to the watched memory locations.)
Plant in share group	If selected, TotalView creates a watchpoint for each thread in the share group. Some architectures place limits on the size and number of watchpoints. See "*Platform Restrictions*" for more information.

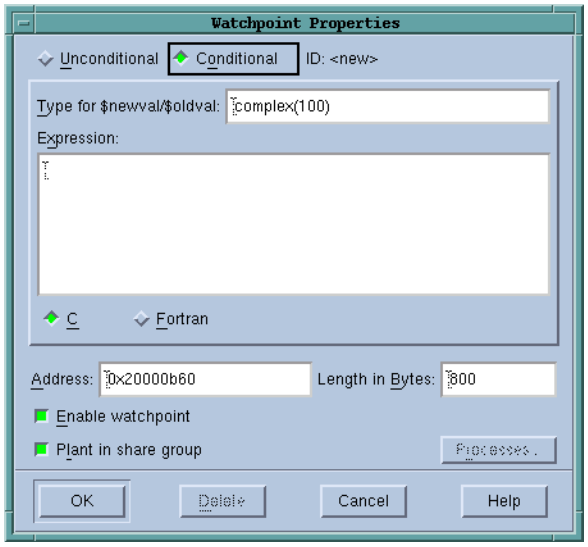
Unconditional Watchpoints

The only control unique to unconditional watchpoints is **When Hit, Stop**, which tells TotalView what should be stopped when a watched location changes. While TotalView will always stop the executing **Thread** (the thread of interest), it can also stop the thread of interest's control **Group** or all thread's in the thread of interest's **Process**.

Conditional Watchpoints

Using the following controls, you can create a watchpoint that triggers only when the condition you specify occurs.

Figure 83: Tools > Watchpoint Dialog Box



Watchpoint Properties

Unconditional [✓ Conditional] ID: <new>

Type for $newval/$oldval: complex(100)

Expression:

[✓ C] Fortran

Address: 0x20000b60 Length in Bytes: 800

[✓] Enable watchpoint

[✓] Plant in share group

Type for $newval/$oldval

If you will be placing the value stored at the memory location into a variable (using **$newval** and **$oldval**), you must define the variable's data by using a scalar type, such as **int**, **integer**, **float**, **real**, or **char**. You cannot use aggregate types such as arrays and structures.

If the size of the watched location matches the size of the data type entered here, TotalView interprets the **$oldval** and **$newval** information as the variable's type. If you are watching an entire array, the watched location can be larger than the size of this type.

Expression

Enter a code fragment. The expression is compiled into interpreted code that is executed each time the watchpoint triggers. These points can be used to implement countdown and conditional watchpoints. For additional information, see Setting Action Points of the TotalView *Users Guide*.

C or Fortran

Indicates the programming language in which you wrote the expression.

The *TotalView Users Guide* contains additional information on watchpoints.

You can create compiled conditional watchpoints when you running programs on an IBM AIX *computer. When watchpoints are compiled, they are evaluated by the process rather than having to be evaluated in* TotalView *main, where all evaluations are single-threaded and must be sent from separately executing processes.*

Tools > Add to Expression List

See "*Expression List Window Overview*" on page 205 for more information.

Tools > Add to Block Properties

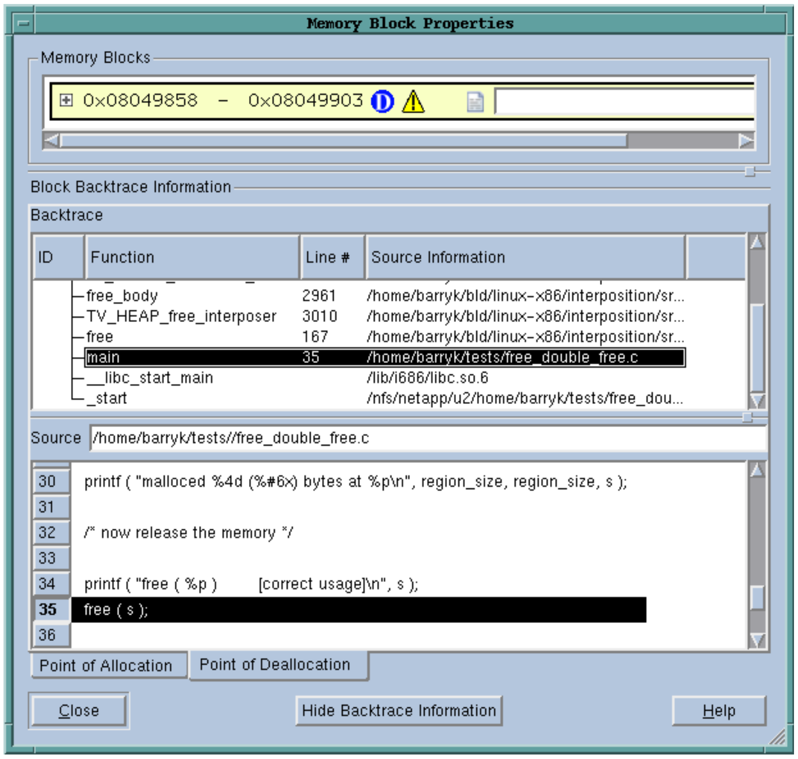
The **Memory Block Properties** Window displays information about all of the blocks that you asked the Memory Debugger to keep track of by using this command. You can only use this command if this variable's memory block was allocated within the heap. For example, assume that you have the following statements within your program:

```
void *s;
s = malloc(sizeof(int)*200);
```

You cannot obtain block properties for variable s as s is allocated on the stack. However, after **malloc()** allocates memory, s points to a memory block. You can now have the Memory Debugger watch this allocated block.

This means that you will need to dive on a pointer so that TotalView can dereference the pointer before you use this command.

You can display this window in two different ways. Pressing the **Hide Backtrace Information** conceals most of what is displayed in Figure 84 on page 147.



0x08049858 – 0x08049903

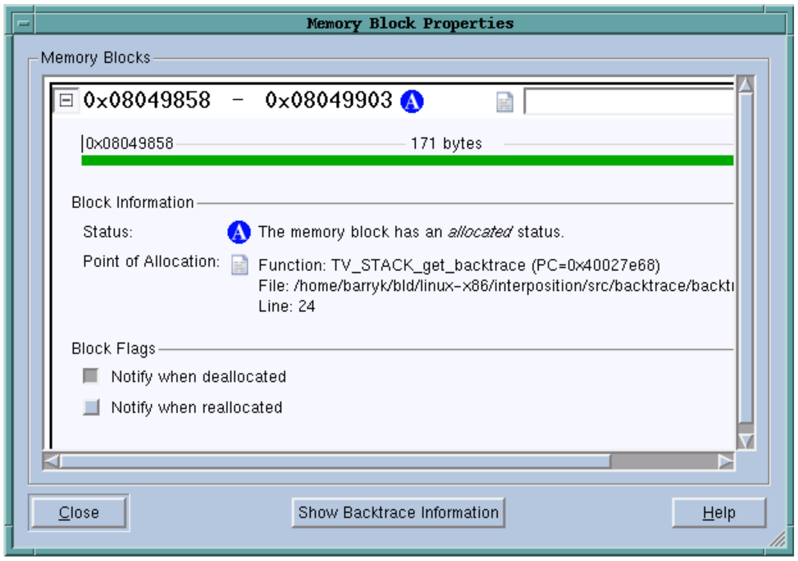
ID	Function	Line #	Source Information
	free_body	2961	/home/barryk/bld/linux-x86/interposition/sr...
	TV_HEAP_free_interposer	3010	/home/barryk/bld/linux-x86/interposition/sr...
	free	167	/home/barryk/bld/linux-x86/interposition/sr...
	main	35	/home/barryk/tests/free_double_free.c
	__libc_start_main		/lib/i686/libc.so.6
	_start		/nfs/netapp/u2/home/barryk/tests/free_dou...

Source /home/barryk/tests//free_double_free.c

```
30  printf ( "malloced %4d (%#6x) bytes at %p\n", region_size, region_size, s );
31
32  /* now release the memory */
33
34  printf ( "free ( %p )       [correct usage]\n", s );
35  free ( s );
36
```

Figure 84: Tools > Memory Block Properties Window

After pressing this button, the window reconfigures itself to that shown in Figure 85 on page 147. When this window is being displayed, pressing **Show Backtrace Information** shifts it back.



0x08049858 – 0x08049903

0x08049858 171 bytes

Status: The memory block has an *allocated* status.

Point of Allocation: Function: TV_STACK_get_backtrace (PC=0x40027e68)
File: /home/barryk/bld/linux-x86/interposition/src/backtrace/backt
Line: 24

Figure 85: Tools > Memory Block Properties Window

As an alternative, you can both make the window larger and use the splitter control to enlarge the top area.

The information within the bottom area is essentially the same as that which is displayed when you generate a Backtrace View in the Memory Debugger. You will find information on the contents of the Block Backtrace Information area within "*Heap Status Page*" on page 203

The first line contains a graphical summary of the blocks information. It contains:

- Memory block extent: the numbers in bold indicate the starting and ending address of the block. For example, the block in Figure 85 indicates the block starts at 0x08049858 and ends at 0x08049903.
- Status: an indicator indicating if the block is allocated, deallocated, leaked, or being hoarded. For example, the Ⓐ in Figure 85 indicates that the block is allocated.
- Point of Allocation: if you place your mouse over the 🗎 icon, TotalView displays a tool tip that shows the function in which the block was allocated, the function's source file, and the line number within that file.
- Comment: if you have more than one or two memory blocks, you might want to enter a comment in the text box to remind yourself what the block is.

The next lines within the Memory Blocks area restate the information that was presented in the summary area. The color of the block's graphic is the same as that used in the Heap Status Page's Graphical View.

The Block Flags area contains two check boxes:

- Notify when deallocated
- Notify when reallocated

Selecting a box tells the Memory Debugger that it should stop execution when the block is deallocated or reallocated and display the Memory Event Details Window. This allows you to track when your program is managing memory and what is being managed. For example, if you have a double free problem, obtaining notification lets you know where the block is first freed.

Tools > Visualize

Displays the Visualizer. The Visualizer lets you graphically display the values contained within a numeric array. For more information, see **Visualizer Window** on page 155.

Tools > Visualize Distribution

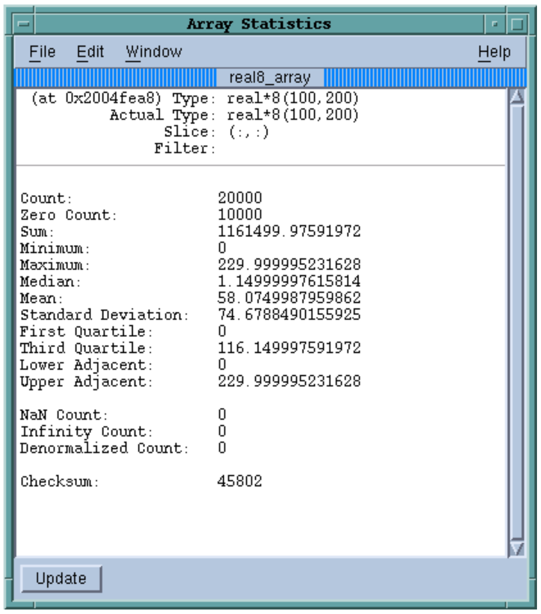
Displays the Visualizer. The Visualizer lets you graphically display the values contained within a numeric array. This differs from a normal visualizer display in that one of the axes is the node upon which the data resides. This command is used with Global Arrays and UPC programs.

For more information, see **Visualizer Window** on page 155.

Tools > Statistics

Displays statistics for the displayed array elements. If the Variable Window contains a slice expression or a filter, only these values are used when TotalView creates the statistical information. (See Figure 86.)

Figure 86: Array Statistics



```
(at 0x2004fea8) Type:  real*8(100,200)
         Actual Type:  real*8(100,200)
               Slice:  (:,:)
              Filter:

Count:                 20000
Zero Count:            10000
Sum:                   1161499.97591972
Minimum:               0
Maximum:               229.999995231628
Median:                1.14999997615814
Mean:                  58.0749987959862
Standard Deviation:    74.6788490155925
First Quartile:        0
Third Quartile:        116.149997591972
Lower Adjacent:        0
Upper Adjacent:        229.999995231628

NaN Count:             0
Infinity Count:        0
Denormalized Count:    0

Checksum:              45802
```

The statistics calculated are:

Adjacents Displays the lower and upper adjacents. The lower adjacent provides an estimate of the lower limit of the array. Values below this limit are called *outliers*. The lower adjacent value is the first quartile value less 1.5 times the difference between the first and third quartiles.

The upper adjacent value provides an estimate of the upper limit of the array. Values above this limit are called outliers. The upper adjacent value is the third quartile value plus 1.5 times the difference between the first and third quartiles.

Checksum A checksum value for the array elements.

Count The number of elements that participated in the statistical calculations. For floating-point arrays, this does not include any NaN or infinity (INF) values.

Denormalized Count
A count of the number of denormalized values found in a floating-point array. This includes both negative and positive denormalized values as defined in the IEEE floating-point standard. These elements do participate in the statistical calculations.

Infinity Count A count of the number of infinity (INF) values found in a floating-point array. This includes both negative and

	positive infinity as defined in the IEEE floating-point standard. These elements do not participate in statistical calculations.
Maximum	The largest value in the array.
Mean	The mean value of the array.
Median	The median value of the array.
Minimum	The smallest value in the array.
NaN Count	A count of the number of NaN values found in a floating-point array. This includes both signaling and quiet NaNs as defined in the IEEE floating-point standard. These elements do not participate in statistical calculations.
Quartile	Displays either the first or third quartile. The first quartile is the 25th percentile value in the array. That is, 25 percent of the array values are smaller than this value and 75 percent are greater. The third quartile is the 75th percentile value in the array. This means that 75 percent of the array values are less than this value and 25 percent are greater.
Standard Deviation	The standard deviation of the array values.
Sum	The summation of all the array elements.
Zero Count	The number of elements that have a value of 0.

Tools > Attach Subset (Array of Ranks)

Lets you indicate which processes TotalView should attach to when these processes begin executing. Limiting the processes to which TotalView attaches is beneficial as TotalView does not have to be concerned with unattached processes. That is, because you know that you will not be interested in a what goes on in within a process, you can cut down on the time that TotalView uses to attach to all or most of your processes.

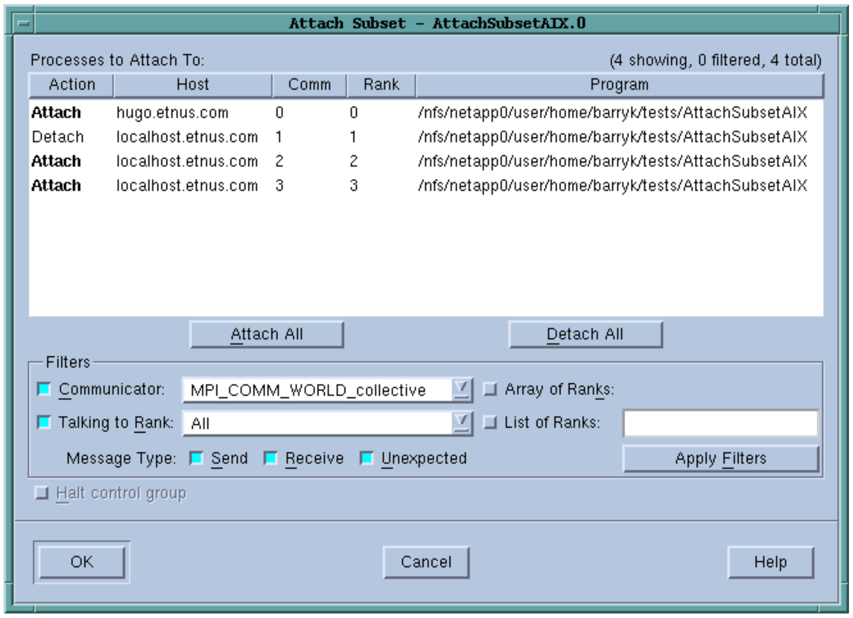
TotalView lets you start MPI *jobs in two ways. One requires that the starter process be under TotalView control and have special instrumentation for TotalView, and the other does not. In the first case, you will enter the name of the starter program on the command line. The other requires that you enter information into the File > New Program or the Process > Startup Parameters dialog boxes. The Attach Subset command is only available if you directly name a starter program on the command line. This is discussed in the TotalView Users Guide.*

Processes to Attach To

Use the controls in this area to specify the processes to which TotalView should attach when they are created. You have three choices:

Selection Area	Individually select or deselect processes
All	Attach to all of the listed processes.

Figure 87: Attach Subset Dialog Box



Attach Subset - AttachSubsetAIX.0

Processes to Attach To: (4 showing, 0 filtered, 4 total)

Action	Host	Comm	Rank	Program
Attach	hugo.etnus.com	0	0	/nfs/netapp0/user/home/barryk/tests/AttachSubsetAIX
Detach	localhost.etnus.com	1	1	/nfs/netapp0/user/home/barryk/tests/AttachSubsetAIX
Attach	localhost.etnus.com	2	2	/nfs/netapp0/user/home/barryk/tests/AttachSubsetAIX
Attach	localhost.etnus.com	3	3	/nfs/netapp0/user/home/barryk/tests/AttachSubsetAIX

Filters

Communicator: MPI_COMM_WORLD_collective

Talking to Rank: All

Message Type: Send, Receive, Unexpected

None Do not attach to any of these processes.

After selecting **All** or **None**, you can individually select or unselect processes. That is, if you only want to select a couple of processes, begin by clicking **None**, then select the few to which TotalView should attach.

Filters You can restrict the list by selecting the controls in this area.

Communicator
The communicators within this list tell TotalView which processes it should display. Selecting one of the communicators contained within this list tells TotalView that it should only display processes using this communicator. You can then select or clear these values in one of the three ways just discussed.

Talking to Rank
TotalView will limit the graph to communicators that receive messages from the indicated ranks. In addition to your rank numbers, TotalView includes two special variables: **All** and **MPI_ANY_SOURCE**.

Message Type
TotalView will only show **Send**, **Receive**, or **Unrestricted** messages.

Array of Ranks
This checkbox is automatically selected by TotalView if you have invoked this command from the **Tools > Attach Subset (Array of Ranks)** command. If the Variable Window is displaying an array, invoking this command tells TotalView that the array's elements indicate ranks.

Halt control group

Selecting this button tells to stop all of the processes in the current process's control group after it attaches to a process. If it isn't selected. TotalView will immediately execute the control group after it attaches to them.

Window Menu Commands

The **Window** Menu contains the following commands:

- **Window > Update** on page 152
- **Window > Update All** on page 152
- **Window > Duplicate** on page 152
- **Window > Memorize** on page 152
- **Window > Memorize all** on page 153
- **Window > Root** on page 153

Window > Update

Updates the display of this window. U*pdate* means that the process is momentarily stopped so that TotalView can determine the program's state and the value of variables. It then updates this Variable Window and other Variable Windows associated with this variable's process so that they contain updated values.

Window > Update All

Tells TotalView to update the contents of all windows. That is, TotalView fetches and then displays the current value of the information in all open windows. U*pdate* means that the process is momentarily stopped so that TotalView can determine the program's state and the value of variables. It then updates this Variable Window and other Variable Windows associated with this variable's process so that they contain updated values.

Window > Duplicate

Tells TotalView to create an identical copy of this Variable Window.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.

TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the File > Preference*'s Options Page.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.

TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

Visualizer Window

chapter 4

The **Tools > Visualize** and **Tools > Visualize Distribution** commands, and the **$visualize** intrinsic send an array's values to the Visualizer so that it can display these values. A "view across" display can also be visualized, in which case, the process or thread index forms one axis of the display.

The **Launch Strings Page** within the **File > Preferences Window** dialog box allows you to specify the command TotalView will use to start the Visualizer.

The **Maximum array rank** field sets the maximum rank of the array that TotalView will export to the Visualizer. The Visualizer cannot visualize arrays of rank greater than two; if you are using another visualizer, however, or if you are dumping binary data, you can set the limit here.

The Visualizer can be used in two ways: it can be launched by TotalView to visualize data as you debug your programs, and it can be run from the command line to visualize data dumped to a file in a previous debugger session.

Visualizing your program's data uses two interactions:

- You interact with TotalView to choose *what* you want to visualize and *when* it should make snapshots of your data.
- You interact with the visualizer to choose *how* you would like your data to be displayed.

The TotalView handles the first of these interactions, extracting data and marshalling it into a standard format that it sends down a pipe. The Visualizer then reads the data from this pipe and displays it for analysis.

The Visualizer has two types of windows:

- *Dataset Window*

  A single main window lists the datasets that you can visualize. You can use this window to set global options and to create views of your datasets.
- *View Window*

  A *data window* contain images of the datasets. By interacting with a View Window, you can change its appearance and set dataset viewing options.

Using the Dataset Window, you can open several View Windows on a single dataset to get different views of the same data.

For more information on the Visualizer, see *TotalView Users Guide*.

Dataset Window

The Dataset Window contains a list of the datasets you can display.

You can select a dataset by left-clicking on it, and you can only select one dataset at a time. The **View** menu commands let you select **Graph** or **Surface** visualizations. Whenever TotalView sends a new dataset, the Visualizer updates its list of datasets. To delete a dataset from the list, use the **File > Delete** command.

File Menu Commands

The following commands are on the **File** pulldown:

- File > Delete
- File > Exit

File > Delete Deletes the currently selected dataset. It removes the dataset from the dataset list and *destroys* any View Windows displaying it.

File > Exit Closes all windows and exits the **Visualizer**.

View Menu Commands

The following commands are on the **View** pulldown:

- View > Graph
- View > Surface

View > Graph Creates a new graph window.

View > Surface Creates a new surface window.

Options Menu Command

The following command is on the **Options** pulldown: *Options > Auto Visualize*.

Options > Auto Visualize This item is a toggle; when enabled, the **Visualizer** automatically visualizes new datasets as they are read.

View Window

View Windows display graphical images of your data. Every View Window contains a menu bar and a drawing area. The View Window title is its dataset identification.

The following table defines general commands that you can use while display a surface view. Command letters can be typed in either upper- or lower-case.

Action	Press
Pick (show value): The Visualizer shows the value of the data point underneath the cursor.	p
Camera mode: Mouse events affect the camera position and focal point. (Axes moves, and you don't.)	c
Actor mode: Mouse events affect the actor that is under the mouse pointer. (You move, not the axes.)	a
Joystick mode: Motion occurs continuously while you are pressing a mouse button.	j
Trackball mode: Motions only occurs when you press the mouse button and you move the mouse pointer.	t
Wireframe view: The Visualizer displays the surface as a mesh. (This is the same as not checking the Surface option.)	w
Surface view: The Visualizer displays the surface as a solid. (This is the same as having checked the Surface option.)	s
Reset: Removes some of the changes you've made to the way the Visualizer displays an object.	r
Initialize: Restores the object to what it was before you interacted with the Visualizer. As this is a menubar accelerator, the window must have focus.	i
Exit or **Quit**: Close the Visualizer or	Ctrl+Q

The following table defines the actions you can perform using your mouse:

Action		Click or Press
Camera mode	**Actor mode**	
Rotate camera around focal point (surface only)	Rotate actor around focal point (surface only)	Left mouse button
Zoom: you appear to get closer to the object.	Scale: the object appears to get larger	Right mouse button
Pan: you move the "camera". For example, moving the camera up means the object moves down.	Translate: The object moves in the direction you pull it.	Middle mouse button or Shift-left mouse button

The **File** menu on the menu bar is the same for all View Windows. Other items on the menu bar are specific to particular types of View Window.

File Menu Commands

The following commands are on the **File** pulldown:

- File > Dataset
- File > Options
- File > Delete
- File > Close
- Window > New Base Window

File > Dataset

Raises the Dataset Window to the front of the desktop. If the Dataset Window is minimized, it is restored.

File > Options

Displays a dialog box containing viewing options. Here is what you'll see if you are displaying a Graph View:

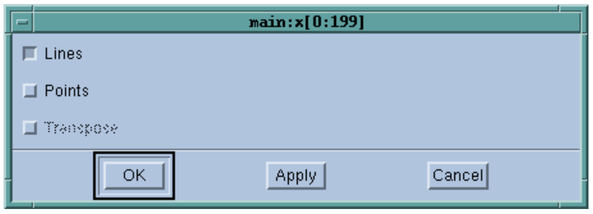


Figure 88: Graph Options Dialog Box

The following describes the meaning of these check boxes:

Lines	When set, the Visualizer displays lines connecting dataset elements.
Points	When set, the Visualizer displays markers for dataset elements.
Transpose	When set, the Visualizer inverts the **X** and **Y** axes of the displayed graph.

Here is what you'll see if you are displaying a Surface View:

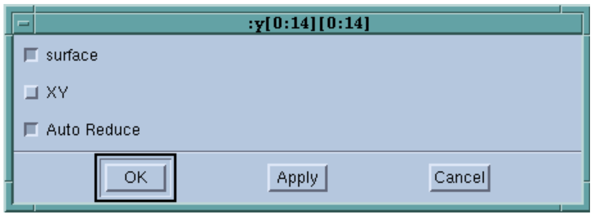


Figure 89: Surface Options Dialog Box

The following describes the meaning of these check boxes:

Surface	If this option is set, the Visualizer displays the array's data as a three dimensional surface. If you don't set this option, the Visualizer displays the surface in two dimensions.
XY	If this option is set, the Visualizer reorients the view's XY axes.

Auto Reduce — If this option is set, the Visualizer derives the displayed surface by averaging over neighboring elements in the original dataset. This speeds up visualization by reducing the resolution of the surface. Clear this option if you want to accurately visualize all dataset elements.

The Auto Reduce option lets you choose between viewing all your data points—which takes longer to appear in the display—or viewing the averaging of data over a number of nearby points.

File > Delete Deletes the View Window's dataset from the dataset list. This also destroys any other View Windows viewing the dataset.

File > Close Closes the View Window.

Window > New Base Window Creates a new View Window using the same visualization method and dataset as the current View Window.

chapter

5

Fortran Modules Window

Selecting the Process Window's **Tools > Fortran Modules** command tells TotalView to display a window containing information about the Fortran modules that are used by a process.

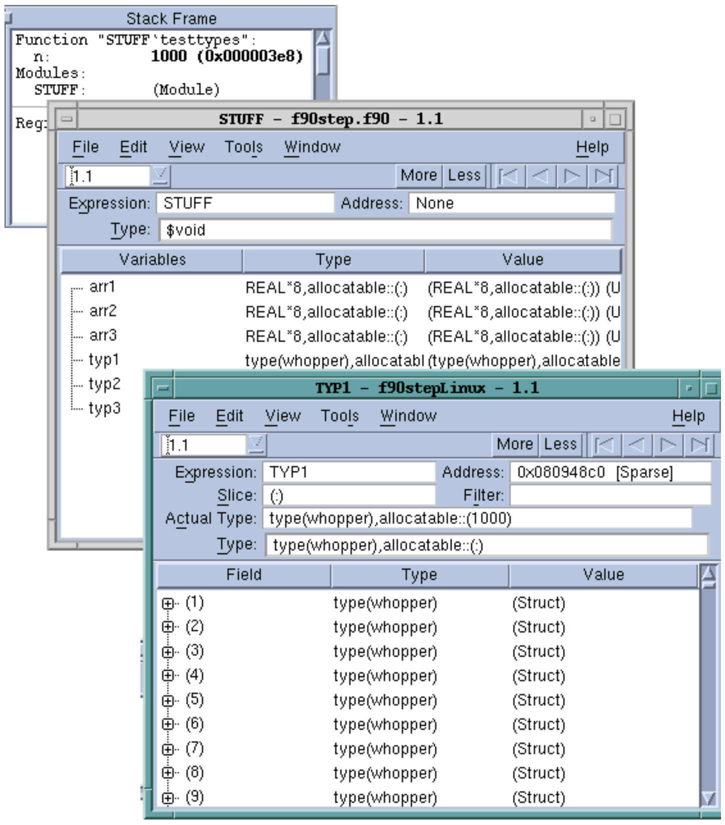
If you compiled your code using the SUNPro F90 *compiler, TotalView must scan all debugging information to locate all module names. As this can be very time-consuming, TotalView only displays information for modules that are named within already read symbol information. If module you want to see is not displayed, use the Process Window's* **View > Lookup Function** *command to open a file using a module.*

To see more information about a module, dive (double-click) on the module's name. TotalView respond's by displaying a Variable Window containing information about that module's variables.

If a module already has a window when you dive into its name in the Fortran Modules Window, TotalView brings its old window to the front of the screen.

To see information about a module without first scrolling to it in the modules Window, select the Process Window's **View > Lookup Variable** command and enter the name of the module into the displayed dialog box.

Figure 90: Fortran Modules


Stack Frame
STUFF - f90step.f90 - 1.1
File Edit View Tools Window Help
1.1
More Less
TYP1 - f90stepLinux - 1.1
File Edit View Tools Window Help
1.1
More Less

```
Function "STUFF`testtypes":
  n:            1000 (0x000003e8)
Modules:
  STUFF:        (Module)
Reg:
```

Expression: STUFF Address: None
Type: $void

Variables	Type	Value
arr1	REAL*8,allocatable::(:)	(REAL*8,allocatable::(:)) (U
arr2	REAL*8,allocatable::(:)	(REAL*8,allocatable::(:)) (U
arr3	REAL*8,allocatable::(:)	(REAL*8,allocatable::(:)) (U
typ1	type(whopper),allocatabl	(type(whopper),allocatable
typ2		
typ3		

Expression: TYP1 Address: 0x080948c0 [Sparse]
Slice: (:) Filter:
Actual Type: type(whopper),allocatable::(1000)
Type: type(whopper),allocatable::(:)

Field	Type	Value
(1)	type(whopper)	(Struct)
(2)	type(whopper)	(Struct)
(3)	type(whopper)	(Struct)
(4)	type(whopper)	(Struct)
(5)	type(whopper)	(Struct)
(6)	type(whopper)	(Struct)
(7)	type(whopper)	(Struct)
(8)	type(whopper)	(Struct)
(9)	type(whopper)	(Struct)

File Menu Commands

The commands on the **File** pull-down are:

- **File > Close Similar** on page 162
- **File > Close** on page 162

File > Close Similar

Closes this windows and close other Fortran Module Windows.

File > Close

Closes this window. No other windows are affected by this command.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 163
- **Edit > Cut** on page 163
- **Edit > Copy** on page 163
- **Edit > Paste** on page 163
- **Edit > Delete** on page 164
- **Edit > Find** on page 164
- **Edit > Find Again** on page 165

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After you make the editing change, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

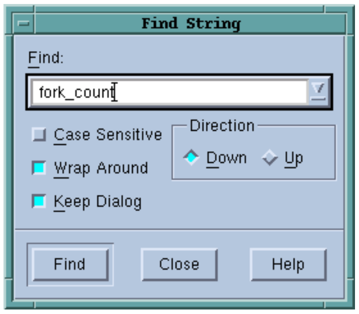
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 91: Edit > Find Dialog Box



The controls in this dialog box are:

Find	Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered.
Case Sensitive	If selected, TotalView only locates text having the same capitalization as the text entered in the **Find** field.
Wrap On Search	If selected, TotalView will continue the search from either the beginning (if **Down** is also selected) or the end (if **Up** is also selected.) For example, you search for "foo" and the **Down** button is selected. If it isn't found 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 had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the **Find** button. If you select this option, you will need to select the **Close** button to dismiss this dialog box.
Direction	Sets the direction in which TotalView searches. **Up** means "search from the current position to the beginning of the file." **Down** means "search from the current position to the end of the file."

Find	Tells TotalView to search for the text within the **Find** box.
Close	Closes the **Find** dialog box.

After you find a string, you can locate another instance of the string by using the **Edit** > **Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the **View** pulldown are:

- **View > Dive** on page 165
- **View > Dive in New Window** on page 165

View > Dive

Tells TotalView to "dive" into the selected item. In all cases, "dive" means that TotalView will show more information. For example:

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

In both cases, if a Process or Variable Window already exists, TotalView brings that window to the top of the display.

View > Dive in New Window

Tells TotalView to open a "dive" into the selected item. In all cases, "dive" means that TotalView will show more information.

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

Even if a Process or Variable Window already exists for a process or variable, TotalView always creates a new window for this information.

Window Menu Commands

The commands on the **Window** pulldown are:

- **Window > Update** on page 166
- **Window > Update All** on page 166
- **Window > Memorize** on page 166
- **Window > Memorize all** on page 166
- **Window > Root** on page 167

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.

TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference***'s Options Page.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.

TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

chapter 6

Program Browser Window

Use this window to examine symbols contained within your program. After you invoke this command, TotalView displays a window containing the libraries and programs that make up your executable. If you dive on any of these, TotalView displays a Variable Window that contains information about the library or program. (See Figure 92 on page 170.)

If you need to see more information about a variable, you can dive on it. In this case, the variable you dove upon replaces the information in the current Variable Window.

If you need to see loader symbols (they are not displayed by default), use the **View** > **Loader Symbols** command from within the Variable Window.

File Menu Commands

The **File** menu contains the following commands:

- **File** > **Close Similar** on page 169
- **File** > **Close** on page 169
- **File** > **Exit** on page 170

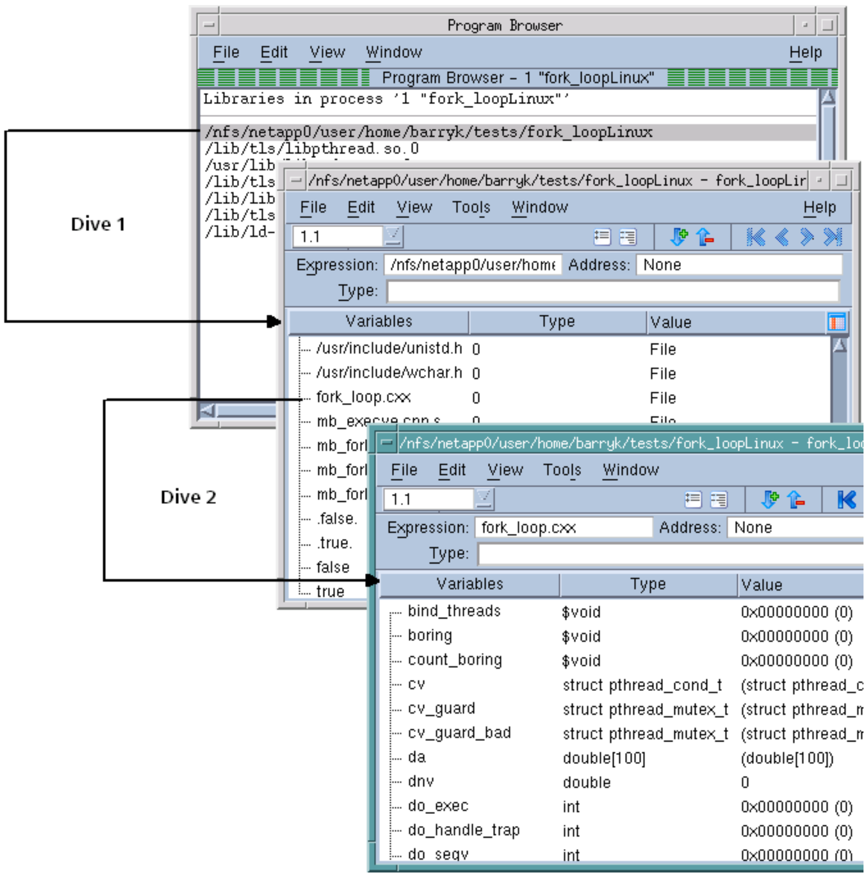
File > Close Similar

Closes all Program Browser Windows.

File > Close

Closes this window. No other windows are affected by this command.

Figure 92: Program Browser


Dive 1
Dive 2
Program Browser
File Edit View Window Help
File Edit View Tools Window Help
File Edit View Tools Window

Program Browser - 1 "fork_loopLinux"

```
Libraries in process '1 "fork_loopLinux"'
/nfs/netapp0/user/home/barryk/tests/fork_loopLinux
/lib/tls/libpthread.so.0
/usr/lib
/lib/tls
/lib/lib
/lib/tls
/lib/ld-
```

/nfs/netapp0/user/home/barryk/tests/fork_loopLinux - fork_loopLir

1.1

Expression: /nfs/netapp0/user/home Address: None

Type:

Variables	Type	Value
/usr/include/unistd.h	0	File
/usr/include/wchar.h	0	File
fork_loop.cxx	0	File
mb_execve.cpp.c	0	File
mb_forl		
mb_forl		
mb_forl		
.false.		
.true.		
false		
true		

/nfs/netapp0/user/home/barryk/tests/fork_loopLinux - fork_loo

1.1

Expression: fork_loop.cxx Address: None

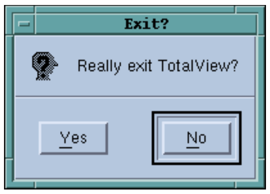
Type:

Variables	Type	Value
bind_threads	$void	0x00000000 (0)
boring	$void	0x00000000 (0)
count_boring	$void	0x00000000 (0)
cv	struct pthread_cond_t	(struct pthread_c
cv_guard	struct pthread_mutex_t	(struct pthread_m
cv_guard_bad	struct pthread_mutex_t	(struct pthread_m
da	double[100]	(double[100])
dnv	double	0
do_exec	int	0x00000000 (0)
do_handle_trap	int	0x00000000 (0)
do_segv	int	0x00000000 (0)

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.)

Figure 93: File > Exit Command



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 171
- **Edit > Cut** on page 171
- **Edit > Copy** on page 171
- **Edit > Paste** on page 171
- **Edit > Delete** on page 172
- **Edit > Find** on page 172
- **Edit > Find Again** on page 173

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After you make the editing change, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

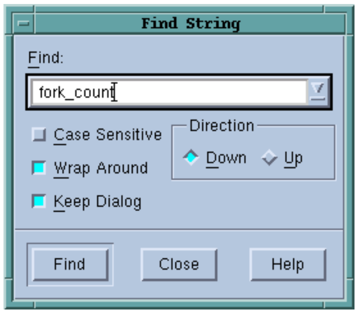
Deletes the selected information. This deleted information is not placed onto the clipboard.

You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.



Find: fork_count

[ ] Case Sensitive
[✓] Wrap Around
[✓] Keep Dialog

Direction: [✓ Down] Up

Figure 94: Edit > Find Dialog Box

The controls in this dialog box are:

Find	Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered.
Case Sensitive	If selected, TotalView only locates text having the same capitalization as the text entered in the **Find** field.
Wrap On Search	If selected, TotalView will continue the search from either the beginning (if **Down** is also selected) or the end (if **Up** is also selected.) For example, you search for "foo" and the **Down** button is selected. If it isn't found 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 had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the **Find** button. If you select this option, you will need to select the **Close** button to dismiss this dialog box.
Direction	Sets the direction in which TotalView searches. **Up** means "search from the current position to the beginning of the file." **Down** means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the **Find** box.

Close	Closes the **Find** dialog box.

After you find a string, you can locate another instance of the string by using the **Edit** > **Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the View menu are:

- **View > Dive** on page 173
- **View > Dive in New Window** on page 173

View > Dive

Tells TotalView to open a "dive" into the selected item. In all cases, "dive" means that TotalView will show more information. TotalView responds by opening a Variable window containing information about that variable.

If the Variable Window already exists for the variable, TotalView brings that window to the top of the display.

View > Dive in New Window

Tells TotalView to open a "dive" into the selected item. In all cases, "dive" means that TotalView will show more information. TotalView responds by opening a Variable window containing information about that variable.

Even if a Variable Window already exists for the variable, TotalView will still create a new window for this information.

Window Menu Commands

The commands on the window pulldown are:

- **Window > Update** on page 174
- **Window > Update All** on page 174

- Window > **Memorize** on page 174
- Window > **Memorize all** on page 174
- Window > **Root** on page 174

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.

TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the File > Preference*'s Options Page.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.

TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

chapter 7

Message Queue Window

Message Queue Window Overview

The **Tools > Message Queue** command tells TotalView that it should display information about the current process's message queues.

Message queues are displayed for the following versions of MPI:

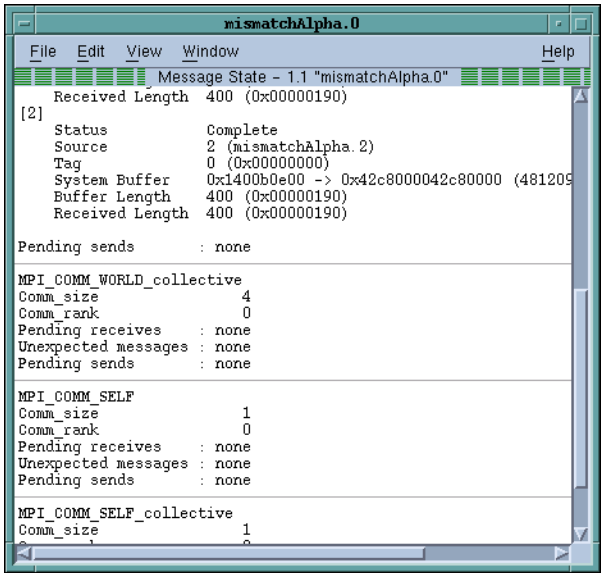
- MPICH version 1.1.0 or later.
- Compaq Alpha MPI (DMPI) version 1.7.
- HP HP-UX version 1.6.
- IBM MPI Parallel Environment (PE) version 2.3 or 2.4; but only for programs using the threaded IBM MPI libraries. TotalView cannot show message queues for earlier releases or with the non-thread-safe version of the IBM MPI library. Therefore, to use the message queue display with IBM MPI applications, you must compile and link your code using the **mpcc_r**, **mpxlf_r**, or **mpxlf90_r** compilers.
- On SGI MPI, you must obtain the Message Passing Toolkit (MPT) release 1.3 or later to display message queues. Check with SGI for availability.

This dialog box displays the state of each of the MPI communicators that exist in the process. In some MPI implementations, such as MPICH, user-visible communicators are implemented as two internal communicator structures, one for point-to-point and the other for collective operations. TotalView displays both.

You cannot edit any of the fields in the Message Queue dialog box.

The contents of the Message Queue dialog box are only valid when a process is stopped.

Figure 95: Message Queue Window


mismatchAlpha.0
File Edit View Window Help
Message State - 1.1 "mismatchAlpha.0"

```
    Received Length  400 (0x00000190)
[2]
    Status           Complete
    Source           2 (mismatchAlpha.2)
    Tag              0 (0x00000000)
    System Buffer    0x1400b0e00 -> 0x42c8000042c80000 (481209
    Buffer Length    400 (0x00000190)
    Received Length  400 (0x00000190)

Pending sends        : none

MPI_COMM_WORLD_collective
Comm_size              4
Comm_rank              0
Pending receives     : none
Unexpected messages  : none
Pending sends        : none

MPI_COMM_SELF
Comm_size              1
Comm_rank              0
Pending receives     : none
Unexpected messages  : none
Pending sends        : none

MPI_COMM_SELF_collective
Comm_size              1
```

For each communicator, TotalView displays the following fields:

Communicator Name
MPI lets you name predefined communicators such as **MPI_COMM_WORLD()**. In addition, MPICH 1.1 and Compaq MPI use the MPI-2 **MPI_NAME_PUT()** and **MPI_NAME_GET()** communicator naming functions that let you associate a name with a communicator. If you use **MPI_NAME_PUT()** to name a communicator, TotalView uses the name you gave it when it displays the communicator.

IBM MPI and SGI MPI do not implement the MPI-2 communicator naming functions, which means that only predefined communicators are named. For user-created communicators, TotalView displays the integer value that represents the communicator. This is the value that a variable of type **MPI_Communicator** has when it represents a communicator.

Comm_size The number of processes in the communicator. This value is the same value as occurs when you apply **MPI_Comm_size()** to the communicator.

Comm_rank The rank in the communicator of the process that owns the Message Queue Window. This information is the same information you would get if you had applied **MPI_Comm_rank** to the communicator in this process.

Pending receive operations

A list of pending receive operations.

Unexpected messages
A list of messages sent to this communicator but that have not yet been matched with a receive.

Pending send operations	A list of pending send operations.

Message Operations

For each communicator, TotalView displays a list of pending receive operations, unexpected messages, and pending send operations. Each operation has an index value displayed in brackets ([*n*]), and each operation can include the following fields:

Actual Source	If the **Status** is **Complete** and the **Source** is **ANY**, this is the receiving process.
Actual Tag	If the **Status** is **Complete** and the **Tag** value is **ANY**, this is the received tag value.
Buffer Length or Received Length	The buffer length in bytes, shown in decimal and hexadecimal.
Function	The MPI function (IBM MPI only). The name of the MPI function associated with the operation; for example, **MPI_Irecv()**.
Source or Target	The source or target process. **Source** is the process from which the message should be received. **Target** is the process to which the message is being sent. This field shows the index of the process in the communicator, and the process name in parentheses. The display shows **ANY** if the message is being received from **MPI_ANY_SOURCE**. Dive into this field to display a Process Window.
Status	The status of the operation. Operation status can be **Pending**, **Active**, or **Complete**.
Tag	The tag value. If the message is being received with **MPI_ANY_TAG**, the display shows **ANY**.
Type	The MPI data type (IBM MPI only). The MPI data type associated with the operation; for example, **MPI_INT()**.
User Buffer, System Buffer, or Buffer	The address of the buffer. Dive into this field to view a Variable Window displaying the buffer contents.

File Menu Commands

The commands on the **File** Menu are:

- **File > Close Similar** on page 178
- **File > Close** on page 178
- **File > Exit** on page 178

File > Close Similar

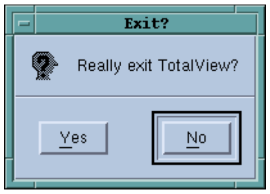
Closes this window and closes windows whose contents are similar to this window.

File > Close

Closes this window. No other windows are affected by this command.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.)



Really exit TotalView?

Figure 96: File > Exit Command

As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 179
- **Edit > Cut** on page 179

- Edit > **Copy** on page 179
- Edit > **Paste** on page 179
- Edit > **Delete** on page 179
- Edit > **Find** on page 180
- Edit > **Find Again** on page 181

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After you make the editing change, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

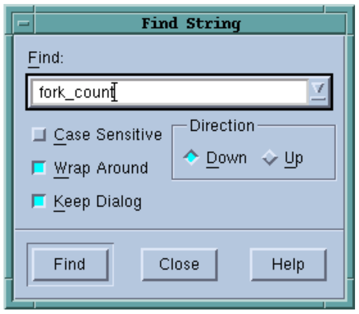
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 97: Edit > Find Dialog Box



The controls in this dialog box are:

Find	Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered.
Case Sensitive	If selected, TotalView only locates text having the same capitalization as the text entered in the **Find** field.
Wrap On Search	If selected, TotalView will continue the search from either the beginning (if **Down** is also selected) or the end (if **Up** is also selected.) For example, you search for "foo" and the **Down** button is selected. If it isn't found 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 had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the **Find** button. If you select this option, you will need to select the **Close** button to dismiss this dialog box.
Direction	Sets the direction in which TotalView searches. **Up** means "search from the current position to the beginning of the file." **Down** means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the **Find** box.
Close	Closes the **Find** dialog box.

After you find a string, you can locate another instance of the string by using the **Edit** > **Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the **View** Menu are:

- **View > Dive** on page 181
- **View > Dive in New Window** on page 181

View > Dive

Tells TotalView to "dive" into the selected item. In all cases, "dive" means that TotalView will show more information. For example:

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

In both cases, if a Process or Variable Window already exists, TotalView brings that window to the top of the display.

View > Dive in New Window

Tells TotalView to open a "dive" into the selected item. In all cases, "dive" means that TotalView will show more information.

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

Even if a Process or Variable Window already exists for a process or variable, TotalView always creates a new window for this information.

Window Menu Commands

The commands on the **Window** pulldown are:

- **Window > Update** on page 182

- **Window > Update All** on page 182
- **Window > Memorize** on page 182
- **Window > Memorize all** on page 182
- **Window > Root** on page 182

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.

TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference***'s Options Page.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.

TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

chapter 8

PVM Tasks Window

Selecting the **Tools > PVM Tasks** command within the Root Window tells TotalView to display a window that contains current information about PVM tasks and hosts. TotalView automatically updates this information as it receives events from PVM.

This window contains two areas. The top area lists the tasks and the bottom area lists the hosts. The top task area is further subdivided to define control groups. The information in the top task area is:

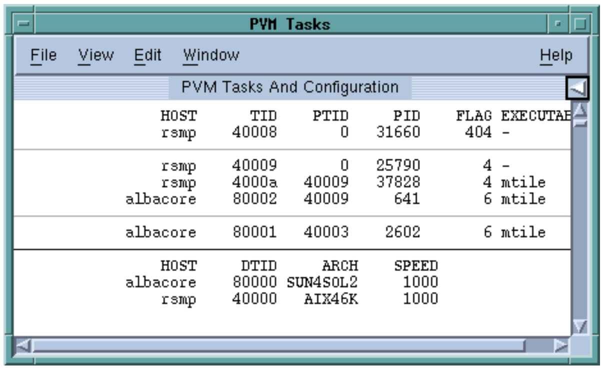
HOST	The name of the host upon which a task is executing.
TID	The parent process's task ID.
PTID	The UNIX process ID.
FLAG	The PVM message tag.
EXECUTABLE	The name of the executable file.

The information in the bottom host area is:

HOST	The name of a host upon which a task is executing.
DTID	The daemon's task ID.
ARCH	The architecture of the computer upon which the task is executing.
SPEED	Your speed setting.

You can attach to a PVM or DPVM task if:

- The machine architecture on which the task is running is the same as the machine architecture on which TotalView is running.
- The task must be created. (This is indicated when flag 4 is set in the PVM Tasks Window.)
- The task must not be a PVM tasker. If flag 400 is clear in the PVM Tasks Window, the process is a tasker.
- The executable name must be known. If the executable name is listed as a dash (–), TotalView cannot determine the name of the executable. (This can occur if a task was not created using the **pvm_spawn()** call.)



```
    HOST      TID     PTID     PID     FLAG EXECUTAB
    rsmp    40008        0   31660      404 -

    rsmp    40009        0   25790        4 -
    rsmp    4000a    40009   37828        4 mtile
albacore    80002    40009     641        6 mtile

albacore    80001    40003    2602        6 mtile

    HOST     DTID     ARCH   SPEED
albacore    80000 SUN4SOL2    1000
    rsmp    40000   AIX46K    1000
```

Figure 98: PVM Window

If the task to which you attached has related tasks that can also be debugged, TotalView asks if you want to attach to these related tasks. If you answer **Yes**, TotalView attaches to them. If you answer **No**, it only attaches to the task you dove on.

After attaching to a task, TotalView looks for attached tasks that are related to this task; if there are related tasks, TotalView places them in the same control group. If TotalView is already attached to a task you dove on, it simply opens and raises the task's Process Window.

File Menu Commands

The commands on the **File** pulldown are:

- **File > Close** on page 184
- **File > Exit** on page 184

File > Close

Closes this window. No other windows are affected by this command.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.)

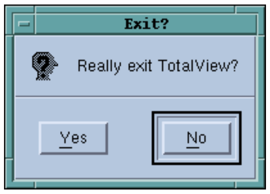


Figure 99: File > Exit Command

As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 185
- **Edit > Cut** on page 185
- **Edit > Copy** on page 185
- **Edit > Paste** on page 186
- **Edit > Delete** on page 186
- **Edit > Find** on page 186
- **Edit > Find Again** on page 187

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After you make the editing change, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

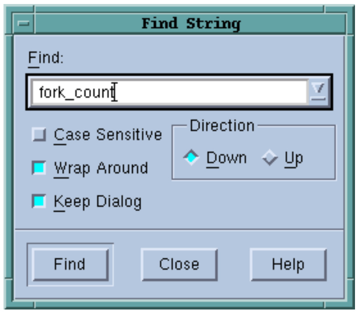
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 100: Edit > Find Dialog Box



Find: fork_count

The controls in this dialog box are:

Find	Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered.
Case Sensitive	If selected, TotalView only locates text having the same capitalization as the text entered in the **Find** field.
Wrap On Search	If selected, TotalView will continue the search from either the beginning (if **Down** is also selected) or the end (if **Up** is also selected.) For example, you search for "foo" and the **Down** button is selected. If it isn't found 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 had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the **Find** button. If you select this option, you will need to select the **Close** button to dismiss this dialog box.
Direction	Sets the direction in which TotalView searches. **Up** means "search from the current position to the beginning of the file." **Down** means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the **Find** box.
Close	Closes the **Find** dialog box.

After you find a string, you can locate another instance of the string by using the **Edit** > **Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the **View** pulldown are:

- **View > Dive** on page 187
- **View > Dive in New Window** on page 187

View > Dive

Tells TotalView to "dive" into the selected item. In all cases, "dive" means that TotalView will show more information. For example:

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

In both cases, if a Process or Variable Window already exists, TotalView brings that window to the top of the display.

View > Dive in New Window

Tells TotalView to open a "dive" into the selected item. In all cases, "dive" means that TotalView will show more information.

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

Even if a Process or Variable Window already exists for a process or variable, TotalView always creates a new window for this information.

Window Menu Commands

The commands on the Window pulldown are:

- **Window > Update** on page 188
- **Window > Update All** on page 188
- **Window > Memorize** on page 188
- **Window > Memorize all** on page 188
- **Window > Root** on page 189

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.

TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference***'s Options Page.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.

TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

chapter 9

Thread Objects Window

This help file contains information for the Thread Objects Window. As the information TotalView displays is specific to your operating system, choose the one for which you want information:

- HP *Tru64* UNIX
- **IBM AIX** on page 194

HP Tru64 UNIX

After you select the **Tools > Thread Object** command, TotalView displays a window containing the following two tabs:

- *Mutexes Page*
- **Condition Variables** on page 193

Mutexes Page

Displays a list of all of a process's mutexes. A *mutex* is a mutual exclusion object that allows multiple threads to synchronize access to shared resources. A mutex has two states: *locked* and *unlocked*. Once a mutex is locked by a thread, other threads attempting to lock it will block. Only after a locking thread unlocks (releases) the mutex can one of the blocked threads acquire (lock) the mutex and proceed.

For each mutex, TotalView displays the following information:

ID	The sequence number that the thread package assigns to a mutex. Diving into this field opens a Variable Window containing a view of the mutex's data.

Type	The mutex type. These types are set using the **pthread_mutexattr_settype()** call on the attribute object before the mutex is initialized.
	This is a mutex type number and a single-character abbreviation of the type name. (While your system may have other types available, TotalView only shows these three types.)
N	A normal mutex.
R	A recursive mutex.
E	An error-check mutex. Error-check mutexes contain additional information for use in debugging, such as the thread ID of the locking thread. During program development, you should use error-check mutexes in place of normal mutexes, and only switch to the simpler version when performance becomes an issue.
Flags	This column contains hex strings that describe the current mutex flags and a one-character abbreviation for some flags:
	0x8 (M): Metered. The mutex contains metering information.
	0x4 (W): Waiters. One or more threads are waiting for this mutex. By default, waiting threads are shown in red. Their color is the same as the thread's error state flag color.
	0x2 (P): Locked. The mutex is locked. By default, locked mutexes are shown in blue; their color is the same as the thread's stopped state flag color.
	0x1 (N): Name. This mutex has a name.
	While your system may use additional flag bits, TotalView only shows names for these flags.
Owner	If the mutex is locked, this field displays the locking thread's system thread ID (TID). This TID is only available for error-check mutexes.
	Diving or selecting on this number tells TotalView to display the locking thread's Process Window. This is the same window that TotalView would display if you dive or select the thread's entry in the Root Window's Attached Page.
	If threads are waiting for this mutex, their system TIDs are shown in the owner field, with one thread ID displayed on each line. You can open a Process Window for these waiting threads by diving or clicking on its number.

If TotalView cannot obtain this information, it does not show blocked thread lines.

Address	This field contains the memory address of the mutex. You can open a Variable Window containing a view of the mutex's data by diving on this field.
Name	If the mutex has a name, it is shown here. If you are using version 4.0D or later of the operating system, the **pthread_mutex_setname_np()** routine provides the mutex's name. However, this routine is not portable.

Condition Variables

The Condition Variables Page lists all the condition variables known in this process.

For each condition variable, TotalView displays the following information:

ID	The ID is the sequence number assigned to this condition variable by the threads package. Diving into this field opens a Variable Window containing a view of the condition variable's data.
Flags	The information in this column is a hex string containing the current condition variable's flags and a one-character abbreviation for some of the flags: **0x8 (M): Metered**. This condition variable contains metering information. **0x4 (W): Waiters**. One or more threads are waiting for this condition variable. By default, this is shown in red, which is the same as the thread's error state flag color. **0x2 (P): Pending**. A wakeup is pending for this condition variable. By default, this is shown in blue; its color is the same as the thread's stopped state flag color. **0x1 (N): Name**. The condition variable has a name. While your system may use more flags, TotalView only shows these four flag names.
Waiters	If threads are waiting for this condition variable, TotalView displays their system thread IDs (TIDs), one thread for each line, on the lines following the condition variable. Diving or selecting entries in the list of waiting threads opens windows for them. *If TotalView cannot obtain this information, it does not show waiting threads.*
Mutex	This field contains the ID of the mutex that guards the condition variable. If TotalView can translate the ID into an address, diving into this field opens a Variable Window containing a view of the guard mutex's data. TotalView can only translate this ID if it has already been initialized. That can be done statically or by using an attributes object. See the **pthread_cond_init**(3) and **pthread_mutex_init**(3) **man** pages for more information.

Address	This field has the condition variable's memory address. Diving into the address field opens a Variable Window containing a view of the actual condition variable's data.
Name	If the condition variable has a name, it is shown here. If you are using version 4.0D or later of the operating system, the **pthread_mutex_setname_np()** routine provides the condition variable's name. This routine is not portable.

IBM AIX

After you select the **Tools > Thread Object** command, TotalView displays a window containing the following four pages:

- **Mutexes Page** on page 194
- **Condition Variables Page** on page 196
- **R/W Locks Page** on page 196
- **Data Keys Page** on page 198

You must set up to six variables when debugging threaded applications. Here's what you would do in the C shell:

```
setenv AIXTHREAD_MNRATIO       "1:1"
setenv AIXTHREAD_SLPRATIO      "1:1"
setenv AIXTHREAD_SCOPE         "S"
setenv AIXTHREAD_COND_DEBUG    "ON"
setenv AIXTHREAD_MUTEX_DEBUG   "ON"
setenv AIXTHREAD_RWLOCK_DEBUG  "ON"
```

The first three variables must be set. Depending upon what you need to examine, you will also need to set one or more of the "DEBUG" variables.

Do not, however, set the **AIXTHREAD_DEBUG** variable. If you have set it, you should unset it before running TotalView

Setting these variables can slow down your application's performance. None of them should be set when you are running non-debugging versions of your program.

Mutexes Page

A mutex is a mutual exclusion object that allows multiple threads to synchronize access to shared resources. A mutex has two states: *locked* and *unlocked*. Once a mutex is locked by a thread, other threads attempting to lock it will block. Only after a locking thread unlocks (releases) the mutex can one of the blocked threads acquire (lock) the mutex and proceed.

This page contains a list of all mutexes known in a process.

For each mutex, TotalView displays the following information:

ID		The sequence number assigned to a mutex by the threads package. Diving into this field opens a Variable Window containing a view of the mutex's data.
Type		The mutex type. These types are set using the **pthread_mutexattr_settype()** call on the attribute object before the mutex is initialized.
		The type is one of the following:
	Normal	A normal mutex.
	Recurs	A recursive mutex.
	ErrChk	An error-check mutex.
	NRecNP	A non-portable, non-recursive mutex.
	RcurNP	A non-portable, recursive mutex.
	FastNP	A non-portable, fast mutex.
State		The mutex lock state is displayed as follows:
	Unlocked	The mutex is unlocked.
	Locked	The mutex is locked. By default, this is shown in blue; its color is the same as the thread's stopped state flag color.
Pshared		This value indicates if the mutex can be shared by other processes.
	Private	The mutex can only be manipulated by threads in the process that initialized the mutex.
	Shared	The mutex can be manipulated by any process that has access to the mutex's memory. (Some versions of IBM's system libraries cannot provide information on shared mutexes. If this information is not available, TotalView only describes private mutexes.)
Owner		If the mutex is locked, this field displays the locking thread's system TID.
		Diving on this number tells TotalView to display the locking thread's Process Window. This is the same window that TotalView would display if you dive or select the thread's entry in the Root Window's Attached Page.
		If threads are waiting for this mutex, their system TIDs are shown beneath the owner field, with one thread ID displayed on each line. You can open a Process Window for these waiting threads by diving or selecting on its number.
		If TotalView cannot obtain this information, it does not show waiting thread system TIDs.
Address		This field contains the memory address of the mutex. You can open a Variable Window containing a view of the mutex's data by diving on this field.

Condition Variables Page

The Condition Variable Page lists all the condition variables known in this process.

For each condition variable, TotalView displays the following information:

ID	The ID is the sequence number assigned to this condition variable by the threads package. Diving into this field opens a Variable Window containing a view of the condition variable's data.
Pshared	This value indicates if the condition variable can be shared by other processes.
Private	The condition variable can only be manipulated by the process that initialized it.
Shared	The condition variable can be manipulated by any process that has access to its memory. (Some versions of IBM's system libraries cannot provide information on shared condition values to TotalView. If this information is not available, TotalView only describes private condition values.)
Waiters	If threads are waiting for this condition variable, TotalView displays their system TIDs, one thread for each line, on the lines following the condition variable. Diving or selecting entries in the list of waiting threads opens windows for them. *If TotalView cannot obtain this information, it does not show waiting threads.*
Mutex	This field contains the ID of the mutex that guards the condition variable. If TotalView can translate the ID into an address, diving into this field opens a Variable Window containing a view of the guard mutex's data. TotalView can only translate this ID if it has already been initialized. That can be done statically or by using an attributes object. See the following mutex and condition variable **man** pages for more information: **pthread_cond_init** (3), **pthread_mutex_init** (3), **pthread_cond_initializer** (3), and **pthread_mutex_initializer** (3).
Address	This field has the condition variable's memory address. Diving into the address field opens a Variable Window containing a view of the actual condition variable's data.

R/W Locks Page

A read-write lock is a mutual exclusion object that allows multiple threads to synchronize access to shared resources. A read-write lock has three states:

- Free
- Read-locked
- Write-locked

A free lock can be locked by any number of readers or by one writer. Once a read-write lock is locked by a thread for one kind of access, other threads attempting to lock it for other kinds of access will block. When locking threads unlock (release) the read-write lock, blocked threads can acquire (lock) it and proceed.

This page lists all read-write locks known in this process.

For each lock, TotalView displays the following information:

ID	This field contains the sequence number assigned to this read-write lock by the threads package. Diving into this field opens a window containing the read-write lock data.
State	This field displays the read-write lock state as follows:
Free	Unlocked.
Read	Locked for reading. By default, this is shown in blue; its color is the same as the thread's stopped state flag color.
Write	Locked for writing. By default, this is shown in blue; its color is the same as the thread's stopped state flag color.
Pshared	This value indicates if the read-write lock can be shared by other processes.
Private	The read-write lock can only be manipulated by the process that initialized it.
Shared	The read-write lock can be manipulated by any process that has access to its memory. (Some versions of IBM's system libraries cannot provide information on shared read-write locks to TotalView. If this information is not available, TotalView only describes private read-write locks.)
Owner	If the read-write lock is locked, this field displays the system TID of a locking thread. Diving or selecting on this number tells TotalView to display the Process Window for that thread. TotalView displays the same window if you dive or select the thread's entry in the Root Window's Attached Page. If threads are waiting for this read-write lock, their system TIDs are shown beneath the system TID in this field, with one thread ID being displayed for each line in the window. That is, threads that are waiting to read and threads waiting to write are grouped together. You can open a Process Window for a waiting thread by diving or selecting its number.

If y cannot obtain this information, it does not show blocked thread lines.

Some versions of IBM*'s system libraries cannot provide the correct owner* TID *for read-write locks locked for reading. In these cases, the owner* TID *can only be trusted when the lock is in its write state.*

Address	The memory address of the read-write lock. You can open a window displaying the read-write lock data by diving on this field.

Data Keys Page

A pthread-specific data key is an object that can have a pointer value of type **void** * associated with it for each pthread in a process.

This window contains a list of all keys known in this process.

TotalView displays information for each key. Many applications initially set keys to zero (the NULL pointer value) using **pthread_set_specific()**. Note that a key's information is not displayed until a thread sets a value for it, even if the value set is NULL.

ID	This field contains the sequence number assigned to this key by the threads package. Only the line for the first thread's value for a key will contain an ID; subsequent lines for the same key omit the ID as a way of visually grouping values with the same ID.
Thread	This field has the system TIDs of the threads that have a value for this key. Diving or selecting on this number tells TotalView to display the Process Window for the thread. TotalView displays the same window if you dive or select the thread's entry in the Root Window's Attached Page.
Value	This field contains the contents of the key for a pthread. Diving into this field opens a window containing a view of the actual key data.

File Menu Commands

The commands on the **File** pulldown are:

- **File > Close Similar** on page 199
- **File > Close** on page 199
- **File > Exit** on page 199

File > Close Similar

Closes this window and closes windows whose contents are similar to this window.

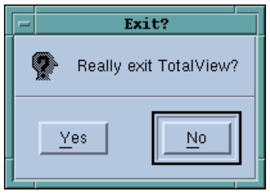
File > Close

Closes this window. No other windows are affected by this command.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.)

Figure 101: File > Exit Command



Exit?

Really exit TotalView?

As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 199
- **Edit > Cut** on page 200
- **Edit > Copy** on page 200
- **Edit > Paste** on page 200
- **Edit > Delete** on page 200
- **Edit > Find** on page 201
- **Edit > Find Again** on page 201

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After you make the editing change, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

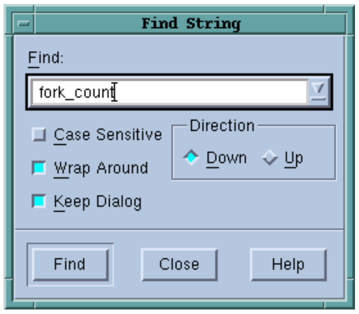
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 102: Edit > Find Dialog Box



The controls in this dialog box are:

Find	Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered.
Case Sensitive	If selected, TotalView only locates text having the same capitalization as the text entered in the **Find** field.
Wrap On Search	If selected, TotalView will continue the search from either the beginning (if **Down** is also selected) or the end (if **Up** is also selected.) For example, you search for "foo" and the **Down** button is selected. If it isn't found 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 had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the **Find** button. If you select this option, you will need to select the **Close** button to dismiss this dialog box.
Direction	Sets the direction in which TotalView searches. **Up** means "search from the current position to the beginning of the file." **Down** means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the **Find** box.
Close	Closes the **Find** dialog box.

After you find a string, you can locate another instance of the string by using the **Edit** > **Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the **View** Pulldown are:

- **View > Dive ...** on page 202
- **View > Dive ... in New Window** on page 202

View > Dive ...

Tells TotalView to "dive" into the selected item. In all cases, "dive" means that TotalView will show more information. For example:

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

In both cases, if a Process or Variable Window already exists, TotalView brings that window to the top of the display.

Note that the Data Keys Page also has a **View > Dive Thread** command. This command tells TotalView to show a Process Window containing this thread.

View > Dive ... in New Window

Tells TotalView to open a "dive" into the selected item. In all cases, "dive" means that TotalView will show more information.

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

Even if a Process or Variable Window already exists for a process or variable, TotalView always creates a new window for this information.

Note that the Data Keys Page also has a **View > Dive Thread in New Window** command. This command differs from **View > Dive Thread** in that information TotalView will not reuse an existing Process Window if the process isn't being displayed.

Window Menu Commands

The commands in the **Window** pulldown are:

- **Window > Update** on page 203
- **Window > Update All** on page 203

- **Window > Memorize** on page 203
- **Window > Memorize all** on page 203
- **Window > Root** on page 203

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.

TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference***'s Options Page.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.

TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

Expression List Window

chapter

10

Expression List Window Overview

Selecting the Process Window's **Tools > Expression List** command displays a window containing a list of variables and expressions. TotalView also displays this window after you select the **Add to Expression List** command from a context menu.l

This window can contain the values of as many variables and expressions so that you can monitor changes that occur as your program executes. That is, whenever your program halts in the thread listed in the **Threads** box, TotalView will reevaluate the value of everything in the **Expression** column.

Entering Variables and Expression into the Expression List Window

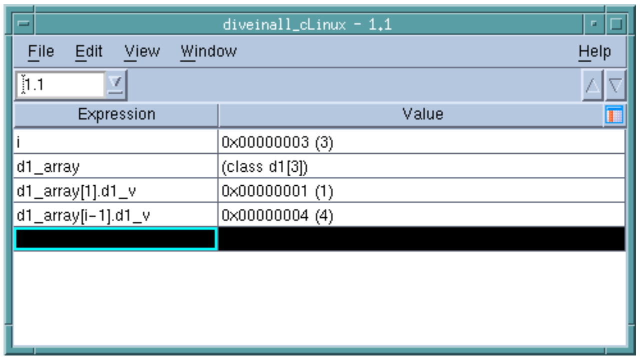
There are a number of ways to get information into the first column of this window:

- You can type information text into the bottom-most cell in the **Expression** column or edit text already entered in this column.

  The variable or expression that you enter is evaluated in the context of the current PC in the thread listed in the **Threads** box. For example, if you typed **my_var** into the window shown here, the value for it that TotalView displays is the value of **my_var** in process 1, thread 1 and its scope is defined by the current PC. This means, for example, if you have two variables named **my_var**, TotalView will not change the context when the other **my_var** is in scope.

  If you would like TotalView to change the scope in which it evaluates an expression, right-click on a row and select the **Compilation Scope >**

Figure 103: Tools > Expression List Window

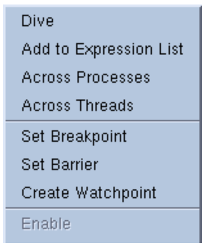


Expression	Value
i	0x00000003 (3)
d1_array	(class d1[3])
d1_array[1].d1_v	0x00000001 (1)
d1_array[i-1].d1_v	0x00000004 (4)

Floating command. For more information, see "**View > Compilation Scope > Floating**" on page 139.

- Right click on something in the Process Window's Source or Stack Frame Panes. From the displayed context menu, select **Add to Expression List**. Here's is the context window that displays within the Process Window:

Figure 104: Context Menu



- Right click on something in the Variable Window. Select **Add to Expression List** from the displayed context menu. You can also use the **View > Add to Expression List** command.

When sending something to the **Expression List** Window, the cursor and your selection matter. If you click on a variable or select a row in the Variable Window, TotalView sends the variable to the **Expression List** Window. If you instead select some text with the Source or Stack Frame Panes, TotalView sends only that text. What's the difference? In the figure at the beginning of this topic, notice that there are three different **d1_array** expressions.

- The first was added by just selecting part of what was displayed in the Source Pane.
- The second was added by selecting a row within the Variable Window.
- The third was added by duplicating the previous entry and then modifying the index.

In addition, you can select any part of your program that is an expression and place the selection in this window.

The scope that TotalView uses when looking up a variable when execution stops is the scope that existed when the variable was entered. If you want the scope to float so that a variable can be evaluated in different scopes, right-click within the variable's row and select the **Compilation Scope >**

Floating command. Selecting **Compilation Scope** > **Fixed** tells TotalView that it should only evaluate the variable in its original scope. For more information, see:

- **View** > **Compilation Scope** > **Fixed** on page 139.
- **View** > **Compilation Scope** > **Floating** on page 139.

Opening and Closing the Expression List Window

If you close the **Expression List** Window and then reopen it, TotalView remembers what you had previously entered. In other words, it doesn't delete what you enter. Instead, you must explicitly delete expressions using the **Edit** > **Delete Expression** or **Edit** > **Delete All Expressions** commands.

What Can You Enter in the Expression Column

You can type a variable or an expression within a cell in the **Expression** column or, if you select the **Add to Expression List** command, TotalView can add an entry. The expressions that enter are limited. They cannot contain function calls and they cannot create side-effects. A previous figure showed four different expressions.

i
A variable with one value. The **Value** column shows its value.

d1_array
An aggregate variable; that is, an array, a structure, a class, and so on. It's value cannot be displayed in one line. Consequently, TotalView just gives you some information about the variable. To see more information, dive on it. After diving, TotalView displays the variable in a Variable Window.

Whenever you place an aggregate variable in the **Expression** column, you will need to dive on it to get more information.

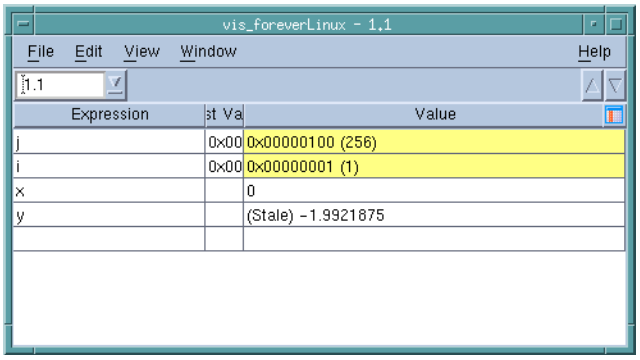
d1_array[1].d1_v
This entity is an element within an array of structures. If TotalView can resolve what you enter in the **Expression** column into a single value, it will display a value in the **Value** column. If it can't, TotalView displays information in the same way that it displays information in the **d1_array** example.

d1_array[i-1].d1_v
This differs from the previous example in that the array index is an expression. Whenever execution stops in the current Thread, TotalView reevaluates **i**, so that the element within the array that is evaluated may change.

The expressions you enter cannot include function calls.

TotalView can tell you when a variable's value changes in several ways.

- When your program stops at a breakpoint, TotalView adds a yellow highlight to the variable's value if it has changed. This is shown in Figure 105 on page 208.

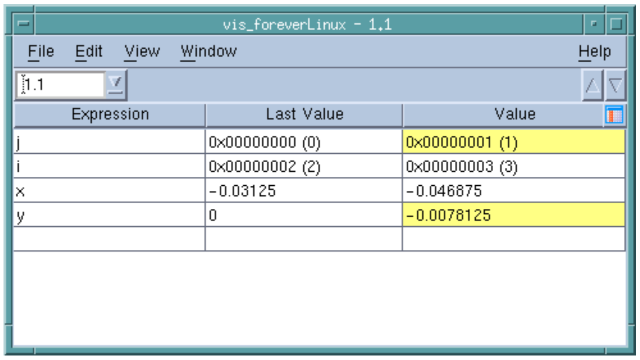


Expression	st Va	Value
j	0x00	0x00000100 (256)
i	0x00	0x00000001 (1)
x		0
y		(Stale) -1.9921875

Figure 105: Expression List Window With "Change" Highlighting

If the thread is stopped for another reason—for example, you've stepped the thread—and the value has changed, TotalView does not add yellow highlighting to the line.

- You can tell TotalView to display the **Last Value** column. Do this by selecting **Last Value** in the column menu, which is displayed after you click on the column menu ( ) icon.



Expression	Last Value	Value
j	0x00000000 (0)	0x00000001 (1)
i	0x00000002 (2)	0x00000003 (3)
x	-0.03125	-0.046875
y	0	-0.0078125

Figure 106: Expression List Window Showing Last Value Column

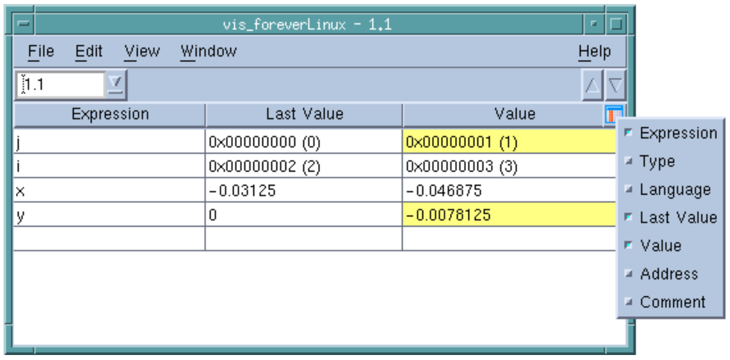
Manipulating the Expression List Window

When you first bring up the **Expression List** Window, it contains two columns. However, TotalView can display additional columns. If you right click on a column headings, TotalView displays a context menu that shows your choices.

Here are operations you can perform by directly manipulating elements within the window:

- The up and down arrows (▲▼) on the right side of the toolbar allow you to change the order in which rows are displayed. For example, clicking on the down arrow moves the currently selected row one row lower in the display.

Figure 107: Tools > Expression List Window



Expression	Last Value	Value
j	0x00000000 (0)	0x00000001 (1)
i	0x00000002 (2)	0x00000003 (3)
x	-0.03125	-0.046875
y	0	-0.0078125

- You can change an expression by clicking within its text, then typing new characters and deleting others.
- You can sort the contents of a column by clicking on the column header. For example, you could sort the **Value** column into an ascending order. After you click on the header, TotalView adds an indicator indicating that the column was sorted and the way in which it was sorted. Reclicking re-sorts the column into a descending order. Clicking a third time removes the results of sorting; that is, TotalView restores the window to what it was before you clicked on the column heading.
- Similarly, you can change a value in the **Value** column if that value is stored in memory. After entering a new value, TotalView replaces the old memory value with the one you just entered. This change cannot be undone.

Multiprocess/Multithreaded Behavior

You can change the thread in which TotalView evaluates elements within the **Expressions** column by changing the value in the **Threads** box contained within the window's toolbar.

When you send a variable to the **Expression List** Window and a window is open, TotalView checks that window's thread. If it is the same as the thread associated with the Process or Variable Window from which you are sending the expression, TotalView adds it to the bottom of the list. If it is different, TotalView still adds it to the bottom of the list. It then duplicates the window and changes the thread of the newly created window. That is, adding a variable to the list can create a new window. In this example, each has the same list of expressions. The value of these expressions may differ as they show the values of the expressions within different threads.

In all cases, the list of expressions will always be the same. What differs is the context in which TotalView evaluates the window's expressions.

Similarly, if TotalView is displaying two or more **Expression List** Windows and you send from yet another process and thread combination, TotalView adds the variable to all of them, duplicates one of them, and then changes the duplicated window to this new combination.

File Menu Commands

The commands on the **File** pulldown are:

- **File > Preferences** on page 210
- **File > Save Pane** on page 210
- **File > Close Similar** on page 211
- **File > Close** on page 211
- **File > Exit** on page 211

File > Preferences

Use this dialog box to set preferences for how TotalView will behave situations, as well as define some general characteristics. For more information, see "**File > Preferences**" on page 13, which is within the Root Windows help.

File > Save Pane

Use this dialog box to write the contents of the selected page, pane, or window.

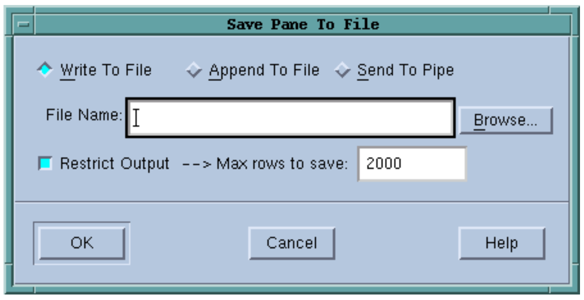


Figure 108: File > Save Pane Dialog Box

Write to File Tells TotalView to write information to a file. You can either enter the name of the file in the **File Name** field or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView overwrites it. If the file does not exist, TotalView creates the file before writing this information.

Append To File Tells TotalView to add information to a file. You can either enter the name of the file in the **File Name** edit box or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView adds this information to the end of the file. If the file does not exist, TotalView creates the file before writing this information.

Send To Pipe Sends the data to the program or script named in the **File Name** field.

Restrict Output --> Max rows to save
If checked, TotalView limits how much information it should send. If the default value of 2000 rows is not what you want, you can specify how many rows TotalView should write.

File > Close Similar

Closes this window and closes windows whose contents are similar to this window.

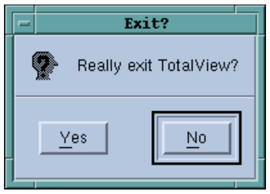
File > Close

Closes this window. No other windows are affected by this command.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.)

Figure 109: File > Exit Command



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 212
- **Edit > Reset Default** on page 212
- **Edit > Cut** on page 212
- **Edit > Copy** on page 212

- Edit > **Paste** on page 212
- Edit > **Delete** on page 213
- Edit > **Delete Expression** on page 213
- Edit > **Delete All Expressions** on page 213
- Edit > **Find** on page 213
- Edit > **Find Again** on page 214

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After you make the editing change, you cannot undo your edit.

Edit > Reset Default

If you have made changes within the **Expression** or **Type** field, selecting this command removes all changes you had made so that what is displayed is what was entered or existed originally in window.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Delete Expression

Removes the currently selected row from the list of displayed. You cannot undo this operation.

Edit > Delete All Expressions

Removes all rows from the Expression List window. You cannot undo this operation.

Edit > Duplicate Expression

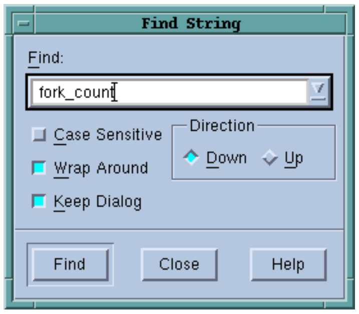
Makes a copy of the currently selected row and pastes it into the bottom row of the Expression List Window. TotalView duplicates the entire row.

This is useful when you want to track a group of similarly named variables at the same time. For example, you might duplicate my_array_var[random_index] so that TotalView is also tracking my_array_var[random_index+10].

Edit > Find

Use this command to search for text within a page or a pane.

Figure 110: Edit > Find Dialog Box



The controls in this dialog box are:

Find	Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered.
Case Sensitive	If selected, TotalView only locates text having the same capitalization as the text entered in the **Find** field.
Wrap On Search	If selected, TotalView will continue the search from either the beginning (if **Down** is also selected) or the end (if **Up** is also selected.) For example, you search for "foo" and the **Down** button is selected. If it isn't found 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 had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the **Find** button. If you select this option, you will need to select the **Close** button to dismiss this dialog box.
Direction	Sets the direction in which TotalView searches. **Up** means "search from the current position to the beginning of the file." **Down** means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the **Find** box.
Close	Closes the **Find** dialog box.

After you find a string, you can locate another instance of the string by using the **Edit** > **Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The command on the **View** pulldown is:

- **View** > **Dive** on page 214

View > Dive

Tells TotalView to open a Variable Window containing the variable or expression contained with the **Value** column of the selected row.

Window Menu Commands

The commands on the Window pulldown are:

- **Window > Update** on page 215
- **Window > Update All** on page 215
- **Window > Duplicate** on page 215
- **Window > Memorize** on page 215
- **Window > Memorize all** on page 215
- **Window > Root** on page 216

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Duplicate

Tells TotalView to create an identical copy of this Variable Window.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.

TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference***'s Options Page.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.

TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

chapter 11

Other Topics

This chapter describes the following commands and topics:

- **Ambiguous Line Dialog Box** on page 217
- **Ambiguous Function Dialog Box** on page 218
- **Installing tvheap_mr.a** on page 223
- **Setting Search Paths Using TotalView Variables** on page 225

Other Dialog Boxes

Ambiguous Line Dialog Box

TotalView displays this dialog box when it cannot locate the function you've named or dove upon. You'll see this dialog box when:

- You've misspelled a function's name.
- There is more than one symbol that has the same name. For example, you used the **Action Point > At Location** command and have named a function that exists in more than one static scope.

To continue your command's action, select the line containing which function or instance you want, and then hit **OK**.

If you type a letter, TotalView scrolls the list to that letter.

If there is more than one function with the same name and you can't tell which one you want, check the **Show full path names** box. TotalView will then show additional information about each function.

Ambiguous Function Dialog Box

TotalView displays this dialog box when it cannot locate the function you've named or dove upon. You'll see this dialog box when:

- You've misspelled a function's name.
- There is more than one symbol that has the same name. For example, you used the **Action Point > At Location** command and have named a function that exists in more than one static scope.

To continue your command's action, select the line containing which function or instance you want, and then hit **OK**.

If you type a letter, TotalView scrolls the list to that letter.

If there is more than one function with the same name and you can't tell which one you want, check the **Show full path names** box. TotalView will then show additional information about each function.

Process Load Libraries

This question gives you the opportunity to stop the loading of libraries so that you can set breakpoints.

Memory Debugging in Parallel Environments

This topic describes using MemoryScape within various environments. The sections within this topic are:

- MPICH
- IBM PE
- SGI MPI
- RMS MPI

MPICH

Here's how to use MemoryScape when launching MPICH MPI codes from the command line.

1. You must link your parallel application with MemoryScape's agent as described in "*Linking Your Application With the Agent*" on page 220. On most Linux x86 systems, you'll type:
   ```
   mpicc -g test.o -o test -Lpath -ltvheap -Wl,-rpath,path
   ```
2. Start TotalView using the –**tv** command-line option to the **mpirun** script in the usual way. For example:
   ```
   mpirun -tv mpirun-args test args
   ```
   TotalView will start up on the rank 0 process.

3 Because you will have linked in MemoryScape's agent, memory debugging is automatically selected in your rank 0 process.

4 If you need to modify the memory debugging options, you should do it now in MemoryScape.

5 Run the rank 0 process.

6 Select **Debug** > **Open MemoryScape** from the menu to open MemoryScape to examine your memory usage.

IBM PE

Here's how to use the MemoryScape with IBM PE MPI codes.There are two alternatives. The first is to place the following **proc** within your **.tvdrc** file:

```
# Automatically enable memory error notifications
# (without enabling memory debugging) for poe programs.
proc enable_mem {loaded_id} {
   set mem_prog poe
   set executable_name [TV::image get $loaded_id name]
   set file_component [file tail $executable_name]

   if {[string compare $file_component $mem_prog] == 0} {
      puts "Enabling Memory Debugger for $file_component"
      dheap -notify
   }
}

# Append this proc to TotalView's image load
# callbacks so that it runs this macro automatically
dlappend TV::image_load_callbacks enable_mem
```

Here's the second method:

1 You must prepare your parallel application to use MemoryScape's agent as in "*Linking Your Application With the Agent*" on page 220 and in "*Installing tvheap_mr.a*" on page 223. Here is an example that usually works:

```
mpcc_r -g test.o -o test -Lpath_mr -Lpath \
      path/aix_malloctype.o
```

"*Installing tvheap_mr.a*" on page 223 contains additional information.

2 Start TotalView on **poe** as usual:

```
totalview poe -a test args
```

Because **tvheap_mr.a** *is not in poe's* LIBPATH, *enabling memory debugging upon the* **poe** *process will cause problems because* **poe** *will not be able to locate the* **tvheap_mr.a** *malloc replacement library.*

3 If you want TotalView to notify you when a heap error occurs in your application (and you probably do):

➤ Select the **Debug**> **Stop on Memory Error** command from the Process Window showing **poe**. This command turns on notification in the **poe** process. The MPI processes to which TotalView will attach inherit notification.

4 Run the **poe** process.

5 Select **Debug** > **Open MemoryScape** from the menu to open MemoryScape to examine your memory usage.

SGI MPI

There are two ways to use MemoryScape on SGI MPI code. In most cases, all you need do is select the **Debug** > **Enable Memory Debugging** command upon the **mpirun** process. Once in a while, this may cause a problem. If it does, here's what you should do:

1 Link your parallel application with MemoryScape's agent as described in the "*Linking Your Application With the Agent*" on page 221. Roughly:

```
cc -n32 -g test.o -Lpath -ltvheap -rpath path \
      -lmpi -o test
```

2 Start TotalView on **mpirun**. For example:

```
totalview mpirun -a mpirun-args test args
```

3 If you need to modify the memory debugging options, you should do it now in MemoryScape.

4 Run the **mpirun** process.

5 Select **Debug** > **Open MemoryScape** from the menu to open MemoryScape to examine your memory usage.

RMS MPI

Here's how to use MemoryScape with Quadrics RMS MPI codes.

1 There is no need to link the application with MemoryScape's agent because **prun** propagates environment variables to the rank processes. However, if you'd like to link the application with MemoryScape's agent, you can.

2 Start TotalView on **prun**. For example:

```
totalview prun -a prun-args test args
```

3 Enable memory debugging using the **Debug** > **Enable Memory Debugging** command from the Process Window showing **prun**. If you had linked in the agent, **Enable memory debugging** is automatically selected.

4 If you want TotalView to notify you when a heap error occurs in your application (and you probably do), select the **Debug** >**Stop on Memory Errors** command from the Process Window. It may be selected already.

5 Run the **prun** process.

6 Select **Debug** > **Open MemoryScape** from the menu to open MemoryScape to examine your memory usage.

Linking Your Application With the Agent

MemoryScape puts its heap agent between your program and its heap library. This allows the agent to intercept the calls that your program makes to this library. After it intercepts the call, it checks the call for errors and then sends it on to the library so that it can be processed. The MemoryScape agent does not replace standard memory functions; it just monitors what they do. For more information, see "*Behind the Scenes*" on page 6.

In some situations, you need to explicitly link the MemoryScape agent directly to your program, such as when attaching to a process or debugging a core file. For example, if you are debugging an MPI program and you name the starter program on the command line, your starter program might not propagate environment variables. (If you start your MPI program by entering information in the **File > New Program** and **Process > Startup Parameters** dialog boxes, TotalView will propagate this information for you.)

On AIX, Blue Gene and Cray computers, you must always link your program so that **malloc()** *can find the heap replacement and agent. In addition, you must set your* LIBPATH *environment variable to include the TotalView tvheap library. If it isn't, your program might not load. On these computers, you must use the* **–L** *options listed in the following table. For special considerations on AIX, see "Installing tvheap_mr.a" on page 223.*

On Apple Mac OS X, you cannot link the agent into your program.

The following table lists additional linker command-line options that you must use when you link your program:

Platform	Compiler	Binary Interface	Additional linker options
Cray XT3	–	64	–L*path* –lgmalloc –ltvheap_xt3
HP Tru64 Alpha (version 5)	Compaq/KCC	64	–L*path* –ltvheap –rpath *path*
	GCC	64	–L*path* –ltvheap –Wl,–rpath,*path*
IBM RS/6000 (all)	IBM/GCC	32/64	–L*path_mr* –L*path*
	KCC	32	–L*path_mr* –L*path* --static_libKCC
		64	–L*path_mr* –L*path*
AIX 5	IBM/GCC/KCC	32	–L*path_mr* –L*path* *path*/aix_malloctype.o
		64	–L*path_mr* –L*path* *path*/aix_malloctype64_5.o
IBM Blue Gene/L	–	64	–L*path* –ltvheap_bluegene
IBM Blue Gene/P	–	64	–L*path* –ltvheap_bluegene_p
Linux x86	GCC/Intel/PGI	32	–L*path* –ltvheap –Wl,–rpath,*path*
	KCC	32	–L*path* –ltvheap –rpath *path*

Platform	Compiler	Binary Interface	Additional linker options
Linux x86-64	GCC/PGI	32	–L*path* –ltvheap –Wl,–rpath,*path*
		64	–L*path* –ltvheap_64 –Wl,–rpath,*path*
Linux IA64	GCC/Intel	64	–L*path* –ltvheap –Wl,–rpath,*path*
Sun	Sun/KCC/ Apogee	32	–L*path* –ltvheap –R *path*
	Sun/KCC	64	–L*path* –ltvheap_64 –R *path*
	GCC	32	–L*path* –ltvheap –Wl,–R,*path*
		64	–L*path* –ltvheap_64 –Wl,–R,*path*

The following list describes the options in this table:

path The absolute path to the agent in the TotalView installation hierarchy. More precisely, this directory is:

installdir/toolworks/totalview.*version*/*platform*/lib

installdir The installation base directory name.

version The TotalView version number.

platform The platform tag.

path_mr The absolute path of the heap replacement library. This value is determined by the person who installs the TotalView malloc replacement library.

Since it is easy to misinterpret the path specifications, you may want to see what value TotalView uses when it sets a path. Here's the procedure:

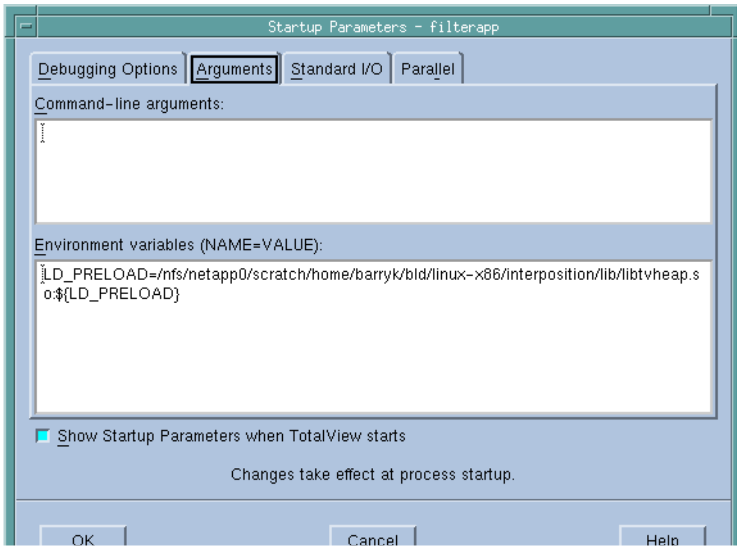
1 Start TotalView. You'll need to start TotalView on a program, but it need not be the programming you are debugging.
2 Enable tmemory debugging by selecting the **Debug > Enable Memory Debugging** command.
3 Select the **Process > Startup Parameters** command and then select the **Environment** Page. Type a value that is the same as or similar to the one in the following figure:

Installing **tvheap_mr.a** *on* AIX *requires that the system have the* **bos.adt.syscalls** *System Calls Application Toolkit page installed.*

Installing tvheap_mr.a

You must install the **tvheap_mr.a** library on each node upon which you will be running the Memory Debugger's agent. One method is to place a symbolic link in **/usr/lib** to the **tvheap_mr.a** library. If you do this, you do not need to add special –**L** command line options to your build. In addition, In addition, there will not be any special requirements when using **poe**.

Figure 111: Groups > Startup: Environment Page



Command-line arguments:

Environment variables (NAME=VALUE):

```
LD_PRELOAD=/nfs/netapp0/scratch/home/barryk/bld/linux-x86/interposition/lib/libtvheap.s
o:${LD_PRELOAD}
```

☑ Show Startup Parameters when TotalView starts

Changes take effect at process startup.

The rest of this section describes who you need to do if you cannot create symbolic links. This section will emphasize what you must do on AIX.

Most of what you need to do is encapsulated within the **aix_install_-tvheap_mr.sh** script. You'll find this script in the following directory:

toolworks/totalview.version/rs6000/lib/

For example, after you become root, enter the following commands:

```
cd toolworks/totalview.6.3.0-0/rs6000/lib
mkdir /usr/local/tvheap_mr
./aix_install_tvheap_mr.sh ./tvheap_mr.tar /usr/local/tvheap_mr
```

Use **poe** to create **tvheap_mr.a** on multiple nodes.

The pathname for the **tvheap_mr.a** library must be the same on each node. That means that you cannot install this library on a shared file system. Instead, you must install it on a file system that is private to the node. For example, because **/usr/local** is usually only accessible from the node upon which it is installed, you might want to install it there.

The **tvheap_mr.a** library depends heavily on the exact version of **libc.a** that is installed on a node. If **libc.a** changes, you must recreate **tvheap_mr.a** by re-executing the **aix_install_tvheap_mr.sh** script.

LIBPATH and Linking

This section discusses compiling and linking your AIX programs. To begin with, the following command adds *path_mr* and *path* to the your program's default **LIBPATH**:

```
xlc -Lpath_mr -Lpath -o a.out foo.o
```

When **malloc()** dynamically loads **tvheap_mr.a**, it should find the library in *path_mr*. When **tvheap_mr.a** dynamically loads **tvheap.a**, it should find it in *path*.

Note that the AIX linker allows you to relink executables. This means that you can make an already complete application ready for the Memory Debugger's agent; for example:

```
cc a.out -Lpath_mr -Lpath -o a.out.new
```

Here's an example that does not link in the malloc replacement. Instead, it allows you to dynamically set **MALLOCTYPE**:

```
xlC -q32 -g \
   -L/usr/local/tvheap_mr \
   -L/home/totalview/interposition/lib prog.o -o prog
```

The next example shows how you allow your program to access the Memory Debugger's agent by linking in the **aix_malloctype.o** module:

```
xlc -q32 -g \
   -L/usr/local/tvheap_mr \
   -L/home/totalview/interposition/lib prog.o \
     /home/totalview/interposition/lib/aix_malloctype.o \
   -o prog
```

You can check that the paths made it into the executable by running the **dump** command:

```
% dump -Xany -Hv tx_memdebug_hello

   tx_memdebug_hello:

           ***Loader Section***
         Loader Header Information
  VERSION#      #SYMtableENT    #RELOCent     LENidSTR
  0x00000001    0x0000001f      0x00000040    0x000000d3

  #IMPfilID     OFFidSTR        LENstrTBL     OFFstrTBL
  0x00000005    0x00000608      0x00000080    0x000006db

           ***Import File Strings***
  INDEX  PATH              BASE          MEMBER
  0      /.../interpos/lib:/usr/.../lib:/usr/lib:/lib
  1                        libc.a            shr.o
  2                        libC.a            shr.o
  3                        libpthreads.a     shr_comm.o
  4                        libpthreads.a     shr_xpg5.o
```

Index 0 within the **Import File Strings** section shows the search path the runtime loader uses when it dynamically loads a library. Some MPI systems propagate the preload library environment to the processes they will run. Others, do not. If they do not, you will need to manually link them with the **tvheap** library.

In some circumstances, you may want to link your program instead of setting **MALLOCTYPE**. If you set the **MALLOCTYPE** environment variable for your program and it fork/execs a program that is not linked with the agent, then your program will terminate because it fails to find **malloc()**.

Setting Search Paths Using TotalView Variables

The following search path discussion has the following sections:

- "*Search Path Variables That You Can Set*" on page 225
- "*Search Path Variables That TotalView Sets*" on page 226
- "*TotalView Built-in Functions*" on page 228
- "*Using the File > Search Path Dialog Box*" on page 229

If TotalView cannot locate your program's source files using the paths named within the Search Path dialog box's **EXECUTABLE_PATH** tab or your **PATH** environment variable, you may need to take additional steps. How to do this is described in this topic and the topics linked to it.

There are seven state variables that you can set from either the CLI or from the GUI by using the **File > Search Path** command.

Search Path Variables That You Can Set

Setting Search Paths Using TotalView Variables

Search Path Variables That TotalView Sets

TotalView Builtin functions

Using the File > Search Path Dialog Box

TotalView uses the following variables when it searches for source, object, shared library, and program files:

- SOURCE_SEARCH_PATH
- OBJECT_SEARCH_PATH
- SHARED_LIBRARY_SEARCH_PATH
- EXECUTABLE_SEARCH_PATH

Each contains a list of paths, with paths separated by a colon.

Related to these variables are four additional similarly named variables. Each contains pairs of regular expressions and replacement strings—these replacements are called *mappings*—separated by colons. TotalView applies these mappings to the search paths before it looks for source, object, shared library, and program files:

- SOURCE_SEARCH_MAPPINGS
- OBJECT_SEARCH_MAPPINGS
- SHARED_LIBRARY_SEARCH_MAPPINGS
- EXECUTABLE_SEARCH_MAPPINGS

The syntax for mapping strings is:

```
+regular_exp+=+replacement+ :+regular_exp+=+replacement+
```

This example shows two pairs, each delimited by a colon (":"). Each element within a pair is delimited by any character except a colon. The first character you enter is the delimiter. This example uses a "+" as a delimiter. (Traditionally, forward slashes are used as delimiters. This is not a good idea as a forward slash is also used to separate components of a pathname. For example, **/home/my_dir** contains forward slashes.)

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

Here is an example:

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

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

TOTALVIEW_SRC

The **TOTALVIEW_SRC** variable is the directory in a TotalView installation that contains the source files shipped with TotalView.

PATH

The **PATH** variable contains all of the directories in your **PATH** variable.

EXECUTABLE_PATH

The **EXECUTABLE_PATH** variable contains directories that TotalView looks in when it searches for files. The contents of this variable is a colon-separated list of directory names. TotalView only uses this variable if you include it within one of the following variables:

- SOURCE_SEARCH_PATH
- OBJECT_SEARCH_PATH
- SHARED_LIBRARY_SEARCH_PATH
- EXECUTABLE_SEARCH_PATH

Search Path Variables That TotalView Sets

Setting Search Paths Using TotalView Variables

Search Path Variables That You Can Set

TotalView Builtin functions

Using the File > Search Path Dialog Box

The following sections discuss variables that you can include within the **SOURCE_SEARCH_PATH**, **OBJECT_SEARCH_PATH**, and **EXECUTABLE_SEARCH_PATH** variables. These variables are:

- COMPILATION_DIRECTORY_COMPONENT on page 227
- COMPILATION_WORKING_DIRECTORY on page 227

- **COMPILATION_DIRECTORY** on page 227
- **EXECUTABLE_DIRECTORY_COMPONENT** on page 227
- **EXECUTABLE_WORKING_DIRECTORY** on page 228
- **EXECUTABLE_DIRECTORY** on page 228

TotalView sets these six variables and they are read-only. That is, you cannot change their values. TotalView sets their values based on information provided by compilers or that you provide when you enter pathnames to TotalView.

COMPILATION_DIRECTORY_COMPONENT

The **COMPILATION_DIRECTORY_COMPONENT** variable contains the path component for the file that is named on the compilation command. For example, assume that you've compiled your program in the following way:

```
cc -g src/test/hello.c
```

COMPILATION_DIRECTORY_COMPONENT would be **src/test**. This variable can be empty because not all compilers add this information to the generated debug information.

COMPILATION_WORKING_DIRECTORY

The **COMPILATION_WORKING_DIRECTORY** variable contains the full pathname of the directory in which the compiler was invoked. This variable can be empty because not all compilers add this information to the generated debug information.

COMPILATION_DIRECTORY

The **COMPILATION_DIRECTORY_COMPONENT** is the result of TotalView applying the following rules to combine the **COMPILATION_DIRECTORY_COMPONENT** variable with **COMPILATION_WORKING_DIRECTORY**:

- If **COMPILATION_DIRECTORY_COMPONENT** is an absolute path, **COMPILATION_DIRECTORY** is the same as **COMPILATION_DIRECTORY_COMPONENT**.
- If **COMPILATION_WORKING_DIRECTORY** ends in a '/', the **COMPILATION_DIRECTORY** is
  ${COMPILATION_WORKING_DIRECTORY}
  ${COMPILATION_DIRECTORY_COMPONENT}
- Otherwise, the **COMPILATION_DIRECTORY** is:
  ${COMPILATION_WORKING_DIRECTORY}/
  ${COMPILATION_DIRECTORY_COMPONENT}

EXECUTABLE_DIRECTORY_COMPONENT

The **EXECUTABLE_DIRECTORY_COMPONENT** contains the executable file's directory name. This directory name can either be absolute or relative.

TotalView lets you enter an executable file name in several places. For example, you can enter it on the command line, as an argument to the **dload** or **dattach** CLI commands, and in the **File > New Program** dialog box.

If the executable file name contains a slash ("/") path separator, TotalView does not search for the executable file. Instead, it uses this name, setting **EXECUTABLE_DIRECTORY_COMPONENT** to the directory portion of the provided name.

If an executable file name does not contain a path separator, TotalView sets **EXECUTABLE_DIRECTORY_COMPONENT** to the first directory named in the **EXECUTABLE_SEARCH_PATH** where it finds the executable file.

Example 1 If the executable file is **/bin/ls**, **EXECUTABLE_DIRECTORY_COMPONENT** is set to **/bin/**.

Example 2 If the executable file is **../../bin/a.out**, **EXECUTABLE_DIRECTORY_COMPONENT** is set to **../../bin**.

EXECUTABLE_WORKING_DIRECTORY

The **EXECUTABLE_WORKING_DIRECTORY** is the absolute path name of the current working directory; that is, this is the value of **pwd** of TotalView at the time the executable file is loaded.

EXECUTABLE_DIRECTORY

The value of **EXECUTABLE_DIRECTORY** is the result of TotalView applying the following rules that combine the **EXECUTABLE_DIRECTORY_COMPONENT** variable with **EXECUTABLE_WORKING_DIRECTORY**.

- If **EXECUTABLE_DIRECTORY_COMPONENT** is an absolute path, **EXECUTABLE_DIRECTORY** is the same as **EXECUTABLE_DIRECTORY_COMPONENT**.
- If **EXECUTABLE_DIRECTORY_COMPONENT** is not absolute, **EXECUTABLE_DIRECTORY** is **EXECUTABLE_WORKING_DIRECTORY** is joined with **EXECUTABLE_DIRECTORY_COMPONENT**.

The rules for entering information here are similar to those that for entering information for **COMPILATION_DIRECTORY**. Note that you cannot set these values using **dset**.

Example If **EXECUTABLE_WORKING_DIRECTORY** is **/lib** and **EXECUTABLE_DIRECTORY_COMPONENT** is **../bin**, the **EXECUTABLE_DIRECTORY** is **/lib/../bin**.

TotalView Built-in Functions

Setting Search Paths Using TotalView Variables
Search Path Variables That You Can Set
Search Path Variables That TotalView Sets
Using the File > Search Path Dialog Box

TotalView has two built-in functions that can help you set search paths. Both take a string argument and return a mapped string (MS). If the MS does not name a directory, nothing is searched.

$tree(*string*)	If the MS names a directory, TotalView search all the directories contained in the MS, including the MS itself.
$link(*string*)	If the MS names a directory and the file is not in the MS or the file is in the MS but is not a symbolic link, TotalView nothing is returned. Since you can have multiple levels of symbolic links, TotalView keeps following links until it finds the actual executable file. It will then look in this directory for your file. If the file isn't there, TotalView backs up the chain of links until either it finds the file or determines that it cannot find your file.

When you use one of these functions, you must delimit it with colons. Also, you cannot include one of these functions within the other.

You'll find an example of how these functions are used in the **Search Path** dialog box.

Using the $tree() Function

The **$tree()** function takes a colon-separated list of directories, and uses a simple tree-walking algorithm to add each directory under the top level to the search path. Previously, TotalView would look only in the specific directories named, which could be tedious to add to the search path if a large number of directories contained source files. **$tree()** does this automatically. For example:

```
$tree(/usr/project/src_A:/home/test/foo)
```

This adds all the directories and sub-directories under both **/usr/project/src_A** and **/home/test/foo** to the current search path. The $tree() list is walked every time the search path is looked up, so its use may incur a slight performance penalty if this involves a deeply nested directory structure.

The **$tree()** function cannot be the first or last entry in the Source tab, but you can have more than one **$tree()** entry.

Using the File > Search Path Dialog Box

Setting Search Paths Using TotalView Variables

Search Path Variables That You Can Set

Search Path Variables That TotalView Sets

TotalView Builtin Functions

The **File** > **Search Path** dialog box has the following four tabs:

- Programs: corresponds to the **EXECUTABLE_SEARCH_PATH** and **EXECUTABLE_SEARCH_MAPPINGS** variables

- Sources: corresponds to the **SOURCE_SEARCH_PATH** and **SOURCE_SEARCH_MAPPINGS** variables
- Objects: corresponds to the **OBJECT_SEARCH_PATH** and **OBJECT_SEARCH_MAPPINGS** variables
- EXECUTABLE_PATH

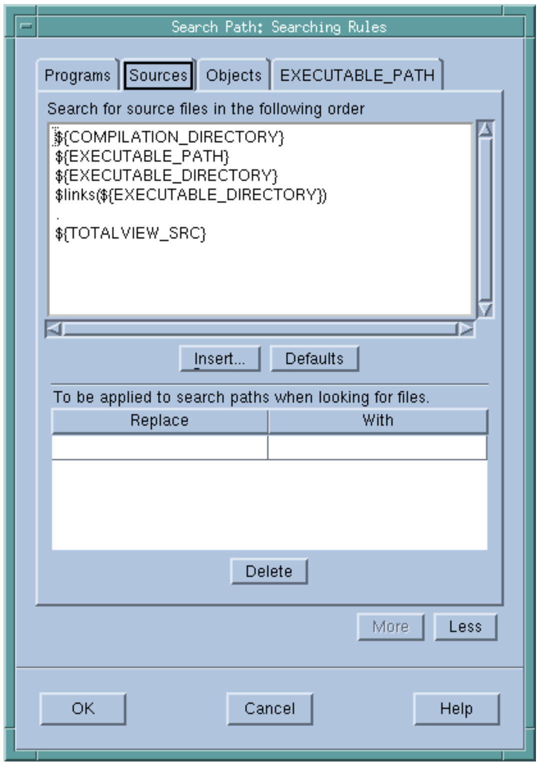
The first three let you specify the directories in which TotalView should look for your executable programs, and source and object files. The fourth tab lets you set the value of the **EXECUTABLE_PATH** variable. This information is then made available elsewhere.

That is, enter directories here and TotalView uses this information as one of the entries it uses when it creates the directories defined in other tabs.

You cannot set or view **SHARED_LIBRARY_SEARCH_PATH** *or* **SHARED_LIBRARY_SEARCH _MAPPINGS** *from the* **File > Search Path** *dialog box; you must use the command line interface.*

Here's an example of this dialog box:

Figure 112: File > Search Path Dialog Box, Sources Tab



Search for source files in the following order

```
${COMPILATION_DIRECTORY}
${EXECUTABLE_PATH}
${EXECUTABLE_DIRECTORY}
$links(${EXECUTABLE_DIRECTORY})
.
${TOTALVIEW_SRC}
```

To be applied to search paths when looking for files.

Replace	With

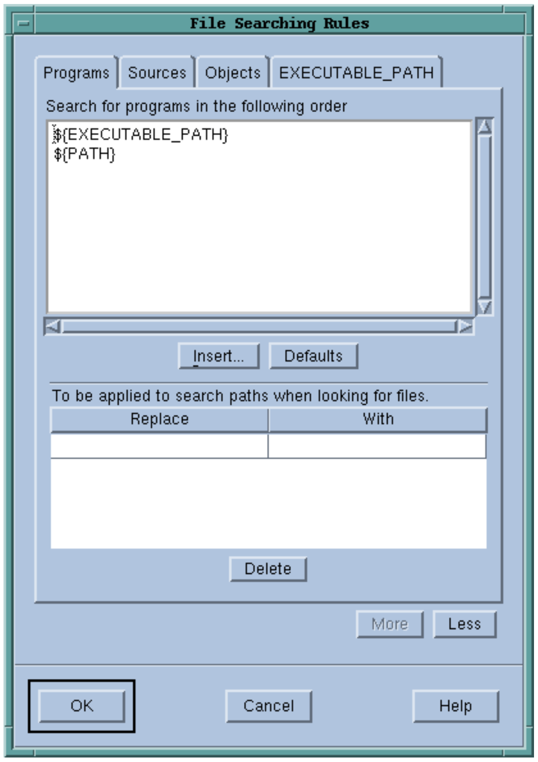
Notice how the dialog box shows the CLI variables that TotalView uses to construct a search path.

Each tab has four buttons:

- **Insert**: tells TotalView to display a dialog box that you can use to locate directories in your file system. Notice that there is no "delete" command as you can delete information from this dialog box by deleting characters.

- **Defaults**: tells TotalView to discard your changes and restore this dialog box to what it was before you began making changes.
- **More**: tells TotalView to display controls that allow you to create regular expressions for mapping paths.
- **Less**: tells TotalView to hide the controls that allow you to create regular expressions for mapping paths.

The next figureFigure 113 on page 231 shows these regular expressions controls:



Search for programs in the following order

```
${EXECUTABLE_PATH}
${PATH}
```

To be applied to search paths when looking for files.

Figure 113: File > Search Path Dialog Box, Sources Tab

Enter information in the **Replace** and **With** sections using the rules described in "*Search Path Variables That You Can Set*" on page 225. There are two data entry differences:

- You do not separate the **Replace** and **With** components with an equals sign ("="),
- You do not have to use delimiters surrounding either the **Replace** or **With** components.

Index

Symbols
!= operator 129
$denorm intrinsic 129
$inf intrinsic 129
$nan intrinsic 129
$nanq intrinsic 129
$nans intrinsic 129
$ndenorm intrinsic 129
$newval intrinsic 146
$ninf intrinsic 129
$oldval intrinsic 146
$pdenorm intrinsic 129
$pinf intrinsic 129
$tree() function 229
$visualize intrinsic 155
%B expansion character 20
%C expansion character 20, 24, 25
%D expansion character 20, 24, 25
%E expansion character 22
%F expansion character 21, 22
%H expansion character 20, 25, 26
%K expansion character 20
%L expansion character 20, 25, 26
%N expansion character 22, 24
%P expansion character 20, 25, 26
%R expansion character 20, 25
%S expansion character 20, 22, 25, 26
%t expansion character 25
%t1 file replacement character 21
%t2 file replacement character 21
%V expansion character 20, 25, 26
.tvd file 13
< icon 61, 62, 137
< operator 129
< undive icon 63
== operator 129
> indicator 61
> operator 129
|< icon 137

A
action point
  defaults 16
Action Point > At Location command 91, 92
Action Point > Create Watchpoint command 93
Action Point > Delete All command 99
Action Point > Delete command 93
Action Point > Disabled command 93
Action Point > Enabled command 93
Action Point > Load All command 18, 99, 100
Action Point > Properties command 50, 93
Action Point > Save All command 100
Action Point > Set Barrier command 91
Action Point > Set Breakpoint command 91
Action Point > Suppress All command 52, 99
action point identifier 52
Action Point menu commands
  Process window 91
action points 49
  enabling 95
  loading 18
  planting in share groups 18
  saving 18
  what else is stopped 16
Action Points > Enable command 52
Action Points page 16
Action Points tab 52
  diving 61
active threads 49
Add new host 5
address as <void> 64
Address command 66
address locking 138
Adjacents 149
agent linking 220
aix_install_ tvheap_mr.sh script 223
All Methods in Class action point setting 92
All Virtual Functions and Override action point setting 92
allocation backtrace 107
allocation point 107
Ambiguous Function Dialog Box 92
ambiguous function dialog box 218
ambiguous line dialog box 217
appending to a file 38, 57, 132, 210
Arguments page 8, 81
Arguments tab 82
arguments, special characters in 9, 82
Array of Ranks option 151
arrays
  filtering 125
arrays subscripts 128
arrays, diving into 130
Ask to stop when loading dynamic libraries preference 27
Assembler > Symbolically 65
Assembler command 65
At Location command 91, 92
Attach Subset command 71, 150
Attach to a new process area 5
Attach to all preference 30
Attach to none preference 30
Attached Page 1
Attached page 1
attaching to base process 42, 120
attaching to parallel option 42, 120
attaching to processes 6, 71, 150
attaching to relatives 5, 6
Auto Visualize command 156
autolaunch 18, 23
averaging data points 159
averaging surface display 159

axis, transposing 158

B

B state indicator 3, 51
backtrace
  allocation 107
  current 107
  deallocation 107
Backtrace View 148
backtraces 106
barrier point properties 94
barrier points 96
  setting 91
barrier preferences 17
barriers
  conditional 99
  releasing threads 96
  what else is stopped 17
Baseline check box 103
Baseline command 102
baselines 102
Block Details tab 107
Block Flags area 106, 148
block information 106
block status
  Memory block properties
    status 105, 148
Block Status command 142
Break lines at newlines command, Variable Window 140
Breakpoint command 91
breakpoint state indicator 3, 51
breakpoints
  conditional 99
  countdown 98
  deleting 99
  disabling 99
  loading 99, 100
  opening Process window 18
  removing 99
  saving 100
  sets 92
  setting 91
  setting at a location 91
  suppressing 99
breakpoints, pending 93
Bulk Launch page 23

C

C++ exceptions 15
Call Graph command 109
call stack 49, 69
–callback expansion option 20, 26
Case Sensitive option 40, 60, 134, 164, 172, 180, 186, 201, 214
casting 127
Checkpoint command 117
checkpointing 42, 120
  Restart Checkpoint command 120
  socket problem 118
Checksum 149
Circle Constraints area 115
clearing signals 90
clipboard 39, 58, 59, 134, 163, 171, 179, 185, 200, 212
Close command
  Fortran Modules window 162
  Globals window 169
  Message Queue window 178
  Process window 57
  PVM Tasks window 184, 211
  Thread Objects window 199
  Variable window 132
  Visualizer window 159
Close Relatives command
  Process window 57
Close Similar command
  Fortran Modules window 162
  Globals window 169
  Message Queue window 178
  Thread Objects window 199
  Variable window 132
closing windows 57
code fragments, evaluating 108
Collapse All command
  Root window 41, 137
Comm_rank 176
Comm_size 176
Command Line command
  Process window 122
  Root window 45
command-line editing preference 13
commands
  Tools > Attach Subset (Array of Ranks) 73, 151
communicator name 176
communicators 72, 151
Compilation Scope > Fixed command 139
Compilation Scope > Floating command 139
Compiled in Scope 128
compilers
  mpcc_r 175
  mpxlf_r 175
  mpxlf90_r 175
condition variables 193, 196
  address of 194, 196
  flags 193
  mutex guard 193, 196
  name of 194
  process shared value 196
  sequence number 193, 196
  waiters value 193, 196
Condition Variables page
  Compaq 193
  IBM 196
conditional barriers 99
conditional breakpoints 99
conditional watchpoints 144, 145
connection timeout preference 25
Continuation Signal command 90
control group
  halting 73, 152
control group, stopping on error 15
Copy command
  Fortran Modules window 163
  Globals window 171
  Message Queue window 179
  Process window 59
  PVM Tasks window 185, 212
  Root window 39
  Thread Objects window 200
  Variable window 134
core file
  operning 7
countdown breakpoints 98
Create Checkpoint command 117
Create command 80
Create Watchpoint command 93
creating a process 67
creating a process window 41
creating process windows 41
creating processes 80
Ctrl+C accelerator 39, 59, 134, 163, 171, 179, 185, 200, 212
Ctrl+C to terminate TotalView 14
Ctrl+V accelerators 39, 59, 134, 163, 171, 179, 185, 200, 212
current backtrace 107
current working directory 12, 25, 53
Cut command
  Fortran Modules window 163
  Globals window 171
  Message Queue window 179
  Process window 58
  PVM Tasks window 185, 212
  Root window 39
  Thread Objects window 200
  Variable window 134
Cycle Detection tab 116

D

data keys page (IBM) 198
data type
  changing 127
data window 155, 157
dataset 155
  deleting 156
  selecting 156
Dataset command 158
Dataset window 155, 156
  menu commands 156
dbfork 29
dbfork library 6
dcheckpoint command 118
deallocation backtrace 107
deallocation point 107
debug server launch 19
Debugger Loaded Libraries command 110

Default Action Point preferences 16
default editor launch string 22
deferred reading 161
Delete All command 99
Delete command 93
  Fortran Modules window 164
  Globals window 172
  Message Queue window 179
  Process window 59
  PVM Tasks window 186, 213
  Root window 39
  Thread Objects window 200
  Variable window 134
  Visualizer window 156, 159
Denormalized Count 149
denormalized number 129
Detach command 80
Disabled command 93
disabling breakpoints 99
DISPLAY environment variable 9, 83
Display Exited Threads command
  Root window 42
Display Manager Threads command
  Root window 42
Display Managers command
  Process window 66
Dive Anew command
  Fortran Modules window 165
  Globals window 173
  Message Queue window 181
  Process window 61
  PVM Tasks window 187
  Root window 41
  Thread Objects window 202
  Variable window 136
Dive command
  Fortran Modules window 165
  Globals window 173
  Message Queue window 181
  Process window 61
  PVM Tasks window 187, 214
  Root window 41
  Thread Objects window 202
  Variable window 136
Dive in All command 136
dive stack 62, 137
Dive Thread command
  Thread Objects window 202
diving 1, 61, 137
  in Fortran Modules window 161
  into arrays 130
  into pointers 130
  into structures 130
diving on a process 2
diving on a thread 2
dlopen() 110
DMPI 175
DOT button 117
drawing options 158
drestart command 43, 120
drun command
  poe issues 85
Duplicate command 123
  Variable window 152, 215
dynamic libraries
  symbol loading 28
  symbol reading order within preference 28
Dynamic Libraries page 27
dynamic libraries preference
  stopping before executing 27
dynamic link information 57
dynamic program representation 109
dynamic symbol library loading 28

E

E state indicator 3, 51
Edit > Copy command
  Fortran Modules window 163
  Globals window 171
  Message Queue window 179
  Process window 59
  PVM Tasks window 185, 212
  Root window 39
  Thread Objects window 200
  Variable window 134
Edit > Cut command
  Fortran Modules window 163
  Globals window 171
  Message Queue window 179
  Process window 58
  PVM Tasks window 185, 212
  Root window 39
  Thread Objects window 200
  Variable window 134
Edit > Delete command
  Fortran Modules command 164
  Globals window 172
  Message Queue window 179
  Process window 59
  PVM Tasks window 186, 213
  Root window 39
  Thread Objects window 200
  Variable window 134
Edit > Find Again 40, 60, 135, 165, 173, 180, 187, 201, 214
Edit > Find Again command
  Fortran Modules window 165
  Globals window 173
  Message Queue window 181
  Process window 60
  PVM Tasks window 187, 214
  Root window 40
  Thread Objects window 201
  Variable window 135
Edit > Find command
  Fortran Modules window 164
  Globals window 172
  Message Queue window 180
  Process window 59
  PVM Tasks window 186, 213
  Root window 40
  Thread Objects window 201
  Variable window 134
Edit > Paste command
  Fortran Modules window 163
  Globals window 171
  Message Queue window 179
  Process window 59
  PVM Tasks window 186, 212
  Root window 39
  Thread Objects window 200
  Variable window 134
Edit > Reset Defaults command
  Variable window 133
Edit > Undo command
  Fortran Modules window 163
  Globals window 171
  Message Queue window 179
  Process window 58
  PVM Tasks window 185, 212
  Root window 39
  Thread Objects window 199
  Variable window 133
Edit menu commands
  Fortran Modules window 163
  Globals window 171
  Message Queue window 178
  Process window 58
  PVM Tasks window 185, 211
  Root window 38
  Thread Objects window 199
  Variable window 133
Edit Source command
  Process window 56
EDITOR environment variable 22
editor launch string 56
  default 22
Enable command for memory debugging 102
Enable command line editing preference 13
Enable debug server bulk launch preference 23
enable memory debugging 102
Enable memory debugging checkbox 5
Enable memory debugging option 103
Enable single debug server launch preference 19
Enable tooltips preference 14
Enable use of dbfork preference 29
Enable Visualizer launch preference 22
Enabled command 93
enabling action points 95
enabling watchpoints 145
Environment page 9, 82
environment variables
  inheriting 9, 82
  PATH 13, 53
Error signal option 55

error state 44, 112
error state indicator 3, 51
error, stopping control group 15
errors, raising Process window 15
Evaluate command 108
evaluating expressions 108
evaluation point properties 94
evaluation points 98
Examine Format > None command 140
Examine Format > Raw command 141
Examine Format > Structured command 140
exec() 7
EXECUTABLE_PATH tab 53
executing to a line 69
execve() 4
Exit command
    Globals window 170
    Message Queue window 178
    Process window 58
    PVM Tasks window 184, 211
    Root window 38
    Thread Objects window 199
    Variable window 132
    Visualizer window 156
Expand All command
    Root window 41
    Variable window 137
expansion characters
    %B 20
    %C 20, 24, 25
    %D 20, 24, 25
    %E 22
    %F 21, 22
    %H 20, 25, 26
    %K 20
    %L 20, 25, 26
    %N 22, 24
    %P 20, 25, 26
    %R 20, 25
    %S 20, 22, 25, 26
    %t 25
    %V 20, 25, 26
expansion options
    –callback 20
    –callback option 26
    –n 20
    –set_pw 20, 26
    –working_directory 20
expression field 126
expressions, evaluating 108

F

file
    appending to 38, 57, 132, 210
    writing data to 37, 56, 132, 210
File > Close command
    Fortran Modules window 162
    Globals window 169
    Message Queue window 178
    Process window 57
    PVM Tasks window 184, 211
    Thread Objects window 199
    Variable window 132
    Visualizer window 159
File > Close Relatives command
    Process window 57
File > Close Similar command
    Fortran Modules window 162
    Globals window 169
    Message Queue window 178
    Thread Objects window 199
    Variable window 132
File > Dataset command 158
File > Delete command
    Visualizer window 156, 159
File > Edit Source command
    Process window 56
File > Exit command
    Globals window 170
    Message Queue window 178
    Process window 58
    PVM Tasks window 184, 211
    Root window 38
    Thread Objects window 199
    Variable window 132
    Visualizer window 156
File > New Base Window command
    Visualizer window 159
File > New Program command 82
    Root window 4
File > Open Source command
    Process window 56
File > Options command 158
File > Preferences
    Action Points page 16
    Dynamic Libraries page 27
    Fonts page 31
    Launch Strings page 18
    Options page 13
    Parallel page 29
File > Preferences Bulk Server Page 21
File > Preferences command
    Process window 55, 210
    Root window 13
File > Rescan Libraries command 57
File > Save Pane command
    Process window 56
    Root window 37
    Variable window 132
File > Search Path command 5, 12, 53, 65
    Process window 53
    Root window 12
    search order 12, 53
File menu commands
    Fortran Modules window 162
    Globals window 169
    Message Queue window 178
    Process window 53
    PVM Tasks window 184, 210
    Root window 4
    Thread Objects window 198
    Variable window 131
    Visualizer window 156, 158
file path prefix doesn't match preference 27
file prototypes preference 25
files, finding 62
Filter Tab 116
filtering 128
    operators 128
filtering arrays 125
Find Again command 40, 60, 135, 165, 173, 180, 187, 201, 214
    Fortran Modules window 165
    Globals window 173
    Message Queue window 181
    Process window 60
    PVM Tasks window 187, 214
    Root window 40
    Thread Objects window 201
    Variable window 135
Find command
    Fortran Modules window 164
    Globals window 172
    Message Queue window 180
    Process window 59
    PVM Tasks window 186, 213
    Root window 40
    Thread Objects window 201
    Variable window 134
finding files 62
finding functions 62
finding variables 62
Fixed (compilation scope) command 139
Floating (compilation scope) command 139
font family 31
font size 31
Fonts page 31
Force window positions (disables window manager placement modes) preference 14
fork() 6
Fortran Modules 161
    Window > Memorize all command 166
    Window > Memorize command 166
Fortran Modules window 161
    Edit > Copy 163
    Edit > Cut 163
    Edit > Delete 164
    Edit > Find 164
    Edit > Find Again 165
    Edit > Paste 163
    Edit > Undo 163

File > Close 162
File > Close Similar 162
View > Dive 165
View > Dive Anew 165
Window > Root 167
Window > Update 166
Window > Update All 166
Function or Line action point setting 92
functions, finding 62

G

Global Arrays command 122
Globals command 109
Globals window 169
Edit > Copy 171
Edit > Cut 171
Edit > Delete 172
Edit > Find 172
Edit > Find Again 173
Edit > Paste 171
Edit > Undo 171
File > Close 169
File > Close Similar 169
File > Exit 170
View > Dive 173
View > Dive Anew 173
Window > Root 174
Window > Update 174
Window > Update All 174
Go command 67, 75, 86
Graph command 156
Graph visualization menu 156
Grid Constraints area 115
Group > Attach Subset command 30, 71, 80, 150
Group > Custom Groups command 73
Process Window 73
Group > Go command 67
Group > Halt command 68
Group > Hold command 71
Group > Kill command 74
Group > Next command 68
Group > Next Instruction command 70
Group > Out command 69
Group > Release command 71
Group > Restart command 74
Group > Run To command 69
Group > Step command 68
Group > Step Instruction command 71
Group menu commands
Process window 67
groups, stopping 16

H

H state indicator 3, 51
Halt command 68, 76, 86
Halt control group option 152
Halt execution at process exit option 102
Heap Change Summary command 103
heap debugging
IBM PE 219
LIBPATH environment variable 221
linker command-line options 221
linking 220
MPICH 218
RMS MPI 220
SGI MPI 220
tvheap_mr.a library 223
held state indicator 3, 51
Hide Backtrace Information button 105, 146
hierarchy button 130
history of source locations 62, 137
Hold command 71, 79, 89
Hold Threads 79
Hold Threads command 79
holding processes 79
holding threads, selectively 79
hostname 25, 26
hostname option 20

I

I state indicator 7
identifier, action point 52
idle state indicator 7
Ignore signal option 55
INF 129
Infinity Count 149
info state 44, 112
inheriting environment variables 9, 82
Instrumentation > Enable 103
Instrumentation > Enable command 103
Instrumentation > Heap Change Summary command 103
Process Window 103
Instrumentation > Memory Block Properties command
Process Window 105
Instrumentation > Memory Debugging command 102
Instrumentation > Memory Event Details command 107
Instrumentation > Set Heap Baseline (in Group) command 102, 103
Instrumentation > Set Heap Baseline (in Process) command 103
Instrumentation > Set Heap Baseline command 102
intrinsics
$denorm 129
$inf 129
$nan 129
$nanq 129
$nans 129
$ndenorm 129
$newval 146
$ninf 129
$oldval 146
$pdenorm 129
$pinf 129
$visualize 155
inverting axis 158

K

K state indicator 3, 51
Keep Dialog option 40, 60, 135, 164, 172, 180, 187, 201, 214
Keep Nodes As Positioned option 115
kernel state indicator 3, 51
Key List Information window 198
keys
contents of 198
sequence number 198
system TID 198
Kill command 74
killing parent process 5

L

Last Value column 131
launch command 20
launch directory 24
launch string 23
source code editor 22
visualizer 21
Launch Strings page 18, 155
launching the debug server 19
Layout area 115
Less button 127
LIBPATH and linking 223
libraries, rescanning 57
linked structures, diving 130
linking agent 220
linking the Memory Tracker agent 220
Load action points file automatically 18
Load All command 99, 100
Load All Symbols in Stack Context Menu Command 29
Load from these libraries list 28
Loader Symbol command 140
loading action points 18
loading shared libraries 110
lockstep threads 68
Log page 44, 111
Long String Maximum Display Length preference 34
Lookup Function command 62
Lookup Variable 63

M

M state indicator 3
MALLOCTYPE heap debugging environment variable 223
matching process 68
matching thread 68

Maximum array rank preference 22
Maximum Display Length preference 34
Memorize all 153
Memorize all command 47, 166
Memorize command 46, 124, 152, 166
memory block extent 105, 148
Memory block properties
  memory block extent 105, 148
Memory Block Properties command 105, 146
memory debugging baselines 102
memory debugging, enable 102
Memory Error Details command 106
memory, viewing 126
message operations 177
Message Passing Toolkit 175
Message Queue Graph command 113
Message Queue window 175
  Edit > Copy 179
  Edit > Cut 179
  Edit > Delete 179
  Edit > Find 180
  Edit > Find Again 181
  Edit > Paste 179
  Edit >Undo 179
  File > Close 178
  File > Close Similar 178
  File > Exit 178
  View > Dive 181
  View > Dive Anew 181
  Window > Root 182
  Window > Update 182
  Window > Update All 182
messages 73, 151
mixed state indicator 3
modify watchpoints 142
More button 127
mpcc_r compilers 175
MPI
  attaching to processes 73, 151
  communicators 175
  MPI-2 communicator not implemented 176
MPI_ANY_SOURCE 177
MPI_ANY_TAG 177
MPI_Comm_rank 176
MPI_Comm_size() 176
MPI_COMM_WORLD() 176
MPI_INT() 177
MPI_NAME_GET() 176
MPI_NAME_PUT() 176
MPICH
  and heap debugging 218
mpxlf_r compiler 175
mpxlf90_r compiler 175
multiprocess programs
  attaching to 5
mutexes
  data window 191, 195
  flags 192
  guard for condition variables 193, 196
  lock state 195
  memory address 193, 195
  name of 193
  owner 192, 195
  process shared value 195
  sequence numbers 191, 195
  states 191, 194
  type 192, 195
  using to synchronize 196
Mutexes page (Compaq) 191
Mutexes pages (IBM) 194
mutual exclusion objects 196

N

–n expansion option 20
name of the substructure is immediately visible. To display a substructure, 126
NaN (Not a Number) 129
NaN Count 150
New Allocations button 104
New Base Window command
  Visualizer window 159
New Leaks button 104
New Program command
  Root window 4
Next command 68, 76, 87, 89
Next Instruction command 70, 78
Next Process command 64
Notify when deallocated 106, 148
Notify when reallocated 106, 148
number of servers 24

O

Open process window at breakpoint preference 18
Open process window on error preference 15
Open Source command
  Process window 56
opening a core file 7
OpenMP slave thread 50
OpenMP special stack frame 50
operators
  != 129
  < 129
  == 129
  > 129
  filtering 128
options
  surface data display 159
Options > Auto Visualize command 156
Options command 158
Options menu commands
  Visualizer window 156
Options page 13
Out command 69, 77, 87

P

Padding command 140
Parallel page 29
Parallel system, nameing 85
parallel system, naming 11
parent process, killing 5
password 20
password expansion character 20
password list 25
passwords 26
Paste command
  Fortran Modules window 163
  Globals window 171
  Message Queue window 179
  Process window 59
  PVM Tasks window 186, 212
  Root window 39
  Thread Objects window 200
  Variable window 134
PATH environment variable 5, 13, 53
PC 4, 62, 89
PC indicator 52
PE 175
pending breakpoint 93
pending receive operations 176
pending send operations 177
PID 1, 49
pipe for Visualizer 155
piping 38, 57, 132, 211
Plant in share group option 95
Plant in share group preference 18
platform restrictions for watchpoints 143
Point of Allocation page 107
Point of Deallocation page 107
pointers
  diving into 130
port list 25
port number 20
preferences
  saving 13
Preferences command
  Process window 55, 210
  Root window 13
prefix matching for dynamic libraries 27
Previous Process command 64
Process > Create command 80
Process > Detach command 80
Process > Enable Memory Debugging command 5
Process > Go command 75
Process > Halt command 76
Process > Hold command 79
Process > Hold Threads command 79
Process > Next command 76
Process > Next Instruction command 78
Process > Out command 77
Process > Release Threads 80

Process > Run To command 77
Process > Signals command
    Process window 54
Process > Startup Parameters
    Arguments page 8, 81
    Environment page 9, 82
Process > Startup Parameters command 9, 11, 80
Process > Startup Parameters dialog box 150
Process > Step command 76
Process > Step Instruction command 78
process bar 49
process barrier breakpoint 96
process ID 49
process lamination 139
process lists 1
Process menu commands
    Process window 75
Process Window
    Group > Custom Groups command 73
    Heap Change Summary command 103
    Instrumentation > Memory Error Details command 106
    Instrumentation > Set Heap Baseline command 102
    Memorize all command 124
    Tools > Debugger Loaded Libraries 110
    Tools > Memory Block Properties 105, 146
    Window > Memorize command 124
Process window 49, 63, 111
    Action Point > At Location 91
    Action Point > Delete 93
    Action Point > Delete All 99
    Action Point > Disabled 93
    Action Point > Enabled 93
    Action Point > Load All 99, 100
    Action Point > Properties 93
    Action Point > Save All 100
    Action Point > Set Barrier 91
    Action Point > Set Breakpoint 91
    Action Point > Suppress All 99
    Action Points tab 52
    creating 41
    Display Managers 66
    Edit > Copy 59
    Edit > Cut 58
    Edit > Delete 59
    Edit > Find 59
    Edit > Find Again 60
    Edit > Paste 59
    Edit > Undo 58
    File > Close 57
    File > Close Relatives 57
    File > Edit Source 56
    File > Exit 58
    File > Open Source 56
    File > Preferences 55, 210
    File > Rescan Libraries 57
    File > Save Pane 56
    File > Search Path 53
    File > Signals 54
    Group > Attach Subset 71, 150
    Group > Go 67
    Group > Halt 68
    Group > Hold 71
    Group > Kill 74
    Group > Next 68
    Group > Next Instruction 70
    Group > Out 69
    Group > Release 71
    Group > Restart 74
    Group > Run To 69
    Group > Step 68
    Group > Step Instruction 71
    opening on breakpoint 18
    Process > Create 80
    Process > Detach 80
    Process > Go 75
    Process > Halt 76
    Process > Hold 79
    Process > Hold Threads 79
    Process > Next 76
    Process > Next Instruction 78
    Process > Out 77
    Process > Release Threads command 80
    Process > Run To 77
    Process > Startup Parameters 80
    Process > Startup Parameters, Arguments page 8, 81
    Process > Startup Parameters, Environment page 9, 82
    Process > Step 76
    Process > Step Instruction 78
    Set PC 89
    Source pane 50
    Stack Frame pane 50
    Stack Trace pane 50
    Thread > Continuation Signal 90
    Thread > Go 86
    Thread > Halt 86
    Thread > Hold 89
    Thread > Next 87
    Thread > Next Instruction 89
    Thread > Out 87
    Thread > Run To 88
    Thread > Step 87
    Thread > Step Instruction 89
    Threads tab 51
    Tools > Call Graph 109
    Tools > Command Line 122
    Tools > Create Checkpoint 117
    Tools > Evaluate 108
    Tools > Fortran Modules 161
    Tools > Global Arrays 122
    Tools > Globals 169
    Tools > Message Queue 175
    Tools > Message Queue Graph 113
    Tools > Restart Checkpoint command 120
    Tools > Thread Objects 112, 191
    Tools > Warnings 112
    Tools >Globals 109
    View > Assembler > Symbolically 65
    View > Dive 61
    View > Dive Anew 61
    View > Lookup Function 62
    View > Next Process 64
    View > Previous Process 64
    View > Redive 62
    View > Reset 62
    View > Source As > Assembler 65
    View > Source As > Both 65
    View > Source As > Source 65
    View > Undive 62
    View> Assembler > By Address 66
    Window > Duplicate 123
    Window > Root 124
    Window > Update 123
    Window > Update All 123
process, creating 67
processes
    changing 130
Processes to Attach To area 150
processes, attaching 71, 150
processes, attaching to a subset 71, 150
processes, stopping 16
Processes/Ranks Tab 49
program counter 89
program representation 109
Properties command 93
pthread_mutexattr_settype() function 192, 195
pthread_mutexattr_settype_np() 192, 195
PVM Tasks window 183, 205
    Edit > Copy 185, 212
    Edit > Cut 185, 212
    Edit > Delete 186, 213
    Edit > Find 186, 213
    Edit > Find Again 187, 214
    Edit > Paste 186, 212
    Edit > Undo 185, 212
    File > Close 184, 211
    File > Exit 184, 211
    View > Dive 187, 214
    View > Dive Anew 187
    Window > Root 189, 216
    Window > Update 188, 215

Window > Update All 188, 215
pvm_spawn() 183

R

R state indicator 3, 7, 51
R/W locks page (IBM) 196
raising process window on error 15
rank of visualizer 22
ranks 73, 151
read-write locks 196
 lock state 197
 memory address of 198
 owner system TID 197
 process shared value 197
 sequence number 197
Redive All command 138
Redive command 137
 Process window 62
register variables, viewing 126
Relative to Baseline check box 102, 103
relatives
 attaching to 5, 6
Release command 71
Release Threads command 80
releasing threads 90
 at barrier 96
remember memory state 102
remote host list 25
remote hosts 25
removing breakpoints 99
Rescan Libraries command 57
Resend signal option 55
Reset command 62
Reset Defaults command
 Variable window 133
Restart Checkpoint 42
Restart Checkpoint command
 Process window command 120
Restart command 74
Restrict Output --> Max rows to save preference 38, 57, 132, 211
Root command
 Fortran Modules window 167
 Globals window 174
 Message Queue window 182
 Process window 124
 PVM Tasks window 189, 216
 Thread Objects window 203
 Variable window 153
Root Window
 Window > Memorize 46
 Window > Memorize all 47
Root window 44
 Attached page 1
 Display Exited Threads 42
 Display Manager Threads 42
 Edit > Copy 39
 Edit > Cut 39
 Edit > Delete 39
 Edit > Find 40
 Edit > Find Again 40
 Edit > Paste 39
 Edit > Undo 39
 File > Exit 38
 File > New Program 4
 File > Preferences 13
 File > Save Pane 37
 File> Search Path 12
 Tools > Command Line 45
 Tools > PVM Tasks 183, 205
 Tools > Restart Checkpoint command 42
 Tools > Warnings 45
 Unattached page 12
 View > Collapse All 41, 137
 View > Dive 41
 View > Dive Anew 41
 View > Expand All 41
 Window > Update 46
 Window > Update All 46
RS/6000 checkpoints 118
Run the group preference 30
Run To command 69, 77, 88
running program to a line 69
running state indicator 3, 7, 51

S

S state indicator 7
satisfaction group 97
satisfaction set 17, 97
Save action points file at exit preference 18
Save All command 100
Save As DOT File button 117
Save as Spreadsheet File button 117
Save Layout button 117
Save Pane command
 Process window 56
 Root window 37
 Variable window 132
Save preferences file on exit preference 15
saving action points 18
saving breakpoints 100
saving preferences 15
Scale To Drawing Area option 115
Search Path command 12, 53
 Process window 53
 Root window 12
 search order 12, 53
Select by family and size preference 31
Select by full name preference 31
selecting a routine 69
selectively holding threads 79
sending to a pipe 38, 57, 132, 211
server launch 19
servers, number of 20, 24
Set Barrier command 91
Set Breakpoint command 91
Set Heap Baseline 102
Set Heap Baseline command 103
Set Heap Baseline command, Process Window 102
Set PC 89
Set PC command 89
–set_pw expansion option 20, 26
sets of breakpoints 92
setting barrier points 91
setting breakpoints 91
setting memory state boundary 102
SGI MPI checkpoints 118
share groups
 planting action points in 95
 planting in breakpoints 18
 planting watchpoint in 145
Show Backtrace Information button 105, 147
showing system thread ID 15
SIGINT signal preference 14
signals 90
 clearing 90
Signals command
 Process window 54
SIGSTOP issue 54
SIGTRAP isue 54
SILENT state 45, 112
Single Debug Server Launch 18
single debug server launch 19
slave thread, OpenMP 50
sleeping state indicator 7
slice 125
slice descriptor 128
sorting columns 130
Source As > Assembler 65
Source As > Both 65
Source As > Source 65
source code 49
Source Code Editor 22
source code editor launch string 22, 56
Source command 65
source display 65
Source Pane 49
Source pane 50, 61
spelling correction procedure 62
Stack Frame Pane 49
Stack Frame pane 50
 diving 61
Stack Trace Pane 49, 50
Stack Trace pane 50
 diving 61
Standard Deviation 150
Standard Error 10
Standard Error redirection 84
Standard Input 9
Standard Input redirection 83
Standard Output 10
Standard Output redirection 84
Startup Parameters

Arguments page 8, 81
Environment page 9, 82
Startup Parameters command 80
state indicators 3
Statistics command 149
stderr redirection 9, 83
stdin redirection 9
stdio redirection 9, 83
stdout redirection 9, 83
Step command 68, 76, 87
Step Instruction command 71, 78, 89
STL containers, viewing 15
Stop control group on error signal preference 15
Stop on Memory Errors checkbox 5
Stop on Memory Errors command 102
Stop signal option 55
Stop the group preference 30
stopped state indicator 3, 7, 51, 55
stopping before executing 27
stopping related processes 55
strides 128
structures
diving into 130
suffix matching for dynamic libraries 27
Suppress All command 99
Surface command 156
Surface visualization window 156
Symbolically command 65
synchronizing processes and threads 96
synchronizing using mutexes 196
system thread ID 15

T

T state indicator 3, 7, 51, 55
tab width 15
Tab width preference 15
Talking to Rank option 151
tasker 183
TCP/IP port number 20, 26
temporary files 25
terminating TotalView 14
Thread > Continuation Signal command 80, 90
Thread > Go command 86
Thread > Halt command 86
Thread > Hold command 89
Thread > Next command 87
Thread > Next Instruction command 89
Thread > Out command 87
Thread > Run To command 88
Thread > Set PC command 89
Thread > Step command 87
Thread >Step Instruction command 89
thread bar 49
thread ID 15
thread lamination 139
thread lists 1
Thread menu commands
Process window 86
Thread Objects command 112
Thread Objects thread window
R/W Locks page (IBM) 196
Thread Objects window 191
condition variables (Compaq) 193
condition variables (IBM) 196
data keys (IBM) 198
Edit > Copy 200
Edit > Cut 200
Edit > Delete 200
Edit > Find 201
Edit > Find Again 201
Edit > Paste 200
Edit > Undo 199
File > Close 199
File > Close Similar 199
File > Exit 199
Mutexes page (Compaq) 191
Mutexes page (IBM) 194
View > Dive 202
View > Dive Anew 202
View > Dive Thread 202
Window > Root 203
Window > Update 203
Window > Update All 203
thread variables 49
threads
changing 130
selectively holding 79
Threads Tab 49
Threads tab 51
diving 61
threads, stopping 16
TID 49
tid 192, 195
timeout 25
timeout preference 19
Toolbar style preference 16
Tools > Attach Subset (Array of Ranks) command 73, 151
Tools > Call Graph command 109
Tools > Command Line command 122
Root window 45
Tools > Create Checkpoint command 117
Tools > Debugger Loaded Libraries command
Process Window 110
Tools > Evaluate command 108
Tools > Fortran Modules 161
Tools > Global Arrays command 122
Tools > Globals command 109
Tools > Memory Block Properties command
Process Window 146
Tools > Message Queue 175
Tools > Message Queue Graph command 113
Tools > Program Browser command 50
Tools > PVM Tasks 183, 205
Tools > Restart Checkpoint command
Process window 120
Root window 42
Tools > Statistics 149
Tools > Statistics command 149
Tools > Thread Objects 191
Tools > Thread Objects command 112
Tools > Visualize 155
Tools > Visualize command 148, 155
Tools > Visualize Distribution command
Variable Window 148
Tools > Warnings, Process window 112
Tools > Warnings, Root window 45
Tools > Watchpoint command 142
Tools menu commands
Process window 108
Variable window 142
tooltips preference 14
TotalView
interactions with Visualizer 155
terminating 14
tracer configuration flags 21
transposing axis 158
tree() function 229
tvd file 13
tvdsvr 18, 24, 25
tvdsvr launch 19
tvdsvr password 20
TVDSVRLAUNCHCMD launch command variable 20, 24, 25
tvheap_mr.a
aix_install_tvheap_mr.sh script 223
and aix_malloctype.o 224
creating using poe 223
dynamically loading 223
libc.a requirements 223
pathname requirements 223
relinking executables on AIX 223
tvheap_mr.a library 223
type transformations, activating 15

U

Unattached page 12
unconditional watchpoints 144, 145
Undive All command
Variable window 137
Undive command
Process window 62
Variable window 137
undive icon 63
Undo command

Fortran Modules window 163
Globals window 171
Message Queue window 179
Process window 58
PVM Tasks window 185, 212
Root window 39
Thread Objects window 199
Variable window 133
undo of find or lookup 62
unexpected messages 176
UNIX signals 54
Update All command
Fortran Modules window 166
Globals window 174
Message Queue window 182
Process window 123
PVM Tasks window 188, 215
Root window 46
Thread Objects window 203
Variable window 152
Update command
Fortran Modules window 166
Globals window 174
Message Queue window 182
Process window 123
PVM Tasks window 188, 215
Root window 46
Thread Objects window 203
Variable window 152
updating variable window contents 138

V

Valid in Scope 127
Variable Window
Tools > Visualize Distribution command 148
View > Dive in All command 136
View > Loader Symbol command 140
View > Padding command 140
View > Redive All command 138
View > Redive command 137
Window > Memorize command 152
Windows > Memorize all command 153
Variable window 140, 149, 155
Close > Similar 132
Edit > Copy 134
Edit > Cut 134
Edit > Delete 134
Edit > Find 134
Edit > Find Again 135
Edit > Paste 134
Edit > Reset Defaults 133
Edit > Undo 133
File > Close 132
File > Exit 132
File > Save Pane 132
Tools > Visualize 148
Tools > Watchpoint 142
View > Dive 136
View > Dive Anew 136
View > Expand All 137
View > Undive 137
View > Undive All 137
View > View Across None 139
View > View Across Process 139
View > View Across Thread 139
Viw > Break lines at newlines command 140
Window > Duplicate 152, 215
Window > Root 153
Window > Update 152
Window > Update All 152
variables
finding 62
viewing distributes 139
variables, changing data type 127
verbosity 25, 26
verbosity level 20
View > Assembler > By Address command 66
View > Assembler > Symbolically command 65
View > Block Status command 142
View > Break lines at newlines command, Variable Window 140
View > Collapse All command 126
Root window 41, 137
View > Compilation Scope > Fixed command 139
View > Compilation Scope > Floating command 139
View > Display Exited Threads command
Root window 42
View > Display Manager Threads command
Root window 42
View > Display Managers command
Process window 66
View > Dive Anew command
Fortran Modules window 165
Globals window 173
Message Queue window 181
Process window 61
PVM Tasks window 187
Root window 41
Thread Objects window 202
Variable window 136
View > Dive command 50
Fortran Modules window 165
Globals window 173
Message Queue window 181
Process window 61
PVM Tasks window 187, 214
Root window 41
Thread Objects window 202
Variable window 136
View > Dive in All command
Variable Window 136
View > Dive Thread command
Thread Objects window 202
View > Examine Format > None command 140
View > Examine Format > Raw command 107, 140, 141
View > Examine Format > Structured command 140
View > Expand All command
Root window 41
Variable window 137
View > Expand all command 126
View > Freeze command 138
View > Graph command 156
View > Loader Symbols command
Variable Window 140
View > Lookup Function command 62
View > Lookup Variable command 63
View > Next Process command 64
View > Padding command
Variable Window 140
View > Previous Process command 64
View > Redive All command
Variable Window 138
View > Redive command
Process window 62
Variable Window 137
View > Reset command 62
View > Source As > Assemble command 65
View > Source As > Both command 65
View > Source As > Source command 65
View > Surface command 156
View > Undive All command
Variable window 137
View > Undive command
Process window 62
Variable window 137
View > View Across None command 139
View > View Across Process command 139
View > View Across Thread command 139
View Across None command 139
View Across Process command 139
View Across Thread command 139
View menu commands
Fortran Modules window 165
Globals window 173
Message Queue window 181
Process window 60
PVM Tasks window 187, 214
Root window 41
Thread Objects window 202

Variable window 135
Visualizer window 156
View simplified STL containers preference 15
viewing distributed variables 139
viewing memory 126
viewing STL containers 15
visualization, deleting a dataset 156
Visualize command 148
Visualize Distribution command 148
Visualizer
actor mode 157
camera mode 157
data window 155, 157
dataset window 155, 156
how implemented 155
interactions with TotalView 155
joystick mode 157
pan 157
picking 157
pipe 155
rank 22
rotate 157
scale 157
selecting datasets 156
surface data display options 159
surface view 157
trackball mode 157
windows, types of 155
wireframe view 157
zoom 157
Visualizer Launch 21
visualizer launch string 21
Visualizer window 155
File > Close 159
File > Dataset 158
File > Delete 156, 159
File > Exit 156
File > New Base window 159
File > Options 158
Options > Auto Visualize 156
View > Graph 156
View > Surface 156
visualizing data 156
vommand-line arguments 81

W

W state indicator 3, 51
waiters 193, 196
Warn about C++ exceptions during single step operations preference 15
warning state 44, 112
Warnings dialog box 45, 112
Watchpoint command 142
Watchpoint Properties dialog box 93
watchpoint state indicator 3, 51
watchpoints
conditional 144, 145
enabling 145
supported platforms 143
unconditional 144, 145
When a job goes parallel or calls exec() preferences 30
When a job goes parallel preferences 30
When barrier done, stop preference 17
When barrier hit, stop preference 17
When breakpoint hit, stop preference 16
When Hit, Stop buttons 96
When the file suffix matches preference 27
wildcards
dynamic library names 28
Window > Duplicate command 138
Process window 123
Variable window 152, 215
Window > Memorize all
Fortran Modules window 166
Window > Memorize all command 124
Process Window 124
Root Window 47
Variable Window 153
Window > Memorize command
Fortran Modules 166
Process Widow 124
Root Window 46
Variable Window 152
Window > Root command
Fortran Modules window 167
Globals window 174
Message Queue window 182
Process window 124
PVM Tasks window 189, 216
Thread Objects window 203
Variable Window 153
Window > Update All command
Fortran Modules window 166
Globals window 174
Message Queue window 182
Process window 123
PVM Tasks window 188, 215
Root window 46
Thread Objects window 203
Variable window 152
Window > Update command
Fortran Modules window 166
Globals window 174
Message Queue window 182
Process window 123
PVM Tasks window 188, 215
Root window 46
Thread Objects window 203
Variable window 152
Window menu commands
Fortran Modules window 166
Globals window 173
PVM Tasks window 188, 215
Root window 46
Thread Objects window 202
Variable window 152
window postions preference 14
windows 57
Windows menu commands
Process window 123
windows, closing 57
working directory 25
working directory option 20
–working_directory expansion option 20
Wrap On Search option 40, 60, 134, 164, 172, 180, 186, 201, 214
writing to a file 37, 56, 132, 210

X

xlsfonts program 31

Y

yellow arrow PC indicator 52

Z

Z state indicator 7
Zombie state indicator 7