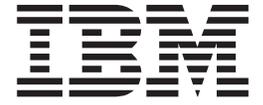


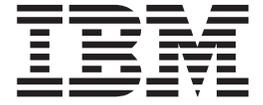
IBM Parallel Environment for AIX



PE Benchmark User's Guide

Version 3 Release 1

IBM Parallel Environment for AIX



PE Benchmark User's Guide

Version 3 Release 1

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 161.

LIMITED RIGHT NOTICE

(a) These data are submitted with limited rights under Subcontract No. B331593 (and lower-tier subcontract, if appropriate). These data may not be reproduced and used by the University and the Government with the express limitation that they will not, without written permission of the Subcontractor, be used for purposes of manufacture nor disclosed outside the University or the Government; except that the University or the Government may disclose these data outside the University or the Government for the following purposes, if any; provided that the University and the Government makes such disclosure subject to prohibition against further use and disclosure*:

(b) This Notice shall be marked on any reproduction of these data in whole or in part.

(End of Notice)

*The purposes shall be identified in the subcontract schedule when this clause is used. (1)

(1) There are none to date.

Beta Documentation (October 2000)

This manual is Beta Documentation only.

Copyright International Business Machines Corporation 2000. All rights reserved. Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

© Copyright International Business Machines Corporation 2000. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About This Book	vii
Who Should Use This Book	vii
How This Book is Organized	vii
Overview of Contents	vii
Chapter 1. What is the PE Benchmarker?	1
Chapter 2. Using the Performance Collection Tool	5
Using the Performance Collection Tool's Graphical User Interface.	5
Performance Collection Tool (Graphical User Interface) Overview.	5
Starting the Performance Collection Tool	8
Loading and Starting a New Application.	10
Connecting to a Running Application	13
Connecting to One or More Processes of the Loaded Application	15
Selecting (At Tool Start-up Time) the Type of Probe Data To Be Collected	16
Selecting (After Tool Startup Time) the Type of Probe Data To Be Collected	18
Starting and Stopping Application Execution	20
Disconnecting From One or More Processes of the Loaded Application	20
Viewing Application Source Code	20
Setting User Preferences	21
Searching for Functions in the Application Source Code.	22
Examining Output From, and Sending Input To, the Application	23
Specifying MPI Trace Data to Be Collected	24
Adding User Markers to Processes	33
Specifying Profile Data To Be Collected	36
Removing Performance Collection Probes From One or More Processes	39
Removing User Markers From Processes	40
Terminating Connected Processes.	42
Exiting the Performance Collection Tool	42
Using the Performance Collection Tool's Command-Line Interface	43
Performance Collection Tool (Command-Line Interface) Overview	43
Starting the Performance Collection Tool In Command-Line Mode	46
Grouping Tasks of a POE Application.	47
Loading and Starting a New Application.	48
Connecting to a Running Application	49
Suspending and Resuming Application Execution	50
Sending Standard Input Text to the Application	51
Displaying the Contents of a Source File	52
Selecting Type of Probe Data To Be Collected	53
Collecting MPI Trace and Custom User Marker Information	54
Collecting Hardware and Operating System Profile Information	59
Terminating Connected Processes.	61
Disconnecting From the Application	62
Exiting the Performance Collection Tool	63
Chapter 3. Using the Profile Visualization Tool	65
Using the Profile Visualization Tool's Graphical User Interface	65
Profile Visualization Tool (Graphical User Interface) Overview.	65
Starting the Profile Visualization Tool	67
Loading Files for Processing	69
Viewing Profile Data	70
Viewing Selected Objects	76
Finding Data.	80

Generating Reports of Profile Data	81
Saving Summary Data	84
Exporting Profile Data	86
Specifying User Preferences	87
Exiting the Profile Visualization Tool	88
Using the Profile Visualization Tool's Command Line Interface	88
Profile Visualization Tool (Command Line Interface) Overview	88
Starting the Profile Visualization Tool in Command-Line Mode	89
Loading Files	89
Creating a Summary File	89
Generating Reports	89
Exporting Files	90
Exiting the Profile Visualization Tool	90

Chapter 4. Creating, Converting, and Viewing Information Contained In, UTE Interval Files	91
Converting AIX Trace Files Into UTE Interval Trace Files	92
Generating Statistics Tables From UTE Interval Trace Files	93
Converting UTE Interval Files Into SLOG Files Required By Argonne National Laboratory's Jumpshot Tool	94

Appendix A. Dynamic Probe Class Library (DPCL) Enhancements	97
New Functions of Class SourceObj	97
bget_function_list	98
get_function_list	99
Class FunctionId	100
Constructors	101
get_demangled_name	102
get_demangled_name_length	103
get_mangled_name	104
get_mangled_name_length	105
get_module_name	106
get_module_name_length	107
operator=	108
operator==	109
Class FunctionList	110
Constructors	111
get_count	112
get_entry	113
operator=	114

Appendix B. PE Benchmark Command Reference	115
convert	116
pct	118
Subcommands of the pct Command	120
connect Subcommand (of the pct Command)	120
destroy Subcommand (of the pct Command)	120
disconnect Subcommand (of the pct Command)	121
exit Subcommand (of the pct Command)	122
file Subcommand (of the pct Command)	122
find Subcommand (of the pct Command)	123
function Subcommand (of the pct Command)	124
group Subcommand (of the pct Command)	125
list Subcommand (of the pct Command)	126
load Subcommand (of the pct Command)	127
point Subcommand (of the pct Command)	128

profile add Subcommand (of the pct Command)	130
profile remove Subcommand (of the pct Command)	131
profile set path Subcommand (of the pct Command)	132
profile show Subcommand (of the pct Command)	132
resume Subcommand (of the pct Command)	133
select Subcommand (of the pct Command)	134
set Subcommand	134
show Subcommand (of the pct Command)	135
start Subcommand (of the pct Command)	137
stdin Subcommand (of the pct Command)	137
suspend Subcommand (of the pct Command)	137
trace add Subcommand (of the pct Command)	138
trace remove Subcommand (of the pct Command)	140
trace set Subcommand (of the pct Command)	140
trace show Subcommand (of the pct Command)	141
wait Subcommand (of the pct Command)	142
pvt	144
Subcommands of the pvt Command	146
exit Subcommand (of the pvt Command)	146
export Subcommand (of the pvt Command)	146
load Subcommand (of the pvt Command)	146
report Subcommand (of the pvt Command)	146
sum Subcommand (of the pvt Command)	147
slogmerge	148
utemerge	150
utestats	152
Appendix C. PE Benchmark Messages	155
Notices	161
Trademarks.	162
Acknowledgements	163
Glossary of Terms and Abbreviations	165
Bibliography	173
Information Formats	173
Finding Documentation on the World Wide Web	173
Accessing PE Documentation Online	173
RS/6000 SP Publications.	174
SP Hardware and Planning Publications	174
SP Software Publications	174
AIX and Related Product Publications	175
DCE Publications	175
Red Books	175
Non-IBM Publications	175
Index	177

About This Book

This book is an informal Beta document that describes the PE Benchmark tool set still under development. It describes the various tools for collecting and analyzing program event trace or hardware performance data. Specifically, it describes:

- the Performance Collection Tool for collecting event trace or hardware performance information from an executing serial or POE program.
- the Profile Visualization Tool for displaying graphs and charts of the performance information gathered by the Performance Collection Tool.
- Several Unified Trace Environment (UTE) utilities for converting and displaying the event trace information gathered by the Performance Collection Tool.

Who Should Use This Book

This book is intended for application developers working in an AIX environment who wish to analyze their serial or POE programs using the PE Benchmark. You should, therefore, understand programming concepts and the AIX operating system before reading this book. Furthermore, if you plan to analyze POE programs, you should understand parallel programming concepts and know how to run programs in the Parallel Operating Environment. Where necessary, this book provides some background information related to these issues. More commonly, this book refers you to the appropriate documentation.

How This Book is Organized

This book is organized into four chapters plus appendices. The chapters provide an overview of the PE Benchmark tool set — the Performance Collection Tool, the Profile Visualization Tool, and the UTE utilities — and then describes, in detail, each of the tools in turn. The appendices provide AIX man-page style reference information for the PE Benchmark commands and some new Dynamic Probe Class Library (DPCL) classes and functions that have been added for this Beta release.

Overview of Contents

This book contains the following information:

- “Chapter 1. What is the PE Benchmark?” on page 1 describes the tools at a high level and illustrates how they are used together to analyze serial or POE programs.
- “Chapter 2. Using the Performance Collection Tool” on page 5 describes how to collect MPI and user event traces or hardware and operating system profiles for a particular serial or POE program’s run. It describes how you can use the Performance Collection Tool’s graphical user interface or command line interface to:
 - connect to a running application, or, if the application you want to examine is not already running, load and connect to it.
 - select the type of data to collect (either MPI and user event traces or hardware and operating system profiles).
 - start and stop execution of the target application.
 - install performance collection probes into the target application to collect the MPI, user event traces, or hardware profile information.

- remove the performance collection probes from the target application when you are through collecting the performance data.
- Disconnect from, or terminate, the target application processes.
- “Chapter 3. Using the Profile Visualization Tool” on page 65 describes how to view and generate reports for the hardware and operating system profiles collected by the Performance Collection Tool. It describes how you can use the Profile Visualization Tool’s graphical user interface or command line interface to:
 - load one or more profile data files for processing.
 - view bar charts of the collected profile data.
 - generate reports of the collected profile data.
- “Chapter 4. Creating, Converting, and Viewing Information Contained In, UTE Interval Files” on page 91 describes several utilities designed to enable you to view the MPI and user event traces collected by the Performance Collection Tool. The trace data collected by the Performance Collection Tool is stored as AIX trace files. This chapter describes how, in order to view the data, you can:
 - convert the AIX trace files into UTE files (using the **convert** utility), and generate statistics tables of the information in the UTE files (using the **utestats** utility). If desired, you can first merge multiple UTE files into a single UTE file (using the **utemerge** utility) before using the **utestats** utility.
 - convert the AIX trace files into UTE files (using the **convert** utility), and merge the UTE files into a single SLOG file (using the **slogmerge** utility). The SLOG file can then be analyzed by Jumpshot — a public domain tool developed at Argonne National Laboratory.
- “Appendix A. Dynamic Probe Class Library (DPCL) Enhancements” on page 97 describes the classes and functions we have added to DPCL in order to implement the Performance Collection Tool. This information is designed to supplement the *IBM Parallel Environment for AIX: DPCL Class Reference* and the *IBM Parallel Environment for AIX: DPCL Programming Guide*, and enables you to incorporate the new classes and functions into the DPCL analysis tools you create.
- “Appendix B. PE Benchmark Command Reference” on page 115 contains the manual pages for the PE Benchmark commands discussed throughout this book.
- “Appendix C. PE Benchmark Messages” on page 155 contains an explanation of PE Benchmark messages you may encounter.

Chapter 1. What is the PE Benchmarker?

The PE Benchmarker is a suite of applications and utilities that you can use to analyze the performance of programs run within the IBM AIX Parallel Environment. The PE Benchmarker suite consists of:

- **the Performance Collection Tool.** This tool enables you to collect either MPI and user event data or hardware and operating system profiles for one or more application processes (or "tasks"). This tool is built on our dynamic instrumentation technology, the *Dynamic Probe Class Library (DPCL)*. Unlike more traditional tools for collecting message-passing and other performance information, the Performance Collection Tool, because it is built on DPCL, enables you to insert and remove instrumentation probes into the target application while the target application is running. More traditional tools require the application to be instrumented through compilation or linking. This often results in more instrumentation being inserted into the application than is actually needed, and so such tools are more likely to create situations in which the instrumented version of the application is no longer representative of the actual, uninstrumented, version of the application. Since the Performance Collection Tool enables you to make the decision of what data is collected at run time, this typically results in a more acceptable intrusion cost of the instrumentation. What's more, the files output by the Performance Collection Tool are output on each machine running instrumented processes rather than on a single, centralized, machine. This means that your analysis can be efficiently scaled to collect information on a large number of processes running on a large number of nodes.
- **a set of Unified Trace Environment (UTE) utilities.** When you collect MPI and user event traces using the Performance Collection Tool, the collected information is saved, on each machine running instrumented processes, as a standard AIX event trace file. The UTE utilities enable you to convert one or more of these AIX trace files into UTE interval files. While an AIX event trace file has a time stamp indicating the point in time when an event occurred, UTE interval files take this information to also determine how long an event lasts before encountering the next event. Because they include this duration information, UTE interval files are easier to visualize than traditional AIX event trace files. The UTE utilities are:
 - The **convert** utility which converts AIX event trace records into UTE interval trace files.
 - The **utemerge** utility which merges multiple UTE interval files into a single UTE interval file.
 - The **utestats** utility which generates statistics tables from UTE interval files.
 - The **slogmerge** utility which converts and merges UTE interval files into a single SLOG file for analysis within Argonne National Laboratory's Jumpshot tool.
- **the Profile Visualization Tool.** When you collect hardware and operating system profiles using the Performance Collection Tool, the collected profile information is saved, on each machine running instrumented processes, as NetCDF (network Common Data Form) files. The Profile Visualization Tool can read NetCDF files and summarize the profile information in reports.

The following figure illustrates how the various tools in the PE Benchmarker toolset work together to enable you to analyze the performance of programs run within the IBM AIX Parallel Environment. Please note that Jumpshot is not part of the PE Benchmarker toolset, but is instead a public domain tool developed at Argonne

National Laboratory. It is shown in the figure below, because the PE Benchmarker provides the **slogmerge** utility for converting UTE files into the SLOG format required by Jumpshot.

PE Benchmarker

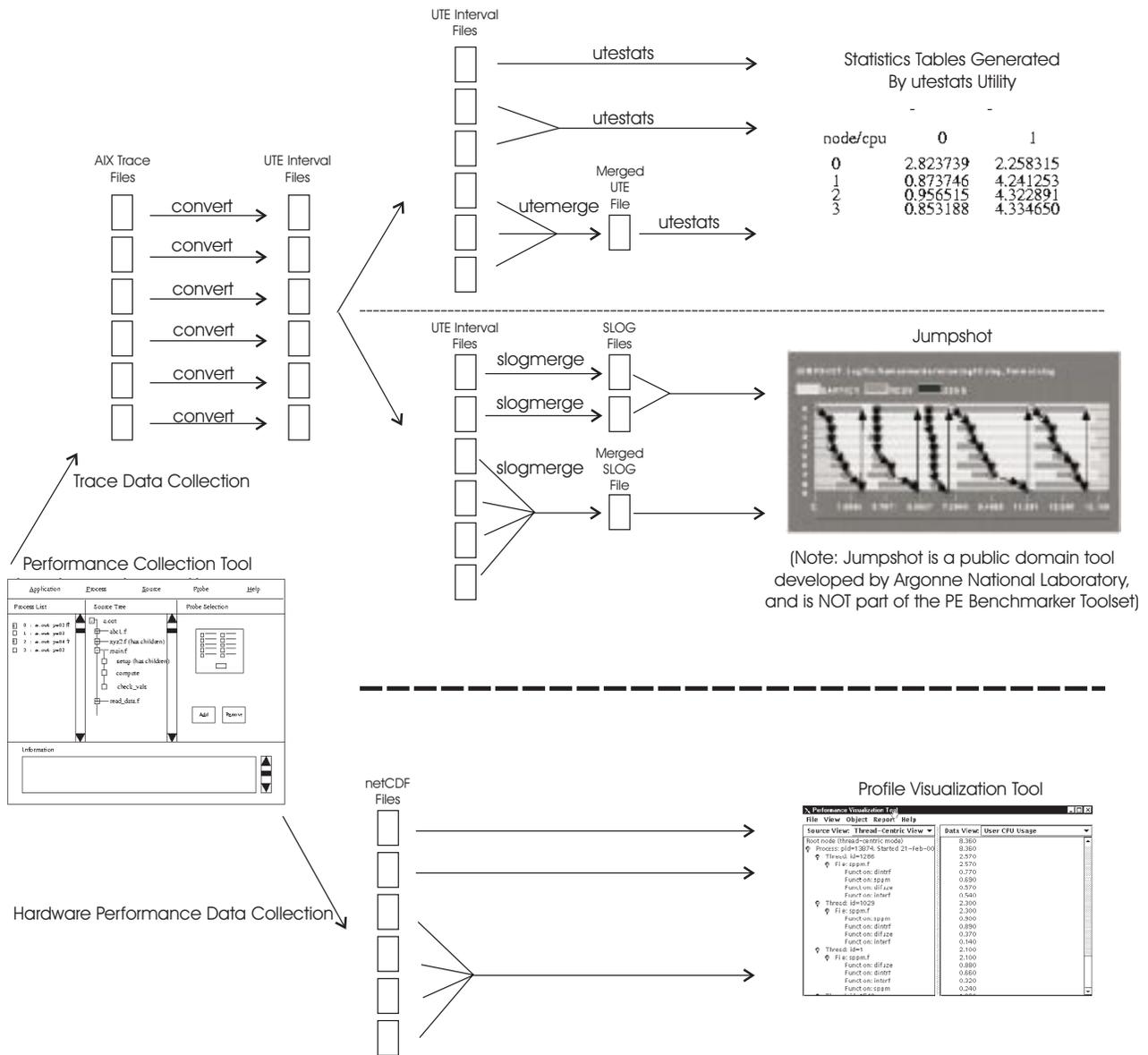


Figure 1. Overview of the PE Benchmarker Toolset

The preceding figure illustrates the procedure for collecting and analyzing data using the PE Benchmarker toolset. This procedure starts with the Performance Collection Tool. When using the Performance Collection Tool, you must select the type of data you are collecting — either MPI and user event trace data or hardware and operating system performance data. You use the Performance Collection Tool to connect to existing processes, or start processes running (which also connects to the processes). By "connect to processes" we mean the Performance Collection Tool establishes a communication connection that enables it to control the process'

execution (suspend, resume, and terminate the process), and also instrument the process with data collection probes. Data files containing the collected information will be generated on each machine running at least one instrumented process. The format of the files generated depends on the type of data you are collecting.

- If you are collecting MPI and user event trace data, standard AIX trace files will be generated. You will first need to take the AIX trace files output by the Performance Collection Tool and convert them, using the **convert** utility, into UTE interval files. If you want to view statistical tables of the information contained in the UTE interval files, you can use the **utestats** utility. You can optionally merge multiple UTE files into a single UTE file using the **utemerge** utility before using the **utestats** utility to generate the statistical tables. If you instead want to view the information contained in the UTE interval files graphically, you can convert them into SLOG files which are readable by Argonne National Laboratory's Jumpshot tool. To convert UTE interval files into SLOG files, you use the **slogmerge** utility. The **slogmerge** utility can convert a single UTE interval file into a single SLOG file, or it can convert multiple UTE interval files into a single, merged, SLOG file.
- If you are collecting hardware performance data, NetCDF files will be generated. You can use the Profile Visualization Tool to generate graphs and reports of the information contained in the NetCDF files.

Chapter 2. Using the Performance Collection Tool

This chapter describes how to collect MPI and user event traces or hardware and operating system profiles for a particular serial or POE program's run. It describes how you can use the Performance Collection Tool's graphical user interface or command-line interface to:

- connect to a running application, or (if the application you want to examine is not already running) load an application and connect to it.
- select the type of data to collect (either MPI and user event traces, or hardware and operating system profiles).
- start and stop execution of the target application.
- install performance collection probes into the target application to collect the MPI, user event trace, or hardware profile information.
- remove the performance collection probes from the target application when you are through collecting the performance data.
- Disconnect from, or terminate, the target application processes.

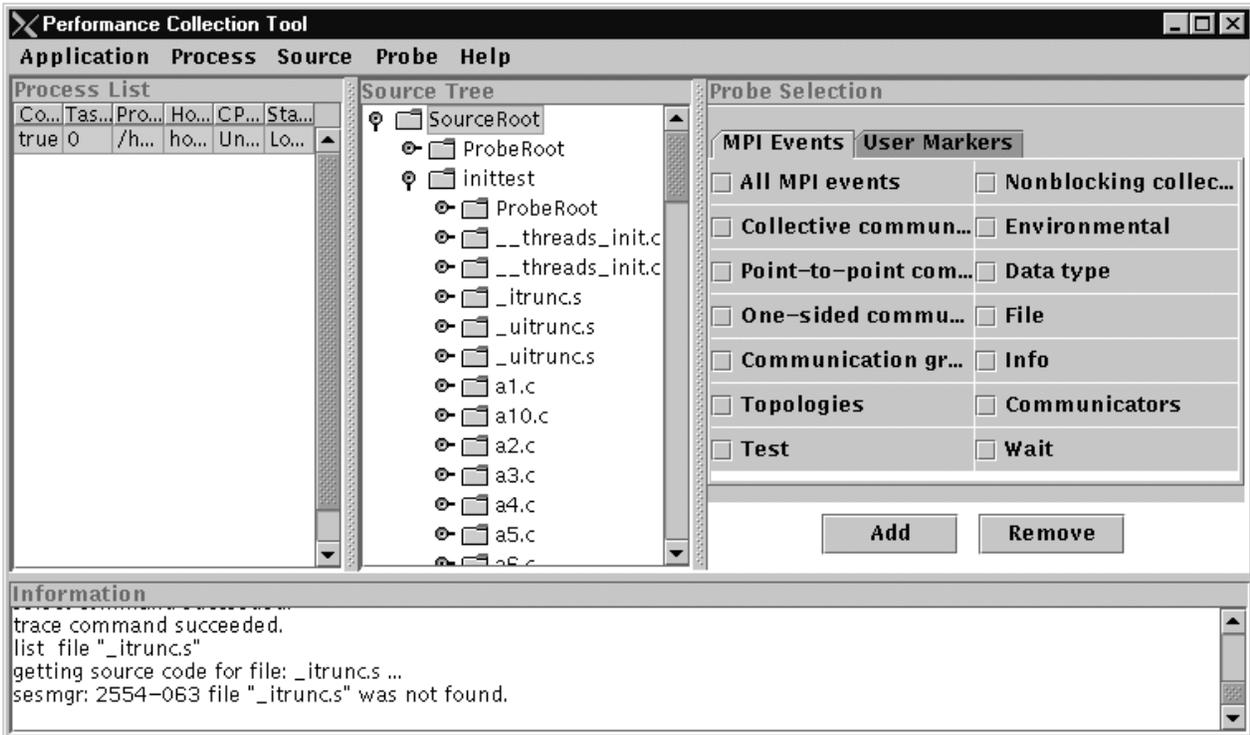
For information on the tool's graphical user interface, refer to "Using the Performance Collection Tool's Graphical User Interface". For information on the tool's command-line interface, refer to "Using the Performance Collection Tool's Command-Line Interface" on page 43.

Using the Performance Collection Tool's Graphical User Interface

This section describes how you can use the Performance Collection Tool's graphical user interface to collect either MPI and user event traces, or hardware and operating system profiles. This section begins with a brief overview of the tasks you can perform using the Performance Collection Tool's graphical user interface, and then describes each of these tasks in more detail. You can also operate the Performance Collection Tool using its command-line interface. For information on the tool's command-line interface, refer to "Using the Performance Collection Tool's Command-Line Interface" on page 43.

Performance Collection Tool (Graphical User Interface) Overview

Performance Collection Tool Main Window



Here's an overview of the steps you'll follow when using the Performance Collection Tool's graphical user interface to collect either MPI and user event traces, or hardware and operating system profiles. More detailed instructions on each of the tasks summarized are provided later in this chapter. To use the Performance Collection Tool, you:

1. Start the Performance Collection Tool by using the **pct** command. For more information, refer to "Starting the Performance Collection Tool" on page 8.
2. Either load and start a new application, or connect to a running application.
 - To load and start a new application, use the Load Application Dialog to load either a serial or POE application. Using the Load Application Dialog, you can select whether you would like to merely load the application, or load the application and start its execution. If you choose to merely load the application, its execution will be suspended at its first executable instruction. This enables you to install performance collection probes before later starting application execution. For more information, refer to "Loading and Starting a New Application" on page 10.
 - To connect to a running application, use the Connect Application Dialog. Using the Connect Application Dialog, you can connect to a serial or POE application. If connecting to a POE application, you can select whether you would like to connect to all processes in the POE application, or just the controlling, "home node", POE process. Connecting to only the controlling POE process will enable you to later connect to select tasks in the POE application, and may be desirable for performance reasons. For more information, refer to "Connecting to a Running Application" on page 13.
3. Select the type of data you will be collecting using the Performance Collection Tool. You can collect either:
 - MPI and user event traces for analysis using the **utestats** utility or a graphical visualization tool like Jumpshot.

- hardware and operating system profiles for analysis within the Profile Visualization Tool.

For more information, refer to “Selecting (At Tool Start-up Time) the Type of Probe Data To Be Collected” on page 16, and “Selecting (After Tool Startup Time) the Type of Probe Data To Be Collected” on page 18.

4.

If:	Then:
<p>You are collecting MPI and user event traces.</p>	<p>Use the Probe Selection Panel of the Performance Collection Tool's Main Window to specify which MPI events you want to collect data for. For example, you can select "All MPI events", "Collective communication", "Point-to-point communication", and so on. In addition to specifying MPI trace data to be collected, you can also add user markers to processes to mark events or states of interest. Marking these states or events of interest gives you a frame of reference when analyzing the trace record in a graphical visualization tool like Jumpshot. You can also use user markers to mark locations where tracing should be stopped or started. Since you can add MPI probes only at a program, file, or function level (meaning that the entire program, file, or function will be traced), this gives you more control over which part of your program is traced.</p> <p>For more information on adding MPI trace probes or user markers to one or more connected processes, refer to "Specifying MPI Trace Data to Be Collected" on page 24 and "Adding User Markers to Processes" on page 33. For information on removing these probes when you are through collecting the necessary information, refer to "Removing Performance Collection Probes From One or More Processes" on page 39 and "Removing User Markers From Processes" on page 40.</p>
<p>You are collecting hardware and operating system profile information.</p>	<p>Use the Probe Selection Panel of the Performance Collection Tool's Main Window to specify the hardware and operating system information you want to collect for later analysis within the Profile Visualization Tool. For more information, refer to "Specifying Profile Data To Be Collected" on page 36. For information on removing the probes when you are through collecting the necessary information, refer to "Removing Performance Collection Probes From One or More Processes" on page 39.</p>

5. When you are done collecting data, you can terminate connected processes, disconnect from the processes, and/or exit the Performance Collection Tool. For more information, refer to “Terminating Connected Processes” on page 42, “Disconnecting From One or More Processes of the Loaded Application” on page 20, and “Exiting the Performance Collection Tool” on page 42.

In addition to the tasks summarized above, you can also:

- display the contents of source files in the View Source window. For more information, refer to “Viewing Application Source Code” on page 20.
- set user preferences. In particular, you can determine whether the scripting commands produced as a result of your manipulating the Performance Collection Tool's graphical user interface are displayed in the Main Window's Information Area and/or saved to a file. Since your actions on the graphical user interface are translated into the same scripting commands you would issue to the Performance Collection Tool's command-line interface, this enables you to:

- more quickly learn the scripting commands and
- create reusable scripts for the command-line interface.

For more information, refer to “Setting User Preferences” on page 21. For general information on the Performance Collection Tool’s command-line interface, refer to “Using the Performance Collection Tool’s Command-Line Interface” on page 43.

- use a search string to locate functions within the Main Window’s Source Tree. For more information, refer to “Searching for Functions in the Application Source Code” on page 22.
- start and stop execution of connected processes. You might, for example, wish to suspend execution of your application prior to instrumenting it, and resume execution after probes have been added. For more information, refer to “Starting and Stopping Application Execution” on page 20.
- examine standard output and error from, and send standard input to, the application using the I/O Console Window. For more information, refer to “Examining Output From, and Sending Input To, the Application” on page 23.

Starting the Performance Collection Tool

Welcome Dialog

You can start the Performance Collection Tool in either graphical-user-interface mode or command-line mode. For instructions on starting the Performance Collection Tool in command-line mode, refer to “Using the Performance Collection Tool’s Command-Line Interface” on page 43. To start the Performance Collection Tool in graphical-user-interface mode:

1. Enter the **pct** command at the AIX command prompt.

```
$ pct
```

Doing this starts the Performance Collection Tool in graphical-user-interface mode and opens its first window -- the Welcome Dialog.



Note: If you want to generate diagnostic log files for the Performance Collection Tool’s run, you can optionally specify a diagnostic log setting using the **-d** command-line option when issuing the **pct** command.

```
$ pct -d diag_log_setting
```

Where *diag_log_setting* is one of the values outlined in the following table, and determines the amount of logging information that will be generated. The log files generated when you use the *diag_log_setting* flag are intended for identifying problems through IBM. When diagnostic logging is turned on, the Performance Collection Tool will generate a log file in the directory /tmp on each host machine running target application processes to which the tool connects. The log file is generated by a daemon process that handles communication between the Performance Collection Tool and the target application process. The log file saved to the /tmp directory will be named dpc1d.nnnn (where nnnn is the AIX process ID of the daemon process).

If <i>diag_log_setting</i> is:	Then:
severe	The daemon process will generate messages for fatal and severe error conditions only.
warning	In addition to fatal and severe error conditions, the daemon process will generate warning messages.
trace	In addition to fatal, severe, and warning messages, the daemon process will also generate function entry/exit trace information.
detail	The most detailed level of diagnostic messages will be generated by the daemon process. In addition to the severe, error, warning, and function entry/exit trace information, the daemon process will generate other, more general, information.

2. The Welcome Dialog provides option buttons that enable you to select whether you would like to load a new application or connect to an existing one.

If:	Then:
You want to examine an application that is not already running.	<p>Select the Load a new application option button and click the OK command button.</p> <p>Doing this closes the Welcome Dialog, and opens the Load Application Dialog. The Load Application Dialog will enable you to specify the serial or POE program you wish to run. For information on filling in the fields of the Load Application Dialog, refer to “Loading and Starting a New Application” on page 10.</p>
You want to examine an application that is already running.	<p>Select the Connect to a running application option button and click the OK command button.</p> <p>Doing this closes the Welcome Dialog, and opens the Connect Application Dialog. The Connect Application Dialog will enable you to specify the serial or POE program to which you want to connect. For information on using the Connect Application Dialog to connect to an application, refer to “Connecting to a Running Application” on page 13.</p>

If:	Then:
<p>You do not want to make the decision between whether to load a new, or connect to an existing, application at this time.</p>	<p>Click on the Cancel command button.</p> <p>Doing this closes the Welcome Dialog and opens the Performance Collection Tool's Main Window. Since you have neither loaded a new application, nor connected to an existing application, the Main Window will not provide any application information at this time. You will eventually have to load a new application (as described in "Loading and Starting a New Application") or connect to an existing application (as described in "Connecting to a Running Application" on page 13).</p>

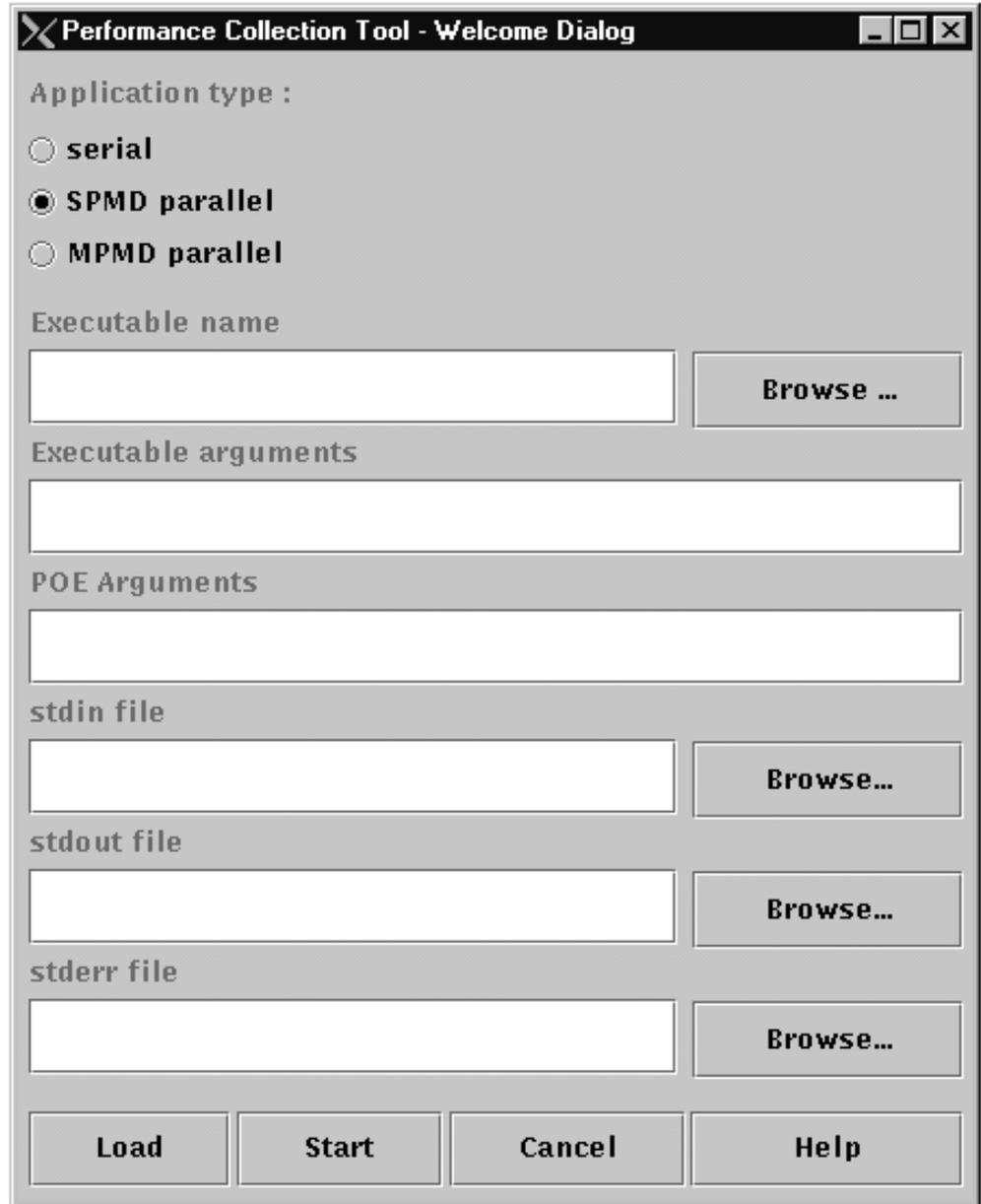
Loading and Starting a New Application

Load Application Dialog

Using the Load Application Dialog, you can load and optionally start execution of, a serial or POE program. To do this:

1. If the Load Application Dialog is not already open, select the **Application → Connect...** menu item off the Main Window's menu bar.

Doing this opens the Load Application Dialog.



2. Indicate the type of application you will be loading.

If you want to load:	Then:
A serial program.	Select the serial option button.
A POE application that follows the Single Program Multiple Data (SPMD) model.	Select the SPMD parallel option button.
A POE application that follows the Multiple Program Multiple Data (MPMD) model.	Select the MPMD parallel option button.

3. Indicate the executable file(s) you want to load.

If you are loading:	Then:
A serial program or an SPMD parallel program.	In the Executable name field, type in the absolute path to the executable file you want to load. If you are unsure of the executable file name or path, click on the Browse... command button to the right of the Executable name field. Doing this will display a standard file selection dialog that you can use to locate the executable file.
An MPMD parallel program.	<p>In the POE commands file field, type in the absolute path to a POE commands file (which lists the individual programs to load). For more information on creating a POE commands file for loading multiple programs, refer to the manual <i>IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment</i>.</p> <p>If you are unsure of the POE commands file name or path, click on the Browse... command button to the right of the POE commands file field. Doing this will open a standard file selection dialog that you can use to locate the POE commands file.</p>

4. In the **Executable arguments** field, type in any arguments that you want to pass to the executable you are loading. Note that these are not POE arguments, which must instead be typed into another field of this dialog. If you do not wish to pass any arguments to the executable, leave the **Executable arguments** field blank.

Note: If you are loading an MPMD parallel program, the **Executable args** field will be grayed out (input disabled). If you want to pass arguments to the executables of the MPMD program, you must instead specify the arguments in your POE commands file. For more information on POE commands files, refer to the manual *IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment*.

5. If you are loading a POE program, and would like to supply POE arguments to influence how the executable you are loading will run, type the arguments in the **POE arguments** field. For a complete description of POE arguments, refer to the manual *IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment*. If you are loading a serial program, the **POE arguments** field will be grayed out (input disabled); you cannot and need not type any information in the field.
6. If you want the application you are loading to read standard input from a file, type the name of the file in the **stdin file** field. To select a file for redirected standard input using a file selection dialog, click on the **Browse...** button to the right of the **stdin file** field. If you do not need to redirect standard input, leave the **stdin file** field empty.
7. If you want to redirect standard output from an application to a file, type the name of the file in the **stdout file** field. To select a file for redirected standard output using a file selection dialog, click on the **Browse...** button to the right of the **stdout file** field. If you leave this field empty, standard output will be displayed in the I/O Console Window as described in "Examining Output From,

and Sending Input To, the Application” on page 23. If you do not need to redirect standard output, leave the **stdout file** field empty.

8. If you would like to redirect standard error from the application to a file, type the name of the file in the **stderr file** field. To select a file for redirected standard error using a file selection dialog, click on the **Browse...** button to the right of the **stderr file** field. If you leave this field empty, standard error will be displayed in the I/O Console Window as described in “Examining Output From, and Sending Input To, the Application” on page 23. If you do not need to redirect standard error, leave the **stderr file** field empty.
9. Once you have filled in the various fields of the **Load Application** dialog, you can choose to either merely load the application or load the application and start its execution. If you chose to merely load the application, execution of the process(es) will be stopped before the first executable instruction. This enables you to add performance collection probes to the application process(es) first before then starting execution.

If :	Then:
You want to merely load the application.	<p>Click on the Load command button.</p> <p>Doing this loads the application process(es) and closes the Load Application Dialog. The process(es) will be loaded in a “stopped state” – execution will be stopped before the first executable instruction. This enables you to add performance collection probes to one or more application processes before starting execution. For instructions on adding performance collection probes to one or more processes, refer to “Specifying MPI Trace Data to Be Collected” on page 24, “Adding User Markers to Processes” on page 33, and “Specifying Profile Data To Be Collected” on page 36. For instructions on starting a loaded application, refer to “Starting and Stopping Application Execution” on page 20.</p>
You want to load the application and start its execution.	<p>Click on the Start command button.</p> <p>Doing this loads the application and starts its execution. The Load Application Dialog closes.</p>

Unless you have already selected the type of probe data you plan to collect, the Probe Data Selection Dialog will open after the Load Application Dialog closes. For information on filling in the fields of the Probe Data Selection Dialog, refer to “Selecting (At Tool Start-up Time) the Type of Probe Data To Be Collected” on page 16.

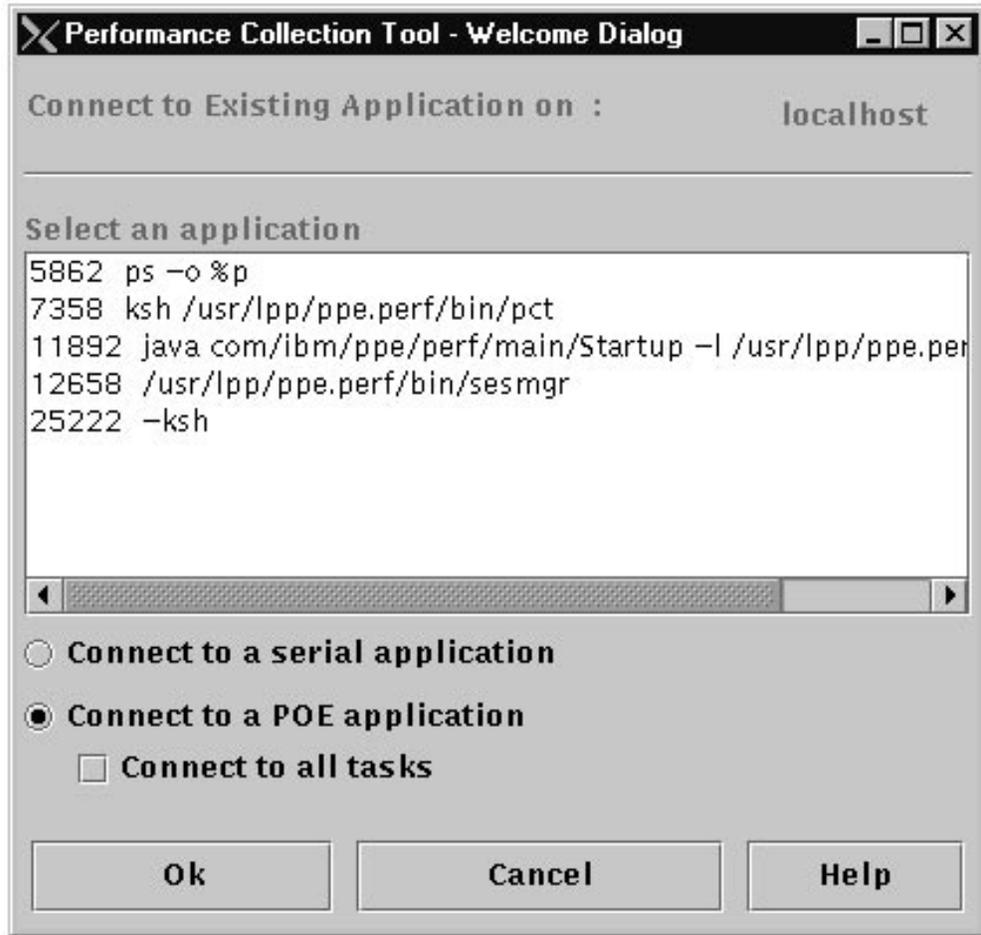
Connecting to a Running Application

Connect Application Dialog

Using the Connect Application Dialog, you can connect the Performance Collection Tool to a serial or POE program that is already executing. To do this:

1. If the Connect Application Dialog is not already open, select the **Application → Connect...** menu item off the Main Window’s menu bar.

Doing this opens the Connect Application Dialog



2. In the Connect Application Dialog, select the serial or parallel POE application you want to connect to by clicking on its process entry in the **Select an application** list box. To connect to a POE application, you should select the controlling POE process (the executing instance of the POE command).
3. Use the Connect Application Dialog's option buttons to indicate whether the application you have selected to connect to is a serial or POE application.

If you are connecting to:	Then:
A serial application.	Select the Connect to a serial application option button.

If you are connecting to:	Then:	
A parallel application	Select the Connect to a POE application option button. Doing this activates the Connect to all tasks check box so that it is now selectable	
	If you would like to connect to:	Then:
	All processes in the POE application.	Select the Connect to all tasks check box.
	Only the controlling, "home node" POE process.	Do not select the Connect to all tasks check box. You can later connect to individual tasks of the POE application as described in "Connecting to One or More Processes of the Loaded Application".

4. Click the **OK** command button.

Doing this connects the Performance Collection Tool to the selected serial or POE application, and closes the Connect Application Dialog.

Unless you have already selected the type of probe data you plan to collect, the Probe Data Selection Dialog will open after the Connect Application Dialog closes. For information on filling in the fields of the Probe Data Selection Dialog, refer to "Selecting (At Tool Start-up Time) the Type of Probe Data To Be Collected" on page 16.

Connecting to One or More Processes of the Loaded Application

When you connect to a POE application as described in "Connecting to a Running Application" on page 13, you can choose to connect only to the controlling, "home node", POE process. If you choose to do this, you are not connected to any of the POE application's individual tasks. You might choose to do this for performance reasons, since the POE application could have many tasks.

To connect to one or more processes of an application that has already been loaded into the Performance Collection Tool, follow the instructions below. If the application has not been loaded into the Performance Collection Tool, refer to "Loading and Starting a New Application" on page 10 instead.

1. Select one or more of the unconnected processes listed in the Main Window's Process List area. The process' entry in the "Connected" column will read either "true" or "false" to indicate whether it is connected.

If you want to:	Then:
Connect to a single process or selected processes of the application.	Click on the process entry in the Process List area. To select more than one process, hold the shift or control key down while you click on each process entry.
Connect to all processes in the application.	Select the Process → Select all menu item off the Main Window's menu bar.

Selecting a process entry in the Main Window's Process List area highlights the entry to show that the process is selected. Once one or more processes are selected, be aware that certain further actions you can request using the

Performance Collection Tool (such as connecting or adding data collection probes) will be performed on the selected processes only.

2. Select the **Process → Connect** menu item off the Main Window's menu bar. Doing this connects the Performance Collection Tool to the selected process(es).

Once a process is connected, the Performance Collection Tool can:

- start or stop its execution (as described in “Starting and Stopping Application Execution” on page 20).
- add data collection probes to the process (as described in “Specifying MPI Trace Data to Be Collected” on page 24, “Adding User Markers to Processes” on page 33, and “Specifying Profile Data To Be Collected” on page 36).
- remove data collection probes from the process (as described in “Removing Performance Collection Probes From One or More Processes” on page 39).
- terminate the process (as described in “Terminating Connected Processes” on page 42) When you no longer need to perform such tasks on the process, you can disconnect it (as described in “Disconnecting From One or More Processes of the Loaded Application” on page 20).

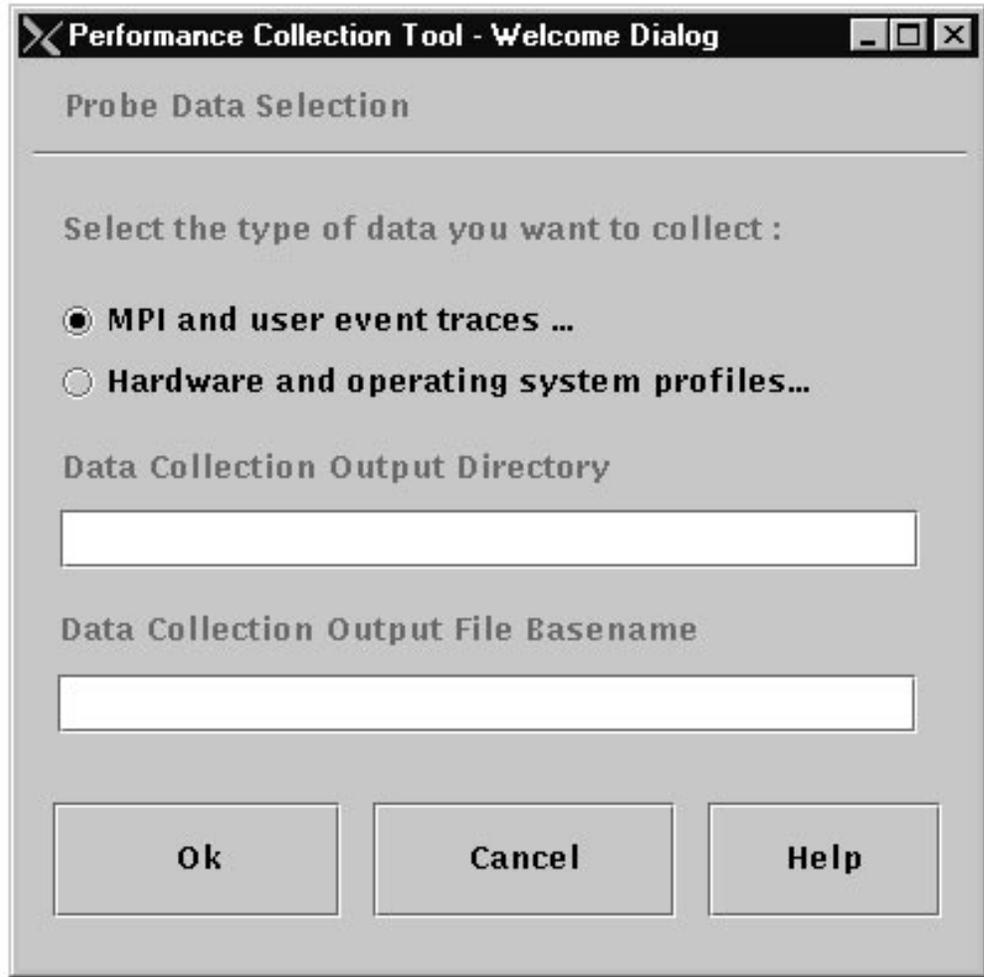
Selecting (At Tool Start-up Time) the Type of Probe Data To Be Collected

Probe Data Selection Dialog

The Performance Collection Tool is capable of collecting two different types of information. It can collect:

- MPI and user event traces for analysis using the **utestats** utility or a graphical visualization tool like Jumpshot (a public domain tool developed at Argonne National Laboratory). For more information on the **utestats** utility, as well as utilities for converting the AIX trace files created by the Performance Collection Tool into a format readable by **utestats** and Jumpshot, refer to “Chapter 4. Creating, Converting, and Viewing Information Contained In, UTE Interval Files” on page 91.
- Hardware and operating system profiles for analysis within the Profile Visualization Tool. For more information on the Profile Visualization Tool, refer to “Chapter 3. Using the Profile Visualization Tool” on page 65.

Upon starting the Performance Collection Tool (as described in “Starting the Performance Collection Tool” on page 8), you are given the choice to either load a new application using the Load Application Dialog (as described in “Loading and Starting a New Application” on page 10) or connect to an existing application using the Connect Application Dialog (as described in “Connecting to a Running Application” on page 13). After clicking the **OK** button on either of these dialogs, the Probe Data Selection Dialog opens.



If the Probe Data Selection Dialog is not already open, refer to “Selecting (After Tool Startup Time) the Type of Probe Data To Be Collected” on page 18 instead of the following instructions.

1. Specify the type of data you want to collect by selecting either the **MPI and user event traces** or the **Hardware and operating system profiles** option button.

If:	Then:
You want to collect MPI or custom user event traces for analysis using the utestats utility or a graphical visualization tool like Jumpshot.	Select the MPI and user event traces option button.
You want to collect hardware and operating system profiles for analysis within the Profile Visualization Tool.	Select the Hardware and operating system profiles option button.

2. The data collected by the Performance Collection Tool will be stored to an output directory on each host machine where a connected process is running. Type the name of the desired output directory in the **Data Collection Output Directory** field.
3. The data collected by the Performance Collection Tool will be saved as a file on each host machine where an instrumented process is running. The file name

will consist of a "base name" that you supply followed by a node-specific suffix supplied by the Performance Collection Tool. Type the "base name" for the output file(s) in the **Data Collection Output File Basename** field.

4. Click the **OK** command button.

Doing this closes the Probe Data Selection Dialog and opens the Main Window. The Probe Selection area of the Main Window will now show options only for the type of data – MPI trace or hardware/operating system profiles – that you have selected. When you add performance collection probes to one or more processes (as described in "Specifying MPI Trace Data to Be Collected" on page 24, "Adding User Markers to Processes" on page 33, and "Specifying Profile Data To Be Collected" on page 36), the data collected will be saved, on each host running instrumented processes, to the directory specified with the file basename specified.

Note: You can select the type of data to collect only one time per load or connect.

Selecting (After Tool Startup Time) the Type of Probe Data To Be Collected

Select Data Collection Output Location Dialog

The Performance Collection Tool is capable of collecting two different types of information. It can collect:

- MPI and user event traces for analysis using the **utestats** utility or a graphical visualization tool like Jumpshot (a public domain tool developed at Argonne National Lab). For more information on the **utestats** utility, as well as utilities for converting the AIX trace files created by the Performance Collection Tool into a format readable by **utestats** and Jumpshot, refer to "Chapter 4. Creating, Converting, and Viewing Information Contained In, UTE Interval Files" on page 91.
- Hardware and operating system profiles for playback within the Profile Visualization Tool. For more information on the Profile Visualization Tool, refer to "Chapter 3. Using the Profile Visualization Tool" on page 65.

Upon starting the Performance Collection Tool (as described in "Starting the Performance Collection Tool" on page 8), you are presented with a number of dialogs requesting information before the Main Window is displayed. One of these dialogs is the Probe Data Selection Dialog. If the Probe Data Selection Dialog is open, refer to "Selecting (At Tool Start-up Time) the Type of Probe Data To Be Collected" on page 16 instead of the following instructions.

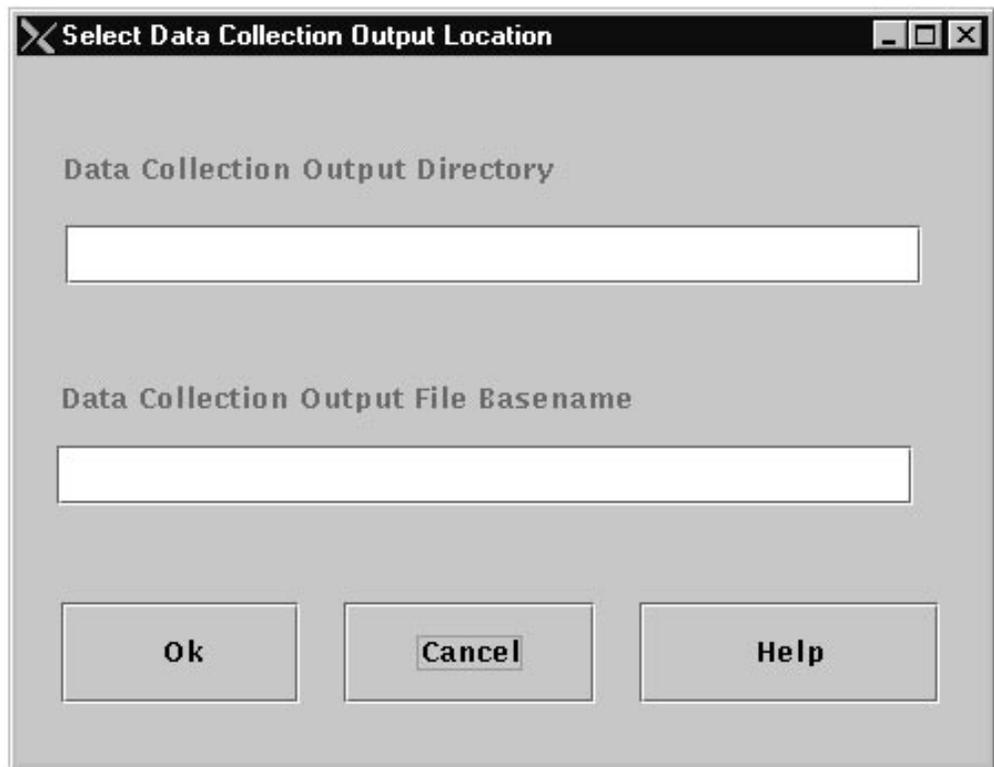
1. Select the type of data you want to collect by selecting either the **Probe → Collect event trace data...** or **Probe → Collect profile data...** menu item off the Main Window's menu bar.

If these options are grayed out (unselectable), this means that you have already specified the type of probe data that you wish to collect. You can select the type of data to collect only one time per load and connect. If you want to change your selection, you will have to exit the tool (as described in "Exiting the Performance Collection Tool" on page 42) and then reload or reconnect the application.

If:	Then:
You want to collect MPI or custom user event traces for analysis using the utestats utility or a graphical visualization tool like Jumpshot.	Select the Probe → Collect event trace data... menu item off the Main Window's menu bar.

If:	Then:
You want to collect hardware and operating system profiles for analysis within the Profile Visualization Tool.	Select the Probe → Collect profile data... menu item off the Main Window's menu bar.

Selecting either the **Probe → Collect event trace data...** or **Probe → Collect profile data...** menu item opens the Select Data Collection Output Location Dialog.



2. The data collected by the Performance Collection Tool will be stored to an output directory on each host machine where an instrumented process is running. Type the name of the desired output directory in the **Data Collection Output Directory** field.
3. The data collected by the Performance Collection Tool will be saved as a file on each host machine where a connected process is running. The file name will consist of a "base name" that you supply followed by a node-specific suffix supplied by the Performance Collection Tool. Type the "base name" for the output file(s) in the **Data Collection Output File Basename** field.
4. Click the **OK** command button.
 Doing this closes the Select Data Collection Output Location Dialog. The Probe Selection area of the Main Window will now show options only for the type of data – MPI trace or hardware/operating system profiles – that you have selected. When you add data collection probes to one or more processes (as described in "Specifying MPI Trace Data to Be Collected" on page 24, "Adding User Markers to Processes" on page 33, and "Specifying Profile Data To Be Collected" on page 36), the data collected will be saved, on each host running instrumented processes, to the directory specified with the file basename specified.

Starting and Stopping Application Execution

Once a process of the loaded application is connected, you can use the Performance Collection Tool to start and stop execution of the process as desired. A process is connected if the "Connected" column in the Process List area reads "true". For information on connecting processes, refer to "Connecting to One or More Processes of the Loaded Application" on page 15.

- To start execution of the connected processes in the currently loaded application, select the **Application → Start** menu item off the Main Window's menu bar. If the **Application → Start** menu item is grayed out (unselectable), this means that no processes are connected or the connected processes are already executing.
- To stop execution of the connected processes in the currently loaded application, select the **Application → Suspend** menu item off the Main Window's menu bar. If the **Application → Suspend** menu item is grayed out (unselectable), this means that no processes are connected or the connected processes are already stopped. When you stop execution of a process, the process' entry in the "Status" column will read "suspended".

When you start execution of a process, the process' entry in the "Status" column will read "running".

Disconnecting From One or More Processes of the Loaded Application

To disconnect from one or more connected processes of the loaded application:

1. Select the connected process(es) you want to disconnect.

If:	Then:
You want to disconnect all connected processes in the application.	Select the Process → Select All Connected menu item off the Main Window's menu bar.
You want to disconnect individual connected processes.	In the Main Window's Process List area, click on the connected process(es) you want to disconnect. A process is connected if the "Connected" column in the Process List area reads "true".

Selecting one or more connected processes as described in the preceding table, highlights the selected process(es) in the Main Window's Process List area. Once one or more processes are selected, be aware that certain further actions you can request using the Performance Collection Tool (such as disconnecting processes) will be performed for the selected processes only.

2. Select the **Process → Disconnect** menu item off the Main Window's menu bar. The Performance Collection Tool disconnects the selected process(es). The "Connected" column in the Process List area reads "false" for each disconnected process.

Viewing Application Source Code

View Source Window

The Source Tree area of the Performance Collection Tool's Main Window provides a hierarchical view of the source code associated with the currently selected process. By selecting entries in this tree, you can add or remove data collection probes to/from the process(es) selected in the Process List area of the Performance Collection Tool's Main Window. (For more information on adding and removing data collection probes, refer to "Specifying MPI Trace Data to Be Collected" on page 24, "Adding User Markers to Processes" on page 33, "Specifying Profile Data To Be

Collected” on page 36, and “Removing Performance Collection Probes From One or More Processes” on page 39.) Since the information in the Source Tree area is just a summary of the source code structure, you may, if you require more detailed information, want to display the actual source code. To do this:

1. If the desired Source Tree is not already displayed, go to the Process List area and click on the process associated with the source code you want to view. Doing this displays the Source Tree for the selected process in the Source Tree area.
2. In the Source Code tree, do one of the following:

Either:	Or:	Or:
double click on the name of the file whose source code you want to view.	<ol style="list-style-type: none"> a. Right click on the name of the file whose source code you want to view. Doing this displays a pop-up menu. b. Select View source... off the pop-up menu. 	<ol style="list-style-type: none"> a. Click on the name of the file whose source code you want to view. Doing this highlights the file’s entry to show that it is selected. b. Select the Source → View Source... menu item off the Main Window’s menu bar.

Doing any one of the above opens the View Source Window.

Note: The Performance Collection Tool searches for the source file using a search path that, by default, is the directory in which the tool was started. If the Performance Collection Tool is unable to locate the source file, it is because the source file is not in the search path. To modify the search path, refer to “Setting User Preferences”

Screen Capture (Showing the **View Source Window**) Will Be Inserted Here.

When you are through viewing the selected source code, you can close the View Source Window by selecting its **File → Close** menu item.

Setting User Preferences

Preferences Dialog

There are a number of user preferences you can set using the Preferences Dialog. Specifically, you can:

- determine whether the scripting commands produced as a result of your manipulating the Performance Collection Tool’s graphical user interface are displayed in the Main Window’s Information Area and/or saved to a file. Since your actions on the graphical user interface are translated into the same scripting commands you would issue to the Performance Collection Tool’s command-line interface, this enables you to:
 - more quickly learn the scripting commands or
 - create reusable scripts for the command-line interface.

For more information on the Performance Collection Tool’s command-line interface, refer to “Using the Performance Collection Tool’s Command-Line Interface” on page 43.

- Specify the search path that the Performance Collection Tool uses to locate source code files to be displayed within the View Source Window. Refer to “Viewing Application Source Code” on page 20 for more information on the View Source Window.

To set user preferences:

1. Select the **Application → User Preferences...** menu off the Main Window’s menu bar.

Doing this opens the Preferences Dialog.

Screen Capture (Showing the **Preferences Dialog**) Will Be Inserted Here.

2. If you want to display the script commands that result from your interface interactions, select the **display script commands in information window** check box.
3. If you would like to save the script commands that result from your interface interactions to a file:
 - a. Select the **save script commands to a file** check box.
Doing this activates the **save script commands to a file** text entry field, so that you can now type in it.
 - b. Type a file name in the **save script commands to a file** text entry field.
To select a file using a standard file selection dialog, click on the **Browse...** command button to the right of the text entry field.
4. By default, the search path used to locate source files displayed in the View Source Window consists only of the directory in which the Performance Collection Tool was started. If you want to modify this path, type the new path or paths in the **source path for displaying source files** text entry field. To separate paths, use the pipe character (|).
5. Click the **OK** command button.
Doing this records your preferences and closes the Preferences Dialog.

Searching for Functions in the Application Source Code

Function Search Dialog

The Function Search Dialog enables you to use a search string to locate functions within the source tree. To do this:

1. If the Source Tree is not already displayed, go to the Process List area and click on the process associated with the source code you want to search.
Doing this displays the source tree for the selected process in the Source Tree area.
2. Select the **Source → Find function...** menu item off the Main Window’s menu bar.

Doing this opens the Function Search Dialog.

Screen Capture (Showing the **Function Search Dialog**) Will Be Inserted Here.

3. Type the search string that identifies the function(s) you wish to locate in the Source Tree, and click the **Find** command button.
Doing this displays any function names that match the supplied search string. If there are no functions listed, this means that no functions matching your search string were found; repeat this step using a different search string.

Screen Capture (Showing the **Function Search Dialog** including a search string) Will Be Inserted Here.

4. In the **Search Results** area, click on the name of the function you want to locate in the Source Tree.
Doing this highlights the function name to show that it is selected.
5. Locate the function in the Source Tree by clicking either the **Goto** or **Apply** command button as described in the following table.

If you want to:	Then:
Locate the function in the Source Tree and close the Function Search Dialog.	<p>Click the Goto command button.</p> <p>Doing this highlights the function in the Source Tree. The Source Tree is expanded if necessary, and the Function Search Dialog closes.</p>
Locate the function in the Source Tree, but keep the Function Search Dialog open.	<p>Click the Apply command button.</p> <p>Doing this highlights the function in the Source Tree. The Source Tree is expanded if necessary.</p> <p>Since the Function Search Dialog remains open, you can repeat the steps outlined above to perform another search. To close the Function Search Dialog, click the Done command button.</p>

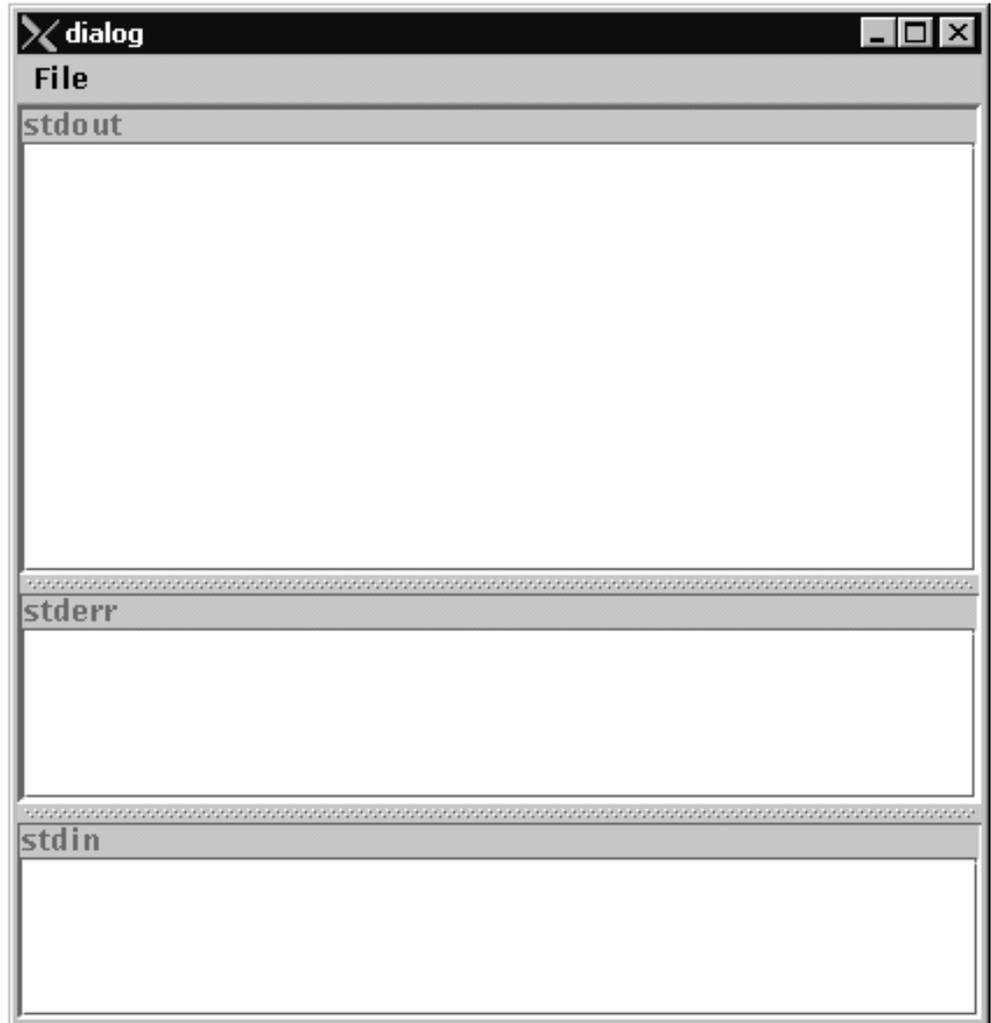
Examining Output From, and Sending Input To, the Application

I/O Console Window

The I/O Console Window enables you to examine standard output and standard error from, and send standard input to, the loaded application (provided you did not redirect the standard output, standard error, or standard input when loading the application as described in “Loading and Starting a New Application” on page 10). If you redirected standard output, it will not appear in the I/O Console Window. If you redirected standard error, it will not appear in the I/O Console Window. If you redirected standard input, you cannot send text to standard input using the I/O Console Window. To open the I/O Console Window from the Application pulldown menu on the Main Window.

If the I/O Console Window is not already open, select the **Application → Show I/O Console...** menu item off the Main Window’s menu bar.

Doing this opens the I/O Console Window.



The top area of the I/O Console Window displays standard output from the loaded application, while the middle area displays standard error.

To send standard input to the application, type the standard input text into the bottom area of the I/O Console Window and select the **File → Submit Stdin** menu item off the I/O Console Window's menu bar. To send an end-of-file character to the application, select the **File → Submit Stdin EOF** menu item off the I/O Console Window's menu bar.

To close the I/O Console Window when you are through examining output from, or sending input to, the loaded application, select the **File → Close** menu item off the I/O Console Window's menu bar.

Specifying MPI Trace Data to Be Collected

MPI Events Tab



Using the MPI Events Tab of the Main Window's Probe Selection Area, you can specify which MPI events you want to collect data for. The MPI Events Tab is displayed in the Main Window's Probe Selection Area only if you have indicated that you want to collect MPI or custom user event traces.

To specify which MPI events you want to collect:

1. Select the connected process(es) you want to instrument with performance collection probes. You can either instrument all connected processes of the application, or individual connected processes.

If:	Then:
You want to instrument all connected processes in the application.	Select the Process → Select All Connected menu item off the Main Window's menu bar.
You want to instrument individual connected processes.	In the Main Window's Process List area, click on the connected process(es) to which you want to add the performance collection probes. A process is connected if the "connected" column in the Process List area reads "True".

Selecting one or more connected processes, as described in the preceding table, highlights the selected process(es) in the Main Window's Process List area. Once one or more processes are selected, be aware that certain further actions you can request using the Performance Collection Tool (such as adding performance collection probes) will be performed on the selected processes only.

2. In the Main Window's Source Tree area, select the location(s) where you want the performance collection probes added.

If:	Then:
You want to instrument the entire executable.	<p>Click on the executable name in the source tree.</p> <p>Doing this highlights the executable name to show that it is selected.</p>
You want to instrument one or more of the executable's source files.	<p>Click on the source file name(s) in the Source Tree. If the source file names are not visible, you may need to expand the source tree. A handle icon to the left of entries in the Source Tree indicates that there is additional, unexpanded, information under that entry. To expand the Source Tree to see the additional information, click on the handle icon.</p> <p>Selecting a source file name in the source tree highlights the file name to show that it is selected.</p>
You want to instrument one or more functions.	<p>Click on the function name(s) in the source tree. If the function names are not visible, you may need to expand the source tree. A handle icon to the left of entries in the Source Tree indicate that there is additional, unexpanded, information under that entry. To expand the Source Tree to see the additional information, click on the handle icon.</p> <p>Selecting a function name in the Source Tree highlights the name to show that it is selected.</p>

3. In the MPI Events Tab, select the MPI events you want to collect data for.

If you want to collect data for:	Then:
All MPI events	<p>Click on the All MPI events check box.</p> <p>Doing this automatically selects all the remaining check boxes in the MPI Events Tab. The Performance Collection Tool will now generate trace records for all of the MPI functions listed in this table.</p>

If you want to collect data for:	Then:
Blocking collective communication	<p>Click on the Collective communication check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Allgather • MPI_Allgatherv • MPI_Allreduce • MPI_Alltoall • MPI_Alltoallv • MPI_Barrier • MPI_Bcast • MPI_Gather • MPI_Gatherv • MPI_Op_create • MPI_Op_free • MPI_Reduce • MPI_Reduce_scatter • MPI_Scan • MPI_Scatter • MPI_Scatterv
Point-to-point communication	<p>Click on the Point-to-point communication check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Bsend • MPI_Bsend_init • MPI_Buffer_attach • MPI_Buffer_detach • MPI_Cancel • MPI_Get_count • MPI_Ibsend • MPI_Iprobe • MPI_Irecv • MPI_Irsend • MPI_Isend • MPI_Issend • MPI_Probe • MPI_Recv • MPI_Recv_init • MPI_Request_free • MPI_Rsend • MPI_Rsend_init • MPI_Send • MPI_Send_init • MPI_Sendrecv • MPI_Sendrecv_replace • MPI_Ssend • MPI_Ssend_init • MPI_Start_mpi • MPI_Startall • MPI_Test_cancelled

If you want to collect data for:	Then:
One-sided communication	<p>Click on the One-sided communication check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Accumulate • MPI_Get • MPI_Put • MPI_Win_complete • MPI_Win_create • MPI_Win_create_errhandler • MPI_Win_create_keyval • MPI_Win_delete_attr • MPI_Win_fence • MPI_Win_free • MPI_Win_free_keyval • MPI_Win_get_attr • MPI_Win_get_errhandler • MPI_Win_get_group • MPI_Win_lock • MPI_Win_post • MPI_Win_set_attr • MPI_Win_set_errhandler • MPI_Win_start • MPI_Win_unlock
Communication groups	<p>Click on the Communication groups check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Comm_group • MPI_Group_compare • MPI_Group_difference • MPI_Group_excl • MPI_Group_free • MPI_Group_incl • MPI_Group_intersection • MPI_Group_range_excl • MPI_Group_range_incl • MPI_Group_rank • MPI_Group_size • MPI_Group_translate_ranks • MPI_Group_union

If you want to collect data for:	Then:
Topologies	<p>Click on the Topologies check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Cart_coords • MPI_Cart_create • MPI_Cart_get • MPI_Cart_map • MPI_Cart_rank • MPI_Cart_shift • MPI_Cart_sub • MPI_Cartdim_get • MPI_Dims_create • MPI_Graph_create • MPI_Graph_get • MPI_Graph_map • MPI_Graph_neighbors • MPI_Graph_neighbors_count • MPI_Graphdims_get • MPI_Topo_test
Test operations	<p>Click on the Test check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Test • MPI_Testall • MPI_Testany • MPI_Testsome • MPI_Win_test
Nonblocking collective communication	<p>Click on the Nonblocking collective communication check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Iallgather • MPI_Iallgatherv • MPI_Iallreduce • MPI_Ialltoall • MPI_Ialltoallv • MPI_Ibarrier • MPI_Ibcast • MPI_Igather • MPI_Igatherv • MPI_Ireduce • MPI_Ireduce_scatter • MPI_Iscan • MPI_Iscatter • MPI_Iscatterv

If you want to collect data for:	Then:
Environmental	<p>Click on the Environmental check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Abort • MPI_Comm_create_errhandler • MPI_Comm_get_errhandler • MPI_Comm_set_errhandler • MPI_Errhandler_create • MPI_Errhandler_free • MPI_Errhandler_get • MPI_Errhandler_set • MPI_Error_class • MPI_Error_string • MPI_File_create_errhandler • MPI_File_get_errhandler • MPI_File_set_errhandler • MPI_Finalize • MPI_Get_processor_name • MPI_Get_version • MPI_Init • MPI_Initialized • MPI_Pcontrol • MPI_Wtick • MPI_Wtime
Derived data types	<p>Click on the Data type check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Address • MPI_Alloc_mem • MPI_Free_mem • MPI_Get_elements • MPI_Pack • MPI_Pack_size • MPI_Type_commit • MPI_Type_contiguous • MPI_Type_create_darray • MPI_Type_create_keyval • MPI_Type_create_subarray • MPI_Type_delete_attr • MPI_Type_dup • MPI_Type_extent • MPI_Type_free • MPI_Type_free_keyval • MPI_Type_get_attr • MPI_Type_get_contents • MPI_Type_get_envelope • MPI_Type_hindexed • MPI_Type_hvector • MPI_Type_indexed • MPI_Type_lb • MPI_Type_set_attr • MPI_Type_size • MPI_Type_struct • MPI_Type_ub • MPI_Type_vector • MPI_Unpack

If you want to collect data for:	Then:
File operations	<p>Click on the File check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_File_close • MPI_File_delete • MPI_File_get_amode • MPI_File_get_atomicsity • MPI_File_get_byte_offset • MPI_File_get_group • MPI_File_get_info • MPI_File_get_position • MPI_File_get_position_shared • MPI_File_get_size • MPI_File_get_type_extent • MPI_File_get_view • MPI_File_iread • MPI_File_iread_at • MPI_File_iread_shared • MPI_File_iwrite • MPI_File_iwrite_at • MPI_File_iwrite_shared • MPI_File_open • MPI_File_preallocate • MPI_File_read • MPI_File_read_all • MPI_File_read_all_begin • MPI_File_read_all_end • MPI_File_read_at • MPI_File_read_at_all • MPI_File_read_at_all_begin • MPI_File_read_at_all_end • MPI_File_read_ordered • MPI_File_read_ordered_begin • MPI_File_read_ordered_end • MPI_File_read_shared • MPI_File_seek • MPI_File_seek_shared • MPI_File_set_atomicsity <p>(continued...)</p>

If you want to collect data for:	Then:
File operations (continued)	<ul style="list-style-type: none"> • MPI_File_set_info • MPI_File_set_size • MPI_File_set_view • MPI_File_sync • MPI_File_write • MPI_File_write_all • MPI_File_write_all_begin • MPI_File_write_all_end • MPI_File_write_at • MPI_File_write_at_all • MPI_File_write_at_all_begin • MPI_File_write_at_all_end • MPI_File_write_ordered • MPI_File_write_ordered_begin • MPI_File_write_ordered_end • MPI_File_write_shared • MPI_Register_datarep
Info operations	<p>Click on the Info check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Info_create • MPI_Info_delete • MPI_Info_dup • MPI_Info_free • MPI_Info_get • MPI_Info_get_nkeys • MPI_Info_get_nthkey • MPI_Info_get_valuelen • MPI_Info_set

If you want to collect data for:	Then:
Communicators	<p>Click on the Communicators check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Attr_delete • MPI_Attr_get • MPI_Attr_put • MPI_Comm_compare • MPI_Comm_create • MPI_Comm_create_keyval • MPI_Comm_delete_attr • MPI_Comm_dup • MPI_Comm_free • MPI_Comm_free_keyval • MPI_Comm_get_attr • MPI_Comm_rank • MPI_Comm_remote_group • MPI_Comm_remote_size • MPI_Comm_set_attr • MPI_Comm_size • MPI_Comm_split • MPI_Comm_test_inter • MPI_Intercomm_create • MPI_Intercomm_merge • MPI_Keyval_create • MPI_Keyval_free
Wait operations	<p>Click on the Wait check box. The Performance Collection Tool will now generate trace records for the following MPI functions:</p> <ul style="list-style-type: none"> • MPI_Wait • MPI_Waitall • MPI_Waitany • MPI_Waitsome • MPI_Win_wait

4. Click the **Add** command button below the MPI Events Tab, or select the **Probe** → **Add** menu item off the Main Window's menu bar.

Doing this adds the probes you have selected in the MPI Events Tab to the process(es) selected in the Process List. The selected data will be collected for the locations selected in the Source Tree. The trace output file will be stored to the location you specified in either the Probe Data Selection Dialog or the Select Data Collection Output Location Dialog.

Adding User Markers to Processes

User Markers Tab

As described in “Specifying MPI Trace Data to Be Collected” on page 24, you can specify, by selecting various check boxes in the MPI Events Tab, the type of MPI events you want to collect data for. In addition to adding trace data probes for collecting standard MPI events (such as collective, point-to-point, or one-sided communication events), you can also add *user markers* to processes. *User markers* are special types of probes that you can install at specific instrumentation points in your application code. With user markers, you can:

- Mark events of interest (such as program function calls) using a *simple marker*. A simple marker will appear in the trace record as a single point; its position gives you a frame of reference when analyzing the trace record in a graphical visualization tool like Jumpshot.
- Mark a state of interest using a *begin state marker* and an *end state marker*. A state marked by begin and end state markers will appear in the trace record as a region. Like the simple markers, this gives you a frame of reference when analyzing the trace record in a graphical visualization tool like Jumpshot.
- Force tracing on or off using a *trace on marker* or a *trace off marker*. Since you can add MPI probes only at a program, file, or function level (meaning that the entire program, file, or function will be traced), the trace on/trace off markers give you more control over which part of your program is traced.

To add user markers to one or more processes:

1. Select the connected process(es) you want to instrument.

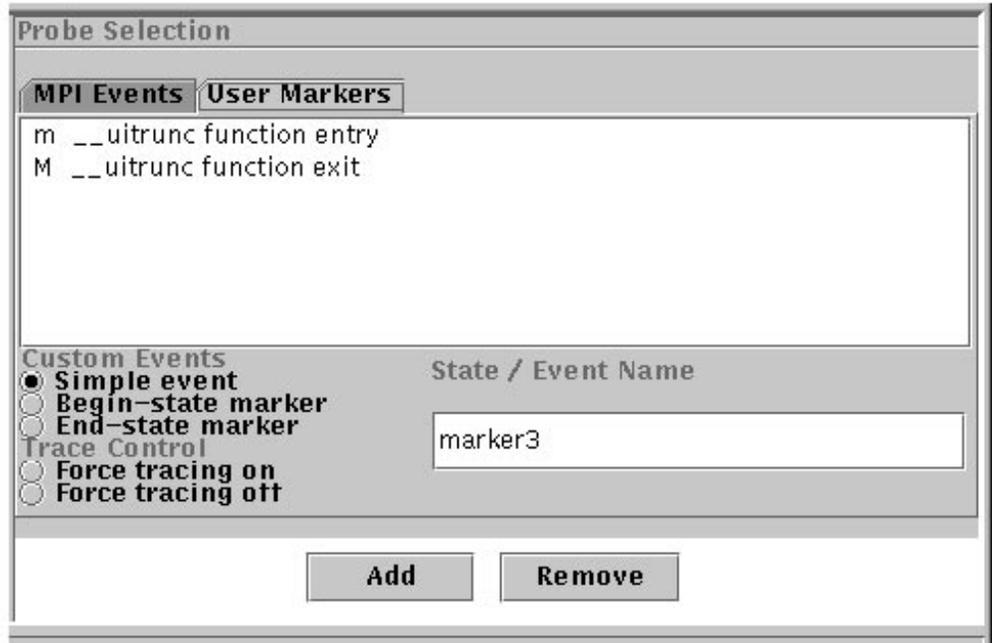
If:	Then:
You want to instrument all connected processes in the application.	Select the Process → Select All Connected menu item off the Main Window's menu bar.
You want to instrument individual connected processes.	In the Main Window's Process List area, click on the connected process(es) to which you want to add the user markers. A process is connected if the "Connected" column in the Process List area reads "True".

Selecting one or more connected processes, as described in the preceding table, highlights the selected process(es) in the Main Window's Process List area. Once one or more processes are selected, be aware that certain further actions you can request using the Performance Collection Tool (such as adding user markers) will be performed on the selected processes only.

2. In the Main Window's Source Tree area, click on the name(s) of the source file(s) you want to instrument. If the source file names are not visible, you may need to expand the source tree. A handle icon to the left of the entries in the Source Tree indicates that there is additional, unexpanded, information under the entry. To expand the Source Tree to see the additional information, click on the handle icon.

Selecting a source file name in the Source Tree highlights the file name to show that it is selected.

3. In the Main Window's Probe Selection area, click on the **User Markers** tab.



The User Markers Tab displays the instrumentation points associated with the selected source file(s). The instrumentation points are the various locations in the source code where you can insert the user markers. If you are unfamiliar with the program you are instrumenting, and need more detailed information on the source code than is presented in this dialog, refer to “Viewing Application Source Code” on page 20.

4. Add the simple marker, begin and end state markers, or trace on and trace off markers as described in the following table.

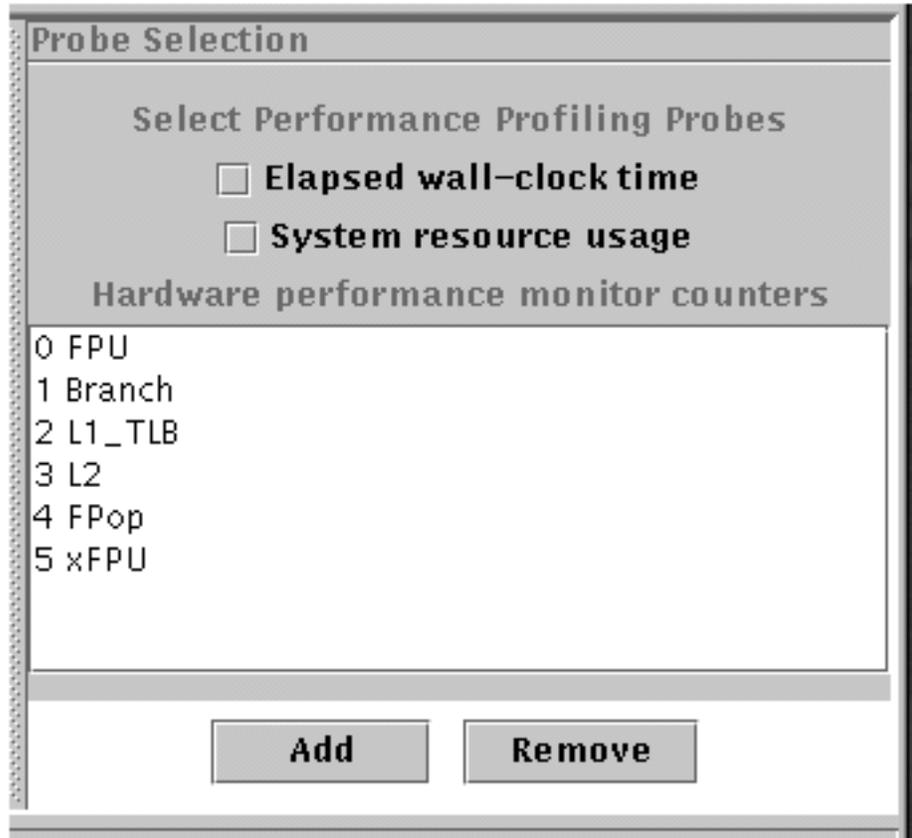
If:	Then:
<p>You want to trace a simple event (one that does not measure a state and so requires only a single user marker inserted in the code).</p>	<p>Insert a simple marker. To do this:</p> <ol style="list-style-type: none"> a. In the User Markers tab, select the Simple event option button. b. In the User Markers tab, click on the instrumentation point location where you want to insert your simple marker. c. In the User Markers tab, type the name for the simple marker in the State/Event Name field. Note that the name you give the simple marker must be unique among all user markers you add. d. Click the Add command button. <p>Doing this adds a simple marker at the selected instrumentation point.</p>

If:	Then:
You want to trace a state by marking the beginning and end of the state.	<p>Insert begin state and end state markers. To do this:</p> <ol style="list-style-type: none"> In the User Markers tab, type the common name for the begin- and end-state markers in the State/Event Name field. Note that you can only use a particular name for one begin marker/end marker pair, and it cannot be a name already assigned to a simple marker. In the User Markers tab, select the Begin-state marker option button. In the User Markers tab, click on the instrumentation point location where you want to insert the begin state marker. Click the Add command button below the User Markers tab. Doing this adds a begin state marker with the name indicated at the selected instrumentation point. In the User Markers tab, select the End-state marker option button. In the User Markers tab, click on the instrumentation point location where you want to insert the end state marker. Click the Add command button below the User Markers tab. Doing this adds an end state marker with the name indicated at the selected instrumentation point. <p>Note: When adding begin and end state markers, be aware that you should not nest markers of the same state. Executing a begin-state probe while already in that state, or executing an end-state probe before a begin state probe has executed, will have no effect. Attempting to, or accidentally, nesting state markers will not result in an error, but may yield unusual or incorrect information in the resulting trace file.</p>
You want to force tracing off or on.	<p>Insert a trace off or trace on marker. To do this:</p> <ol style="list-style-type: none"> In the User Markers Tab, select either the Force tracing on or Force tracing off option button. In the User Markers tab, click on the instrumentation point at which you want to force tracing on or off. Click the Add command button. Doing this adds the trace off or trace on marker at the selected instrumentation point.

- Repeat the steps outlined in the above table as many times as needed until all the desired user markers have been added. In the User Markers tab's list of instrumentation points, a lowercase "m" indicates that a single user marker is installed at that instrumentation point. An uppercase "M" indicates that there are multiple user markers installed at the same instrumentation point.

Specifying Profile Data To Be Collected

Select Performance Profiling Probes Panel



Using the Select Performance Profiling Probes Panel of the Performance Collection Tool's Main Window, you can specify the hardware and operating system information you want to collect for analysis within the Profile Visualization Tool. The Select Performance Profiling Probes Panel is displayed in the Main Window's Probe Selection Area only if you have indicated that you want to collect hardware and operating system profile information as described in "Selecting (At Tool Start-up Time) the Type of Probe Data To Be Collected" on page 16 and "Selecting (After Tool Startup Time) the Type of Probe Data To Be Collected" on page 18. If the Select Performance Profiling Probes panel is not already displayed in the Performance Collection Tool's Main Window, refer to "Selecting (After Tool Startup Time) the Type of Probe Data To Be Collected" on page 18.

To specify the profile data to be collected:

1. Select the connected process(es) you want to instrument with performance collection probes. You can either instrument all connected processes of the application, or individual connected processes.

If:	Then:
You want to instrument all connected processes in the application.	Select the Process → Select All Connected menu item off the Main Window's menu bar.
You want to instrument individual connected processes.	In the Main Window's Process List area, click on the connected process(es) to which you want to add the performance collection probes. The process is connected if the "Connected" column in the Process List area reads "True".

Selecting one or more connected processes, as described in the preceding table, highlights the selected process(es) in the Main Window's Process List area. Once one or more processes are selected, be aware that certain further actions you can request using the Performance Collection Tool (such as adding performance collection probes) will be performed on the selected processes only.

2. In the Main Window's Source Tree area, select the location(s) where you want the performance collection probes added. You can instrument the entire executable, one or more of the executable's source files, or one or more functions. Be aware, however, that instrumenting a file will result in probes being added to each function or subroutine in the file. Similarly, instrumenting an entire executable will result in probes being added to every function or subroutine in every file in the executable. For these reasons, instrumenting an entire executable or file will be more time and resource intensive than instrumenting selected functions.

If:	Then:
You want to instrument the entire executable.	Click on the executable name in the source tree. Doing this highlights the executable name to show that it is selected.
You want to instrument one or more of the executable's source files.	Click on the source file name(s) in the Source Tree. If the source file names are not visible, you may need to expand the source tree. A handle icon to the left of entries in the Source Tree indicates that there is additional, unexpanded, information under that entry. To expand the Source Tree to see the additional information, click on the handle icon. Selecting a source file name in the source tree highlights the file name to show that it is selected.
You want to instrument one or more functions.	Click on the function name(s) in the source tree. If the function names are not visible, you may need to expand the source tree. A handle icon to the left of entries in the Source Tree indicates that there is additional, unexpanded, information under that entry. To expand the Source Tree to see the additional information, click on the handle icon. Selecting a function name in the Source Tree highlights the name to show that it is selected.

3. In the Main Window's Select Performance Profiling Probes Panel, select the MPI events you want to collect data for.

If you want to collect data for:	Then:
Elapsed wall clock time	click on the Elapsed wall-clock time check box.
System resource usage	Click on the System resource usage check box.

If you want to collect data for:	Then:
Hardware performance monitor counters.	<p>All tasks you want to instrument must be running on the same type of CPU (either all running on 604e CPUs or all running on 630 CPUs). In the Hardware counters list, click on the specific hardware counter group for which you want to collect information. Clicking on the name of a hardware counter group highlights its entry in the list to show that it is selected.</p> <p>Note: If your application is running on mixed CPUs (in other words, some tasks running on 604e CPUs and some tasks running on 630 CPUs), no hardware counter groups will be listed.</p>

- Click the **Add** command button below the Select Performance Profiling Probes Panel, or select the **Probe → Add** menu item off the Main Window’s menu bar. Doing this adds the probes you have selected in the Select Performance Profiling Probes Panel to the process(es) selected in the Process List. The selected data will be collected for the locations selected in the Source Tree. The output file will be stored to the location you specified in either the Probe Data Selection Dialog (as described in “Selecting (At Tool Start-up Time) the Type of Probe Data To Be Collected” on page 16) or the Select Data Collection Output Location Dialog (as described in “Selecting (After Tool Startup Time) the Type of Probe Data To Be Collected” on page 18).

Removing Performance Collection Probes From One or More Processes

When you add performance collection probes to one or more processes (as described in “Specifying MPI Trace Data to Be Collected” on page 24 and “Specifying Profile Data To Be Collected” on page 36), the entries for the installed probes will appear in the Main Window’s Source Tree area. When you install a number of probe types at the same time, note that they are installed as a probe set and will be represented by one or more entries (depending on the number of locations you selected) in the Main Window’s Source Tree. If you install a probe or a set of probes at the executable level, its entry will appear in a “ProbeRoot” folder under the executable’s entry in the tree. If you install a probe or set of probes at the file level, its entry will appear in a “ProbeRoot” folder under the file’s entry in the tree. If you install a probe or set of probes at the function level, its entry will appear in a “ProbeRoot” folder under the function’s entry in the tree.

To delete an installed probe or probe set:

- Select the connected process(es) from which you want to remove the probe or probe set. You can either select all connected processes of the application, or individual connected processes.

If:	Then:
You want to select all connected processes in the application.	Select the Process → Select All Connected menu item off the Main Window’s menu bar.

If:	Then:
You want to select individual connected processes.	In the Main Window's Process List area, click on the connected process(es) from which you want to remove the performance collection probes. A process is connected if the "Connected" column in the Process List area reads "true".

Selecting one or more connected processes, as described in the preceding table, highlights the selected process(es) in the Main Window's Process List area. Once one or more processes are selected, be aware that certain further actions you can request using the Performance Collection Tool (such as adding performance collection probes) will be performed on the selected processes only.

2. Click on the probe or probe set's entry in the Source Tree. The probe or probe set's entry will appear in the "ProbeRoot" directory under the application, file, or function where the probe or probe set was installed.

Selecting a probe entry in the Source Tree highlights the name to show that it is selected.

3. Click on the **Remove** command button in the Probe Selection area, or else select the **Probe → Remove...** menu item off the Main Window's menu bar.

Removing User Markers From Processes

As described in "Adding User Markers to Processes" on page 33, you can add user markers to processes in order to:

- mark simple events (using simple markers)
- mark states (using begin state markers and end state markers)
- force tracing on or off (using a trace on marker or a trace off marker)

To remove a custom user marker:

1. Select the connected process(es) from which you want to remove the user marker(s).

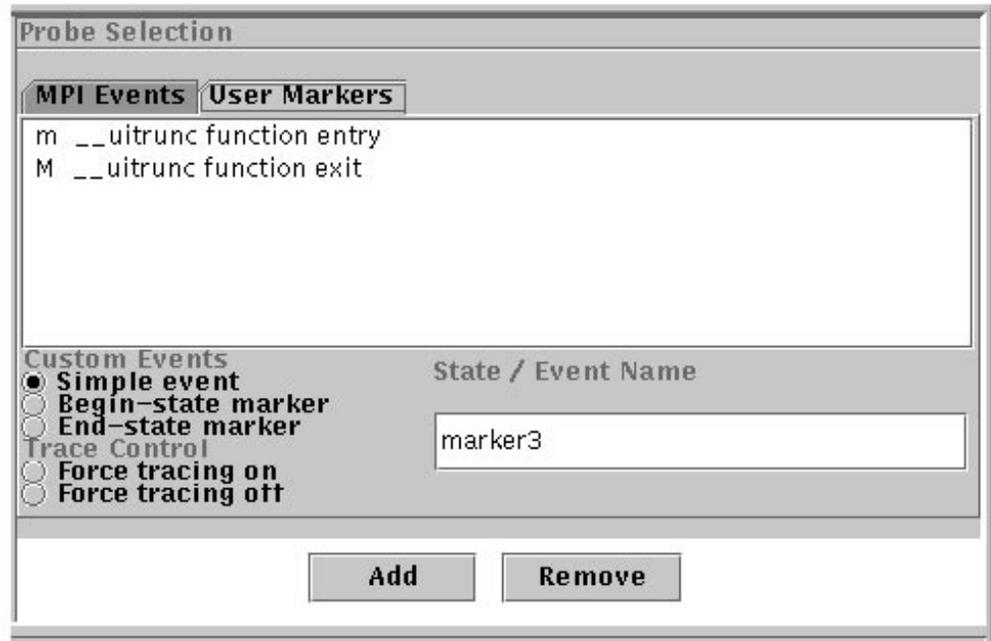
If:	Then:
You want to select all connected processes in the application.	Select the Process → Select All Connected menu item off the Main Window's menu bar.
You want to select individual connected processes.	In the Main Window's Process List area, click on the connected process(es) from which you want to remove the user markers. A process is connected if the "Connected" column in the Process List area reads "true".

Selecting one or more connected processes, as described in the preceding table, highlights the selected process(es) in the Main Window's Process List area. Once one or more processes are selected, be aware that certain further actions you can request using the Performance Collection Tool (such as removing user markers) will be performed on the selected processes only.

2. In the Main Window's Source Tree area, click on the name(s) of the source file(s) from which you want to remove the user marker(s). If the source file names are not visible, you may need to expand the source tree. A handle icon to the left of the entries in the Source Tree indicates that there is additional, unexpanded, information under the entry. To expand the Source Tree to see the additional information, click on the handle icon.

Selecting a source file name in the Source Tree highlights the file name to show that it is selected.

3. In the Main Window's Probe Selection area, click on the **User Markers** tab.



The User Markers Tab displays instrumentation points associated with the selected source files. In the list of instrumentation points, a lowercase "m" indicates that a single user marker is installed at that instrumentation point. An uppercase "M" indicates that there are multiple user markers installed at the same instrumentation point.

4. In the User Markers tab, click on the instrumentation point where you want to remove a custom user marker.
5. Click on the **Remove** command button. If there is only one user marker installed at the selected instrumentation point, the user marker it represents is removed. If there are multiple user markers installed at the selected instrumentation point, the Remove Custom Marker Dialog opens.



The Remove Custom Marker Dialog lists the various user markers installed at the selected instrumentation point. In the Remove Custom Marker Dialog's list, click on the entry for the user marker you want to remove, and then click on the **OK** command button.

Terminating Connected Processes

Terminate Application Confirmation Dialog

If you are done collecting performance data for one or more connected processes, you may wish to terminate the process(es). To terminate any connected target application process(es):

1. Select the **Application** → **Terminate** menu item off the Main Window's menu bar.

Doing this opens the Terminate Application Confirmation Dialog.



2. The Terminate Application Dialog is displayed in order to prevent accidental termination of the target application processes. (If, for example, you accidentally select the **Application** → **Terminate** menu item).

If:	Then:
You are sure you want to terminate execution of the target application.	Click on the OK command button. Doing this terminates any connected target application process and closes the Terminate Application Confirmation Dialog.
You do not want to terminate execution of the target application.	Click on the Cancel command button. Doing this closes the Terminate Application Confirmation Dialog. The target application continues executing.

Note: When working with a POE application, be aware that terminating any process of the application will cause POE to terminate all of the application's processes. This termination of all processes is a function of POE, not of the Performance Collection Tool. For more information, refer to *IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment*.

Exiting the Performance Collection Tool

Exit Tool Dialog

To exit the Performance Collection Tool:

1. Select the **Application** → **Exit** menu item off the Main Window's menu bar.

Doing this opens the Exit Tool Dialog.



2. The Exit Tool Dialog asks if you would also like to "Terminate the target application?"

If:	Then:
You wish to also terminate the target application to which the Performance Collection Tool is connected.	Click on the Yes command button. Doing this terminates the target application as well as the Performance Collection Tool, and closes all of the Performance Collection Tool windows.
You wish to allow the target application to continue running.	Click on the No command button. Doing this terminates the Performance Collection Tool, but leaves the target application running.

Using the Performance Collection Tool's Command-Line Interface

This section describes how you can use the Performance Collection Tool in command-line mode to collect either MPI and user event traces or hardware and operating system profiles. The purpose of this section is to illustrate how the various subcommands of the **pct** command can be used to instrument serial or POE programs. Note, however, that this section does not necessarily describe all the options of all the **pct** subcommand. For complete reference information on any of the subcommands describe in this section, refer to the **pct** command's man page in "Appendix B. PE Benchmark Command Reference" on page 115

This section begins with a brief overview of the tasks you can perform using the Performance Collection Tool's command-line interface, and then describes each of these tasks in more detail. You can also operate the Performance Collection Tool using its graphical user interface. For information on how to do this, refer to "Using the Performance Collection Tool's Graphical User Interface" on page 5.

Performance Collection Tool (Command-Line Interface) Overview

To use the Performance Collection Tool's command-line interface to collect either MPI and user event traces or hardware and operating system profiles:

1. Start the Performance Collection Tool in command-line mode by issuing the **pct** command with its **-cmd** option. You can optionally specify the **-s** option to instruct the Performance Collection Tool to read its subcommands from a script file. For more information, refer to "Starting the Performance Collection Tool In Command-Line Mode" on page 46.
2. Either load and start a new application, or connect to a running application.

- To load and start a new application, use the **load** subcommand to load either a serial or POE application. When you load an application, its process execution will be suspended at its first executable instruction. To start execution of one or more loaded application processes, issue the **start** subcommand. For more information, refer to “Loading and Starting a New Application” on page 48.
- To connect to a running application, use the **connect** subcommand. You can connect to a serial or POE home node process using this subcommand. Once connected to a POE home node process, you can issue the **connect** subcommand again to connect to one or more of its individual tasks. For more information, refer to “Connecting to a Running Application” on page 49.

When you load or connect to a serial or POE application, two task groups are created. A *task group* is simply a named set of tasks — in this case, the task groups are named “*all*” and “*connected*”. Task groups are intended for when you are working with POE applications as opposed to serial applications. The *all* task group represents all the tasks in the POE application, while the *connected* task group represents the POE application’s connected tasks only. You can also create your own named task groups. Task groups enable you to more easily manipulate the tasks of a POE application, since many of the Performance Collection Tool’s subcommands are designed to operate upon one or more tasks. By default, the tasks operated upon are those in a “current task group” that you specify. By default, the current task group is the automatically-created task group *connected*. If you are instrumenting a serial application, you naturally do not need to concern yourself with task groups. You should be aware, however, that the *all* and *connected* groups are still created by the Performance Collection Tool. For more information on task groups, refer to “Grouping Tasks of a POE Application” on page 47.

3. Select the type of data you will be collecting using the Performance Collection Tool. You can collect either:
 - MPI and user event traces for analysis using the **utestats** utility or a graphical visualization tool like Jumpshot.
 - hardware and operating system profiles for analysis within the Profile Visualization Tool.

To specify which type of data you’ll be collecting, use the **select** subcommand. For more information, refer to “Selecting Type of Probe Data To Be Collected” on page 53.

4.

If:	Then:
<p>you are collecting MPI and user event traces.</p>	<p>a. Set the output location for the trace files that are generated by the Performance Collection Tool. To do this, use the trace set subcommand. For more information, refer to “Setting the Output Location and Other Preferences for the AIX Trace Files Generated” on page 55.</p> <p>b. Add MPI trace probes and/or custom user markers using the trace add subcommand. For more information, refer to “Adding MPI Trace Probes to Processes” on page 55 and “Adding User Markers to Processes” on page 57.</p> <p>When you are done collecting the trace data, you can remove the probes using the trace remove subcommand. For more information, refer to “Removing MPI Trace Probes From Processes” on page 57 and “Removing User Markers From Processes” on page 58.</p>
<p>you are collecting hardware and operating system profile information.</p>	<p>a. Set the output location for the profile files that are generated by the Performance Collection Tool. To do this, use the profile set path subcommand. For more information, refer to “Setting the Output Location for the NetCDF Files Generated” on page 59.</p> <p>b. Add the profile probes to processes using the profile add subcommand. For more information, refer to “Adding Hardware Profile Probes to Processes” on page 59.</p> <p>When you are done collecting the profile data, you can remove the probes using the profile remove subcommand. For more information, refer to “Removing Hardware Profile Probes From Processes” on page 61.</p>

5. When you are done collecting data, you can terminate connected processes using the **destroy** subcommand, or disconnect from the processes using the **disconnect** subcommand. To exit the Performance Collection Tool, issue the **exit** subcommand. For more information, refer to “Terminating Connected Processes” on page 61, “Disconnecting From the Application” on page 62, and “Exiting the Performance Collection Tool” on page 63.

In addition to the tasks summarized above, you can also:

- suspend and resume execution of connected processes by issuing the **suspend** and **resume** subcommands. You might, for example, wish to suspend execution of your application prior to instrumenting it, and resume execution after the probes have been added. For more information, refer to “Suspending and Resuming Application Execution” on page 50.

- send standard input text to your application using the **stdin** subcommand. For more information, refer to “Sending Standard Input Text to the Application” on page 51.
- Display the contents of source files using the **list** subcommand. For more information, refer to “Displaying the Contents of a Source File” on page 52.

Starting the Performance Collection Tool In Command-Line Mode

To start the Performance Collection Tool in command-line mode, enter, at the AIX command prompt, the **pct** command with its **-cmd** option:

```
pct -cmd
```

You can optionally specify the **-s** option to instruct the Performance Collection Tool to read subcommands from a particular script file of Performance Collection Tool subcommands. As described in “Setting User Preferences” on page 21, you can, when operating the Performance Collection Tool in graphical-user-interface mode, have the tool save the command-line equivalent of your interface interactions to a script file. You can later use the **-s** option to read this script file when running in command-line mode. You can also modify the saved script file, or create your own from scratch, using a text editor. For example, to have the Performance Collection Tool read the subcommands in the script file *myscript.cmd*:

```
pct -cmd -s myscript.cmd
```

The first thing you’ll want to do after starting the Performance Collection Tool is either connect to a running application, or load and connect to new application. If the application you wish to examine is already running, you can connect to it; refer to “Connecting to a Running Application” on page 49. If the application you wish to examine is not already running, you can load it; refer to “Loading and Starting a New Application” on page 48. If you are going to connect or load a POE application, you need to understand the concept of task groups; refer to “Grouping Tasks of a POE Application” on page 47.

Note: If you want to generate diagnostic log files for the Performance Collection Tool’s run, you can optionally specify a diagnostic log setting using the **-d** command-line option when issuing the **pct** command.

```
$ pct -d diag_log_setting
```

Where *diag_log_setting* is one of the values outlined in the following table, and determines the amount of logging information that will be generated. The log files generated when you use the *diag_log_setting* flag are intended for identifying problems through IBM. When diagnostic logging is turned on, the Performance Collection Tool will generate a separate log file in the directory */tmp* on each host machine running target application processes to which the tool connects. The log file is generated by a daemon process that handles communication between the Performance Collection Tool and the target application process. The log file saved to the */tmp* directory will be named *dpc1d.nnnn* (where *nnnn* is the AIX process ID of the daemon process).

If <i>diag_log_setting</i> is:	Then:
severe	The daemon process will generate messages for fatal and severe error conditions only.
warning	In addition to fatal and severe error conditions, the daemon process will generate warning messages.

If <i>diag_log_setting</i> is:	Then:
trace	In addition to fatal, severe, and warning messages, the daemon process will also generate function entry/exit trace information.
detail	The most detailed level of diagnostic messages will be generated by the daemon process. In addition to the severe, error, warning, and function entry/exit trace information, the daemon process will generate other, more general, information.

Grouping Tasks of a POE Application

In the Parallel Operating Environment, the multiple cooperating processes of your program are referred to as "tasks". Many of the Performance Collection Tool subcommands are designed to operate on one or more tasks of a POE application. By default, the tasks operated upon are those in a "current task group" that you can specify. A task group is simply a named set of tasks. Two such task groups — *all* and *connected* — are created automatically when you either connect to a running application using the **connect** subcommand, or load a new application using the **load** subcommand. The *all* task group represents all the tasks in the POE application. The *connected* task group is the current task group by default — it represents the POE application's connected tasks only. You can also create your own task groups.

By default, the current task group will be *connected*; the subcommands you issue will act upon all connected tasks in the POE application. You can change the current task group to be the automatically created group *all*, or a task group that you have created. You can also, for all of the subcommands that act upon task groups, specify a set of tasks or a task group when issuing the subcommand. If you do this, the subcommand will operate on the tasks specified rather than the current task group. For example, consider the **suspend** subcommand for suspending execution of one or more tasks. If you issue this subcommand without options as in:

```
suspend
```

The tasks in the current task group are suspended. However, if, on the **suspend** subcommand, you specify a task list using the **task** clause, you suspend execution for the tasks specified — in this next example tasks 0 through 5:

```
suspend task 0:5
```

Note: When using the task clause, the tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

You can also specify a named task group (other than the current task group) using the **group** clause:

```
suspend group workers
```

To understand why you might want to specify a task group, consider the following example. Say that the application you're examining follows the master/workers model in which one task (the "master") coordinates the activities of all the other tasks — the "workers". You could create two task groups — one containing just the

master task, and the other containing all the other tasks. To do this, you would use the **group** subcommand with its **add** clause. To create a task group *master* containing just task 0:

```
group add master 0
```

To create a task group *workers* containing the tasks 1 through 10:

```
group add workers 1:10
```

Once these groups are created, you can make either the current task group. To do this, you would use the **group** subcommand with its **default** clause. For example, the following subcommand sets the current task group to be the task group *master*:

```
group default master
```

While *master* is the current task group, any subcommands that operate upon tasks will operate only upon task 0 — the only task in the group *master*. To make the group *workers* the current task group:

```
group default workers
```

While you cannot modify or delete the two groups that the Performance Collection Tool automatically creates (*all* and *connected*), you can modify and delete the groups that you have created. To add tasks 11 through 20 to the task group *workers*:

```
group add workers 11:20
```

To delete task 11 from the task group *workers*:

```
group delete workers 11
```

To delete the entire task group *workers*:

```
group delete workers
```

Notes:

1. If you are instrumenting a serial application, you naturally do not need to concern yourself with task groups. You should be aware, however, that the *all* and *connected* groups are still created by the Performance Collection Tool.
2. You can list the existing task groups, or the members of a particular task group, using the **show** subcommand. For example, the following subcommand lists the existing task groups:

```
>show groups
Default      Group Name
             all
@            connected
```

To list the tasks in the task group *all*:

```
>show group all
Tid Program Name                               Host                Cpu Type State
-----
0  /home/strofino/dpctest/WORK/prod_cons        pe04.pok.ibm.com    Unknown  Loaded
1  /home/strofino/dpctest/WORK/prod_cons        pe04.pok.ibm.com    Unknown  Loaded
2  /home/strofino/dpctest/WORK/prod_cons        pe04.pok.ibm.com    Unknown  Loaded
3  /home/strofino/dpctest/WORK/prod_cons        pe04.pok.ibm.com    Unknown  Loaded
```

Loading and Starting a New Application

If the serial or POE application you wish to examine is not already running, you can load it onto one or more nodes. When you load an application using the **load** subcommand, it is loaded in a stopped state with execution suspended at the first executable instruction. You can then start its execution using the **start**

subcommand. To load a serial application, you simply supply the **load** subcommand with the absolute path to the executable. The **exec** clause indicates the executable path. If the application takes arguments, you can specify them using the **args** clause. For example:

```
> load exec /u/example/bin/foo args "a b c"
```

If loading a POE application, you specify the **poe** clause, and can also supply any POE arguments using the **poeargs** clause. For information on the POE command-line flags available to you, refer to the manual *IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment*.

The procedure for loading a POE application differs depending on whether the application follows the Single Program Multiple Data (SPMD) or Multiple Program Multiple Data (MPMD) model. If your program follows the SPMD model, you specify the absolute path to the executable using the **exec** clause:

```
> load poe exec /u/example/bin/parallel_foo poeargs "-procs 4 -hfile /tmp/host.list"
```

If your program follows the MPMD model, you supply the absolute path to a POE commands file (which lists the individual programs to load) using the **mpmcmd** clause:

```
> load poe mpmcmd /u/example/bin/foo.cmds poeargs "-procs 3 -hfile /tmp/host.list"
```

For information on creating a POE commands file for loading multiple programs, refer to the manual *IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment*.

The **load** subcommand also enables you to specify that standard input, standard output, or standard error should be redirected. To read standard input from a file, use the **stdin** clause:

```
> load exec /u/example/bin/foo args "a b c" stdin input_file
```

To redirect standard output to a file, use the **stdout** clause:

```
> load exec /u/example/bin/foo args "a b c" stdout output_file
```

To redirect standard error to a file, use the **stderr** clause:

```
> load exec /u/example/bin/foo args "a b c" stderr error_file
```

When you load an application, two task groups — *all* and *connected* — are automatically created, and *connected* is made the current task group. Task groups are important to know about only if you are working with a POE application and are described in “Grouping Tasks of a POE Application” on page 47. Also note that the application is loaded in a stopped state with execution suspended at the first executable instruction. To start execution of the application, use the **start** subcommand:

```
start
```

Connecting to a Running Application

If the serial or POE application you wish to examine is already running, you can connect to it using the **connect** subcommand. To list the processes to which you can connect, use the **show** subcommand with its **ps** clause:

```
> show ps
Pid  Command
-----
10652 /home/strofino/dpctest/WORK/prod_cons
13256 /etc/dpcld /tmp/dpc1sd
```

```

13316 /home/strofino/dpctest/WORK/prod_cons
14302 /usr/lpp/ppe.dpcl/dpcl_beta/bin/poe
18108 /home/strofino/dpctest/WORK/prod_cons
20614 /u/alfeng/public/perf/seqsleep
21996 /u/alfeng/bin/sesmgr
22644 /home/strofino/dpctest/WORK/prod_cons
22802 java com/ibm/ppe/perf/main/Startup -l /u/alfeng/bin/sesmgr -cmd
23236 -ksh
24894 /etc/dpclid /tmp/dpclid
27632 -ksh
>

```

If you are connecting to a serial application, you simply supply the process ID of the process you wish to connect to using the **pid** clause of the **connect** subcommand.

```
connect pid 12345
```

If you are connecting to a POE application, you connect to the processes in two steps. First, you issue the **connect** subcommand to connect to the controlling, home node, POE process. Once connected to the controlling POE process, you can then reissue the **connect** subcommand to connect to any of its processes. For example, to connect to the application whose AIX process ID is 12345:

```
connect poe pid 12345
```

When you connect to the POE home node process, the Performance Collection Tool creates two task groups — *all* and *connected*. The *all* task group refers to all of the tasks in the application, while the *connected* task group refers only to connected tasks. The *connected* task group will initially be empty since no tasks are connected. You can list the existing task groups by issuing the **show** subcommand with its **groups** clause:

```

>show groups
Default      Group Name
              all
@            connected

```

To connect to all tasks in the POE application:

```
connect group all
```

To connect to select tasks in the POE application, use the **task** clause:

```
connect task 2,3
```

Suspending and Resuming Application Execution

The Performance Collection Tool enables you to suspend and resume execution of connected processes by issuing the **suspend** and **resume** subcommands. You might, for example, wish to suspend execution of your target application prior to instrumenting it as described in “Collecting MPI Trace and Custom User Marker Information” on page 54. Once your performance collection probes have been added to the application, you could resume the application’s execution. By default, the **suspend** and **resume** subcommands act upon the current task group. Unless you have specified another task group to be the current task group (as described in “Grouping Tasks of a POE Application” on page 47), the current task group will be the task group *connected*. The task group *connected* is created automatically by the Performance Collection Tool when you either connect to or load an application (as described in “Connecting to a Running Application” on page 49 and “Loading and Starting a New Application” on page 48). The task group *connected* consists of all connected tasks in a POE application. If you are instrumenting a serial application, you do not need to concern yourself with task groups. If you are instrumenting a

POE application, however, it is useful to understand the concept of task groups as described in “Grouping Tasks of a POE Application” on page 47.

To suspend execution of the tasks in the current task group:

```
> suspend
```

To suspend execution of tasks in a specific task group (in this case, the task group *connected*), use the **group** clause on the **suspend** subcommand:

```
> suspend group connected
```

To suspend a specific set of tasks in a POE application, use the **task** clause on the **suspend** subcommand. To determine how many tasks are available, you can use the **show group** subcommand to list the tasks in the task group *all*:

```
> show group all
Tid Program Name                               Host                Cpu Type State
-----
0  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
1  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
2  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
3  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
.
.
.
> suspend task 1,3
```

The **resume** subcommand works in the same way. By default, it operates on the current task group:

```
> resume
```

But you can override this by specifying a task group:

```
> resume group connected
```

or supplying a task list:

```
> resume task 1,5
```

Sending Standard Input Text to the Application

If you have loaded an application (as described in “Loading and Starting a New Application” on page 48), you can use the **stdin** subcommand to send standard input text to your application. However, if you have instead merely connected to an application (as described in “Connecting to a Running Application” on page 49), you cannot send standard input text to the application using the **stdin** subcommand.

If you are instrumenting a serial application, the standard input text will be set to that application process. If you are instrumenting a POE application, the standard input text will be sent to the controlling, “home node”, POE process. As described in “Loading and Starting a New Application” on page 48, you can, when loading an application using the **load** subcommand, specify that standard input should be read from a file. If you are reading standard input from a file, you cannot use the **stdin** subcommand.

To send a standard input string to the application, specify the string on the **stdin** subcommand. The string must be enclosed in double quotes:

```
> stdin "Now is the time for all good men"
```

If desired, you can use embedded formatting characters (such as **\n**) in your standard input string:

```
> stdin "Now is the time \nfor all good men"
```

To send a newline character to the input stream reading this input data, issue the **stdin** command without any text string:

```
stdin
```

To send an end-of-line character to the input stream reading this input data, use the **eof** clause on the **stdin** subcommand:

```
> stdin eof
```

Displaying the Contents of a Source File

Using the **list** subcommand, you can display the contents of source files. Unless you are certain of the file name of the source file you want to examine, you may want to list the available source files using the **file** subcommand. The **file** subcommand lists, for one or more connected tasks, the associated source file names that match a regular expression you supply. By default, the **file** subcommand acts upon the current task group. Unless you have specified another task group to be the current task group (as described in “Grouping Tasks of a POE Application” on page 47), the current task group will be the task group *connected*. The task group *connected* is created automatically by the Performance Collection Tool when you either connect to or load an application (as described in “Connecting to a Running Application” on page 49 and “Loading and Starting a New Application” on page 48). The task group *connected* consists of all connected tasks in a POE application. If you are instrumenting a serial application, you do not need to concern yourself with task groups. If you are instrumenting a POE application, however, it is useful to understand the concept of task groups as described in “Grouping Tasks of a POE Application” on page 47.

You supply the **file** subcommand with an AIX regular expression file-matching pattern (enclosed in double quotation marks) to match the source files you want to list. The files that match the supplied expression(s) are listed in the form *task_identifier.file_identifier.file_name*. For example, to list all the available source files in the current task group:

```
> file "*"
Tid      File Id   File Name   Path
0        0         bar.c       ../../lib/src
0        1         foo1.c      ../../lib/src
0        2         foo2.c      ../src
```

Although this subcommand, by default, acts upon the current task group, you can specify that it should instead act upon a different task group, or all the tasks in a task list that you supply. This is done by using the **task** or **group** clause on the **file** subcommand. For more information on the **task** and **group** clauses, refer to “Grouping Tasks of a POE Application” on page 47.

After issuing the **file** subcommand, you’ll have both the file name and the file identifier of the source file(s) you want to examine. Now you can use the **list** subcommand to display the contents of one or more files. Like the **file** subcommand, the **list** subcommand will, by default, act upon the current task group. Using either the **file** or **fileid** clause of the **list** subcommand, you indicate the file(s) whose contents you want listed.

Using the **file** clause, you supply the **list** subcommand with an AIX regular expression file-matching pattern (enclosed in double quotation marks) to match the source file(s) whose contents you want to list. If desired, you can supply additional

regular expressions separated by commas (file "f*", "b*"). For example, the following subcommand lists the contents of the file *bar.c*:

```
list file "bar.c"
```

While this subcommand lists the contents of the first file found in the application that begins with the letter "f":

```
list file "f*"
```

Using the **fileid** clause, you identify the file whose contents you want to list using the process identifier(s) returned by the **file** subcommand. For example, the following subcommand lists the contents of the file *bar.c* (whose file identifier is 0):

```
list fileid 0
```

Note: When listing the contents of files using the **list** subcommand, the Performance Collection Tool uses a special source path to locate the source files. This source path is, by default, the directory in which the Performance Collection Tool was started, and can be displayed using the **sourcepath** clause on the **show** subcommand as in:

```
> show sourcepath
PATH (BASE)
-----
/u/alfeng/bin/
./
>
```

To modify the source path so that the Performance Collection Tool can locate source files that are not located in the directory in which the tool was started, use the **set** subcommand. As with setting your AIX **PATH** environment variable, you separate the various directories in your source path using colons. For example:

```
> set sourcepath "/afs/aix/u/jbrady:/afs/aix/u/dlecker"
```

Selecting Type of Probe Data To Be Collected

The Performance Collection Tool is capable of collecting two different types of information. It can collect:

- MPI and user event traces for analysis using the **utestats** utility or a graphical visualization tool like Jumpshot (a public domain tool developed at Argonne National Lab). For more information on the **utestats** utility, as well as utilities for converting the AIX trace files created by the Performance Collection Tool into a format readable by **utestats** and Jumpshot, refer to "Chapter 4. Creating, Converting, and Viewing Information Contained In, UTE Interval Files" on page 91.
- Hardware and operating system profiles for analysis within the Profile Visualization Tool. For more information on the Profile Visualization Tool, refer to "Chapter 3. Using the Profile Visualization Tool" on page 65.

Be aware that, before you can collect either type of information, you must specify, using the **select** subcommand, which type you are interested in:

If you want to collect:	Then:
MPI and user event traces.	Specify the trace clause on the select subcommand: select trace

If you want to collect:	Then:
Hardware and operating system profiles.	Specify the profile clause on the select subcommand: select profile

Note: You can select the type of data to collect only once per load and connect.

Collecting MPI Trace and Custom User Marker Information

Using the Performance Collection Tool, you can collect MPI and user event traces for:

- analysis using the **utestats** utility.
- eventual analysis within a graphical visualization tool like Jumpshot.

The trace information collected is stored as an AIX trace file on each node running instrumented processes. After you have generated these AIX trace files, you can convert them into the Unified Trace Environment (UTE) format (using the **convert** utility) for analysis using the **utestats** utility. You can then also convert the UTE files into the SLOG format (using the **slogmerge** utility) for analysis within Jumpshot. For more information on the utilities for converting the AIX trace files output by the Performance Collection Tool into formats readable by the **utestats** utility and Jumpshot, refer to “Chapter 4. Creating, Converting, and Viewing Information Contained In, UTE Interval Files” on page 91.

Before you can use any of the MPI trace collection subcommands described in this section, you must first specify that you are collecting MPI trace information rather than hardware profile information. Refer to “Selecting Type of Probe Data To Be Collected” on page 53 for more information. Once you have indicated that you’ll be collecting MPI and/or user event traces, you can select the output location for the trace files generated by the Performance Collection Tool. To do this, you simply supply an output directory and “base name” (file prefix) for the trace files. Refer to “Setting the Output Location and Other Preferences for the AIX Trace Files Generated” on page 55 for more information. You can collect information about:

- Standard MPI messaging events such as collective communication, point-to-point communication, or one-sided communication. This is done by adding MPI data collecting probes to one or more application task. Refer to “Adding MPI Trace Probes to Processes” on page 55 for more information.
- Events of interest (such as program function calls). This is done by installing a simple user marker into one or more application task at an instrumentation point in the code. Instrumentation points are locations in the code (such as function call sites) where it is safe to install probes. A simple marker will appear in the trace record as a single point; its position gives you a frame of reference when analyzing a trace record in a graphical visualization tool like Jumpshot.
- states of interest. This is done by installing beginning and ending state user markers in the code at particular instrumentation points. A state will appear in the trace record as a region and, like the simple markers, gives you a frame of reference when analyzing a trace record in a graphical visualization tool like Jumpshot.

Note: The set of tasks in which you will add the probes cannot include different executables in an MPMD application. For example, if an MPMD application consists of executables *a.out* and *b.out*, then this command cannot be applied to a task group that contains both *a.out* and *b.out* tasks.

Setting the Output Location and Other Preferences for the AIX Trace Files Generated

The trace information collected by the Performance Collection Tool is stored as a separate AIX trace file on each node running instrumented processes. You can select the output location and other preferences for the trace files using the **trace set** subcommand.

To specify:	Use this clause of the trace set subcommand:	For example:
The output location and a "base name" prefix for the generated files.	path	trace set path "/tracefiles/mytrace"
The AIX trace buffer size in Kilobytes. (This value can be at most 1024, which is the default.)	bufsize	trace set bufsize 1000
The type of events (MPI events or process dispatch events) that are traced. (By default, MPI events are traced.)	event	trace set event mpi trace set event process
The maximum trace file size in Megabytes. (The default is 20.)	logsize	trace set logsize 25

Adding MPI Trace Probes to Processes

By adding MPI trace probes to processes, you can trace such MPI events as collective communication, point-to-point communication, and one-sided communication. To add MPI trace probes, you'll need to know the specific MPI probe type identifier or name as returned by the **trace show** subcommand. To list the available MPI probe type identifiers and names, specify the **probetypes** clause on the **trace show** subcommand:

```
> trace show probetypes
MPI Id MPI Name   Description
-----
0      all          all MPI events
1      blkcollcomm    blocking collective communication
2      pttopt        point-to-point communication
3      onesided      one-sided communication
4      commgroup     communication groups
5      topo         topologies
6      collcomm     non-blocking collective communications
7      env          environmental
8      data         data type
9      file         file
10     info         information
11     comm        communicators
12     wait        wait calls
13     test        test calls
```

Once you have the probe type information, you can use the **trace add** subcommand to add one or more probe types to one or more processes. You can add the probes at the file level, in which case the MPI events for the entire file will be traced, or at the function level. If that granularity is not small enough and you want to trace only a portion of a function, you can use special markers to force tracing on and off at particular points.

By default, the **trace add** subcommand acts upon the current task group. Unless you have specified another task group to be the current task group (as described in “Grouping Tasks of a POE Application” on page 47), the current task group will be the task group *connected*. The task group *connected* is created automatically by the Performance Collection Tool when you either connect to or load an application (as described in “Connecting to a Running Application” on page 49 and “Loading Files for Processing” on page 69). The task group *connected* consists of all connected tasks in a POE application. If you are instrumenting a serial application, you do not need to concern yourself with task groups. If you are instrumenting a POE application, however, it is useful to understand the concept of task groups as described in “Grouping Tasks of a POE Application” on page 47.

If you are tracing at the file level, you’ll need to specify the files using either the **file** or **fileid** clause on the **trace add** subcommand. To do this, you’ll need the file identifier or file name information as returned by the **file** subcommand. To list all available source files in the current task group:

```
> file "*"
Tid   File Id   File Name   Path
0     0         bar.c      ../../lib/src
0     1         foo1.c     ../../lib/src
0     2         foo2.c     ../src
```

To add a certain type of MPI probe, you supply the **trace add** subcommand with the MPI probe type and file information. You can specify the MPI probe type by supplying the:

- MPI probe type identifier using the **mpiid** clause
- MPI probe type name using the **mpiname** clause

Similarly, you can specify the file information by supplying the:

- file identifier using the **fileid** clause
- file name using a regular expression

For example:

```
> trace add mpiid 0 to fileid 0
> trace add mpiname all to file "bar.c"
```

You can also specify multiple MPI probe types or multiple files:

```
> trace add mpiid 1,2 to fileid 0,1
> trace add mpiname collcom,pttopt to file "bar.c","f*"
```

If you would like to trace at a function level rather than tracing an entire file, you need to specify the function(s) using either the **function** or **funcid** clause. You’ll need the function identifier or function name information as returned by the **function** subcommand. To list all functions in the file *bar.c*:

```
> function file "bar.c" "*"
Tid   File Id   Function Id   File Name   Function Name
0     1         1            bar.c      func0
0     1         1            bar.c      func1
```

Note:

If you wish to instrument a particular function, but do not know which file the function is located in, you can use the **find** subcommand. For example, to search all files in task 0 for functions that match the regular expression *comp**:

```
> find task 0 function "comp*"
Tid      File Id   File Name  Function Name
0        23       main.c     compute
0        23       main.c     compare
0        25       sort.c     compare2
```

You can then specify the function on the **trace add** subcommand using either its identifier or name:

```
> trace add mpiid 0 to file "bar.c" function "func0"
> trace add mpiid 0 to file "bar.c" funcid 0
```

You can also specify multiple functions:

```
> trace add mpiid 0 to file "bar.c" funcid 0,1
> trace add mpiid 0 to file "bar.c" function "*"
> trace add mpiid 0 to file "bar.c" function "func0","func1"
```

Removing MPI Trace Probes From Processes

When you issue the **trace add** subcommand to install MPI trace probes, the probes are given a unique probe index. You can use the probe index on the **trace remove** subcommand to remove the probes. To ascertain the probe index, use the **trace show** subcommand with its **probes** clause as in:

```
> trace show probes
Probe Id Command
-----
0      trace add mpiid 0 to file "prod_cons.c" function "alarm_handler"
1      trace add mpiid 0 to file "prod_cons.c" function "consume"
```

In the example above, the number in brackets is the probe index. To remove the probe set whose index is 0:

```
> trace remove probe 0
```

Adding User Markers to Processes

User markers are special types of probes that you can install at specific instrumentation points in your application code. You can:

- Mark events of interest (such as program function calls) using a *simple marker*. A simple marker will appear in the trace record as a single point; its position gives you a frame of reference when analyzing the trace record in a graphical visualization tool like Jumpshot.
- Mark a state of interest using a *begin state marker* and an *end state marker*. A state marked by begin and end state markers will appear in the trace record as a region. Like the simple markers, this gives you a frame of reference when analyzing the trace record in a graphical visualization tool like Jumpshot.
- Force tracing on or off using a *trace on marker* or a *trace off marker*.

To install a user marker, you'll need to identify not only the file and function, but also the instrumentation point at which you want the probe installed. To list instrumentation points, issue the **point** subcommand.

```
> point task 0 file bar.c
Tid      File Id   Point Id   Point Type  Callee Name
0        491       0          0           
```

```

0      491      1      1
0      491      2      2      g1
0      491      3      3      g1

```

To:	Use:	For example:
mark a state of interest.	the simplemarker clause on the trace add subcommand.	> trace add simplemarker "simple" to file bar.c funcid 0 pointid 0
mark a region	the beginmarker and endmarker clauses on the trace add subcommand. You must mark the beginning and end of the range with the same "marker name" (a string that will be used to identify the user state in the trace record. You can only use a particular name for one begin marker/end marker pair. The state will appear in the trace record as a region.	> trace add beginmarker "green" to file bar.c funcid 1 pointid 0 > trace add endmarker "green" to file bar.c funcid 1 pointid 1
force tracing on or off	the traceon or traceoff clause on the trace add subcommand.	> trace add traceoff to file bar.c funcid 0 pointid 0 > trace add traceon to file bar.c funcid 0 pointid 1

Removing User Markers From Processes

When you issue the **trace add** subcommand to install a custom user marker, the marker is given a unique marker index. You can use this marker index on the **trace remove** subcommand to remove the markers. To ascertain the marker index, use the **trace show** subcommand with its **markers** clause as in:

```

> trace show markers
[0] 0:0:0:1 beginmarker "green"
[1] 0:0:0:1 simplemarker "simple"
[2] 0:0:0:4 endmarker "green"
[3] 0:0:0:5 traceoff

```

In the example above, the number in brackets is the marker index. To remove the marker whose index is 3:

```
> trace remove marker 3
```

Collecting Hardware and Operating System Profile Information

Using the Performance Collection Tool, you can collect hardware and operating system profiles for analysis within the Profile Visualization Tool.

The profile information collected is stored in NetCDF (network Common Data Form) format on each node running instrumented processes. The Profile Visualization Tool can read NetCDF files and summarize the profile information in reports. For more information on using the Profile Visualization Tool to read NetCDF files output by the Performance Collection Tool, refer to “Chapter 3. Using the Profile Visualization Tool” on page 65.

Before you can use any of the profile collection subcommands described in this section, you must first specify that you are collecting hardware profile information rather than MPI and user event traces. Refer to “Selecting Type of Probe Data To Be Collected” on page 53 for more information. Once you have indicated that you’ll be collecting hardware profile information, you can select the output location for the NetCDF files generated by the Performance Collection Tool. To do this, you simply supply an output directory and “base name” (file prefix) for the NetCDF files. Refer to “Setting the Output Location for the NetCDF Files Generated” for more information.

Setting the Output Location for the NetCDF Files Generated

The hardware profile information is saved as a separate NetCDF file on each node running instrumented processes. Using the **profile set path** subcommand, you can specify the output location and “base name” file prefix for these files. For example:

```
profile set path "profile/output"
```

Adding Hardware Profile Probes to Processes

By adding hardware profile probes to processes, you can collect hardware and operating system information such as elapsed wall-clock time, process resource usage, and hardware counters. To add hardware profile probes, you need to know the specific probe type identifier or name as returned by the **profile show** subcommand. To list available probe type identifiers and names, specify the **probetypes** clause on the **profile show** subcommand. The list of available probe types will differ depending on whether the current or supplied task group:

- has tasks running only on 604e CPUs
- has tasks running only on 630 CPUs
- has tasks running on mixed CPUs

For example:

```
> profile show probetypes
Prof Id Prof Name Description
-----
0      wclock    wall clock
1      rusage    resource usage
2      hwcount   hardware counter
```

For hardware counters, you can also display an option list of the specific hardware counter information you can collect. To list these options, specify the **probetype** clause followed by the probe type name on the **profile show** subcommand:

```
> profile show probetype hwcount
Prof Type Name      Description
-----
0      FPU operations
1      FXU operations
2      LSU operations
3      Branch operations
```

4	L1 cache operations
5	TLB operations
6	Snoop operations
7	Load miss operations
8	Pipeline operations

Once you have the probe type and probe type option information, you can use the **profile add** subcommand to add one or more probe types to one or more processes. You can add the probes at the file level, in which case profile information for the entire file will be produced, or at the function level.

By default, the **profile add** subcommand acts upon the current task group. Unless you have specified another task group to be the current task group (as described in “Grouping Tasks of a POE Application” on page 47), the current task group will be the task group *connected*. The task group *connected* is created automatically by the Performance Collection Tool when you either connect to or load an application (as described in “Connecting to a Running Application” on page 49 and “Loading and Starting a New Application” on page 48). The task group *connected* consists of all connected tasks in a POE application. If you are instrumenting a serial application, you do not need to concern yourself with task groups. If you are instrumenting a POE application, however, it is useful to understand the concept of task groups as described in “Grouping Tasks of a POE Application” on page 47.

Note: The set of tasks in which you will add the probes cannot include different executables in an MPMD application. For example, if an MPMD application consists of executables *a.out* and *b.out*, then this command cannot be applied to a task group that contains both *a.out* and *b.out* tasks.

If you are collecting profile information at the file level, you’ll need to specify the files using either the **file** or **fileid** clause on the **profile add** subcommand. To do this, you’ll need the file identifier or file name information as returned by the **file** subcommand. To list all available source files in the current task group:

```
> file "*"
Tid      File Id      File Name      Path
0        0            bar.c          ../../lib/src
0        1            foo1.c         ../../lib/src
0        2            foo2.c         ../src
```

To add a certain type of profile probe, you can supply the **profile add** subcommand with the profile probe type and option information, as well as the file information. You can specify:

- the profile probe type by supplying the:
 - profile probe type identifier using the **profid** clause
 - profile probe type name using the **profname** clause
- the hardware profile group using the **groupid** or **groupname** clause
- the file information by supplying the:
 - file identifier using the **fileid** clause
 - file name using a regular expression

Note: To ascertain the probe type identifier or probe type name to supply to the **profile add** subcommand, use the **profile show** subcommand as described in “Appendix B. PE Benchmarking Command Reference” on page 115.

For example:

```
> profile add profname wc to fileid 0
> profile add profid 0 to file "bar.c"
> profile add profname hw groupid 2 to fileid 3
```

You can also specify multiple profile probe types or multiple files:

```
> profile add profname wc profname hw groupid 2 to fileid 3,4
```

If you would like to collect profile information at the function level rather than the entire file, you'll need to specify the function(s) using either the **function** or **funcid** clause. You'll need the function identifier or function name information as returned by the **function** subcommand. To list all the functions in the file *bar.c*:

```
> function file "bar.c" "*"
Tid      File Id      Function Id      File Name      Function Name
0        1            1                bar.c          func0
0        1            1                bar.c          func1
```

You can specify the function on the **profile add** subcommand using its identifier or name:

```
> profile add profname wc to file "bar.c" function "func0"
> profile add profname wc to file "bar.c" funcid 0
```

You can also specify multiple functions:

```
> profile add profname wc to file "bar.c" funcid 0,1
> profile add profname wc to file "bar.c" function "*"
> profile add profname wc to file "bar.c" function "func0","func1"
```

Removing Hardware Profile Probes From Processes

When you issue the **profile add** subcommand to install profile probes, the probes are given a unique probe index. You can use this probe index on the **profile remove** subcommand to remove the probes. To ascertain the probe index, use the **profile show** subcommand with its **probes** clause as in:

```
> profile show probes
Probe Id Command
-----
0      profile add profid 0 to file "prod_cons.c" function "alarm_handler"
1      profile add profid 0 to file "prod_cons.c" function "consume"
```

In the example above, the number in brackets is the probe index. To remove the probe set whose index is 0:

```
> profile remove probe 0
```

Terminating Connected Processes

The Performance Collection Tool enables you to terminate execution of connected processes by issuing the **destroy** subcommand. You might, for example, wish to terminate execution of your target application after you have finished examining it. By default, the **destroy** subcommand acts upon the current task group. Unless you have specified another task group to be the current task group (as described in "Grouping Tasks of a POE Application" on page 47), the current task group will be the task group *connected*. The task group *connected* is created automatically by the Performance Collection Tool when you either connect to or load an application (as described in "Connecting to a Running Application" on page 49 and "Loading and Starting a New Application" on page 48). The task group *connected* consists of all connected tasks in a POE application. If you are instrumenting a serial application,

you do not need to concern yourself with task groups. If you are instrumenting a POE application, however, it is useful to understand the concept of task groups as described in “Grouping Tasks of a POE Application” on page 47.

Note: When working with a POE application, be aware that terminating any process of the application will cause POE to terminate **all** of the application’s processes. This termination of all processes is a function of POE, not of the Performance Collection Tool. For more information, refer to the manual *IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment*.

To terminate execution of all tasks in the current task group:

```
> destroy
```

To terminate execution of tasks in a specific task group (in this case, the task group *connected*), use the **group** clause on the **destroy** subcommand.

```
destroy group connected
```

To terminate a specific set of tasks in a POE application, use the task clause on the **destroy** subcommand. To determine how many tasks are available, you can use the **show group** subcommand to list the tasks in the task group *all*:

```
> show group all
Tid Program Name                               Host                               Cpu Type State
-----
0  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
1  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
2  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
3  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
.
.
.
> destroy task 1,3
```

You can also, optionally, terminate execution of all connected tasks when exiting the Performance Collection Tool. To do this, use the **exit** command (as described in “Exiting the Performance Collection Tool” on page 63).

Disconnecting From the Application

Once you are through examining a particular application, or particular tasks in an application, you can disconnect from the application or application tasks by issuing the **disconnect** subcommand. Once a process is disconnected, the Performance Collection Tool will no longer be able to control execution of, or instrument, the process unless it reconnects to the process. By default, the **disconnect** subcommand acts upon the current task group. Unless you have specified another task group to be the current task group (as described in “Grouping Tasks of a POE Application” on page 47), the current task group will be the task group *connected*. The task group *connected* is created automatically by the Performance Collection Tool when you either connect to or load an application (as described in “Connecting to a Running Application” on page 49 and “Loading and Starting a New Application” on page 48). The task group *connected* consists of all connected tasks in a POE application. If you are instrumenting a serial application, you do not need to concern yourself with task groups. If you are instrumenting a POE application, however, it is useful to understand the concept of task groups as described in “Grouping Tasks of a POE Application” on page 47.

To disconnect all tasks in the current task group:

```
> disconnect
```

To disconnect tasks in a specific task group (in this case, the task group *connected*), use the **group** clause on the **disconnect** subcommand.

```
disconnect group connected
```

To disconnect a specific set of tasks in a POE application, use the **task** clause on the **disconnect** subcommand. To determine how many tasks are available, you can use the **show group** subcommand to list the tasks in the task group *all*:

```
> show group all
Tid Program Name                               Host                Cpu Type State
-----
0  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
1  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
2  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded
3  /home/strofino/dpctest/WORK/prod_cons pe04.pok.ibm.com Unknown Loaded.
.
.
> disconnect task 1,3
```

Exiting the Performance Collection Tool

To exit the Performance Collection Tool and return to your AIX command prompt, issue the **exit** subcommand:

```
exit
```

To terminate execution of all connected processes as you exit the Performance Collection Tool, include the **destroy** clause on the **exit** subcommand.

```
exit destroy
```


Chapter 3. Using the Profile Visualization Tool

The Profile Visualization Tool is a post-mortem analysis tool. It is designed to process profile data files generated by the Performance Collection Tool used in application profiling. For more information on the Performance Collection Tool, refer to “Chapter 2. Using the Performance Collection Tool” on page 5. After processing profile data, you can view the results in the Profile Visualization Tool’s graphical user interface display, outputted to report files, or saved to a summary file. The Profile Visualization Tool provides a command-line interface to process individual profile files directly into a summary file without initializing the graphic display. The command-line interface also enables you to generate textual profile reports. This chapter begins with a discussion of the Profile Visualization Tool’s graphical user interface, followed by a description of the command-line interface.

Using the Profile Visualization Tool’s Graphical User Interface

The Profile Visualization Tool provides a graphical user interface that enables you to process profile data files and view the results. The options available in the graphical user interface correspond to the commands available in the Profile Visualization Tool’s command-line interface. For more information on the command-line interface, refer to “Using the Profile Visualization Tool’s Command Line Interface” on page 88.

Profile Visualization Tool (Graphical User Interface) Overview

The Profile Visualization Tool’s graphical user interface allows you to process and view profile data. You can load one or more files for processing and view the results in a variety of ways. After initializing the graphical user interface, you can choose the appropriate options:

If:	Then:
You wish to load files for processing.	<p>Select File → Load...</p> <p>Doing this opens the Load Files panel. The Load Files panel will enable you to specify what files to load into the tool for processing. You can specify one or more individual profile files, or a summary profile file. For more information on the Load option, refer to “Loading Files for Processing” on page 69.</p>
You wish to control the way profile data is presented.	<p>Select the View option.</p> <p>Doing this opens the View menu. The View menu will enable you to specify how profile data is presented in the Main Display window. You can specify how to sort data, as well as show function call count and resource usage. For more information on the View option, refer to “Viewing Profile Data” on page 70.</p>

You wish to view selected objects.	<p>Select the Object option.</p> <p>Doing this opens the Object menu. The Object menu will enable you to view information such as source code, profile data, and statistics reports for selected objects. For more information on the Object option, refer to "Viewing Selected Objects" on page 76.</p>
You wish to search for a text string.	<p>Select File → Find...</p> <p>Doing this opens the Find panel. The Find panel will enable you to specify the text string for which you want to search. For more information on the Find option, refer to "Finding Data" on page 80.</p>
You wish to generate reports of profile data.	<p>Select the Report option.</p> <p>Doing this opens the Report menu. The Report menu will enable you to select and view a variety of reports, including function call count, CPU usage, and memory usage. For more information on the Report option, refer to "Generating Reports of Profile Data" on page 81.</p>
You wish to save summary data to a file.	<p>Select File → Save Statistic Summary...</p> <p>Doing this opens the Save Statistic Summary panel. This panel will enable you to accept a user-specified file name. The statistic summary data of the input profile file or files will be written to the file. For more information on the Save Statistic Summary option, refer to "Saving Summary Data" on page 84.</p>
You wish to export profile data to a file.	<p>Select File → Export...</p> <p>Doing this opens the Export panel. This panel will enable you to accept a user-specified file name. The profile data that is currently loaded will be written to the file. For more information on the Export option, refer to "Exporting Profile Data" on page 86.</p>
You wish to specify certain things that you would prefer to see as part of the tool.	<p>Select File → Preferences...</p> <p>Doing this opens the Preferences panel. At this time, this panel will enable you to access only one option: source code search paths. There is a text field available that allows you to specify where the source code files reside. For more information on the Preferences option, refer to "Specifying User Preferences" on page 87.</p>

You wish to exit the Profile Visualization Tool.	Select File → Exit... Doing this closes the Main Display window and exits the Profile Visualization Tool. For more information on the Exit option, refer to “Exiting the Profile Visualization Tool” on page 88.
--------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The following sections describe the graphical user interface in greater detail.

Starting the Profile Visualization Tool

You can start the Profile Visualization Tool in either graphical-user-interface (GUI) mode or command-line mode. For instructions on starting the Profile Visualization Tool in command-line mode, refer to “Using the Profile Visualization Tool’s Command Line Interface” on page 88. To start the Profile Visualization Tool in graphical-user-interface mode:

Enter the pvt command at the AIX command prompt.

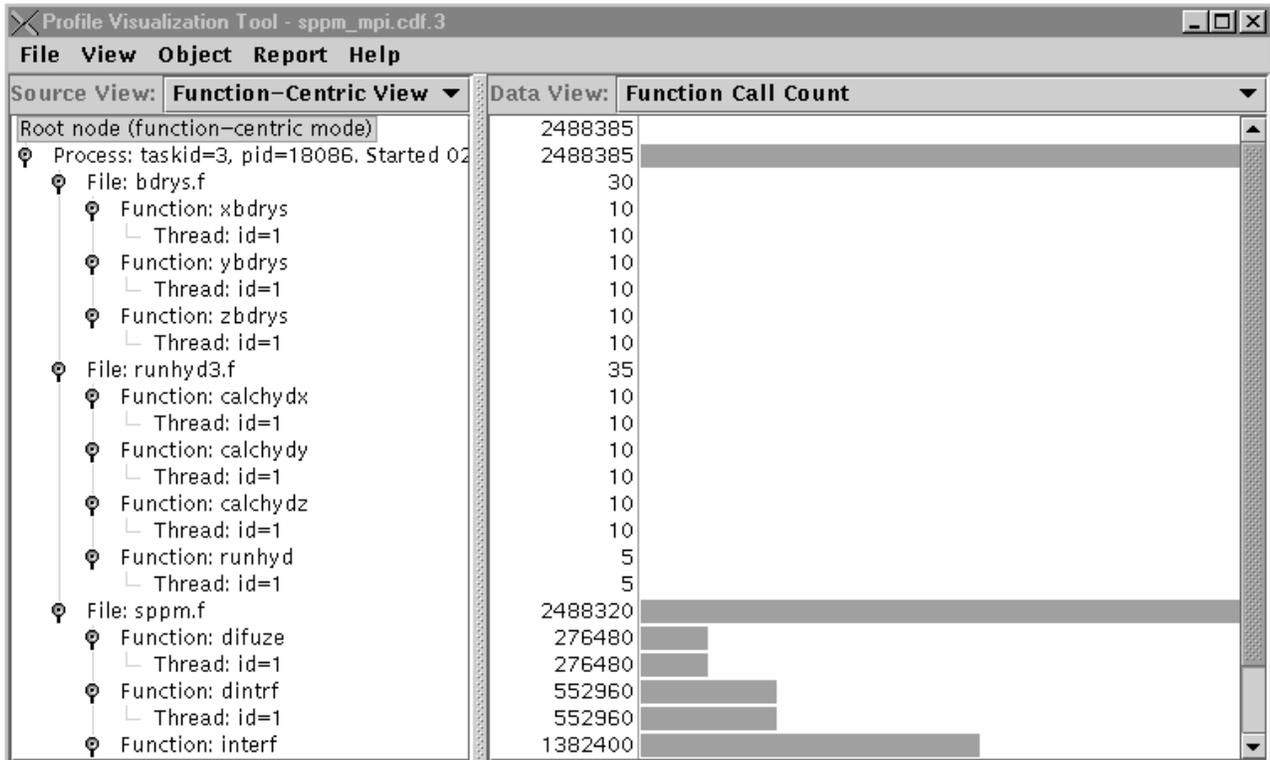
```
$ pvt
```

Doing this starts the Profile Visualization Tool in graphical-user-interface mode and opens its first window – the Main Display.

To start the Profile Visualization Tool in graphical-user-interface mode with input profile data loaded and showing in the Main Display window, enter:

```
$ pvt one_or_more_file_names
```

The following figure shows an example of the Main Display window with input profile data loaded.



The Main Display window shows a hierarchical list of all the functions being profiled. The window is divided into two panes, the left one for viewing source code structure and the right one for viewing profile data. Each pane has a corresponding menu: the **Source View** menu and the **Data View** menu. Both the Source View and Data View menus are grayed out if no input file is loaded. The two panes share the same vertical scroll bar and are scrolled together. You can resize the panes horizontally to change their relative proportion in the Main Display window.

The source code structure pane uses ASCII text to show the identifier of each displayed object. The profile data pane represents a selected profile data field, which uses a bar chart to show the profile data associated with each object. The data value is displayed in front of the bar. When you select an object in the source code structure pane, an object menu opens that provides some actions associated with the selected object. You left-click to select an object, and right-click to bring up the selected object's object menu. When you select an object, the **Object** menu in the Main Display window will become available also, providing the same functions as the popup object menu.

If you load a summary profile file to start the GUI, process objects are labeled as **summary process object** in order to distinguish them from the process objects available in an individual profile file. Each function object has a set of statistics records associated with each profile data field.

Following are explanations of the Source View and Data View menus.

Viewing Source Code Structure

The Source View is a drop-down menu with two options: a **Thread-Centric View** and a **Function-Centric View**. The same options are available under the **View** drop-down menu in the Main Display window. See "Viewing Profile Data" on page 70 for more information. If the input file you are loading to start the GUI is a

summary file, there will be no thread information in the file. The structure displayed will be the same no matter which view is used.

Viewing Selected Profile Data

The Data View is a drop-down menu that enables you to change the type of data to be shown in the Main Display window. The Data View menu options include the following categories:

- **Function Call Count**
- **Wall Clock Time**
- **Resource Usage**
- **Hardware Counters.**

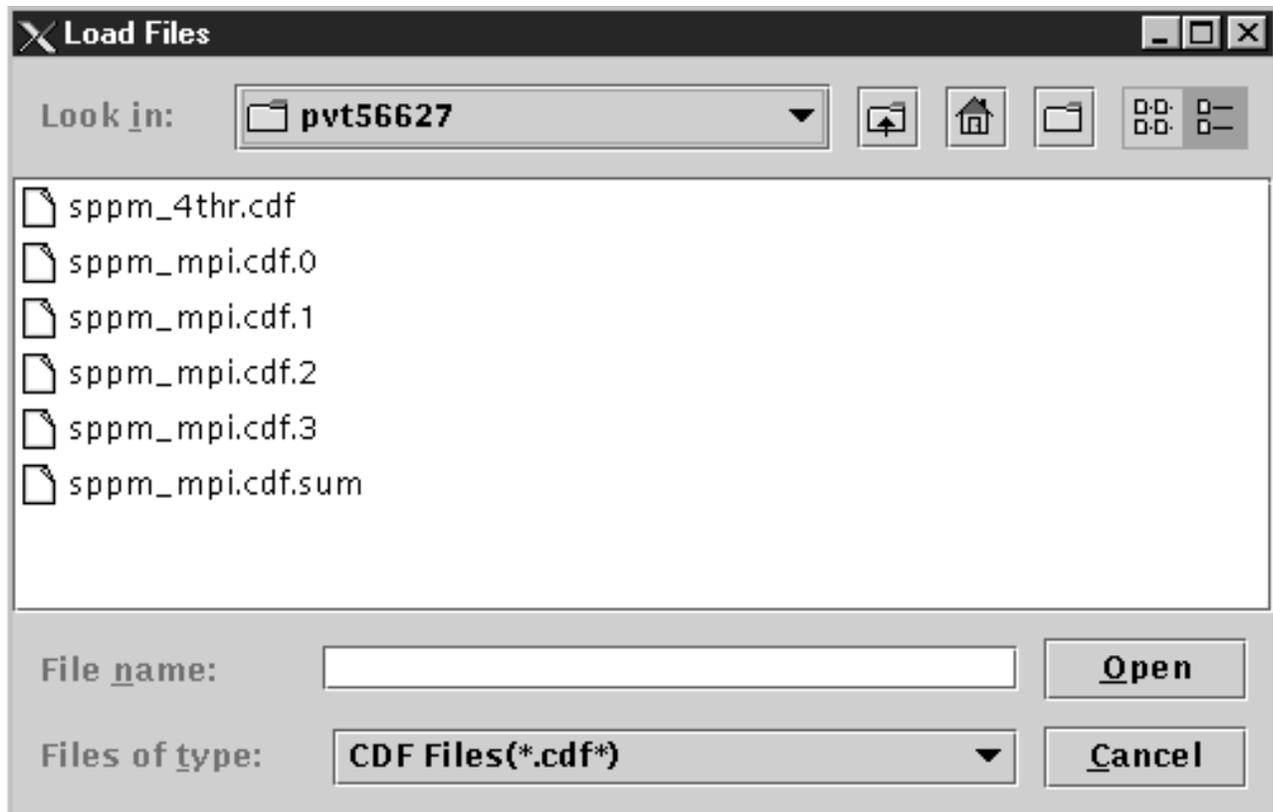
You will find similar options available in the **View** drop-down menu. See “Viewing Profile Data” on page 70 for more information. When a particular data type is unavailable in any of the input data files, its corresponding menu option in the View menu is grayed out. The Data View drop-down menu only shows the options that have corresponding values in the input data files. When a set of files is loaded, **Function Call Count** is the default field in the Data View menu.

Loading Files for Processing

Using the Load Files panel, you can load one or more individual profile files or a summary profile file into the tool for processing. To do this:

1. If the Load Files panel is not already open, select the **File → Load...** menu item off the Main Display window’s menu bar.

Doing this opens the Load Files panel.



All files for processing are CDF files (*.cdf*).

2. You can select a file or files to load from the given list, or search for a file using the appropriate fields.

The **Look in** field provides a drop-down menu for you to select a directory in which to search for a file. You can enter the name of the file that you want to load in the **File name** field. You can also specify the type of file for which to search by using the drop-down menu in the **Files of type** field.

3. After selecting the file or files to load, click on **Open**. The data is loaded for processing and the Load Files panel closes. The profile data is shown in the Main Display window.

When multiple files in the list are selected, only the first file name is shown in the **File name** field. If the **File name** field is then changed, the name in the field is used as the input file (it is treated as a single file name). If the name does not represent any existing file, then the highlighted files will be used. If the **File name** field is never changed after the file selection, then the highlighted files will be used as input files.

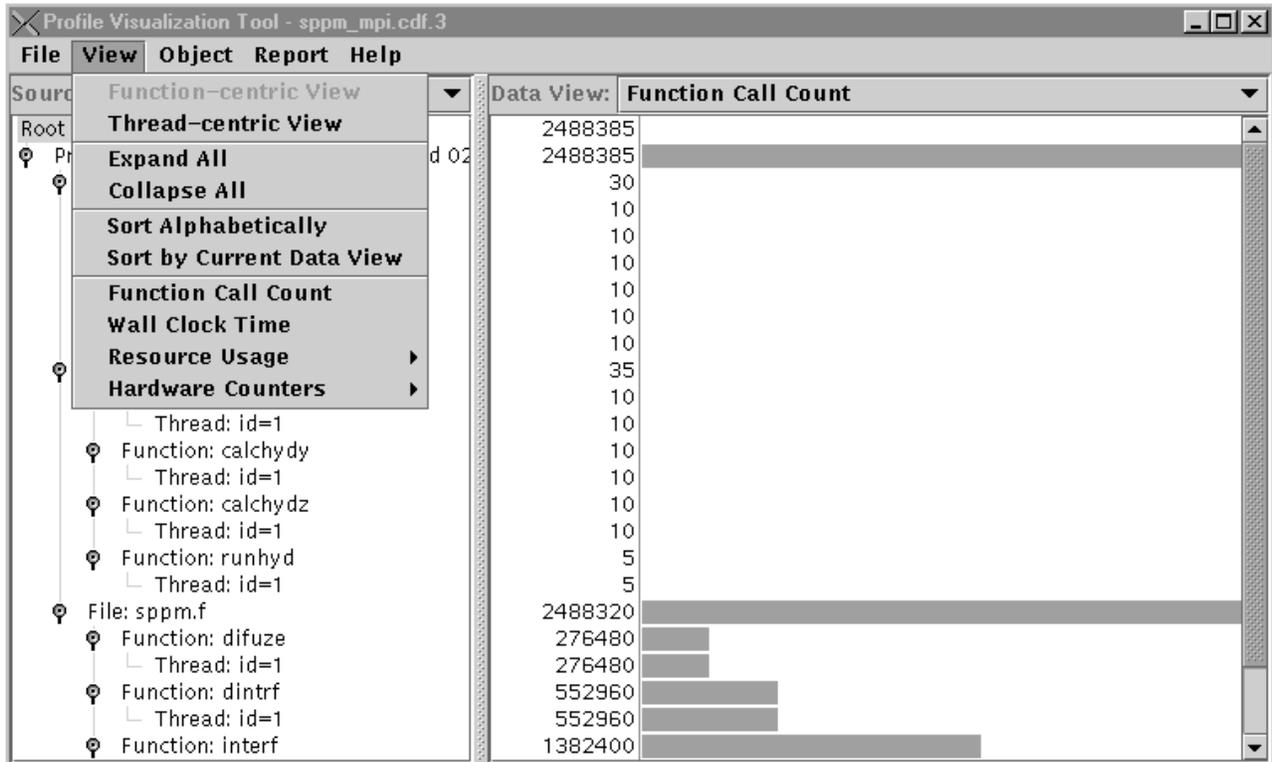
Once you have loaded the file information, you can choose the way you want to view the processed data. The following sections describe in detail how to view profile data.

Viewing Profile Data

Using the View menu, you can specify how profile data is presented in the Main Display window. The options available with this menu enable you to:

- Change source code hierarchy
- Reveal or hide objects
- Sort data
- Select a variety of other options as appropriate, to show profile data values.

To show the View menu, click on **View** on the Main Display window's menu bar.



The following sections describe the View menu options.

Viewing Source Code Hierarchy

You can control how to view source code hierarchy with the first two options in the View menu: **Function-centric View** and **Thread-centric View**. These options are also available in the Source View drop-down menu in the Main Display window (see “Viewing Source Code Structure” on page 68). You can select only one of these options at any given time.

If:	Then:
You want to show the source code hierarchy in the Main Display window so that thread objects are children of function objects.	Select View → Function-centric View
You want to show the source code hierarchy in the Main Display window so that thread objects are children of process objects, and parents of file objects.	Select View → Thread-centric View

Expanding and Collapsing Objects

You can expand or collapse objects in the Main Display window by using the next two View menu options: **Expand All** and **Collapse All**. When a new set of profile data files is loaded into the tool, only a root object and process object or objects are shown in the Main Display window. You can expand all objects in one action, or you can select an object to expand it. Also, you can collapse all objects in one action, or select an object to collapse all objects under it. Options for expanding or collapsing a selected object are also available in the Object menu. Refer to “Viewing Selected Objects” on page 76 for more information.

If:	Then:

You want to expand objects.	If:	Then:
	You want to expand all objects.	Select View → Expand All . This will show all objects, including file, function, and thread, if available, in the Main Display window.
	You want to expand a selected object.	Click on the handle associated with the object.
You want to collapse objects.	If:	Then:
	You want to collapse all objects.	Select View → Collapse All . This will hide all lower level objects. The root and process objects will be the only ones remaining in the Main Display window.
	You want to collapse a selected object.	Click on the handle associated with the object.

Note: You can also use the Object menu to expand or collapse selected objects. Refer to “Viewing Selected Objects” on page 76 for more information.

Sorting Objects

You can sort objects in the Main Display window using the following View menu options: **Sort Alphabetically** and **Sort by Current Data View**.

If:	Then:
You want to re-sort objects by their alphabetic or numeric order.	Select View → Sort Alphabetically . This option sorts objects in each layer by their alphabetic or numeric order.
You want to re-sort objects using the current data view as the key for sorting.	Select View → Sort by Current Data View . This option re-sorts objects using the current data view in the bar chart of the Main Display window as the key for the sorting operation. Objects in each layer are sorted; those with larger values are moved to the top of the layer. Objects of different layers maintain their relative position. Sorting does not alter their relative position in the source hierarchy. After sorting is completed, any object being expanded or collapsed is situated according to the sorting key view. When a new data view is selected for the bar chart values, the sorting order remains unchanged. The sorting key used in the last sorting operation is still used as the measure to place objects. Only when you select this option again will objects be re-sorted.

The following options are also available in the Data View drop-down menu in the Main Display window (see “Viewing Selected Profile Data” on page 69). You can select only one of them at any given time. The item you select becomes the criterion used to construct the bar chart in the Main Display window. Since not every profile data file contains all the possible profile data types, an option that does not belong to any object in the loaded profile data will be grayed out in the View menu, and will not be included in the Data View drop-down menu.

Note: Regardless of which data field is selected, the **leaf nodes** in the source code tree will have runtime performance data associated with them. A leaf node is one that cannot be further expanded. Other intermediate nodes (that is, nodes that can be expanded) will have the roll-up data of their child nodes. The only exceptions are the following fields:

- Maximum Resident Memory Size — the maximum value among child nodes will be used.
- Instructions Per Cycle — each instructions per cycle value will be re-calculated at each intermediate layer.

Viewing Function Call Count

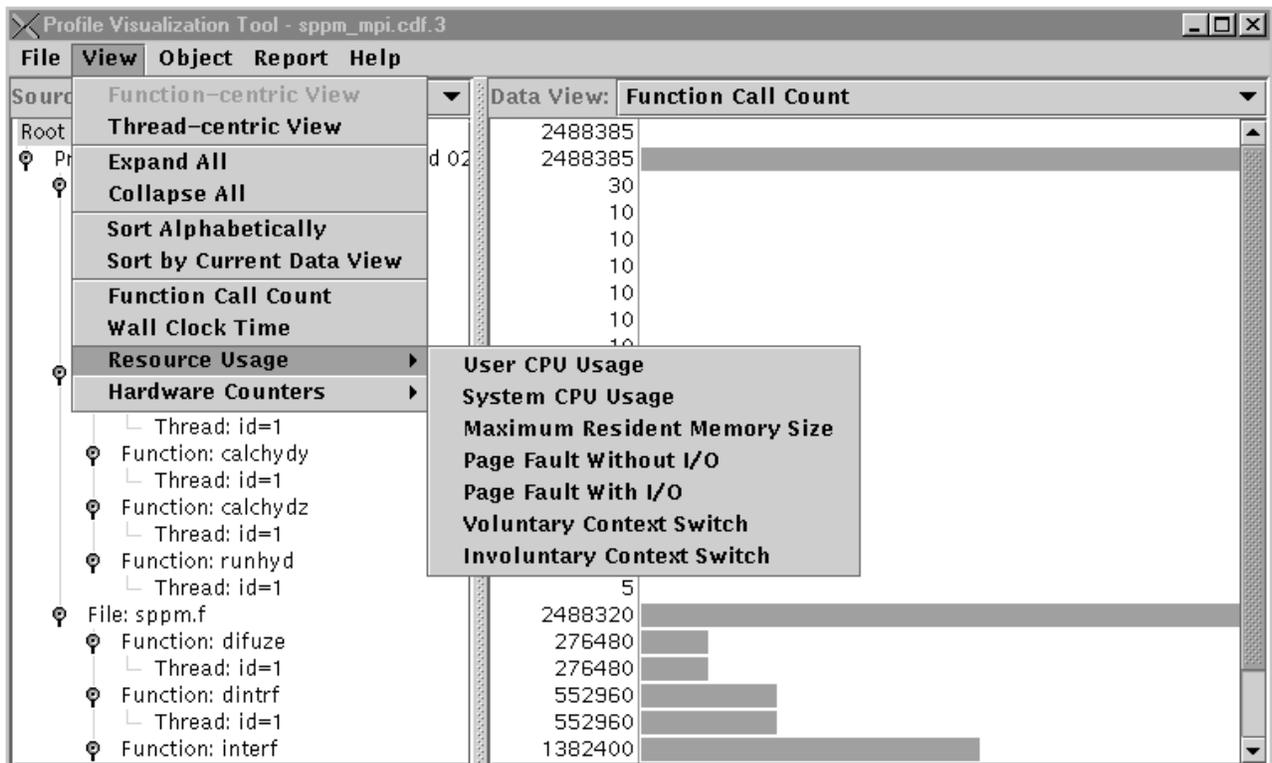
When you select the **Function Call Count** option, function call count values are used to construct the bar chart in the Main Display window.

Selecting Wall Clock Time

When you select the **Wall clock time** option, elapsed wall clock time values are used to construct the bar chart in the Main Display window.

Selecting Resource Usage Options

When you select the **Resource Usage** option, a secondary drop-down menu opens. You can select only one option at a time from this secondary list.



Note: The secondary menu is available only as part of the View menu. The Data View drop-down menu does not support the secondary list.

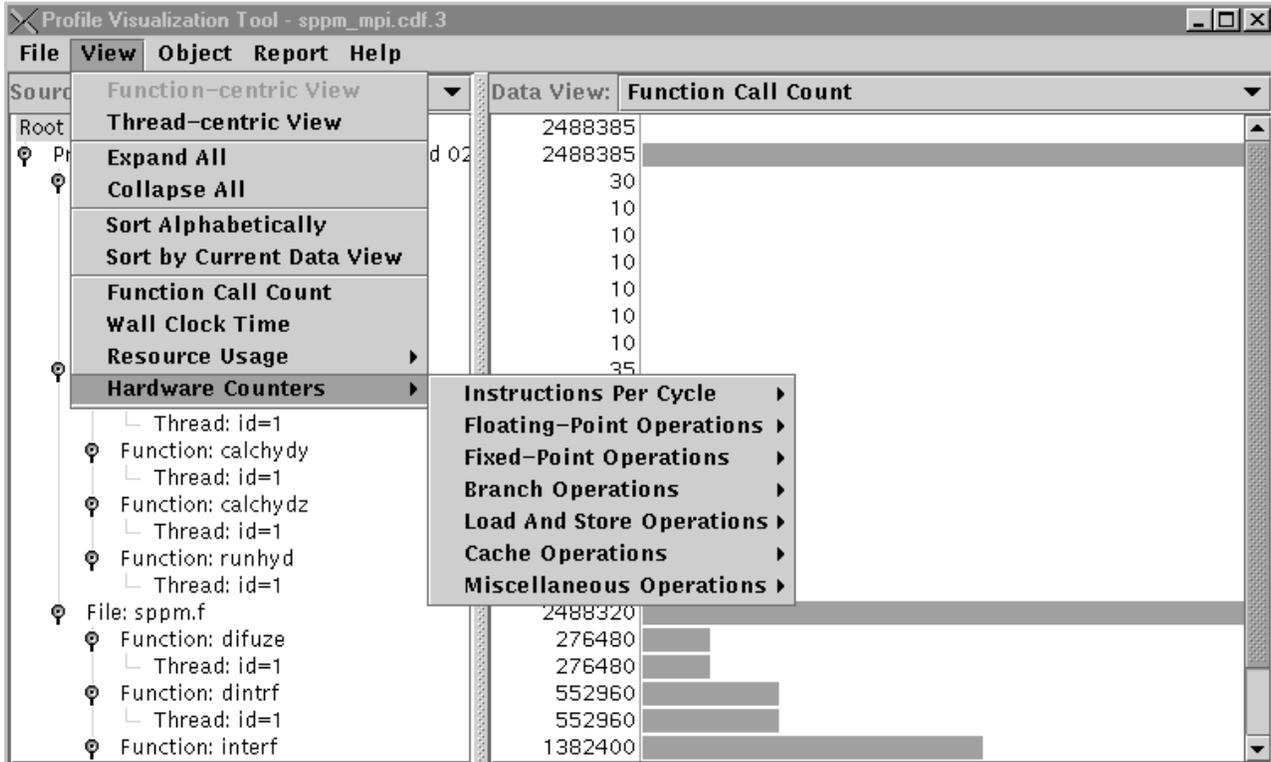
The following list shows the options available from the Resource Usage menu:

- **User CPU Usage**
- **System CPU Usage**
- **Maximum Resident Memory Size**

- **Page Fault Without I/O**
- **Page Fault With I/O**
- **Voluntary Context Switch**
- **Involuntary Context Switch.**

Viewing Hardware Counters

When you select the **Hardware Counters** option, a secondary drop-down menu opens showing a list of supported hardware counter groups. Under each group that you select, there is a list of hardware counter events from which to choose.



Note: The secondary menu is available only as part of the View menu. The Data View drop-down menu does not support the secondary list.

When you select a hardware counter event, it becomes the value used in constructing the bar chart in the Main Display window. Under the View menu, only the hardware counter options that have corresponding values in the profile data are activated; all the rest are grayed out. The following list shows the options available from the Hardware Counters menu, and their corresponding hardware counter events:

- **Instructions Per Cycle**
 - Floating-Point Instructions Per Cycle
 - Fixed-Point Instructions Per Cycle
 - Branch Instructions Per Cycle
 - Load Instructions Per Cycle
 - Store Instructions Per Cycle.
- **Floating-Point Operations**
 - FPU0 (Floating-Point Unit 0) Produced Result

- FPU1 (Floating-Point Unit 1) Produced Result
- FPU Divide Executed
- FPU Multiply-Add Executed
- FPU Add, Subtract, or Multiply Executed
- FPU FSQRT Executed
- FPU FCMP Executed
- FPU (Floating-Point Unit) Produced Result (604e).
- **Fixed-Point Operations**
 - FXU0 (Fixed-Point Unit 0) Produced Result
 - FXU1 (Fixed-Point Unit 1) Produced Result
 - FXU2 (Fixed-Point Unit 2) Produced Result
 - FXU (Fixed-Point Unit) Produced Result (604e).
- **Branch Operations**
 - BPU (Branch Unit) Produced Result
 - A Conditional Branch Was Predicted
 - Global Cancel Due To A Branch Guessed Wrong
 - BPU (Branch Unit) Produced Result (604e)
 - Branch Misprediction Correction From Execute Stage (604e).
- **Load and Store Operations**
 - Number Of Load Instructions Completed
 - Number Of Store Instructions Completed
 - Number Of Cycles Load Stalled Due To Interleave Conflict
 - Number Of Load Instructions Completed (604e)
 - LSU (Load And Store Unit) Produced Result (604e)
 - Number Of Cycles A Load Miss Took (604e).
- **Cache Operations**
 - L1 I-cache Miss
 - A Load Miss Occurred In L1
 - A Store Miss Occurred In L1
 - TLB Miss, Included Both D-cache And I-cache Miss
 - Snoop Hit Occurred And L2 Had The Valid Block
 - Number Of D-cache Prefetch Blocked Due To Four Streams
 - Number Of D-cache Prefetched And Used
 - RWITH Caused L2 Miss
 - Burst Read Caused L2 Miss
 - Instruction Cache Miss (604e)
 - Data Cache Miss (604e)
 - Instruction TLB Miss (604e)
 - Data TLB Miss (604e)
 - Valid Snoop Request Received (604e)
 - Number Of Snoop Hits Occurred (604e).
- **Miscellaneous Operations**
 - Processor Clock Cycles
 - Processor Clock Cycles (604e)
 - Number Of Pipeline Flushing Instructions (604e).

Along with the ability to view how data is presented, you can also view specific data about selected objects. The following sections describe how to view these selected objects.

Viewing Selected Objects

You can view information about objects you select using the Object menu. The functions available with this menu only apply to the selected objects and, in some cases, their child objects. The Object menu enables you to view the following information:

- Source code
- Profile data
- Statistics reports
- Process IDs.

In addition, the Object menu has expand and collapse options that allow you to show or hide the child objects of a selected object.

To show the Object menu, first select an object, then click on **Object** on the Main Display window's menu bar. The object menu is grayed out until an object is selected.

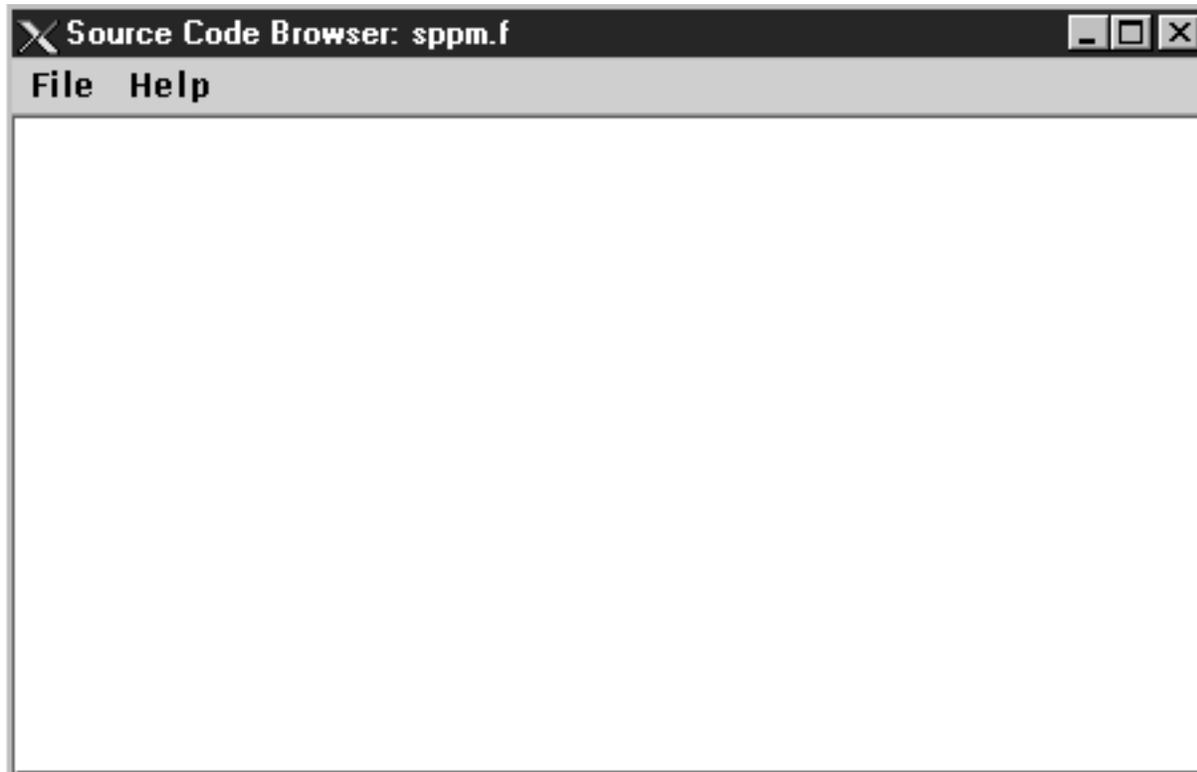
Object	ew:	Function Call Count
Root node (f)	88385	
Process: t	88385	
File: b	30	
Fu	10	
Fu	10	
Thread: id=1	10	
Function: zbdrys	10	
Thread: id=1	10	
File: runhyd3.f	35	
Function: calchdx	10	
Thread: id=1	10	
Function: calchdy	10	
Thread: id=1	10	
Function: calchdz	10	
Thread: id=1	10	
Function: runhyd	5	
Thread: id=1	5	
File: sppm.f	2488320	
Function: difuze	276480	
Thread: id=1	276480	
Function: dintrf	552960	
Thread: id=1	552960	
Function: interf	1382400	

The Object menu also appears as a pop-up menu for a selected object in the Main Display window. Left-clicking on an object in the source code structure section of the Main Display window will select the object, then right-clicking will open the Object menu.

The following sections describe the Object menu options.

Viewing Selected Source Code

You can view the source code of a selected object using the Source Code Browser. To open the Source Code Browser select **Object** → **Show Selected Source Code**.

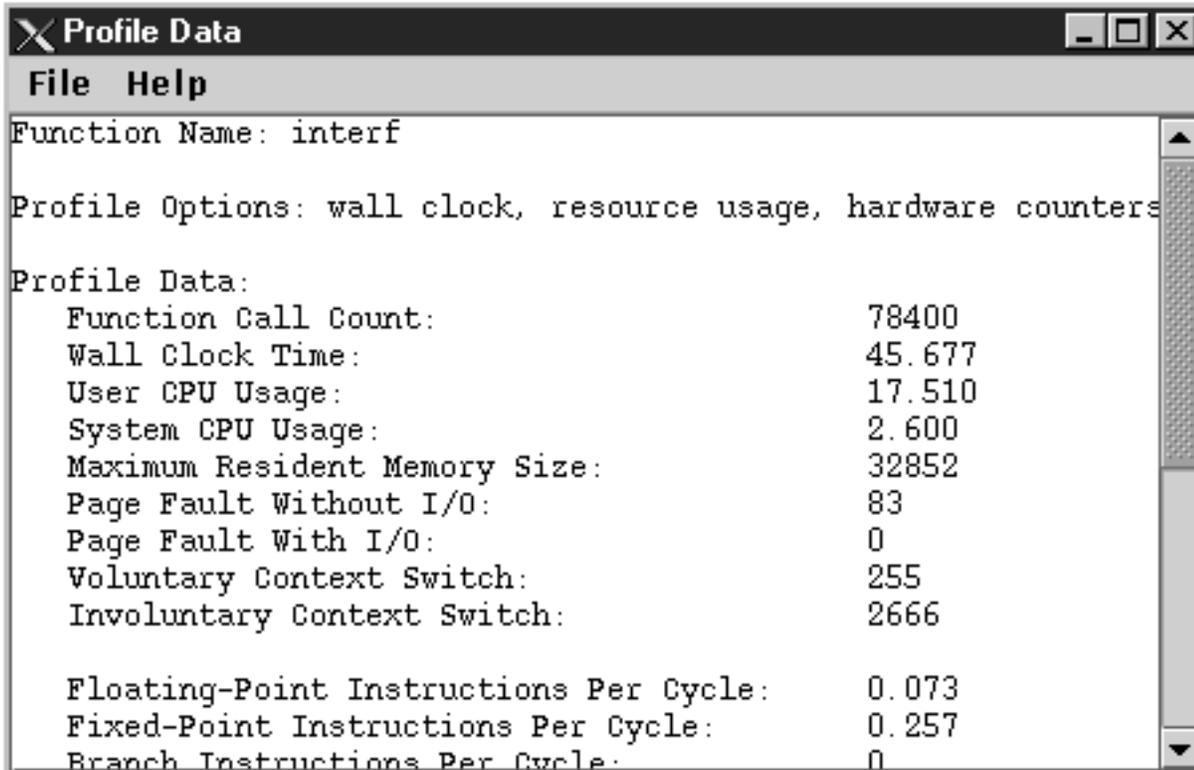


When you choose to view a selected object's source code, the Source Code Browser is used to display the source code. If the object you select is not a file of function object, the Show Selected Source Code option is grayed out. If the source code file cannot be found, an error message panel opens to report the missing file. The Source Code Browser has a File menu that provides one option, **Close**, that is used to close down the browser.

Note: You can select **File** → **Preferences** from the Main Display window menu bar to specify the location of a source code file. Refer to "Specifying User Preferences" on page 87 for more information.

Viewing Selected Profile Data

You can view the profile data of a selected object. The profile data is shown in the Profile Data window. To open the Profile Data window select **Object** → **Show Selected Profile Data**.



Process objects have their own runtime profile data, which is collected by profile probes inserted in a running application by the Performance Collection Tool (refer to “Chapter 2. Using the Performance Collection Tool” on page 5 for more information). Each process object represents a profile session. There are two sets of data associated with each process object. One set is the **profile-session data** collected at runtime. The other set is the summary of each process’ child objects’ profile data, that is, the **function-summation data**. Other objects contain summary data of their child objects’ profile data.

The Profile Data window for a process object includes both the profile-session data and function-summation data associated with each process object.

Viewing Selected Statistics Reports

You can view a statistics report on a selected function object. To do this select **Object → Show Selected Statistics Report**. This opens the Function Statistics Report window.

...	this	Summary	Average	StdDev	Max	Max ID	Min
	78400	311040	311040	0	311040	3	311040
Wall Cloc...	45.677	175.356	175.356	0	175.356	3	175.356
User CPU ...	17.510	66.800	66.800	0	66.800	3	66.800
System C...	2.600	9.180	9.180	0	9.180	3	9.180
Maximum...	32852	32852	32852	0	32852	3	32852
Page Faul...	83	194	194	0	194	3	194
Page Faul...	0	0	0	0	0	0	0
Voluntary...	255	944	944	0	944	3	944
Involunta...	2666	10123	10123	0	10123	3	10123
Floating-...	0.073	0.074	0.074	0	0.074	3	0.074
Fixed-Poi...	0.257	0.259	0.259	0	0.259	3	0.259
Branch In...	0	0	0	0	0	0	0
Load Instr...	0.473	0.477	0.477	0	0.477	3	0.477
Store Inst...	0.168	0.169	0.169	0	0.169	3	0.169
Number ...	4.39246...	1.74286...	1.74286...	0	1.74286...	3	1.74286...
FPU1 (Flo...	160707	514534	514534	0	514534	3	514534
Number ...	1.55898...	6.18562...	6.18562...	0	6.18562...	3	6.18562...
Processor	9.28975	3.65079	3.65079	0	3.65079	3	3.65079

The Show Selected Statistics Report option is only applied to function objects. It is grayed out unless a function object is currently selected.

The Function Statistics Report includes the following profile data:

- Function Call Count
- Wall Clock Time
- User CPU Time
- System CPU Time
- Maximum Resident Set Size
- Page Faults Without I/O
- Page Faults With I/O
- Voluntary Context Switch
- Involuntary Context Switch
- A Hardware Counter Group – which will be a maximum of four (4) rows for 604e CPU architecture, or eight (8) rows for 630 CPU architecture. Each row represents a counter event, for example: L1 I-cache miss.

For each row, there are columns to show the corresponding statistics data. The columns from left to right are:

- Selected object – the selected function object's own profile data value.
- Summary – the summary value of the corresponding function objects in input profile data. Basically these function objects have the same file name, function name, profile options, and ran on the same type of CPU architecture.
- Mean – the average value of the corresponding function objects.
- Standard deviation – the standard deviation value of the corresponding function objects.
- Maximum – the maximum value among the corresponding function objects.
- Maximum id – the task id of the process that has the maximum value.

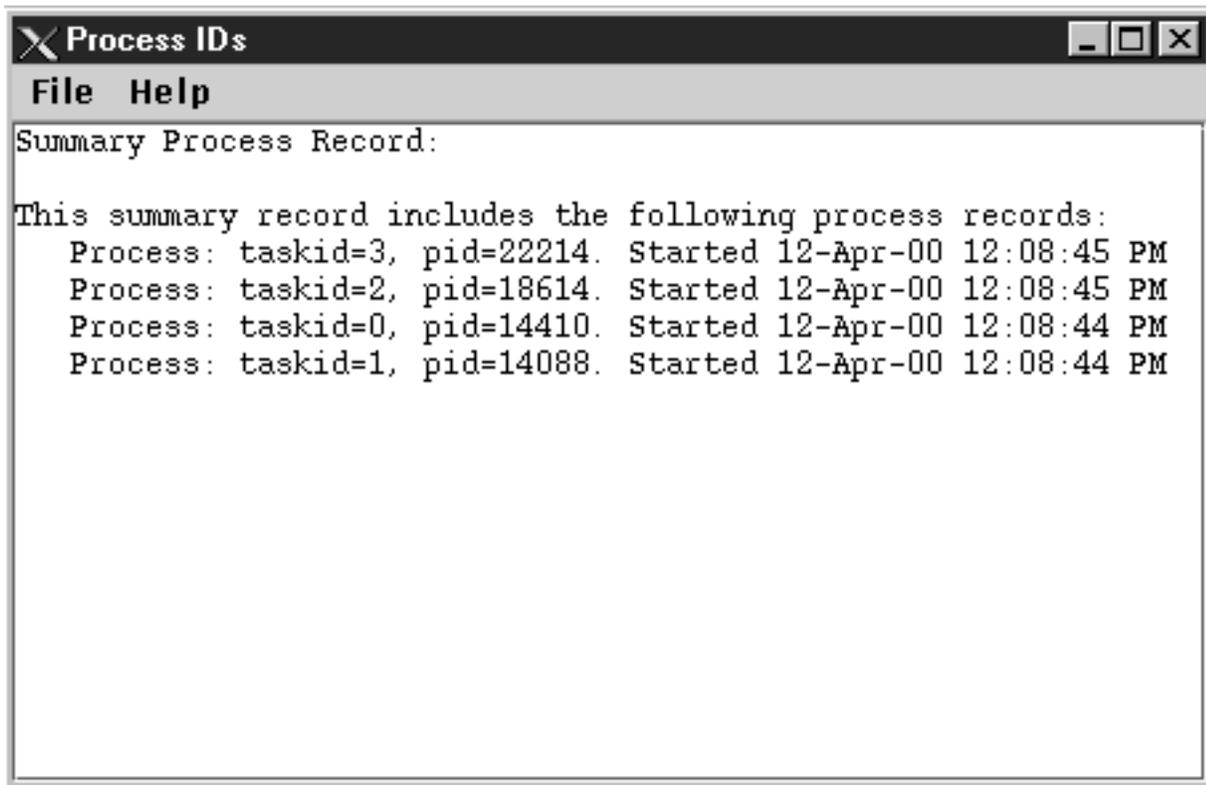
- Minimum – the minimum value among the corresponding function objects.
- Minimum id – the task id of the process that has the minimum value.

The Function Statistics Report window shows the name of the function. The window has a File menu with two options: **Save** and **Close**.

When you save a report, it is saved to a file in plain text format. The title row is saved as the first line in the file. Each row is ended with an end-of-line symbol “\n”, and blank spaces are used as field delimiters to separate data in the same row.

Viewing Selected Process IDs

You can view the process IDs that are included in a summary file by selecting **Object** → **Show Selected Process IDs**. This opens the Process IDs window.



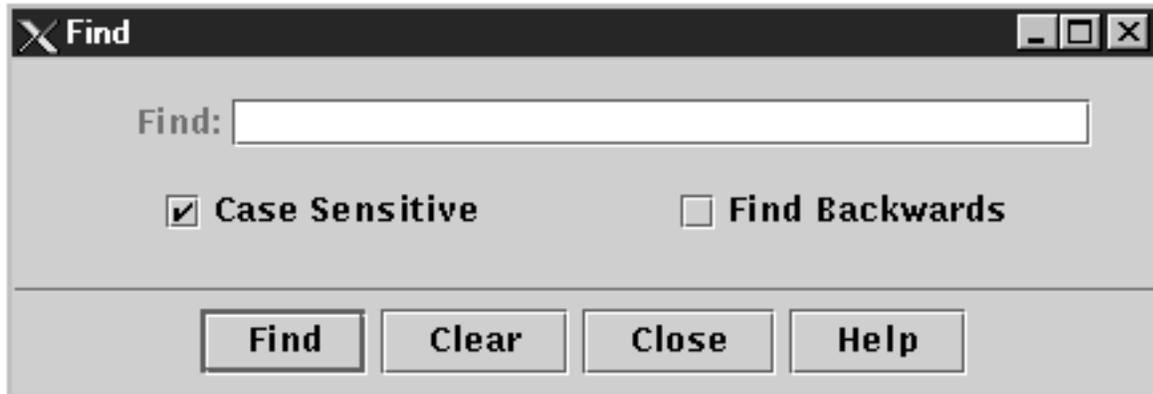
The Show Selected Process IDs option is available only for the summary process objects in a summary file. This option provides the information of how many, and which, individual profile files are used to create a summary file.

Expanding and Collapsing a Selected Object

You can expand or collapse a selected object to show or hide its child objects. To expand a selected object, select **Object** → **Expand Selected Object**. This will show the immediate child objects of the selected object. To collapse a selected object, select **Object** → **Collapse Selected Object**. This will hide all the immediate and distant child objects of the selected object. If the selected object cannot be expanded or collapsed, either option will be grayed out.

Finding Data

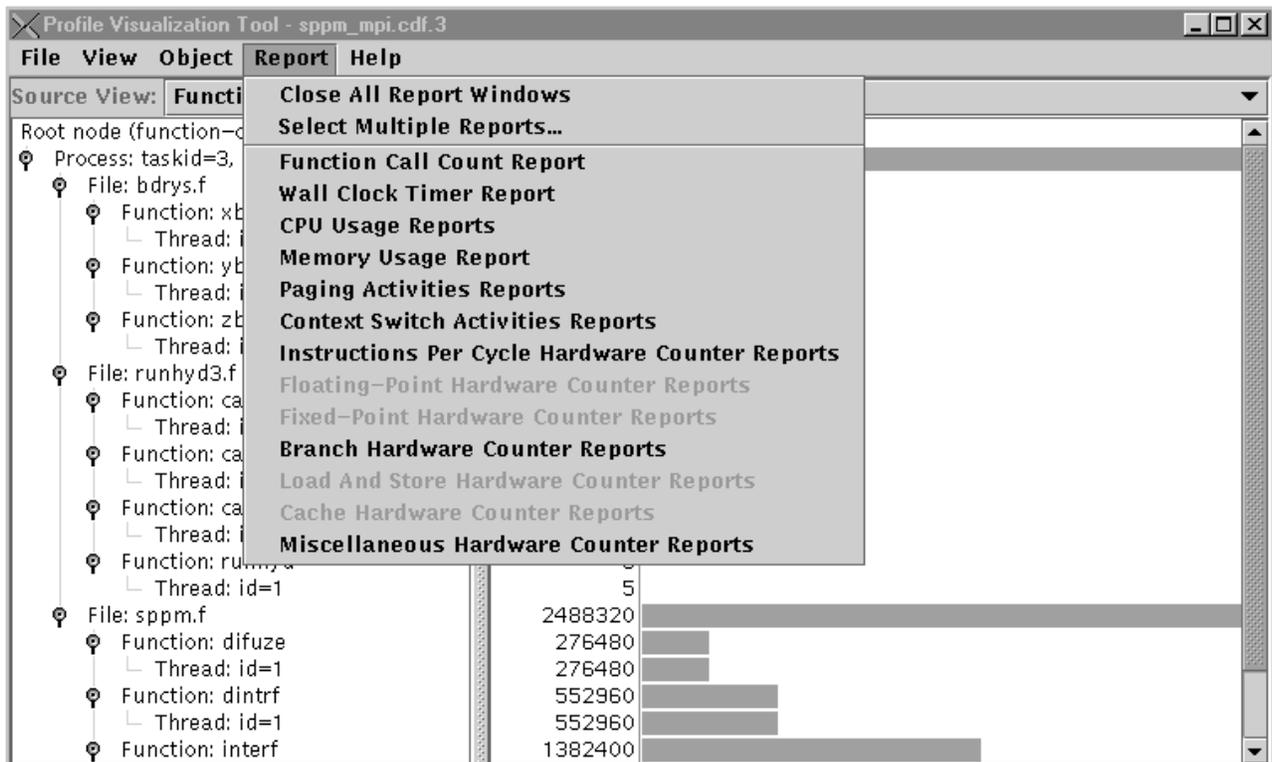
You can specify a text string to search for and find data by using the Find option. Select **File** → **Find...** to open the Find panel.



The Find panel provides a text field to enter the string for which you want to search. Click on **Find** to start the search. The first object in the Main Display window that matches the specified string will be highlighted. The Find panel also provides the **Find Backwards** button to support backward search (the default is forward search). If you de-select the **Case Sensitive** button, the string search is conducted in a case insensitive manner. The search scans through the entire data set, whether or not an object is currently in the Main Display window. A hidden object that matches the search criterion will be made visible in the Main Display window. The Find panel stays open until you select the **Close** button.

Generating Reports of Profile Data

You can generate textual reports of profile data by using the Report menu. To open this menu select **Report**.



The reports you select from the menu will be displayed in report windows, unless you want to select multiple reports. In this case, the Report menu allows you to specify whether a selected report goes to a file or a report window. For multiple reports select **Report** → **Select Multiple Reports...**. This opens the Select Multiple Reports panel.

Select Multiple Reports

Output Report(s) to File: **Browse...**

Output Report(s) to Display Window(s)

Select one or more reports:

- Function Call Count Report**
- Wall Clock Timer Report**
- CPU Usage Reports**
- Memory Usage Report**
- Paging Activities Reports**
- Context Switch Activities Reports**
- Instructions Per Cycle Hardware Counter Reports**
- Floating-Point Hardware Counter Reports**
- Fixed-Point Hardware Counter Reports**
- Branch Hardware Counter Reports**
- Load And Store Hardware Counter Reports**
- Cache Hardware Counter Reports**
- Miscellaneous Hardware Counter Reports**

Select All **Deselect All**

OK **Cancel** **Help**

From this panel you can specify what textual reports you would like to view. You can also specify if you want the selected reports displayed in report windows, or if you want the reports written to a file. If you want the reports written to a file, you need to provide a file name for output.

When you choose one of the other options available in the Report menu, a textual report window opens showing the appropriate data for the profiled application. For example, the following figure shows an example of a textual report for User CPU Usage.

Function N...	Summary	Average	StdDev	Max	Max ID	Min	Min ID	File Name
sppm	341.250	341.250	0	341.250	3	341.250	3	sppm.f
interf	98.630	98.630	0	98.630	3	98.630	3	sppm.f
difuze	72.380	72.380	0	72.380	3	72.380	3	sppm.f
dintrf	34.540	34.540	0	34.540	3	34.540	3	sppm.f
calchdy	28.290	28.290	0	28.290	3	28.290	3	runhyd3.f
calchdz	28.230	28.230	0	28.230	3	28.230	3	runhyd3.f
calchdx	25.910	25.910	0	25.910	3	25.910	3	runhyd3.f
xbdrys	5.390	5.390	0	5.390	3	5.390	3	bdrys.f
zbdrys	5	5	0	5	3	5	3	bdrys.f
ybdrys	4.420	4.420	0	4.420	3	4.420	3	bdrys.f
runhyd	0.780	0.780	0	0.780	3	0.780	3	runhyd3.f

The following list shows the other options available in the Report menu.

Note: For a complete list of the reports available for each group, refer to “Viewing Hardware Counters” on page 74.

- **Function Call Count Report**
- **Wall Clock Timer Report**
- **CPU Usage Reports**
 - User CPU Time
 - System CPU Time.
- **Memory Usage Report**
- **Paging Activities Reports**
 - Page Faults Without I/O
 - Page Faults With I/O.
- **Context Switch Activities Reports**
 - Voluntary Context Switch
 - Involuntary Context Switch.
- **Instructions Per Cycle Hardware Counter Reports**
 - Floating-Point Instructions Per Cycle
 - Fixed-Point Instructions Per Cycle
 - Branch Instructions Per Cycle
 - Load Instructions Per Cycle
 - Store Instructions Per Cycle.
- **Floating-Point Hardware Counter Reports**
 - Floating-point operations – each report represents a floating-point data type.
- **Fixed-Point Hardware Counter Reports**
 - Fixed-point operations – each report represents a fixed-point data type.
- **Branch Hardware Counter Reports**

- Branch operations – each report represents a branch data type.
- **Load and Store Hardware Counter Reports**
- Load and store operations – each report represents a load or store data type.
- **Cache Hardware Counter Reports**
- Cache operations, including L 1/2 cache, TLB, snoop,etc. – each report represents a cache data type.
- **Miscellaneous Hardware Counter Reports**
- Counter events not included in any other counter report groups – each report represents a data type.

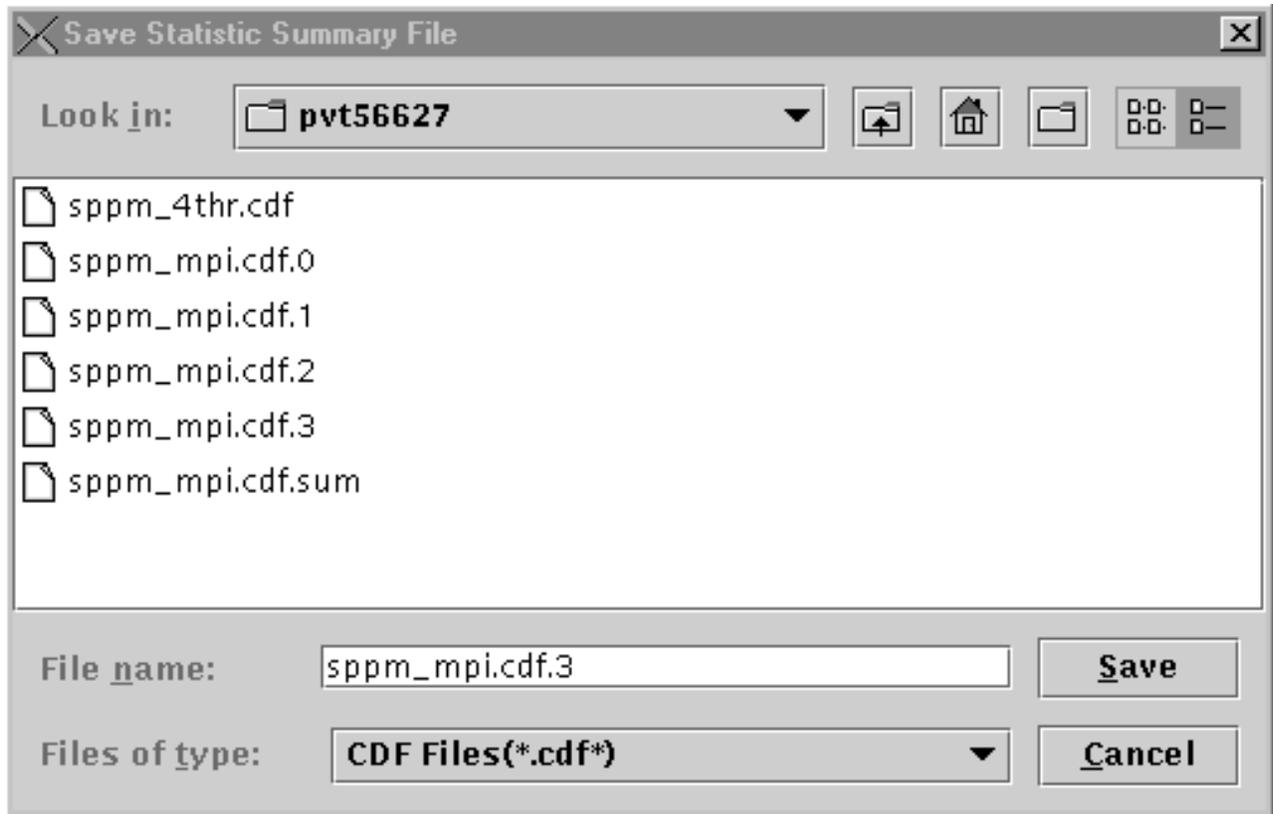
Each row in the reports represents a function object being profiled. Since a function can be profiled with different options in different profile sessions, there can be one function object for each profile session. In other words, there can be multiple rows with the same function name; each row having different profile options. The first column in each row contains a function name. The rows are sorted by the summary value of each function object's statistics data. Following is a list of the columns in each report, in left to right order:

- Function name – the name of each function object.
- Summary – the summary value of the function's corresponding values in all profile data files.
- Mean – the average value of the function's corresponding values in all profile data files.
- Standard deviation – the standard deviation value of the function's corresponding values in all profile data files.
- Maximum value – the maximum value of the function's corresponding values in all profile data files.
- Maximum id – the task id of the process that has the maximum value.
- Minimum value – the minimum value of the function's corresponding values in all profile data files.
- Minimum id – the task id of the process that has the minimum value.
- File name – the name of the source code file that contains this function.

Like the Function Statistics Report window, each textual report type has a File menu. Refer to "Viewing Selected Statistics Reports" on page 78 for more information. There is a **Find** option in the File menu. The Find option acts the same here as it does in the Main Display window's File menu. Refer to "Finding Data" on page 80 for more information. In the Textual Report window, the Find option is used to search for a function name in the report. When there is a matched function name, the matched row is highlighted and brought to the center of the display window.

Saving Summary Data

You can save the statistic summary data of a profile file. Select **File → Save Statistic Summary...** to open the Save Statistic Summary File panel.



From this panel you can specify a file name to which the statistic summary data of the input profile file is written.

When individual profile files are merged and summarized to create a summary profile file, some of the data that exists in individual files is lost in the merging process. For example, there is no thread related data, and you can no longer link a summary function record to its corresponding process object (each process object represents a profile session). This is because thread ids and profile session ids are not consistent across processes. Each process can assign a different id for the same part of code in a program. During the merge process, records that have the following same information are merged together:

- CPU type
- File name
- Function name
- Profile option.

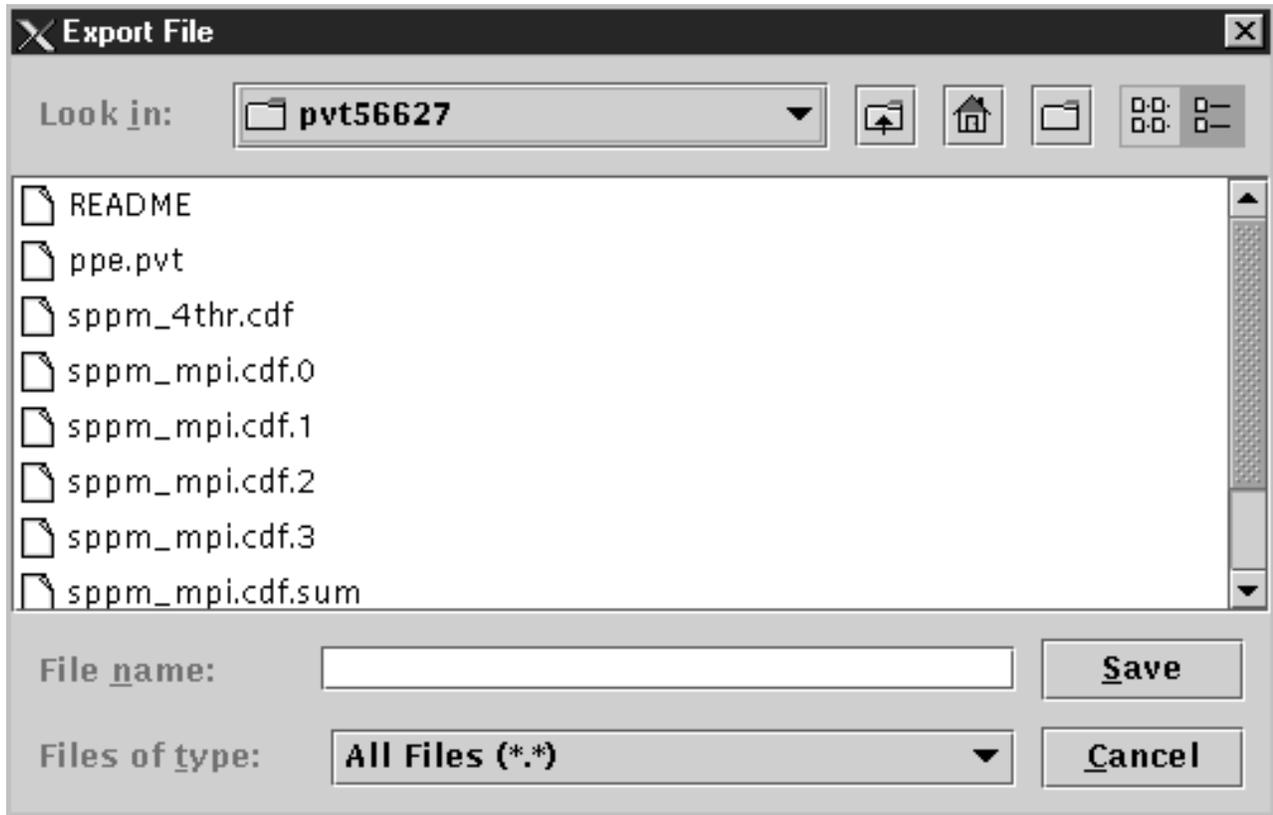
Then the following statistics records are generated:

- Summary record
- Average record
- Max value record
- Max id record (a task id indicating which task has the max value)
- Min value record
- Min id record.

Each individual profile file has one profile-session record and one function-summation record for each profile session. The records are copied over to the newly created summary profile file. The summary file uses `.cdf` as its file name suffix.

Exporting Profile Data

You can export profile data by selecting **File → Export...** The Export option will open a file selection panel that accepts a user-specified file name.



The currently loaded profile data is written to the user-specified file in plain text format, so the data can be loaded easily into a spreadsheet tool, like Lotus 1-2-3.

The export file contains descriptions of the hardware counter events used in the file. It also contains information about each profile session, for example, pid, task id, machine hostname, IP address, CPU type, and OS level. If the input file is a summary file, then the fields mentioned above all have value "N/A". If the records belong to one profile session, each of them represents a thread's profile data. If the input file is a summary file, then each record represents the summary of thread activities within a function.

Each record in an export file contains the following information:

- Record type
- Task id
- Pid
- CPU type
- File name

- Function name
- Thread id
- Profile option
- Call count
- Wall clock
- User CPU
- System CPU
- Max memory size
- Page fault without I/O
- Page fault with I/O
- Voluntary context switch
- Involuntary context switch
- Counter event id [8]
- Counter event value [8].

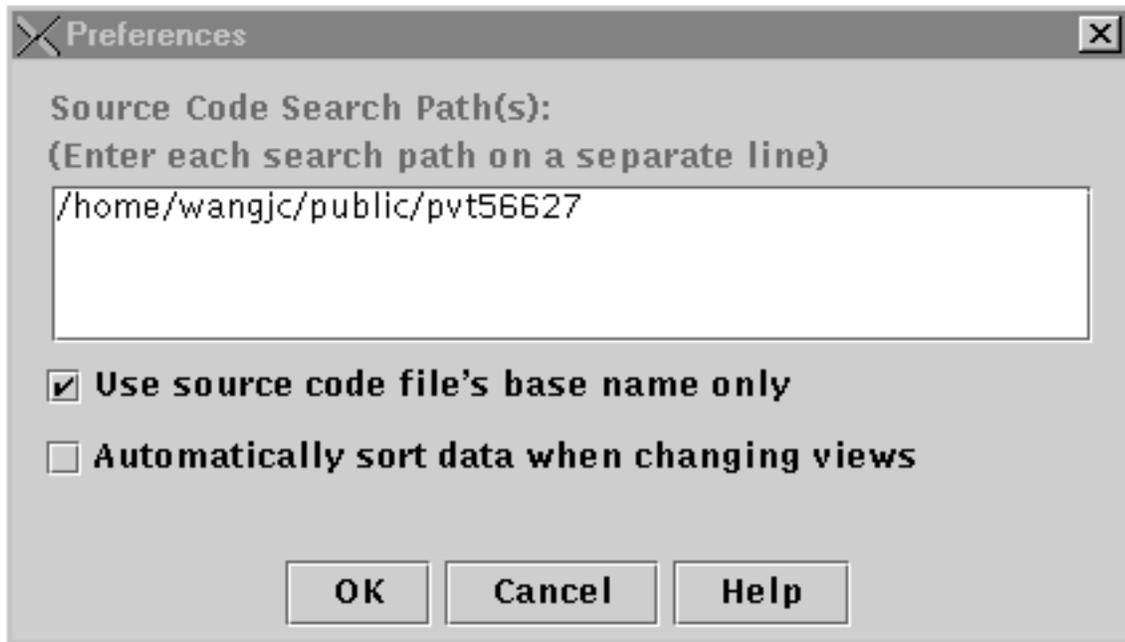
The “counter event id” field and “counter event value” field are arrays with eight entries each. Each entry represents a hardware counter event being profiled and its value, respectively. The word “N/A” is assigned to any “counter event value” entry that is not used during the performance profiling phase.

For each profile session, there is one record that represents the profile session data associated with the session. The record has the same format as other records, but both its file name field and function name field have the value “N/A”.

When you open the Export file selection panel, you are prompted to enter an output file name. All the data available in input profile files is exported. Comment records that start with a number sign (#) are added to the output file to describe what type of data is stored in the file, the runtime environment, and the record format used in the export file. Each record is delimited by an end-of-line symbol (\n), and the fields in each record are separated by semicolon symbols (;). All records have an equal number of fields. Any field that has no meaning in a particular record type is assigned the value “N/A”. The export file uses *.txt* as its file name suffix.

Specifying User Preferences

There is one user preference you can specify: where source code files reside. Select **File** → **Preferences...** to open the Preferences panel.



You can enter the appropriate source code search path in the text field. The Source Code Browser looks for source code files at the specified locations, in addition to the default locations. You can also use this option to change source file search order. You can specify to search default location first, last, or in between two file paths.

The text field for the source code search path is a multi-line text field. Each line represents a specified path. If you select the option “use a file’s base name only”, the base name of a file to be searched will be appended to each specified path and searched in that order. If you de-select the option, then a file’s full name (that is, path, if any, plus the base name) will be appended to each specified path, and the combining string will be used in locating the file.

Exiting the Profile Visualization Tool

To exit the Profile Visualization Tool, select **File → Exit**. This will stop Profile Visualization processing, and close the Main Display window and any panels or menus that are open.

Using the Profile Visualization Tool’s Command Line Interface

The Profile Visualization Tool provides a command-line interface that enables you to process profile files directly without initializing the graphical user interface. The subcommands available in the command-line interface correspond to the options available in the graphical user interface. For more information on the graphical user interface, refer to “Using the Profile Visualization Tool’s Graphical User Interface” on page 65.

Profile Visualization Tool (Command Line Interface) Overview

The Profile Visualization Tool’s command-line interface allows you to process profile data directly without using the graphical user interface. After initializing the command-line interface, you can enter the appropriate subcommands that enable you to:

- Load files for processing
- Create a summary file of all the loaded data
- Generate textual reports of profile data
- Export profile data to a file.

The following sections describe the command-line interface in greater detail.

Starting the Profile Visualization Tool in Command-Line Mode

To start the Profile Visualization Tool in command-line mode, enter:

```
pvt -cmd
```

Doing this starts a command-line session without associated profile data. To start a command-line session with associated profile data, enter:

```
pvt -cmd one_or_more_file_names
```

Once you start a command-line session, the command line prompt changes to **pvt>** and remains this way until you enter the **exit** command to end the command-line session.

The following sections describe the command-line mode subcommands.

Loading Files

You can load a set of profile data files into the session with the **load** command. Enter:

```
load one_or_more_file_names
```

If a set of data already exists, then the existing data is discarded and the newly loaded data becomes the current data to be used in future actions.

Creating a Summary File

You can create a summary file of all the loaded data with the **sum** command. Enter:

```
sum summary_file_name
```

The merged summary data is written to the file that you specify in the command, with a suffix of *.cdf* being appended to the specified file name.

Generating Reports

You can generate textual reports of profile data using the **report** command. You can specify several different options with the report command, depending on what type of output you want. To show a list of available report types, enter:

```
report list
```

The result will look something like:

- **[0] call_count:** function call count report
- **[1] wclock:** wall clock timer report
- **[2] ru_cpu:** CPU usage reports
- **[3] ru_mem:** memory usage report
- **[4] ru_paging:** paging activities reports
- **[5] ru_cswitch:** context switch activities reports
- **[6] pmc_cycle:** instructions per cycle hardware counter reports
- **[7] pmc_fpu:** floating-point hardware counter reports

- **[8] pmc_fxu:** fixed-point hardware counter reports
- **[9] pmc_branch:** branch hardware counter reports
- **[10] pmc_lsu:** load and store hardware counter reports
- **[11] pmc_cache:** cache hardware counter reports
- **[12] pmc_misc:** miscellaneous hardware counter reports

For details of these reports, refer to “Generating Reports of Profile Data” on page 81.

To generate all the available reports to a file, enter:

```
report output_file_name
```

To generate reports by report name to a file, enter:

```
report "one_or_more_report_names" output_file_name
```

For example:

```
report "wclock,ru_cpu" output
```

To generate reports by report id to a file, enter:

```
report "one_or_more_report_ids" output_file_name
```

For example:

```
report "1,2" output
```

The report names or report ids in double quotes must be separated by a comma, with no blank space in between. No matter how many reports are selected in one report command, all the reports are outputted to a single file specified in the report command.

Exporting Files

You can export profile data to a specified file using the **export** command. Enter:

```
export output_file_name
```

A suffix *.txt* will be appended to the specified file name.

The currently loaded profile data is written to the user-specified file in plain text format, so the data can be loaded easily into a spreadsheet tool like Lotus 1-2-3. The data that is loaded into the tool can be grouped into the following types of records:

- Profile-session record associated with each process (that is, profile session)
- Individual function or thread records
- Function statistics records.

For more information on exporting data, refer to “Exporting Profile Data” on page 86.

Exiting the Profile Visualization Tool

You can end a command-line session with the **exit** command. Enter:

```
exit
```

Chapter 4. Creating, Converting, and Viewing Information Contained In, UTE Interval Files

When you collect MPI and user event traces using the Performance Collection Tool (as described in “Chapter 2. Using the Performance Collection Tool” on page 5), the collected information is saved, on each machine running instrumented processes, as a standard AIX event trace file. In order to view the information contained in these standard AIX trace files, you will first need to convert them into UTE (Unified Trace Environment) interval files. While an AIX event trace file has a time stamp indicating the point in time when an event occurred, UTE interval files take this information to also determine how long an event lasts. Because they include this duration information, UTE interval files are easier to visualize than traditional AIX event trace files. The UTE utilities are:

- The **convert** utility which converts AIX event trace files into UTE interval trace files.
- The **utemerge** utility which merges multiple UTE interval files into a single UTE interval file.
- The **utestats** utility which generates statistics tables from UTE interval files.
- The **slogmerge** utility which converts and merges UTE interval files into a single SLOG file for analysis within Argonne National Laboratory’s Jumpshot tool.

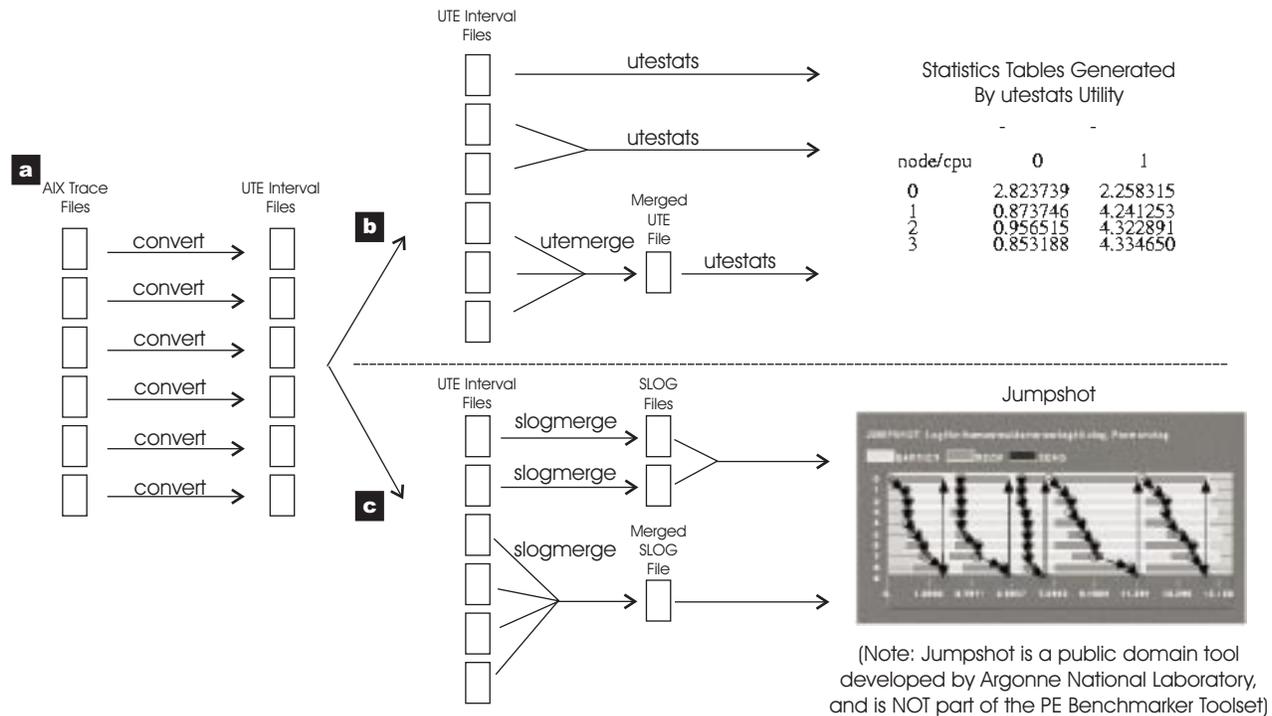


Figure 2. Unified Trace Environment (UTE) Utilities

The preceding figure illustrates the UTE utilities you can use to either generate statistics tables from UTE interval files or view statistics graphically using Argonne National Laboratory’s Jumpshot tool. Regardless of whether you want to view the statistics in simple tables or graphically in Jumpshot, the first thing you’ll need to do is use the **convert** utility to create UTE interval files from the AIX trace files (a). (See “Converting AIX Trace Files Into UTE Interval Trace Files” on page 92 for more

information.) Then, if you want to view the statistics in simple tables (**b**), you can use the **utestats** utility. You can optionally merge multiple UTE files into a single UTE file using the **utemerge** utility before using the **utestats** utility to generate the statistics tables. (See “Generating Statistics Tables From UTE Interval Trace Files” on page 93 for more information.) If you instead want to view the information contained in the UTE interval files graphically (**c**), you can convert them into SLOG files using the **slogmerge** utility. The **slogmerge** utility can either convert a single UTE interval file into a single SLOG file, or it can merge multiple UTE interval files into a single SLOG file. The SLOG files are readable by Argonne National Laboratory’s Jumpshot Tool. (See “Converting UTE Interval Files Into SLOG Files Required By Argonne National Laboratory’s Jumpshot Tool” on page 94 for more information.)

Note: The UTE utilities are intended only for the AIX event trace files generated when you collect MPI and user event traces with the Performance Collection Tool. If you instead collect hardware and operating system profiles, the information is output by the Performance Collection Tool as NetCDF (network Common Data Form) files and these UTE utilities are not necessary. Instead, the NetCDF files can be read directly into the Performance Collection Tool as described in “Chapter 3. Using the Profile Visualization Tool” on page 65.

The following sections provide an overview of the UTE utilities. Note, however, that this section does not attempt to describe all the options available when using these utilities. For complete reference information on any of the utilities described in this section, refer to their man pages contained in “Appendix B. PE Benchmarking Command Reference” on page 115.

Converting AIX Trace Files Into UTE Interval Trace Files

Regardless of whether you want to view the statistics you have collected in simple tables, or graphically in Jumpshot, the first thing you’ll need to do is use the **convert** utility to create UTE interval files from the AIX trace files generated by the Performance Collection Tool. When you collect MPI and user event traces, the collected information is saved, on each machine running instrumented processes, as a standard AIX event trace file. The names of these individual trace files will consist of a common “base name” that you specified using the Performance Collection Tool, followed by a node-specific suffix supplied by the tool itself. Using the **convert** utility, you can convert either a single AIX trace file into a UTE interval file, or a set of AIX trace files with the same prefix into a set of UTE interval files.

To convert a single AIX trace file into a UTE interval file, simply pass the **convert** utility the name of the trace file located in the current directory. For example, to convert the AIX trace file *mytrace* into a UTE interval trace file, enter:

```
convert mytrace
```

Using the **-o** flag, you can optionally specify the name of the output UTE interval file. For example, to specify that the output file should be named *outute*.

```
convert -o outute mytrace
```

To convert a set of AIX trace files into a set of UTE interval files, simply specify the number of files using the **-n** option, and supply the common “base name” prefix shared by the files. For example, to convert five trace files with the prefix *mytraces* into UTE interval files, copy the trace files to a common directory and enter:

```
convert -n 5 mytraces
```

You can optionally use the `-o` option to specify a file name prefix for the resulting UTE interval files.

```
convert -n 5 -o outute mytraces
```

For complete reference information on the **convert** utility, refer to its man page in “Appendix B. PE Benchmark Command Reference” on page 115. If you want to view the statistics information contained in the UTE file(s) in simple tables, refer to “Generating Statistics Tables From UTE Interval Trace Files”. If you want to view the statistics information contained in the UTE file(s) graphically, refer to “Converting UTE Interval Files Into SLOG Files Required By Argonne National Laboratory’s Jumpshot Tool” on page 94.

Generating Statistics Tables From UTE Interval Trace Files

Once you have created UTE interval trace files (as described in “Converting AIX Trace Files Into UTE Interval Trace Files” on page 92), you can generate statistical tables from them using the **utestats** utility. By default, six two-dimensional tables are generated. These tables are:

- Time Bin vs. Node
- Thread vs. Event Type
- Event Type vs. Thread
- Node vs. Event Type
- Event Type vs. Node
- Node vs. Processor

The computed statistic for all tables is the sum or the duration. A Node vs. Processor table would look like the following (where tabs have been replaced by spaces to make the column alignment clearer).

node/cpu	0	1
0	2.823739	2.258315
1	0.873746	4.241253
2	0.956515	4.322891
3	0.853188	4.334650

You can generate these statistics tables for a single UTE interval file or multiple UTE interval files. You can also generate these statistics tables for a merged UTE interval file. A merged UTE interval file is one that consists of multiple UTE interval files that have been merged into one file by the **utemerge** utility.

For example, to generate the statistics tables for the UTE interval file *mytrace.ute*, you would enter:

```
utestats mytrace.ute
```

By default, the statistics tables will be printed to standard output. You can, however, redirect them to a file using the `-o` option on the **utestats** command. For example, to redirect the statistics tables output by the **utestats** utility to the file *stattables*, you would enter:

```
utestats -o stattables mytrace.ute
```

As already stated, you can also specify multiple UTE interval files from which the statistics should be generated.

```
utestats mytrace.ute mytrace2.ute mytrace3.ute
```

Rather than specify multiple UTE interval trace file names on the **utestats** command, you could instead use the **utemerge** utility to first merge the multiple UTE interval trace files into a single UTE interval trace file. To do this, you use the **-n** option on the **utemerge** command to indicate the number of files you want to merge, and supply the common "base name" prefix shared by the files. For example:

```
utemerge -n 3 mytrace
```

The merged UTE interval file generated by the **utemerge** utility will, by default, be named *trcfile.ute*. To specify your own output file name, use the **-o** option.

```
utemerge -n 3 -o mergedtrc.ute mytrace
```

You can then generate statistics for the merged UTE interval file using the **utestats** command.

```
utestats mergedtrc.ute
```

For complete reference information on the **utestats** and **utemerge** utilities, refer to their man pages in "Appendix B. PE Benchmarking Command Reference" on page 115.

Note: Argonne National Laboratory's Jumpshot Tool also includes a statistics view feature that displays the same information as the **utestats** command. Jumpshot also has the ability to display statistics information graphically. The Jumpshot Tool is described next in "Converting UTE Interval Files Into SLOG Files Required By Argonne National Laboratory's Jumpshot Tool".

Converting UTE Interval Files Into SLOG Files Required By Argonne National Laboratory's Jumpshot Tool

If you would like to view the traces collected by the Performance Collection Tool graphically, you can use the Jumpshot tool developed by Argonne National Laboratory. While Jumpshot is a public domain tool and **not** part of the PE Benchmarking Toolset, we do provide a utility — **slogmerge** — for converting UTE interval files into the SLOG files required by Jumpshot. You can use the **slogmerge** utility to:

- convert a single UTE interval file into a single SLOG file.
- merge multiple UTE interval files into a single SLOG file.

If you are dealing with a massively parallel job, it is unlikely that you will be able to display all the process threads in Jumpshot. Rather than merge all the trace files generated from such a job, you will instead want to merge selected trace files. To determine which files to merge, you can first use the **utestats** utility (as described in "Generating Statistics Tables From UTE Interval Trace Files" on page 93) to determine the characteristics of the files.

To convert a single UTE interval file into a single SLOG file, pass the **slogmerge** command the name of the file located in the current directory. For example:

```
slogmerge mytrace.ute
```

By default, the SLOG file output by the **slogmerge** utility will be *trcfile.slog*. Using the **-o** option on the **slogmerge** command, however, you can specify an output file name. For example:

```
slogmerge -o mergedtrc.slog mytrace.ute
```

To merge multiple UTE interval files into a single SLOG file, use the **-n** option to indicate the number of files to merge and pass the **slogmerge** utility the common "base name" prefix of the files. For example, to merge 3 files whose prefix is *mytrace*, enter:

```
slogmerge -n 3 mergedtrc.slog mytrace
```

For complete reference information on the **slogmerge** utility, refer to its man page in "Appendix B. PE Benchmark Command Reference" on page 115.

Appendix A. Dynamic Probe Class Library (DPCL) Enhancements

This appendix summarizes the enhancements we've made to the Dynamic Probe Class Library to enable the Performance Collection Tool, which is built on DPCL, to search for functions in a target application's source code. These enhancements include:

- some additional functions added to the **SourceObj** class. Objects of the **SourceObj** class (which is defined in the header file *SourceObj.h*) are called "source objects" and are used by the DPCL system to represent the source code structure associated with a target application. Two new functions added to this class enable the analysis tool to get, for a particular process, a list of function names that match a regular expression you supply. This function list is returned as an object of class **FunctionList** (described below).
- a new class — the **FunctionList** class — used to represent a list of functions.
- a new class — the **FunctionId** class — used to represent a single entry in a **FunctionList** object's list of functions.

The information in this appendix is designed to supplement the reference information already contained in the *IBM Parallel Environment for AIX: DPCL Class Reference*. If you are new to DPCL, and would like an overview of this class library, refer to the *IBM Parallel Environment for AIX: DPCL Programming Guide*.

New Functions of Class **SourceObj**

In order to enable an analysis tool (specifically our Performance Collection Tool which is built on the DPCL) to search an application's source code for functions that match a particular regular expression pattern, several new functions have been added to the **SourceObj** class. These new functions are summarized in the following AIX man pages.

bget_function_list

Synopsis

```
#include <SourceObj.h>
FunctionList *bget_function_list(
    const Process &proc,
    char *regex,
    AisStatus *sts);
```

Parameters

proc

process to which the **bget_function_list** request applies

regex

regular expression that represents the pattern to be matched when searching for functions within the process specified by the **proc** parameter.

sts

AisStatus that indicates the success or failure of the blocking call

Description

Sends a request to the server to get, for a particular process, a list of function names that match a regular expression you supply.

Return Values

FunctionList

Returns a list of functions (an object of class **FunctionList**) that match the supplied regular expression for the given process.

See Also

get_function_list

get_function_list

Synopsis

```
#include <SourceObj.h>
AisStatus get_function_list(
    const Process &proc,
    char *regexp,
    GCBFuncType fp,
    GCBTagType tag);
```

Parameters

proc

process to which the **get_function_list** request applies

regexp

regular expression that represents the pattern to be matched when searching for functions within the process specified by the **proc** parameter.

fp callback function to be invoked when this operation succeeds or fails

tag

callback tag to be used when the callback function is invoked

Description

Sends a request to the server to get, for a particular process, a list of function names that match a regular expression you supply.

Note that the **get_function_list** function immediately returns control to the caller upon issuing the request to get the function list. The return value indicates only whether the request was successfully submitted.

Callback Data

The callback function is passed a pointer to the **FunctionList** object (containing the list of function names that match the supplied regular expression for the specified process) as the callback object. The callback message is the request status, of type **AisStatus**, which contains one of the following values:

ASC_success

get function list operation was successful

ASC_operation_failed

get function list operation was unsuccessful

Return Values

The return value for the **get_function_list** function indicates whether the request was successfully submitted.

ASC_success

Request was submitted successfully.

ASC_operation_failed

request was not submitted.

See Also

bget_function_list

Class **FunctionId**

Class **FunctionId** contains a single entry in a function list (as returned by the **FunctionList::get_entry** function). The **FunctionId** object returned contains a string representation of the function name, its mangled name, and its associated source file name.

Constructors

Synopsis

```
#include <FunctionId.h>
FunctionId(void);
FunctionId(const char *source_name, const char *function_name,
           const char *mangled_name);
FunctionId(const FunctionId &oldobj);
```

Parameters

source_name

source file name of the function

function_name

name of the function

mangled_name

mangled name of the function

oldobj

object to be copied into the new **FunctionId** object.

Description

The default constructor creates an empty **FunctionId** object.

The standard constructor uses the arguments provided to initialize the object.

The copy constructor uses the values contained in the **oldobj** to initialize the new (constructed) object.

The constructors provided with this class will return a value of **NULL** if they are invalid.

Exceptions

ASC_insufficient_memory

a memory allocation routine failed.

See Also

class **FunctionList**

get_demangled_name

Synopsis

```
#include <FunctionId.h>
char *get_demangled_name(char *buf, unsigned int len) const;
```

Parameters

buf

caller-allocated buffer that holds the demangled name

len

maximum number of bytes the function will place in the buffer. The **len** parameter should include enough space for a null-terminating byte.

Description

This function copies into the buffer a null-terminated character string representing the demangled name of the function. The name may be truncated if the **len** parameter is smaller than the length of the demangled name. A function's demangled name is the name of a function as it appears in the original source code of a program as seen by a compiler.

Return Values

Returns a pointer to the buffer, which will contain at most **len** bytes of the demangled function name. If the **FunctionId** object is uninitialized, then **NULL** is returned.

See Also

[get_demangled_name_length](#), [get_mangled_name](#), [get_mangled_name_length](#)

get_demangled_name_length

Synopsis

```
#include <FunctionId.h>
unsigned int get_demangled_name_length(void) const;
```

Description

Returns the length, including the null-terminating byte, of the demangled name of the function.

Return Values

The length of the object's demangled name is returned. If the **FunctionId** object is uninitialized, **0** is returned.

See Also

get_demangled_name, **get_mangled_name**, **get_mangled_name_length**

get_mangled_name

Synopsis

```
#include <FunctionId.h>
char *get_mangled_name(char *buf, unsigned int len) const;
```

Parameters

buf

caller-allocated buffer that holds the mangled name

len

maximum number of bytes the function will place in the buffer. The **len** parameter should include enough space for a null-terminating byte.

Description

This function copies into the buffer a null-terminated character string representing the mangled name of the function. The name may be truncated if the **len** parameter is smaller than the length of the mangled name. A function's mangled name is the name of a function as it appears to the linker and loader. Name mangling is supported by compilers and linkers to resolve overloaded function names in object-oriented programming languages. In order to distinguish between two functions that have the same programmer-visible name, compilers encode parameter type information into the actual function name as it is seen by the linker and loader. Mangled names include parameter data type information for some languages, notably C++, but not necessarily for all languages.

Return Values

Returns a pointer to the buffer, which will contain at most **len** bytes of the mangled function name. If the **FunctionId** object is uninitialized, then **NULL** is returned.

See Also

get_mangled_name_length, **get_demangled_name**,
get_demangled_name_length

get_mangled_name_length

Synopsis

```
#include <FunctionId.h>
unsigned int get_mangled_name(void) const;
```

Description

Returns the length, including the null-terminating byte, of the mangled name of a function.

Return Values

The length of the object's mangled name is returned. If the **FunctionId** object is uninitialized, **0** is returned.

See Also

get_mangled_name, **get_demangled_name**, **get_demangled_name_length**

get_module_name

Synopsis

```
#include <FunctionId.h>
char *get_module_name(char *buf, unsigned int len) const;
```

Parameters

buf

caller-allocated buffer that holds the module name

len

maximum number of bytes the function will place in the buffer. The **len** parameter should include enough space for a null-terminating byte.

Description

Copies into the buffer a null-terminated character string representation of the file name and path, if available, of the module that contains the function represented by this **FunctionId** object. The name may be truncated if the **len** parameter is smaller than the length of the module name.

Return Values

Returns a pointer to the buffer, which will contain the file name and path, if available, of the module that contains the function represented by this **FunctionId** object. Returns **0** if the **FunctionId** object is uninitialized.

See Also

get_module_name_length

get_module_name_length

Synopsis

```
#include <FunctionId.h>
unsigned int get_module_name_length(void) const;
```

Description

Returns the length, including the null-terminating byte, of the file name and path, if available, of the module that contains the function represented by this **FunctionId** object.

Return Values

Returns the length of the file name and path, if available, of the module that contains the function represented by this **FunctionId** object. Returns **0** if the **FunctionId** object is uninitialized.

See Also

get_module_name

operator=

IBM Confidential, Limited Rights Data

operator=

Synopsis

```
#include <FunctionId.h>
FunctionId & operator=(const FunctionId &rhs);
```

Parameters

rhs

The right hand operand. This is the existing **FunctionId** object whose value you want to assign to the invoking **FunctionId** object.

Description

Assigns the value of the right hand operand (existing **FunctionId** object) to the invoking **FunctionId** object. The left operand is the invoking object.

For example,

```
FunctionId rhs, lhs;
.
.
.
lhs = rhs;
```

assigns the value of **rhs** to **lhs**. This yields a new copy of the **FunctionId** object.

Return Values

Returns a reference to the invoking **FunctionId** object (the left operand).

operator==

Synopsis

```
#include <FunctionId.h>
int operator==(const FunctionId& compare);
```

Parameters

compare

function id to be compared against the invoking object

Description

This function compares two **FunctionId** objects for equivalence. If the two objects represent the same function, this function returns **1**. Otherwise it returns **0**.

Return Values

This function returns **1** if the two objects are equivalent. Otherwise, it returns **0**.

Class **FunctionList**

Class **FunctionList** defines an object that contains a list of functions that match a regular expression search pattern supplied to the **SourceObj::bget_function_list** or **SourceObj::get_function_list** function. An instance of **FunctionList** should contain a list of 0 or more **FunctionId** objects.

Constructors

Synopsis

```
#include <FunctionList.h>
FunctionList(void);
FunctionList(const FunctionList &oldobj);
```

Description

The default constructor creates a **FunctionList** object. This **FunctionList** object will initially contain no entries. Calling the **get_count** function will return **0**.

The copy constructor uses the values contained in the **oldobj** to initialize the new (constructed) object.

See Also

FunctionId, **SourceObj::bget_function_list**, **SourceObj::get_function_list**

get_count

IBM Confidential, Limited Rights Data

get_count

Synopsis

```
#include <FunctionList.h>
int get_count(void) const;
```

Description

Returns the number of entries (**FunctionId** objects) in this function list.

Return Values

Returns an integer indicating the number of entries (**FunctionId** objects) in this function list. If this **FunctionId** object was initialized by a default constructor, this function returns **0**.

See Also

get_entry

get_entry

Synopsis

```
#include <FunctionList.h>
FunctionId get_entry(int index) const;
```

Parameters

index

the position or index of a particular entry (**FunctionId** object) in this function list.

Description

Returns the *i*th **FunctionId** object in this function list.

Return Values

Returns the *i*th **FunctionId** object if this index is valid. In other words, $0 \leq i < \text{get_count}()$;

See Also

get_count

operator=

IBM Confidential, Limited Rights Data

operator=

Synopsis

```
#include <FunctionList.h>
FunctionList & operator=(const FunctionList &rhs);
```

Parameters

rhs

The right hand operand. This is the existing **FunctionList** object whose value you want to assign to the invoking **FunctionList** object.

Description

Assigns the value of the right hand operand (existing **FunctionList** object) to the invoking **FunctionList** object. The left operand is the invoking object.

For example,

```
FunctionList rhs, lhs;
.
.
.
lhs = rhs;
```

assigns the value of **rhs** to **lhs**. This yields a new copy of the **FunctionList** object.

Return Values

Returns a reference to the invoking **FunctionList** object (the left operand).

Appendix B. PE Benchmark Command Reference

This appendix contains the manual pages for the PE Benchmark commands discussed throughout this book. Each manual page is organized into the sections listed below. The sections always appear in the same order, but some appear in all manual pages while others are optional.

NAME Provides the name of the command described in the manual page, and a brief description of its purpose.

SYNOPSIS

Includes a diagram that summarizes the command syntax, and provides a brief synopsis of its use and function.

FLAGS

Lists and describes any required and optional flags for the command.

DESCRIPTION

Describes the command more fully than the **NAME** and **SYNOPSIS** sections.

ENVIRONMENT VARIABLES

Lists and describes any applicable environment variables.

EXAMPLES

Provides examples of ways in which the command is typically used.

FILES Lists and describes any files related to the command.

RELATED INFORMATION

Lists commands, functions, file formats, and special files that are employed by the command, that have a purpose related to the command, or that are otherwise of interest within the context of the command.

convert

NAME

convert – Converts AIX event trace files into UTE internal trace files.

SYNOPSIS

```
convert [-# | -?][ -n number_of_files] [-p profile_file_name]  
[-c number_of_records_per_frame] [-f number_of_frames_per_frame_directory]  
[-o {output_file_name | output_file_name_prefix}] {input_file_name |  
input_file_name_prefix}
```

The **convert** command converts one or more AIX event trace files into one or more UTE interval trace files. The *input_file_name* (for converting a single AIX event trace file) or *input_file_name_prefix* (for converting multiple AIX event trace files) must be the last item on the command line.

FLAGS

-# or -?

Prints out usage information for the **convert** command instead of converting and AIX trace files.

-n *number_of_files*

Specifies the number of AIX event trace files to be converted. If not specified, the default is 1.

-p *profile_file_name*

Specifies the name of the description profile. If not specified, the default profile is the one specified by the environment variable **UTE_PROFILE**, or, if the environment variable **UTE_PROFILE** is not set, the file *profile.ute* in the current directory.

-c *number_of_records_per_frame*

Specifies the number of interval records per frame.

-f *number_of_frames_per_frame_directory*

Specifies the number of frames in each frame directory. When a frame directory is exhausted, this utility automatically creates an additional frame directory and links it with existing frame directories.

-o {*output_file_name* | *output_file_name_prefix*}

If the **-n** option specifies the number of files as 1 (the default), the **-o** option specifies the name of the resulting UTE interval file.

If the **-n** option specifies the number of files as greater than 1, the **-o** option specifies the file name prefix for the resulting UTE interval files. The names of the output files are formed by concatenating the given prefix with a node identifier, starting from 0.

DESCRIPTION

The **convert** command converts one or more AIX event trace files into one or more UTE interval trace files. If the **-n** option specifies the number of files to be converted as 1 (the default), then you supply a single *input_file_name* to the **convert** command. If instead, the **-n** option specifies the number of files to be converted as greater than 1, then an *input_file_name_prefix* is supplied. The *input_file_name* or *input_file_name_prefix* must be the last item on the command line.

ENVIRONMENT VARIABLES

UTE_PROFILE

Specifies the name of the file description profile. If not set, the file *profile.ute* in the current directory is the default description profile. If the **-p** option is used, its specification overrides the **UTE_PROFILE** environment variable's setting.

EXAMPLES

To convert the AIX trace file *mytrace* into a UTE interval trace file:

```
convert mytrace
```

To convert five trace files with the prefix *mytraces* into UTE interval trace files:

```
convert -n 5 mytraces
```

FILES

profile.ute default description profile.

RELATED INFORMATION

Commands: **slogmerge(1)**, **utemerge(1)**, **utestats(1)**

pct

NAME

pct – Invokes the Performance Collection Tool in either its graphical-user-interface or command-line mode.

SYNOPSIS

```
pct [-cmd [-s script_file]] [-d diagnostic_log_setting]
```

The **pct** command starts the Performance Collection Tool in either its graphical-user-interface mode, or, if the **-cmd** flag is specified, its command-line mode.

FLAGS

-cmd

Specifies that the Performance Collection Tool should be started in command-line mode. Refer to “Subcommands of the pct Command” on page 120 for information on the subcommands you can issue once the Performance Collection Tool is running in this mode.

-s *script_file*

When running in command-line mode, instructs the Performance Collection Tool to read its commands from the script file specified. When running in graphical user interface mode, you cannot use this option.

-d *diagnostic_log_setting*

Turns diagnostic logging on for this run of the Performance Collection Tool. When diagnostic logging is turned on, a separate log file will be generated in the directory */tmp* on each host machine running target application processes to which the Performance Collection Tool connects. The log file is generated by a daemon process (called the “DPCL communication daemon”) that handles communication between the Performance Collection Tool and the target application process. The log file saved to the */tmp* directory will be named **dpclid.nnnn** where *nnnn* is the AIX process ID of the DPCL communication daemon process. For more information on the DPCL communication daemon, refer to the manual *IBM Parallel Environment for AIX: Dynamic Probe Class Library Programming Guide*. The *diag_log_setting* specified with this flag can be one of the following:

severe

The DPCL communication daemon will generate messages for fatal and severe error conditions only.

warning

In addition to fatal and severe error conditions, the DPCL communication daemon will generate warning messages.

trace

In addition to fatal, severe, and warning messages, the DPCL communication daemon will also generate function entry/exit trace information.

detail

The most detailed level of diagnostic messages will be generated by the DPCL communication daemon. In addition to the severe, error, warning, and function entry/exit trace information, the DPCL communication daemon will generate other, more general, information.

DESCRIPTION

The Performance Collection Tool is a highly scalable performance monitoring tool built on IBM's dynamic instrumentation technology — the Dynamic Probe Class Library (DPCL). Using the Performance Collection Tool, you can collect:

- MPI and user event traces for eventual analysis by either:
 - Jumpshot (a public-domain tool developed at Argonne National Lab).
 - or
 - the **utestats** utility provided as part of the PE Benchmark Toolset.

Since the MPI and user trace information will be output as standard AIX trace files, we have also supplied, as part of the PE Benchmark tool set, several utilities for converting the AIX trace files created by the Performance Collection Tool into a format readable by Jumpshot and the **utestats** utility.

- Hardware and operating system profiles for playback within the Performance Visualization Tool (as invoked by the **pvt** command).

The Performance Collection Tool can be run in either its graphical-user-interface mode, or, if the **-cmd** flag is specified, its command-line mode. The Performance Collection Tool's graphical user interface is built on top of its command-line interface; in other words, your manipulations of the graphical user interface are translated by the tool into **pct** subcommands. These subcommands are issued, and the information returned is used to update the graphical user interface. You can optionally have the **pct** subcommands that result from your interface interactions:

- displayed in an information area of the Performance Collection Tool's Main Window.
- saved to a file.

Both of these options are provided to help you learn the Performance Collection Tool's command-line interface more quickly. In addition, if you save the subcommands to a file, that file can later be read (as is or modified) as a script file in the Performance Collection Tool's command-line interface (using the **-s** option when issuing the **pct** Command).

The **pct** command's subcommands (for controlling the Performance Collection Tool in command-line mode) are listed alphabetically under "Subcommands of the pct Command" on page 120.

EXAMPLES

To start the Performance Collection Tool in graphical-user-interface mode:

```
pct
```

To start the Performance Collection Tool in command-line mode:

```
pct -cmd
```

To start the Performance Collection Tool in command-line mode, and read commands from the script file *myscript*.

```
pct -cmd -s myscript.cmd
```

RELATED INFORMATION

Commands: **convert(1)**, **pvt(1)**, **slogmerge(1)**, **utemerge(1)**, **utestats(1)**

Subcommands of the `pct` Command

connect Subcommand (of the `pct` Command)

```
connect [{pid process_id | poe pid poe_process_id | task task_list |
group task_group_name]
```

The **connect** subcommand connects the Performance Collection Tool to an existing application. Using this subcommand, you can connect to a single application process, or the controlling, "home node" process in a POE application. Once you are connected to a controlling POE home node process, you can reissue this subcommand to connect to one or more of the POE application's tasks.

all of the application processes in a POE application, or select application processes (or tasks) of a POE application.

pid *process_id*

Specifies the process ID of a single application process to connect.

poe pid *poe_process_id*

Indicates that you are connecting a POE process, and specifies the process ID of the POE home node process (the executing instance of the **poe** command). Only the controlling POE process is connected. To connect to one or more of the POE application's tasks, reissue the **connect** subcommand.

task *task_list*

Specifies a list of POE tasks to connect. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refer to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. To connect to all tasks in a POE application, you can specify the task group *all*, which will have been created by the Performance Collection Tool when you connected to the controlling, home node, POE process. Refer to the **group** subcommand for information on creating task groups.

For example, to connect to the application process whose AIX process ID is 12345:

```
connect pid 12345
```

To connect to the POE "home node" process whose AIX process ID is 12345:

```
connect poe pid 12345
```

The preceding example connects to just the controlling, home node, process in a POE application. To now connect to all of the tasks in the POE application:

```
connect group all
```

destroy Subcommand (of the `pct` Command)

```
destroy [task task_list | group task_group_name]
```

The **destroy** subcommand terminates execution of one or more connected processes. By default, the tasks in the current task group (as previously defined by the **group** subcommand) are the ones terminated. You can override this default, however, by specifying a *task_list* or *task_group_name* when you issue the **destroy** subcommand.

When working with a POE application, be aware that terminating any process of the application will cause POE to terminate all of the application's processes. This termination of all processes is a function of POE, not of the Performance Collection Tool. For more information, refer to *IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment*.

task *task_list*

Specifies the connected tasks to be terminated. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refer to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

For example, to terminate execution of the tasks in the current task group:

```
destroy
```

To terminate task 8:

```
destroy task 8
```

To terminate the tasks in task group *connected*:

```
destroy group connected
```

disconnect Subcommand (of the pct Command)

disconnect [**task** *task_list* | **group** *task_group_name*]

The **disconnect** subcommand disconnects the Performance Collection Tool from one or more connected processes. Disconnecting from a process removes any performance collection probes from the process. Once a process is disconnected, the Performance Collection Tool will no longer be able to control execution of, or instrument, the process. By default, the tasks in the current task group (as previously defined by the **group** subcommand) are the ones that are disconnected. You can override this default, however, by specifying a task list or task group name when you issue the **disconnect** subcommand.

task *task_list*

Specifies the connected POE tasks to be disconnected. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

For example, to disconnect from the tasks in the current task group:

```
disconnect
```

To disconnect from task 8:

```
disconnect task 8
```

To disconnect from the tasks in task group *connected*:

```
disconnect group connected
```

exit Subcommand (of the pct Command)

exit [**destroy**]

The **exit** subcommand exits the Performance Collection Tool and, optionally, terminates execution of all connected processes. Since terminating any process of the POE application will cause POE to terminate all of the POE application's processes, the **destroy** clause effectively terminates the entire POE application.

For example, to exit the Performance Collection Tool, but allow all of its connected processes to continue running:

```
exit
```

To exit the Performance Collection Tool and terminate the target application:

```
exit destroy
```

file Subcommand (of the pct Command)

file [**task** *task_list* | **group** *task_group_name*] "*regular_expression*"

The **file** subcommand lists, for one or more tasks, any associated source file names that match a regular expression that you supply. By default, this subcommand applies to the current task group (as previously defined by the **group** subcommand). You can override this default, however, by specifying a task list or task group name when you issue the **file** subcommand.

The files are listed by this subcommand as a table with column headings for the task identifier, file identifier, file name, and, if available, the path.

The file identifiers are determined by sorting the files alphabetically and numbering them starting from 0. The path will be shown only if the file path information was supplied when you compiled a file.

task *task_list*

Specifies the connected POE tasks whose source file names you want to list. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

"regular_expression"

An AIX regular expression (file-name substitution pattern) enclosed in quotation marks that identifies the files to list. The **file** subcommand will filter the list of file names using this regular expression; only file names that match this regular expression pattern will be listed.

For example, to list all the files in the current task group:

```
> file "*"
Tid    File Id   File Name   Path
0      0         bar.c      ../../lib/src
0      1         foo1.c     ../../lib/src
0      2         foo2.c     ../src
```

To list only the files in task 0 that begin with the letter "f":

```
> file task 0 "f*"
Tid    File Id   File Name   Path
0      1         foo1.c     ../../lib/src
0      2         foo2.c     ../src
```

find Subcommand (of the pct Command)

file [**task** *task_list* | **group** *task_group_name*]

function *"regular_expression_to_match_function_name"*

The **find** subcommand lists all function names that match a regular expression pattern that you supply. This subcommand is intended for situations when you wish to instrument a particular function, but do not know which file the function is located in.

The function names found are listed by this subcommand as a table with column headings for task identifier, file identifier, file name, and function name.

task *task_list*

Specifies the connected POE tasks whose source files you want to search. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

function *"regular_expression_to_match_function_name"*

An AIX regular expression (file-name substitution pattern) enclosed in quotation marks that identifies the functions to locate. Matching is performed using rules of AIX file-name pattern matching. The **find** subcommand will filter the list of function names using this regular expression; only function names that match this regular expression pattern will be listed.

For example, to list all the functions in task 0 that match the regular expression *comp**:

```
> find task 0 function "comp*"
Tid   File Id   File Name   Function Name
0     23       main.c     compute
0     23       main.c     compare
0     25       sort.c     compare2
```

function Subcommand (of the pct Command)

```
function [task task_list | group task_group_name]
{file "regular_expression"[, "regular_expression"] | fileid file_identifier[,file_identifier]...
"regular_expression_to_match_function_name"
```

The **function** subcommand lists, for one or more tasks, the names of the functions (that match a regular expression pattern you supply) contained in a source file. The file whose functions are listed can be specified as a file identifier or as a regular expression that matches the file name; this information can be ascertained by the **file** subcommand, or, if you are unsure which file the function is located in, the **find** subcommand. By default, this subcommand applies to the current task group (as previously defined by the **group** subcommand). You can override this default, however, by specifying a task list or task group name when you issue the **function** subcommand.

The function names are listed by this subcommand as a table with column headings for task identifier, file identifier, function identifier, file name, and function name.

The function identifiers are determined by sorting the functions contained in a file alphabetically starting from 0. Each file's functions are numbered sequentially starting from 0.

task *task_list*

Specifies the connected POE tasks containing the source files whose functions you want to list. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

file "*regular_expression*"[, "*regular_expression*]...

Specifies, using one or more regular expression patterns, the file(s) whose functions you want to list. The regular expression patterns must be contained in quotation marks.

fileid *file_identifier*[,*file_identifier*]...

Specifies, using one or more file identifiers as returned by the **file** subcommand, the file(s) whose functions you want to list.

"regular_expression_to_match_function_name"

A regular expression enclosed in quotation marks that identifies the function names to list. Matching is performed using rules of AIX file-name pattern matching. The **function** subcommand will filter the list of function names using this expression; only function names (for the tasks/file indicated) that match the regular expression will be listed.

For example, to list all the functions in the file "bar.c" in task 0:

```
> function task 0 file "bar.c" "*"
Tid    File Id    Function Id    File Name    Function Name
0      1          1              bar.c        func0
0      1          1              bar.c        func1
```

To list all the functions in the file "bar.c" (using the file identifier) in task 0:

```
> function task 0 fileid 1 "*"
Tid    File Id    Function Id    File Name    Function Name
0      1          1              bar.c        func0
0      1          1              bar.c        func1
```

To list, for task 0, all of the functions in files beginning with "b" or "d":

```
> function task 0 file "b*", "d*" "*"
Tid    File Id    Function Id    File Name    Function Name
0      3          0              bar.c        func0
0      3          1              bar.c        func1
0      3          2              bar2.c       func_xyz
0      4          0              bar2.c       calc
0      4          1              bar2.c       do_math
0      4          2              bar2.c       sum
```

group Subcommand (of the pct Command)

group default *task_group_name*

group add *task_group_name task_list*

group delete *task_group_name [task_list]*

The **group** subcommand can perform three distinct actions related to task groups:

- Using the **default** action of the **group** command:

group default *task_group_name*

you can set the command context on a particular task group. When you do this, the task group you specify becomes the current task group; certain other subcommands that you issue (such as the **file**, **function**, and **point** subcommands) will, by default, apply only to the tasks in the current task group.

- Using the **add** action of the **group** subcommand:

group add *task_group_name task_list*

you can create a new task group, or add tasks to an existing task group.

- Using the **delete** action of the **group** subcommand:

group delete *task_group_name [task_list]*

you can delete, or delete selected tasks from, a task group. If a task list is specified, these tasks are removed from the task group; otherwise, the entire task group is deleted.

In addition to any task groups you create using the **group** subcommand, note that there are two task groups that are created automatically by the Performance Collection Tool when you issue either the **load** or **connect** subcommands. These automatically-created task groups are named *all* and *connected*. The *all* task group

contains all tasks in the current application, while the *connected* task group contains the set of tasks to which the Performance Collection Tool is connected.

task_group_name

refers to the name of the task group that, depending on the particular **group** subcommand action you are executing, you want to:

- make the default task group
- create or add tasks to
- delete or remove tasks from

task_list

Refers to the list of tasks that, depending on the particular **group** subcommand action you are executing, you want to either add to, or delete from, the task group. The tasks can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

For example, to create a task group *master* consisting of task 0, and a task group *workers* consisting of tasks 1 through 20.

```
group add master 0
group add workers 1:20
```

To add tasks 21 through 30 to the task group *workers*:

```
group add workers 21:30
```

To make the group *workers* the default task group:

```
group default workers
```

To remove tasks 21 through 30 from the task group *workers*.

```
group delete workers 21:30
```

To delete the task group *workers*:

```
group delete workers
```

list Subcommand (of the pct Command)

```
list [task task_list | group task_group_name]
[file "regular_expression" [, "regular_expression" ]... |
fileid file_identifer [, file_identifer ]... ] [line line_number_range]
```

The **list** subcommand returns the contents of a file. The first time you issue this subcommand, you should specify a file using the **file** or **fileid** clause. Doing this will list the entire file's contents. To list only a portion of the files contents, specify a line number range using the **line** clause. To minimize typing, the Performance Collection Tool records the number of the last source code line displayed; issuing the **list next** subcommand will display the next few lines of the source code. By default, this form of the subcommand applies to the current task group (as previously defined by the **group** subcommand). You can override this default, however, by specifying a task list or task group name when you issue the **list** subcommand.

task *task_list*

Specifies the connected POE tasks containing the source files whose contents you want to list. The tasks in the POE application can be specified

by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

file "*regular expression*" [*,"regular expression"*]...

Specifies, using one or more regular expressions, the file(s) whose contents you want to list.

fileid *file_identifier*[*,file_identifier*]...

Specifies, using one or more file identifiers as returned by the **file** subcommand, the file(s) whose contents you want to list.

line *line_number_range*

The line number range of the source code you want to list. Use a colon to separate the ends of the range (for example 1:20).

For example, to list lines 1 through 20 of the source file "bar.c":

```
list bar.c 1:20
```

load Subcommand (of the pct Command)

```
load {{exec [poe] absolute_path_to_executable } | {poe
[mpmdcmd path_to_poe_commands_file] [poeargs "poe_arguments_string"]}
[args "program_arguments_string"]}
[poeargs "poe_arguments_string"] [stdout standard_out_file_name]
[stderr standard_error_file_name] [stdin standard_input_file_name]
```

The **load** subcommand loads a serial or POE application for execution. Once an application is loaded, you can instrument it with probes, or control its execution using the **start**, **suspend**, **resume**, and **terminate** subcommands. The **load** subcommand is intended for applications that are not already executing; to connect to applications that are already executing, use the **connect** subcommand. The **poe** clause indicates that the application is a POE application; if not specified, the **load** subcommand assumes you are loading a serial application. The **load** subcommand loads the application into memory in a "stopped state" with execution suspended at its first executable instruction. You can start execution of the application using the **start** subcommand.

exec *absolute_path_to_executable*

Specifies the full path to the executable file. If you are loading a POE application, you must also include the keyword **poe** (as described below) on the command line. You specify the *path_to_executable* as an absolute path.

poe specifies that you are loading a POE program.

mpmdcmd *path_to_poe_commands_file*

Specifies that the POE program you're loading follows the Multiple Program Multiple Data (MPMD) model and indicates the path to the POE commands file listing the executable programs to run. For more information on POE commands files, refer to manual *IBM Parallel Environment for AIX: Operation and Use, Volume 1*.

poeargs *"poe_arguments_string"*

Specifies command-line arguments that are passed to the **poe** command to control various aspects of the Parallel Operating Environment. For a complete listing of the POE arguments you can supply, refer to the manual *IBM Parallel Environment for AIX: Operation and Use, Volume 1*. The POE arguments should be provided as a string delimited by double quotation marks. Embedded quotation marks can be included in the string if each mark is preceded by an escape character (\). Embedded escape characters may also be included if they are preceded by an additional escape character.

args *"program_arguments_string"*

Specifies command-line arguments that are passed to the application. Note that these are not POE arguments, which are instead specified by using the **poeargs** clause. The program arguments should be provided as a string delimited by double quotation marks. Embedded quotation marks can be included in the string if each mark is preceded by an escape character (\). Embedded escape characters may also be included if they are preceded by an additional escape character.

stdout *standard_out_file_name*

Redirects standard output to the file specified.

stderr *standard_error_file_name*

Redirects standard error to the file specified.

stdin *standard_input_file_name*

Reads standard input from a file.

For example, the following command loads the serial executable *foo* and passes it the argument string *"a b c"*:

```
load exec /u/example/bin/foo args "a b c"
```

The following command loads the POE executable *parallel_foo* and passes it POE arguments:

```
load poe exec /u/example/bin/parallel_foo poeargs "-procs 4 -hfile /tmp/host.list"
```

The following command loads an MPMD POE program. The executable files to load are listed in the POE commands file */u/example/bin/foo.cmds*:

```
load poe mpmdcmd /u/example/bin/foo.cmds poeargs "-procs 3 -hfile /tmp/host.list"
```

point Subcommand (of the pct Command)

```
point [task task_list | group task_group_name]
{file regular_expression["regular_expression"]... |
fileid file_identifier["file_identifier"]...}
[function regular_expression["regular_expression"]... |
funcid function_identifier["function_identifier"]...]
```

Lists the instrumentation points (at the task, file, or function level) where custom user markers can be added by the **trace add** subcommand. You only need to identify instrumentation points when installing custom user markers using the **trace add** subcommand. You do not need the instrumentation point information if installing MPI trace probes using the **trace add** subcommand or profile probes using the **profile add** subcommand. By default, this subcommand will list the instrumentation points for the tasks in the current task group (as previously defined by the **group** subcommand). You can override this default, however, by specifying a

task list or task group name when you issue the **point** subcommand. Using the **function** clause, you can specify one or more functions whose instrumentation points you want listed. Using the **file** or **fileid** clause, you can specify a particular file whose instrumentation points you want listed.

The instrumentation points are listed by this subcommand as a table with headings for task identifier, file identifier, function identifier, point identifier, point type, and callee name.

The point identifiers are determined by numbering the points, starting from 0, according to their location in each function. The first instrumentation point in the function is given the identifier 0, the second is given the identifier 1, and so on. Each function's instrumentation points are numbered separately starting from 0.

task *task_list*

Specifies the connected POE tasks whose instrumentation points you want to list. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

file "*regular_expression*"[, "*regular_expression*"]...

Specifies, using one or more regular expressions (file name substitution patterns), the file(s) whose instrumentation points you want to list. The regular expression(s) must be contained in quotation marks.

fileid *file_identifier*[, *file_identifier*]

specifies, using one or more file identifiers as returned by the **file** subcommand, the file(s) whose instrumentation points you want to list.

function "*regular_expression*"[, "*regular_expression*"]...

Specifies, using one or more regular expressions, the function(s) whose instrumentation points you want to list. This regular expression must be contained in quotation marks.

funcid *function_identifier*[, *function_identifier*]

Specifies, using one or more function identifiers as returned by the **function** subcommand, the function(s) whose instrumentation points you want to list.

For example, to list all the instrumentation points in task 0 for the file *bar.c*:

```
> point task 0 file bar.c
Tid File Id Function Id Point Id Point Type Callee Name
-----
0 54 0 0 0
0 54 0 1 2 printf
0 54 0 2 3 printf
0 54 0 3 2 MPI_Abort
0 54 0 4 3 MPI_Abort
0 54 0 5 1
0 54 1 0 0
0 54 1 1 2 printf
0 54 1 2 3 printf
0 54 1 3 2 printf
0 54 1 4 3 printf
0 54 1 5 2 MPI_Recv
```

0	54	1	6	3	MPI_Recv
0	54	1	7	2	consume_data
0	54	1	8	3	consume_data
0	54	1	9	2	printf
0	54	1	10	3	printf
0	54	1	11	1	

profile add Subcommand (of the pct Command)

```

profile add [task task_list | group task_group_name]
{{profname profile_type_name | profid profile_type_identifier}
groupid group_identifier | groupname group_name}}...
[to {file "regular_expression" [, "regular_expression"]... |
fileid file_identifier [, file_identifier]...}
[function "regular_expression" [, "regular_expression"]... |
funcid function_identifier [, function_identifier...]]]

```

The **profile add** subcommand adds one or more probes to collect hardware and operating system profile information. You cannot use this subcommand, or any of the **profile** subcommands, unless you have specified that you are collecting profile data. To specify that you are collecting profile data, issue the **select** subcommand with its **profile** clause:

```
select profile
```

If you add multiple profile probes (by specifying multiple probe types in the *profile_types_list*), be aware that they are considered a single set of probes. When removing profile probes using the **profile remove** subcommand, you will not be able to remove individual probes. Instead, you'll have to remove the entire set of probes.

By default, this subcommand will add the probe(s) to the tasks in the current task group (as previously defined by the **group** subcommand). You can override this default, however, by specifying a task list or task group name when you issue the **profile add** subcommand. Be aware, however, that the set of tasks cannot include different executables in an MPMD application. For example, if an MPMD application consists of executables *a.out* and *b.out*, then this command cannot be applied to a task group that contains both *a.out* and *b.out* tasks.

task *task_list*

Specifies the connected POE tasks to which you want to add the profile probes. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

profname *profile_type_name*

Specifies, using a probe type name, a profile probe type to add. To list the profile probe type names, use the **profile show** subcommand (with its **probetypes** clause specified):

```
profile show probetypes
```

profid *profile_type_identifier*

Specifies, using a probe type identifier, a profile probe type to add. To list the profile probe type identifiers, use the **profile show** subcommand (with its **probetypes** clause specified):

```
profile show probetypes
```

groupid *group_identifier*

If you are collecting hardware counter information, a profile group identifier indicating the specific hardware counter information you want to collect. To get a list of the profile groups available for your hardware, use the command:

```
profile show probetype hw
```

groupname *group_name*

If you are collecting hardware counter information, a profile group name indicating the specific hardware counter information you want to collect. To get a list of the profile groups available for your hardware, use the command:

```
profile show probetype hw
```

file "*regular_expression*"[, "*regular_expression*"]...

Specifies, using one or more regular expressions (file name substitution patterns), the file(s) you wish to instrument with profile probes. The regular expressions must be enclosed in quotation marks.

fileid *file_identifier*[, *file_identifier*]...

Specifies, using one or more file identifiers as returned by the **file** subcommand, the file(s) you wish to instrument with profile probes.

function "*regular_expression*"[, "*regular_expression*"]...

Specifies, using one or more regular expressions, the functions you wish to instrument with the profile probes. The regular expression must be enclosed in quotation marks.

funcid *function_identifier*[, *function_identifier*]...

Specifies, using one or more function identifiers as returned by the **function** subcommand, the functions you wish to instrument with the profile probes.

For example, to add a profile probe to collect wallclock data for the current task group:

```
profile add profname wc to fileid 5 funcid 3
```

To add a profile probe to collect wallclock data, and hardware data using counter group 2:

```
profile add profname wc profname hw groupid 2 to fileid 3
```

profile remove Subcommand (of the pct Command)

```
profile remove [task task_list | group task_group_name] probe probe_index
```

The **profile remove** subcommand removes the profile probe set specified by the supplied *probe_index*. A profile probe set consists of one or more probes as previously installed by the **profile add** subcommand. An installed profile probe's *probe_index* can be ascertained by the **profile show** subcommand (with its **probes** clause) as in:

```
profile show probes
```

By default, the **profile remove** subcommand will remove the profile probe set from all tasks in the current task group (as previously defined by the **group** subcommand). You can override this default, however, by specifying a task list or task group name when you issue the **profile remove** subcommand.

task *task_list*

Specifies the connected POE tasks to which you want to remove the profile probe set. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

probe *probe_index*

Specifies, using a probe index, the profile probe set to be removed. The probe index can be ascertained by issuing the **profile show** subcommand (with its **probes** clause) as in:

```
profile show probes
```

For example, to remove the profile probe set whose index is 3 from all connected tasks (as represented by the task group *connected*):

```
profile remove group connected 3
```

profile set path Subcommand (of the pct Command)

profile set path *path_name/output_file_base_name*

The **profile set path** subcommand specifies the output location and base name for the profile data files generated by profile probes that you install using the **profile add** subcommand.

path_name/output_file_base_name

specifies a relative or full path to the desired location for the profile output files, followed by the output file base name. The base name is needed because the data collected by the Performance Collection Tool will be saved as a file on each host machine where a connected process with probes is running. The file name will consist of the base name you supply followed by a node-specific suffix supplied by the Performance Collection Tool. If a relative path is specified, note that the location will be relative to the application's current working directory.

For example, to specify the relative path *profile/output* as the location for profile output files:

```
profile set path profile/output
```

profile show Subcommand (of the pct Command)

profile show {**probes** | **probetypes** | **probetype** *probe_type_name* | **path**}

The **profile show** subcommand lists, depending on the clause you specify, either the currently installed profile probes, the list of profile probe types that you can install, or the profile file output location.

probes

Specifies that the **profile show** subcommand should list the currently installed profile probes (including the probe index). The probe index information is needed when removing a profile probe using the **profile remove** subcommand.

probetypes

Specifies that the **profile show** subcommand should list the available probe types you can add using the **profile add** subcommand. This information is returned in the form:

```
probe_type_index:probe_name:probe_description
```

probes *probe_type_name*

Specifies that the subcommand should list the options for the specified probe type. Currently, only hardware counter probes have options.

path

Specifies that you want the **trace show** subcommand to return the profile file output location and base name as set by the command:

```
profile set path
```

For example, to list the installed profile probes:

```
profile show probes
```

To list available profile probe types:

```
> profile show probetypes
Prof Id Prof Name Description
-----
0      wclock    wall clock
1      rusage    resource usage
2      hwcount   hardware counter
```

resume Subcommand (of the pct Command)

resume [**task** *task_list* | **group** *task_group_name*]

The **resume** subcommand resumes execution of one or more processes that have previously been suspended by the **suspend** subcommand. By default, the tasks in the current task group (as previously defined by the **group** subcommand) are the ones that have their execution resumed. You can override this default, however, by specifying a task list or task group name when you issue the **resume** subcommand.

task *task_list*

Specifies the connected POE tasks that you want to resume executing. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

For example, to resume execution of all tasks in the current task group:

```
resume
```

To resume execution of tasks 0 through 20:

```
resume task 0:20
```

To resume execution of the tasks in task group *mygroup*:

```
resume group mygroup
```

select Subcommand (of the pct Command)

```
select {trace | profile}
```

The **select** subcommand enables you to select the type of probe data you will be collecting.

trace Specifies that you intend to collect MPI or custom user event traces for eventual playback within Jumpshot.

profile

Specifies that you intend to collect hardware and operating system profiles for playback within the Performance Visualization Tool.

For example, if you will be adding trace probes (using the **trace add** subcommand) for collecting MPI or custom user event data:

```
select trace
```

If, on the other hand, you will be adding profile probes (using the **profile add** subcommand) for collecting hardware and operating system profiles:

```
select profile
```

set Subcommand

```
set sourcepath [relative] path_list
```

The **set** subcommand enables you to set the path used when displaying the contents of a file using the **list** subcommand. The initial value for the source path is the directory in which the tool was started.

relative

Specifies that, if relative path information is included as part of the file name supplied to the **list** subcommand, the relative path should be used together with the directories listed in the *pathlist*.

For example, say one of the source files in the application is named *"../myapp/src/compute.c"* and the source path is *"/tmp:/usr/tmp:/home/mydir/examples/yourapp"*. If the **relative** keyword is used when setting the source path, the Performance Collection Tool searches the following directories when the **list ../myapp/src/compute.c** subcommand is issued.

```
/tmp/../../myapp/src/
/usr/tmp/../../myapp/src/
/home/mydir/examples/yourapp/../../myapp/src/
```

If the **relative** keyword is not used when setting the source path, however, the following directories are searched:

```
/tmp/
/usr/tmp/
/home/mydir/examples/yourapp/
```

path_list

A colon-delimited list that specifies the path the **list** subcommand will use to search for source files.

show Subcommand (of the pct Command)

show events

show group [*task_group_name*]

show groups

show next

show points

show ps

show sourcepath

show tools

The **show** subcommand returns, depending on the form of the subcommand you use, various information about the target application and the Performance Collection Tool.

- Using the form **show events** returns a list of the possible events that, if you place the Performance Collection Tool in an event loop using the **wait** subcommand, can break the Performance Collection Tool out of the loop.
- Using the form:

show group [*task_group_name*]

returns, for each task in the current or specified task group, the task identifier, the task name, and the name of the host machine on which the task is running. If you do not specify a task group name when issuing this subcommand, the tasks in the current task group (as previously defined by the **group** subcommand) are listed.

- Using the form:

show groups

returns a list of task groups. This includes any task groups created by default (the task groups all and connected), and any task groups you created using the **group** subcommand. An ampersand character (@) is displayed to the right of the default task group.

- Using the form:

show points

returns a list of the available instrumentation point types. This enables you to associate the instrumentation point information returned by the **point** subcommand with a numeric point identifier needed when installing custom markers using the **trace add** subcommand.

- Using the form:

show ps

returns a list of the processes you own on the node where you started the Performance Collection Tool. This information is needed when connecting to an application using the **connect** subcommand.

- Using the form:

show sourcepath

returns a list of directories searched when displaying the contents of a file using the **list** subcommand. You can set the source path using the **set** subcommand.

If only the base source path is used to search for files, the word "BASE" will appear in parentheses at the top of the list of directories. For example:

```
PATH (BASE)
/tmp/
/usr/tmp/
/home/mydir/examples
```

If the **relative** keyword was used on the **set** subcommand, then, if relative path information is supplied in the name of the file requested for viewing with the **list** subcommand, the relative path is appended to the directories specified by the **set** subcommand. If the **relative** keyword was specified, the word "RELATIVE" will appear in parentheses at the top of the list of directories.

- Using the form:

show tools

returns a list of the types of information you can collect using the Performance Collection Tool (for this release, "trace" and "profile") This information is needed when selecting the type of data you will be collecting using the **select** subcommand.

group [*task_group_name*]

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

events

Specifies that you want the subcommand to return a list of event names that can break the Performance Collection Tool out of an event loop established by the **wait** subcommand.

groups

Specifies that you want the subcommand to return a list of task groups.

next

Specifies that you want the subcommand to return the next few lines of the source code.

points

Specifies that you want the subcommand to return a list of all tasks in all task groups.

ps

Specifies that you want the subcommand to return a list of the processes you own on this host machine.

tools

Specifies that you want the subcommand to return a list of the type of information you can collect using the Performance Collection Tool.

For example, to show the tasks in the current task group:

```
show group
```

To show the tasks in the task group "connected":

```
show group connected
```

To show the processes that you own on the host machine:

```
show ps
```

start Subcommand (of the pct Command)

start

The **start** subcommand starts execution of an application you have loaded using the **load** subcommand. (The **load** subcommand loads an application into memory in a "stopped state" with execution suspended at the first executable instruction.)

For example, to start execution of the currently-loaded application:

```
start
```

stdin Subcommand (of the pct Command)

stdin [{"string" | eof}]

The **stdin** subcommand sends the supplied string as standard input to the currently loaded application. If no string is supplied, the **stdin** subcommand will send a newline character to the application. If the **eof** option is supplied, the **stdin** subcommand will send an end-of-file character to the application.

You can also, when loading an application using the **load** subcommand, indicate that the application should read standard input from a file specified by the **stdin** option. If the **stdin** option is used when loading an application with the **load** subcommand, note that the **stdin** subcommand cannot be used.

"string"

Specifies a text string to send to standard input. The string should be enclosed in quotes, and embedded formatting characters (such as \n) are permitted. If no string is supplied, the **stdin** subcommand will send a newline character to the application.

eof sends an end-of-file character to the input stream reading this input data.

For example:

```
stdin "now is the time \nfor all good men"
```

suspend Subcommand (of the pct Command)

suspend [task *task_list* | group *task_group_name*]

The **suspend** subcommand suspends execution of one or more processes. By default, the tasks in the current task group (as previously defined by the **group** subcommand) are the ones that are suspended. You can override the default, however, by specifying a task list or task group name when you issue the **suspend** command. You can resume execution of tasks suspended by this subcommand by issuing the **resume** subcommand.

task *task_list*

Specifies the connected POE tasks that you want to suspend. The tasks in the POE application can be specified by listing individual values separated

by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

For example, to suspend execution of all tasks in the current task group:

```
suspend
```

To suspend execution of tasks 0 through 20:

```
suspend task 0:20
```

To suspend execution of the tasks in task group "mygroup":

```
suspend group mygroup
```

trace add Subcommand (of the pct Command)

```
trace add [task task_list | group task_group_name]
{mpiid probetype_number_list | mpiname probe_name_list} [to
{file "regular_expression" [, "regular_expression"] | fileid file_identifier [, file_identifier]}
[function "regular_expression" [, "regular_expression"]... |
funcid function_identifier [, function_identifier]...]
```

```
trace add {simplemarker "marker_name" | {{beginmarker | endmarker} "marker_name"}}
| {traceon | traceoff}} to {file "regular_expression" [, "regular_expression"] |
fileid file_identifier [, file_identifier]}
{function "regular_expression" [, "regular_expression"]... |
funcid function_identifier [, function_identifier]...} pointid point_identifier
```

The **trace add** subcommand enables you to add the following types of probes to one or more tasks. You can add:

- MPI trace probes. If you add multiple MPI trace probes (by specifying multiple probe names in the *probe_name_list*), be aware that they are considered a single set of probes. When removing MPI trace probes using the **trace remove** subcommand, you will not be able to remove selected probes. Instead, you'll have to remove the entire set of probes.
- simple user markers to trace events of interest
- begin user markers and end user markers to trace intervals of interest
- user markers to force tracing on and off

You cannot use this subcommand, or any of the trace subcommands, unless you have specified that you are collecting trace data. To specify that you are collecting trace data, issue the **select** subcommand with its **trace** clause:

```
select trace
```

By default, this subcommand will add the probes to the tasks in the current task group (as previously defined by the **group** subcommand). You can override this default, however, by specifying a task list or task group name when you issue the **trace add** subcommand. Be aware, however, that the set of tasks cannot include different executables in an MPMD application. For example, if an MPMD application

consists of executables *a.out* and *b.out*, then this command cannot be applied to a task group that contains both *a.out* and *b.out* tasks.

task *task_list*

Specifies the connected POE tasks to which you want to add the trace probes or user markers. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

mpiid *probe_number_list*

A probe identifier (or a list of comma-separated probe identifiers) indicating the type of MPI data (collective communication, point-to-point communication, one-sided operations, and so on) that you want to collect. To get a list of the probe identifiers, issue the **trace show** subcommand with its **probetypes** clause as in:

```
trace show probetypes
```

mpiname *probe_name_list*

A probe name (or a list of comma-separated probe names) indicating the type of MPI data (collective communication, point-to-point communication, one-sided operations, and so on) that you want to collect. To get a list of the probe names, issue the **trace show** subcommand with its **probetypes** clause as in:

```
trace show probetypes
```

simplemarker "*marker_name*"

Indicates that the probe is a simple marker being placed in the target application to trace a particular event of interest. A simple marker appears in the trace record as a single point.

{beginmarker | endmarker} "*marker_name*"

Specifies that the probe is a user marker that marks either the beginning or ending of a named user state. You need to mark both the beginning and ending of the range with the same "*marker_name*" (a string that will be used to identify the user state in the trace record). You can only use a particular marker name for one begin marker/end marker pair. The state will appear in the trace record as a region.

{traceon | traceoff}

Specifies that the probe is a user marker that will either force tracing on or off. This provides a finer degree of trace control than is otherwise available when merely specifying the task, file, and function to trace.

file "*regular_expression*"[, "*regular_expression*"]...

Specifies, using one or more regular expressions (file name substitution patterns), the file(s) you wish to instrument. The regular expression must be contained in quotation marks.

fileid *file_identifier* [, *file_identifier*]...

Specifies, using one or more file identifiers as returned by the **file** subcommand, the files you wish to instrument.

function *"regular_expression"* [, *"regular_expression"*]

Specifies, using one or more regular expressions, the function(s) you want to instrument.

funcid *function_identifier* [, *function_identifier*] ...

Specifies, using one or more function identifiers as returned by the **function** subcommand, the function you want to instrument.

pointid *point_identifier*

Specifies, using a point identifier, the instrumentation point at which to add the user markers.

For example, to trace collective communication events in the file *"foo.c"*:

```
trace add mpiname coll to file "foo.c"
```

To add a begin state marker named *"green"* to the second point of the first function of file *"foo.c"*:

```
trace add beginmarker "green" to file "foo.c" funcid 0 pointid 1
```

trace remove Subcommand (of the pct Command)

trace remove {**marker** *marker_id* | **probe** *probe_index*}

The **trace remove** subcommand enables you to remove a custom user marker or a trace probe set.

marker *marker_id*

Specifies the marker identifier of the custom user marker you want to remove. To ascertain the marker identifier, use the **trace show** subcommand with its **markers** clause.

```
trace show markers
```

probe *probe_index*

Specifies, using a probe index, the trace probe set you wish to remove. A trace probe set consists of one or more probes previously installed by the **trace add** subcommand. To ascertain the trace probe set you wish to remove, use the **trace show** subcommand with its **probes** clause as in:

```
trace show probes
```

For example, to remove the trace probe whose probe identifier is *"2"*:

```
trace remove probe 2
```

trace set Subcommand (of the pct Command)

trace set { **path** *path_name/output_file_base_name* | [**bufsize** *buffer_size*] [**event** {**mpi** | **process**}] [**logsize** *maximum_log_size*]

The **trace set** subcommand enables you to specify various settings for event trace collection. You cannot use this subcommand, or any of the trace subcommands, unless you have specified that you are collecting trace data. To specify that you are collecting trace data, issue the **select** subcommand with its **trace** clause:

```
select trace
```

There are two forms of this subcommand.

- Using the form:

trace set path *path_name/output_file_base_name*

enables you to specify the output location and base name for the trace files.

- Using the form:

trace set [**bufsize** *buffer_size*] [**event** {**mpi** | **process**}] [**logsize** *maximum_log_size*]

enables you to specify the:

- size of the AIX trace buffer size
- the type of events (MPI events or process dispatch events) that are traced
- maximum size of the output trace file

The settings you make with this subcommand will stay in effect until you issue the **select** subcommand.

path *path_name/output_file_base_name*

Specifies a relative or full path name to the desired location for trace files followed by the output file base name. The base name is needed because the data collected by the Performance Collection Tool will be stored as a file on each host machine where a connected process with probes is running. The file name will consist of the base name you supply followed by a node specific suffix supplied by the Performance Collection Tool.

bufsize *buffer_size*

Specifies the AIX trace buffer size in Kilobytes. This value can be at most 1024, which is also the default value.

event {**mpi** | **process**}

Specifies the type of events (MPI events or process dispatch events) that are traced. By default, MPI events are traced.

logsize *maximum_log_size*

Specifies the maximum trace file size in Megabytes. The default is 20 M.

For example, to specify the directory tracefiles/mytrace as the output directory for the trace files:

```
trace set path tracefiles/mytrace
```

To specify the buffer size to be 900 K and the maximum trace file size to be 25 M:

```
trace set bufsize 900 logsize 25
```

trace show Subcommand (of the pct Command)

trace show {[**task** *task_list* | **group** *task_group_name*] {**markers** | **probes**} | **probetypes** | **path**}

The **trace show** subcommand lists, depending on the clause you specify, either:

- the currently installed trace probes:

trace show [**task** *task_list* | **group** *task_group_name*] **probes**

- the currently installed user markers:

trace show [**task** *task_list* | **group** *task_group_name*] **markers**

- the list of available **trace probe types** you can add using the **trace add** subcommand:

trace show probetypes

- the trace file output location and base name:

```
trace show path
```

The path is set by the subcommand:

```
trace set path
```

When listing the currently installed trace probes or user markers, the action is performed for the tasks in the current task group (as previously defined by the **group** subcommand). You can override this default, however, by specifying a task list or task group name when you issue the **trace show** subcommand.

task *task_list*

Specifies the connected POE tasks whose trace probes or user markers you want to list. The tasks in the POE application can be specified by listing individual values separated by commas (1,3,8,9), by giving a range of tasks using a colon to separate the ends of the range (12:15 refers to tasks 12, 13, 14, and 15), by giving a range and increment value using colons to separate the range and increment values (20:26:2 refers to tasks 20, 22, 24, and 26), or by using a combination of these (12:18,22,30).

group *task_group_name*

Specifies the name of a task group. Refer to the **group** subcommand for information on creating task groups.

markers

Specifies that you want the **trace show** subcommand to list the currently installed user markers. The markers are listed in the form:

```
[marker_identifier]
task_identifier.file_identifier.function_identifier.point_identifier
```

path Specifies that you want the **trace show** subcommand to return the trace file output location and base name as set by the subcommand:

```
trace set path
```

probes

Specifies that you want the **trace show** subcommand to list the currently installed trace probes.

probetypes

Specifies that you want the **trace show** subcommand to list the available trace probe types you can add using the **trace add** subcommand.

For example, to list the trace probes installed in the tasks in the current task group:

```
trace show probes
```

To list the user markers for the tasks in the task group "workers":

```
trace show group workers markers
```

To list the available probe types:

```
trace show probetypes
```

wait Subcommand (of the pct Command)**wait**

The **wait** subcommand places the Performance Collection Tool in an event loop so that it can wait for asynchronous system events (such as a task terminating) to occur. When one of these asynchronous events occurs, the Performance Collection Tool breaks out of the loop and returns the event that occurred. Be aware that this command is intended only for use within scripts you write, and is not intended for interactive command-line sessions. If you use it during an interactive command-line session, the only way to break out of the loop is to press **<control>-C** which will kill the Performance Collection Tool.

To see a list of the possible events that can break the Performance Collection Tool out of the event loop, issue the subcommand:

```
show commands
```

For example, the following example places the Performance Collection Tool into the event loop. It breaks out of this loop when the target application terminates, and returns the event name "*app_term*":

```
> wait  
app_term
```

pvt

NAME

pvt – Invokes the Profile Visualization Tool in either its graphical-user-interface or command-line mode.

SYNOPSIS

```
pvt [-cmd] [ one_or_more_file_names]
```

The **pvt** command starts the Profile Visualization Tool in either its graphical-user-interface mode, or, if the **-cmd** flag is specified, its command-line mode. In either mode, you can specify one or more file names to start the Profile Visualization Tool with profile data showing.

FLAGS

-cmd

Specifies that the Profile Visualization Tool should be started in command-line mode. Refer to “Using the Profile Visualization Tool’s Command Line Interface” on page 88 for information on the subcommands you can issue once the Profile Visualization Tool is running in this mode.

DESCRIPTION

The Profile Visualization Tool is a post-mortem analysis tool. It is designed to process profile data files generated by the Performance Collection Tool used in application profiling. You can run the Profile Visualization Tool in either its graphical-user-interface mode, or, if the **-cmd** flag is specified, its command-line mode. After processing profile data, you can view the results in the Profile Visualization Tool’s graphical user interface display, outputted to report files, or saved to a summary file. The Profile Visualization Tool provides a command-line interface to process individual profile files directly into a summary file without initializing the graphic display. The command-line interface also enables you to generate textual profile reports.

The **pvt** command’s subcommands (for controlling the Profile Visualization Tool in command-line mode) are listed alphabetically under “Subcommands of the pvt Command” on page 146.

EXAMPLES

To start the Profile Visualization Tool in graphical-user-interface mode showing an empty graphical user interface:

```
pvt
```

To start the Profile Visualization Tool in graphical-user-interface mode with profile data showing:

```
pvt one_or_more_file_names
```

To start the Profile Visualization Tool in command-line mode:

```
pvt -cmd
```

To start the Profile Visualization Tool in command-line mode with profile data showing:

IBM Confidential, Limited Rights Data

`pvt -cmd one_or_more_file_names`

RELATED INFORMATION

Commands: **pct(1)**

Subcommands of the pvt Command

exit Subcommand (of the pvt Command)

exit

The exit subcommand ends the command line session.

export Subcommand (of the pvt Command)

export *output_file_name*

The export subcommand allows you to export profile data to a specified file. The suffix *.txt* will be appended to the specified file name.

The currently loaded profile data is written to the user-specified file in plain text format, so the data can be loaded easily into a spreadsheet tool, like Lotus 1–2–3. The data that is loaded into the tool can be grouped the following types of records:

- Profile-session records associated with each process
- Individual function or thread records
- Function statistics records.

For more information on exporting data, refer to “Exporting Profile Data” on page 86.

load Subcommand (of the pvt Command)

load *one_or_more_file_names*

The load subcommand loads a set of profile data files into the session. If a set of data already exists, then the existing data is discarded and the newly loaded data becomes the current data to be used in future actions.

report Subcommand (of the pvt Command)

report [*list* | *output_file_name* | "*one_or_more_report_names*" *output_file_name* | "*one_or_more_report_ids*" *output_file_name*]

The report subcommand generates textual reports on the profile data. To show a list of available report types, enter:

```
report list
```

The result of the command will look something like:

- **[0] call_count:** function call count report
- **[1] wclock:** wall clock timer report
- **[2] ru_cpu:** CPU usage reports
- **[3] ru_mem:** memory usage report
- **[4] ru_paging:** paging activities reports
- **[5] ru_cswitch:** context switch activities reports
- **[6] pmc_cycle:** instructions per cycle hardware counter reports
- **[7] pmc_fpu:** floating point hardware counter reports

- **[8] pmc_fxu:** fixed-point hardware counter reports
- **[9] pmc_branch:** branch hardware counter reports
- **[10] pmc_lsu:** load and store hardware counter reports
- **[11] pmc_cache:** cache hardware counter reports
- **[12] pmc_misc:** miscellaneous hardware counter reports

For details of these reports, refer to “Generating Reports of Profile Data” on page 81.

To generate all the available reports to a file, enter:

```
report output_file_name
```

To generate reports by report name, enter:

```
report "one_or_more_report_names" output_file_name
```

For example:

```
report "wclock,ru_cpu" output
```

To generate reports by report id, enter:

```
report "one_or_more_report_ids" output_file_name
```

For example:

```
report "1,2" output
```

The report names or report ids in double quotes must be separated by a comma with no blank space in between. No matter how many reports are selected in one report command, all the reports are outputted to a single file specified in the report command.

sum Subcommand (of the pvt Command)

```
sum summary_file_name
```

The sum subcommand creates a summary file of all the loaded data. The merged summary data is written to the file specified in the command.

slogmerge

NAME

slogmerge – Merges multiple UTE interval files into a single SLOG file.

SYNOPSIS

```
slogmerge [-# | -?] [-n number_of_files] [-p profile_file_name]
[-c number_of_bytes_per_frame] [-f number_of_frames_per_frame_directory]
[-o output_file_name] [-s range] [-m number_of_available_markers]
[-r factor] input_file_name_prefix
```

The **slogmerge** command merges multiple UTE interval trace files (whose names begin with the *input_file_name_prefix*) into a single SLOG file. The *input_file_name_prefix* must be the last item on the command line.

FLAGS

-# or -?

Prints out the usage information for the **slogmerge** command instead of performing the actual merge.

-n *number_of_files*

Specifies the number of input UTE interval files to be merged. The default value is 1.

-p *profile_file_name*

Specifies the name of the description profile. If not specified, the default profile is the one specified by the environment variable **UTE_PROFILE**, or, if the environment variable **UTE_PROFILE** is not set, the file *profile.ute* in the current directory.

-c *number_of_bytes_per_frame*

Specifies the number of bytes per frame.

-f *number_of_frames_per_frame_directory*

Specifies the number of frames in each frame directory. When a frame directory is exhausted, the command automatically creates an additional frame directory and links it with existing frame directories.

-o *output_file_name*

Specifies the name for the output file — the merged SLOG file. If not specified, the default value is *trcfile.ute* in the current directory.

-s *range*

Specifies a list of MPI tasks to be merged. The task IDs in the list can be separated by either a comma (,) or a hyphen (-). If used, the hyphen represents a range of tasks. For example, *-s 0,2,4,5-7* indicates that the user wants to merge threads with MPI task IDs 0, 2, 4, 5, 6, and 7. By default, all tasks/threads in all UTE interval files will be merged.

-m *number_of_available_markers*

Specifies the number of spaces to reserve for user markers in the SLOG interval table. The number of available markers should not be less than the actual number of user markers in the UTE trace file, or the **slogmerge** utility will quit. The default number of available markers is 20.

-r *factor*

specifies the factor by which spaces for "pseudo records" are reserved. The

number of reserved slots for pseudo records is the number of threads in the trace file times the *factor*. If not specified, the default is 2.

Pseudo records are SLOG-specific interval records that are duplicates of certain internal records for visualization purposes. The number of pseudo records could be fairly high, depending on the number of nested states and their time span, and the number of internal records crossing SLOG frame boundaries in the trace. If the number of created pseudo records is more than the reserved slots during the merge process, the **slogmerge** utility will quit. If this happens, you should specify a larger number for this option to reserve more slots for pseudo records.

DESCRIPTION

The **slogmerge** command merges multiple UTE interval trace files into a single SLOG file. A number (as indicated by the **-n** option) of UTE files beginning with the *input_file_name_prefix* will be merged into an output file. The name of this output file is the one specified by the **-o** option, or, if the **-o** option is not specified, the file *trcfile.ute* in the current directory by default. The *input_file_name_prefix* must be the last item in the command line.

ENVIRONMENT VARIABLES

UTE_PROFILE

Specifies the name of the file description profile. If not set, the file *profile.ute* in the current directory is the default description profile. If the **-p** option is used, its specification overrides the **UTE_PROFILE** environment variable's setting.

EXAMPLES

To merge 5 UTE interval trace files that begin with the prefix *mytrace* into a single SLOG file:

```
slogmerge -n 5 mytrace
```

The above example will create an SLOG file with the default output file name *trcfile.ute*. To specify your own output file name, use the **-o** option.

```
slogmerge -n 5 -o mergedtrc.ute mytrace
```

To additionally specify that only the MPI tasks 2, 4, and 6 through 9 should be merged into the SLOG file, use the **-s** option.

```
slogmerge -n 5 -o mergedtrc.ute -s 2,4,6-9 mytrace
```

FILES

profile.ute default description profile

RELATED INFORMATION

Commands: **convert(1)**, **utemerge(1)**, **utestats(1)**

utemerge

NAME

utemerge – Merges multiple UTE interval files into a single UTE interval file.

SYNOPSIS

```
utemerge [-# | -?] [-n number_of_files] [-p profile_file_name]  
[-c number_of_records_per_frame] [-f number_of_frames_per_frame_directory]  
[-o output_file_name] [-s range] input_file_name_prefix
```

The **utemerge** command merges multiple UTE interval trace files (whose names begin with the *input_file_name_prefix*) into a single UTE file. The *input_file_name_prefix* must be the last item on the command line.

FLAGS

- # or -?**
Prints out the usage information for the **utemerge** command instead of performing the actual merge.
- n *number_of_files***
Specifies the number of input UTE interval files to be merged. The default value is 1.
- p *profile_file_name***
Specifies the name of the description profile. If not specified, the default profile is the one specified by the environment variable **UTE_PROFILE**, or, if the environment variable **UTE_PROFILE** is not set, the file *profile.ute* in the current directory.
- c *number_of_records_per_frame***
Specifies the number of bytes per frame.
- f *number_of_frames_per_frame_directory***
Specifies the number of frames in each frame directory. When a frame directory is exhausted, the command automatically creates an additional frame directory and links it with existing frame directories.
- o *output_file_name***
Specifies the name for the output file — the merged UTE file. If not specified, the default value is *trcfile.ute* in the current directory.
- s *range***
Specifies a list of MPI tasks to be merges. The task IDs in the list can be separated by either a comma (,) or a hyphen (-). If used, the hyphen represents a range of tasks. For example, *-s 0,2,4,5-7* indicates that the user wants to merge threads with MPI task IDs 0, 2, 4, 5, 6, and 7. By default, all tasks/threads in all UTE interval files will be merged.

DESCRIPTION

The **utemerge** command merges multiple UTE interval trace files into a single UTE interval trace file. A number (as indicated by the **-n** option) of UTE files beginning with the *input_file_name_prefix* will be merged into an output file. The name of this output file is the one specified by the **-o** option, or, if the **-o** option is not specified, the file *trcfile.ute* in the current directory by default. The *input_file_name_prefix* must be the last item in the command line.

ENVIRONMENT VARIABLES

UTE_PROFILE

Specifies the name of the file description profile. If not set, the file *profile.ute* in the current directory is the default description profile. If the **-p** option is used, its specification overrides the **UTE_PROFILE** environment variable's setting.

EXAMPLES

To merge 5 UTE interval trace files that begin with the prefix *mytrace* into a single UTE file:

```
utemerge -n 5 mytrace
```

The above example will create a UTE file with the default output file name *trcfile.ute*. To specify your own output file name, use the **-o** option.

```
utemerge -n 5 -o mergedtrc.ute mytrace
```

To additionally specify that only the MPI tasks 2, 4, and 6 through 9 should be merged into the UTE file, use the **-s** option.

```
utemerge -n 5 -o mergedtrc.ute -s 2,4,6-9 mytrace
```

FILES

profile.ute default description profile

RELATED INFORMATION

Commands: **convert(1)**, **slogmerge(1)**, **utestats(1)**

utestats

NAME

utestats – Generates statistics tables from UTE interval files.

SYNOPSIS

```
utestats [-?] [-p profile_file_name] [-o output_file_name]
[-B number_of_bins] [-f table_specification_file] input_file [input_file]...
```

The **utestats** command generates statistics tables from one or more UTE interval file. By default, six two-dimensional tables are generated. These tables are:

- Time Bin vs. Node
- Thread vs. Event Type
- Event Type vs. Thread
- Node vs. Event Type
- Event Type vs. Node
- Node vs. Processor

The computed statistic for all tables is the sum of the duration. By default, the statistics tables will be written to standard output. You can optionally save the statistics tables to a file using the **-o** flag.

FLAGS

- ? Prints out the usage information for the **utestats** command instead of generating statistics tables.
- p *profile_file_name*
Specifies the name of the description profile. If not specified, the default profile is the one specified by the environment variable **UTE_PROFILE**, or, if the environment variable is not set, the file *profile.ute* in the current directory.
- o *output_file_name*
Specifies the name of a file to which the statistics tables will be saved. If not specified, the statistics tables will be written to standard output.
- B *number_of_bins*
Specifies the number of bins in the Time vs. Node table. The default is 50.
- f *table_specification_file*
Specifies the name of a file containing custom table specifications.

DESCRIPTION

The **utestats** utility is able to take individual UTE interval files or a merged UTE interval file as input. If a number of individual UTE interval files are specified, the timestamps in each file will start at 0 without alignment with respect to global clock values. If, instead, a merged UTE interval file is specified, the timestamps of records from different nodes will already have been adjusted with respect to the global clock value.

By default, six two-dimensional tables are generated. These tables are:

- Time Bin vs. Node
- Thread vs. Event Type

- Event Type vs. Thread (a row/column transposition of the Thread vs. Event Type table)
- Node vs. Event Type
- Event Type vs. Node (a row/column transposition of the Node vs. Event Type table)
- Node vs. Processor

The computed statistic for all the tables is the sum or the duration. As you can see, several tables are simply row/column transpositions of the tables. These transposed tables are provided so that a program used to visualize the tables does not have to be able to transpose a table in order to show a transposed view.

The output of the **utestats** command is written in tab-separated-value format; each line of output is a row of a table, and columns in a row are separated by a tab character. Tables are separated by a Form Feed character (0x0c). This format is used to make it easy to import a **utestats** output file into a spreadsheet program.

A Node vs. Processor table would look like the following (where the tabs have been replaced by spaces to make the column alignment clearer).

node/cpu	0	1
0	2.823739	2.258315
1	0.873746	4.241253
2	0.956515	4.322891
3	0.853188	4.334650

The first value "node/cpu" is the name of the table. It consists of the row title followed by a "/" followed by a column title. This table contains statistics aggregated over interval records whose field values for "node" and "cpu" are the same. The values "node" and "cpu" are the field names as stored in the UTE profile file. The rest of the values in the first row are the column labels; these are the values that appeared in the "cpu" field in at least one interval record.

With other rows, the first field is the row label; it is a value that appeared in the node field in at least one interval record. The other fields in a row are the accumulated duration of all interval records with the same ("node", "cpu") pair of values. For example, the accumulated duration of all interval records for "cpu" 1 of "node" 0 was 2.258315 seconds.

ENVIRONMENT VARIABLES

UTE_PROFILE

Specifies the name of the file description profile. If not set, the file *profile.ute* in the current directory is the default description profile. If the **-p** option is used, its specification overrides the **UTE_PROFILE** environment variable's setting.

EXAMPLES

To generate statistics tables for a single UTE interval file:

```
utestats mytrace.ute
```

The above example will write the statistics tables to standard output. To redirect the output to a file, use the **-o** option.

```
utestats -o statables mytrace.ute
```

You can also specify multiple UTE interval files from which statistics should be generated.

utestats mytrace.ute mytrace2.ute mytrace3.ute

FILES

profile.ute default description profile

RELATED INFORMATION

Commands: **convert**(1), **utemerge**(1)

Appendix C. PE Benchmark Messages

2554-002 Internal error from *number*.

Explanation: An internal error was found at the specified line number.

User Response: Gather information about the problem and follow local site procedures for reporting hardware and software problems.

2554-003 Syntax error, near *string*.

Explanation: A syntax error was found near the location of the command.

User Response: Correct the input command and retry.

2554-004 Duplicate keyword *string* found.

Explanation: The command keyword was repeated more than one time.

User Response: Correct the error and retry.

2554-005 Connect command has both all and task/group qualifier.

Explanation: Both the **all** and **task** group qualifiers are present with the **connect** command.

User Response: Remove either the **all** or **task** qualifier.

2554-006 The task does not exist, is in an inappropriate state, or does not contain the data requested.

Explanation: There was an attempt to retrieve data, that is, a file id, on a task that either does not exist or does not have the data requested. Also, the task could be in a state that does not allow the specified action, such as a disconnected task trying to be destroyed. You can destroy the target if disconnected, but cannot specify individual tasks.

User Response: Try a different task and/or different data.

2554-007 Group name *string* is not defined.

Explanation: The specified group name is not defined.

User Response: Use **show groups** to find all the defined groups.

2554-008 Group name *string* is empty.

Explanation: The specified group name is empty.

User Response: Enter some tasks into the specified group.

2554-009 Cannot find the default group or the default group is empty.

Explanation: The default group could not be found, or the default group is empty.

User Response: If the application is not connected, connect to it. Run the **show group** command to see the tasks.

2554-010 Session needs to connect or load before running the command.

Explanation: The session is not connected to any application.

User Response: Correct the error and retry.

2554-011 Ais_status failure code *number*.

Explanation: DPCL returned a bad status.

User Response: Check the DPCL reference menu.

2554-012 Cannot modify predefined group name *string*.

Explanation: The group name is a predefined name. Tasks cannot be added or removed.

User Response: Use the **connect** or **disconnect** command to add or remove tasks in the connected group.

2554-013 The target has completed or has been killed on its own.

Explanation: The target has either completed or has been stopped using **<Ctrl-C>**. Also the target may have been destroyed in some other manner, external to the tool.

User Response: You must load or connect to a new target before doing any more work.

2554-014 The command failed to allocate enough memory for its use.

Explanation: The current command failed to allocate memory.

User Response: Check the system memory usage, page space, etc.

2554-021 Session is already connected to an application.

Explanation: The session is already connected to an application.

User Response: Correct the error and retry.

2554-022 Cannot disconnect *number*.

Explanation: Cannot disconnect the task.

User Response: Check the task number; it must be connected or suspended.

2554-023 *pid number* specified is not valid.

Explanation: The process id (*pid*) is not valid.

User Response: Double check the *pid*.

2554-024 All tasks in this command are already connected.

Explanation: All tasks in the **connect** command are already connected; or there was an attempt to connect an empty group.

User Response: Correct the error and retry.

2554-025 *pid number* specified is not a valid *poe pid*.

Explanation: The process id (*pid*) is not a valid *poe pid*.

User Response: Double check the *pid*.

2554-026 Cannot find the *string* to help.

Explanation: Cannot find the help information for the specified name.

User Response: Check the name type.

2554-027 Help is not available in the beta release.

Explanation: Cannot find the help information in the beta release.

User Response: None.

2554-031 Cannot suspend *number*.

Explanation: Cannot suspend the task.

User Response: Check the task number; it must be connected or loaded.

2554-033 Cannot resume *number*.

Explanation: Cannot resume the task.

User Response: Check the task number; it must be suspended.

2554-036 No connected target.

Explanation: There is no target for you to destroy.

User Response: Connect to a target.

2554-038 The target was not loaded.

Explanation: The target was connected to "not created".

User Response: Do not try to start this target.

2554-041 The *stdin* command input is not valid.

Explanation: The text part of the **stdin** command is not valid.

User Response: Check the input text; it must be a quoted string.

2554-042 The application loaded cannot take *stdin* as input.

Explanation: The application loaded reads the *stdin* from a file.

User Response: If the application needs to read the input from user input, remove the *stdin* clause from the **load** command.

2554-043 The user closed **STDIN already.**

Explanation: The user has closed *stdin*.

User Response: The user cannot close *stdin* if more input is expected.

2554-051 No *file_spec* was provided.

Explanation: No *file_spec* was given for a command that requires it.

User Response: Type **file *regexp*** where *regexp* is a regular expression such as **"*"**.

2554-052 Session loaded with *string* already.

Explanation: The session loaded a tool.

User Response: Use a separate session.

2554-053 Tool name *string* is not a valid tool name.

Explanation: The tool is not a valid name.

User Response: Use **show tools** to find all the defined tools.

2554-054 Tool *string* failed to load.

Explanation: The tool specified in the tool name field failed to load.

User Response: Report the failure to the owner of the tool.

2554-055 Tool *string* failed to initialize.

Explanation: The tool specified in the tool name field failed to initialize properly.

User Response: Report the failure to the owner of the tool.

2554-060 The file id *string* provided is invalid.

Explanation: The file id provided does not exist.

User Response: Enter a different file id. Use the **file** command to find a list of valid file ids.

2554-061 The function id *string* provided is invalid.

Explanation: The function id provided does not exist.

User Response: Enter a different function id. Use the **function** command to find a list of valid function ids.

2554-062 End of File.

Explanation: The end of the file was reached.

User Response: State an earlier line.

2554-063 File *string* was not found.

Explanation: The file specified was not found.

User Response: State a different file name.

2554-064 Function *string* was not found.

Explanation: The function specified was not found.

User Response: State a different function name.

2554-065 The first line specified is invalid.

Explanation: The file does not have this many lines.

User Response: Enter a smaller line number. The number must be positive, that is, greater than 0.

2554-066 The last line specified is invalid.

Explanation: The file does not have this many lines.

User Response: Enter a smaller line number. The number must be positive, that is, greater than 0, and must be greater than or equal to the start line.

2554-067 No path was set.

Explanation: The path was not yet set. The default path is the current directory.

User Response: Issue a **set *sourcepath***.

2554-068 No previous file set for this command.

Explanation: There is no previous file set for this command.

User Response: Use the **list file** command before the **list next** call.

2554-076 File *string* failed to open.

Explanation: The file specified cannot be opened.

User Response: Check to see if the user has permission to open the file to write.

2554-076 File *string* failed to write.

Explanation: The file specified cannot be written to.

User Response: Check to see if the file system has sufficient disk space to write.

2554-080 *poeargs* was specified but not *poe*.

Explanation: The user tried to specify the **poeargs** argument for a job that was not a **poe**.

User Response: If the job is supposed to be a **poe**, type **poe** on the line; if not, do not use **poeargs**.

2554-081 load command needs an *exec* clause or *mpmdcmd* clause to specify program to load.

Explanation: There is no program specified to attempt to load.

User Response: Try **load** again; this time with an **exec** clause or **mpmdcmd** clause.

2554-082 load failed.

Explanation: PE Benchmark was unable to load the program specified.

User Response: Make sure the correct path was specified, that the program is the correct one, and if **poe**, that either environment variables are set or you specify **poeargs**.

2554-083 load command cannot have both *exec* clause and *mpmdcmd* clause.

Explanation: You cannot specify both the **exec** clause and the **mpmdcmd** clause with the **load** command.

User Response: Try **load** again; this time with only an

`exec` clause or an `mpmcmd` clause.

2554-084 **mpmcmd clause also requires poe keyword.**

Explanation: The `load mpmcmd` command requires the `poe` keyword.

User Response: Try `load` again; this time with the `poe` keyword.

2554-085 **Different version between tool and application.**

Explanation: The tool and the target application do not have the same version.

User Response: Recompile the application.

2554-086 ***string* is not a full path.**

Explanation: The specified path is not a full path.

User Response: Use the full path name.

2554-201 **Path name not set.**

Explanation: The path name is not set yet.

User Response: Set the path name.

2554-202 **Excessive probe data size: *number*.**

Explanation: The size of the probe data memory is too large to allocate.

User Response: Try using a smaller amount.

2554-203 **Error has occurred while updating MPI Event masks.**

Explanation: One of the probe expressions executed failed.

2554-204 **Error has occurred while installing and activating probes.**

Explanation: One or more of the probe installations or activations failed.

2554-205 **The probe id: *number* is out of range.**

Explanation: The probe id is either less than 0 or greater than the number of probes.

User Response: List the probes to see how many there are and choose a valid id.

2554-206 **The task id: *number* has previous errors.**

Explanation: The task id has had an `AisStatus` error on a previous `add`.

User Response: Use a different task or restart the tool.

2554-207 **Invalid probe path: *string*.**

Explanation: The path assigned via the `set path` command is invalid for a given task.

User Response: Make sure the directory exists.

2554-221 **Invalid keyword *string* found in trace help command.**

Explanation: Invalid `help` keyword.

User Response: Check the input text.

2554-231 **Event *string* is unknown to the command.**

Explanation: The event name specified is unknown to this command. No events are set.

User Response: Provide a valid event name.

2554-232 **bufsize specified *number* is outside the valid range *minimum number* — *maximum number*.**

Explanation: The size specified is invalid.

User Response: Provide a valid size.

2554-233 **logsize specified *number* is outside the valid range *minimum number* — *maximum number*.**

Explanation: The size specified is invalid.

User Response: Provide a valid size.

2554-234 **Path *string* is invalid.**

Explanation: The path specified is invalid.

User Response: Provide a valid path.

2554-241 **No functions were found to meet any of the expressions provided.**

Explanation: Unlike specific function ids or ranges, an expression may match nothing and will not produce an error in the internal functions. For the adding of probes and markers, this **is** invalid. Therefore, if not even one function from any function expression provided produces a match, and a valid function id is provided, then the **add** command itself will produce this generic message.

User Response: Provide a valid file id/expression and a valid function id/expression. Use the **file**, **function**, and **point** commands to find out what is and is not valid.

2554–251 More than one instrumentation point can be derived from this command.

Explanation: Only one instrumentation point can occur in one **add marker** command.

User Response: Check the input text and remove the extra files or functions, especially if a regular expression is involved.

2554–252 No instrumentation point can be derived from this command.

Explanation: Only one instrumentation point can occur in one **add marker** command.

User Response: Check the input text, or use the **point** command to find all the available points.

2554–253 No matching marker name *string* found from previous commands.

Explanation: The marker name must be paired with the previous **beginmarker**.

User Response: Issue **trace show markers** to see all the previous defined markers.

2554–254 No marker name found in the command.

Explanation: Missing marker name for **beginmarker** and **endmarker**.

User Response: Add the name.

2554–255 Different tasks between beginmarker and endmarker.

Explanation: Tasks are mismatched for **beginmarker** and **endmarker**.

User Response: Match the tasks.

2554–255 Duplicate name *string* found in marker id *number*.

Explanation: A duplicate name was found in the previously defined marker.

User Response: Change the name.

2554–256 Invalid marker name found.

Explanation: The marker name contains unprintable characters.

User Response: Change the name.

2554–261 Marker id *number* is invalid.

Explanation: The marker id specified is not valid.

User Response: Issue **trace show markers** to find all the valid marker ids.

2554–262 Marker id *number* is already removed.

Explanation: The marker id specified is already removed.

User Response: Issue **trace show markers** to find all the valid marker ids.

2554–301 Path name not set.

Explanation: The path name is not set yet.

User Response: Set the path name.

2554–204 The task id: *number* has previous errors.

Explanation: The task id has had an AisStatus error on a previous add.

User Response: Use a different task or restart the tool.

2554–321 Invalid keyword *string* found in trace help command.

Explanation: Invalid **help** keyword.

User Response: Check the input text.

2554–326 ProfName *string* is unknown to this cpu type.

Explanation: The *ProfName* used in this **show probetype** command is not valid.

User Response: Use the **show probetypes** command to find a valid name.

2554–341 ProfId or ProfName is not valid in this cpu type.

Explanation: The *ProfId* or *ProfName* used in this **add** command is not valid.

User Response: Make sure the tasks do not have mixed cpu type.

2554–342 Option part of the command is not valid in this ProfType.

Explanation: The *Option* part of the command is not valid for the *ProfType*.

User Response: Make sure the option specified is valid.

2554–343 **Proflid has been specified, only one option is allowed.**

Explanation: Only one *Proflid/Option* pair is allowed in the command.

User Response: Remove the duplicates.

2554–344 **Some functions are profiled in the probe id *number*, duplication is not allowed.**

Explanation: Some of the functions are profiled in the previous command. Duplication is not allowed.

User Response: Use the **show probes** command to examine the command in question.

2554–345 **Task *number* has different probe type specified.**

Explanation: The task id had been used before with a different probe type. Only one probe type can be used in a profile session.

User Response: The user needs to plan ahead on what profile type to monitor, and use that type across the profiling session.

2554–346 **There is no function in the specified file and function combination.**

Explanation: The specified file and function combination generates an empty result.

User Response: Use the **function** command to verify the function list.

2554–347 **Too many files/functions in the add command.**

Explanation: The number of functions that can be added in a single **add** command is 200.

User Response: Separate the command into two.

2554–348 **Name *string* is unknown to the command.**

Explanation: The event name specified is unknown to the **add** command.

User Response: Type the correct name.

2554–349 **At least one task *number* does not have PMAPI installed.**

Explanation: The host which runs the specified task id does not have PMAPI installed and cannot collect the hardware counter event information.

User Response: Install the PMAPI on the node.

2554–351 **No previous added profile to be removed.**

Explanation: There is no previous added profile to be removed.

User Response: Issue the **add** command before the **remove** command.

2554–352 **The range started from *number* is too big.**

Explanation: The range specified is outside the boundary of the added profile.

User Response: Issue **show probes** to see the list of profiles.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department LJEB/P905
2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States or other countries or both:

AIX
ESCON
IBM
IBMLink
LoadLeveler

Micro Channel
RS/6000
RS/6000 SP

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, BackOffice, MS-DOS, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Pentium and Pentium II Xeon are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Tivoli Enterprise Console is a trademark of Tivoli Systems Inc. in the United States, other countries, or both.

UNIX is a registered trademark in the United States, other countries, or both and is licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be the trademarks or service marks of others.

Acknowledgements

This product includes software developed by the Apache Software Foundation (<http://www.apache.org>).

Glossary of Terms and Abbreviations

This glossary includes terms and definitions from:

- The *Dictionary of Computing*, New York: McGraw-Hill, 1994.
- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by the symbol (A) after the definition.
- The *ANSI/EIA Standard - 440A: Fiber Optic Terminology*, copyright 1989 by the Electronics Industries Association (EIA). Copies can be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue N.W., Washington, D.C. 20006. Definitions are identified by the symbol (E) after the definition.
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

This section contains some of the terms that are commonly used in the Parallel Environment books and in this book in particular.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard *Vocabulary for Information Processing* (Copyright 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3. ANSI definitions are preceded by an asterisk (*).

Other definitions in this glossary are taken from *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems* (GC20-1699).

A

address. A value, possibly a character or group of characters that identifies a register, a device, a particular part of storage, or some other data source or destination.

AIX. Abbreviation for Advanced Interactive Executive, IBM's licensed version of the UNIX operating system. AIX is particularly suited to support technical computing applications, including high function graphics and floating point computations.

AIXwindows Environment/6000. A graphical user interface (GUI) for the RS/6000. It has the following components:

- A graphical user interface and toolkit based on OSF/Motif
- Enhanced X-Windows, an enhanced version of the MIT X Window System
- Graphics Library (GL), a graphical interface library for the applications programmer which is compatible with Silicon Graphics' GL interface.

API. Application Programming Interface.

application. The use to which a data processing system is put; for example, topayroll application, an airline reservation application.

argument. A parameter passed between a calling program and a called program or subprogram.

attribute. A named property of an entity.

B

bandwidth. The total available bit rate of a digital channel.

blocking operation. An operation which does not complete until the operation either succeeds or fails. For example, a blocking receive will not return until a message is received or until the channel is closed and no further messages can be received.

breakpoint. A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

broadcast operation. A communication operation in which one processor sends (or broadcasts) a message to all other processors.

buffer. A portion of storage used to hold input or output data temporarily.

C

C. A general purpose programming language. It was formalized by ANSI standards committee for the C language in 1984 and by Uniforum in 1983.

C++. A general purpose programming language, based on C, which includes extensions that support an object-oriented programming paradigm. Extensions include:

- strong typing
- data abstraction and encapsulation
- polymorphism through function overloading and templates
- class inheritance.

call arc. The representation of a call between two functions within the Xprofiler function call tree. It appears as a solid line between the two functions. The arrowhead indicates the direction of the call; the function it points to is the one that receives the call. The function making the call is known as the *caller*, while the function receiving the call is known as the *callee*.

chaotic relaxation. An iterative relaxation method which uses a combination of the Gauss-Seidel and Jacobi-Seidel methods. The array of discrete values is divided into sub-regions which can be operated on in parallel. The sub-region boundaries are calculated using Jacobi-Seidel, whereas the sub-region interiors are calculated using Gauss-Seidel. See also *Gauss-Seidel*.

client. A function that requests services from a server, and makes them available to the user.

cluster. A group of processors interconnected through a high speed network that can be used for high performance computing. It typically provides excellent price/performance.

collective communication. A communication operation which involves more than two processes or tasks. Broadcasts, reductions, and the MPI_Allreduce subroutine are all examples of collective communication operations. All tasks in a communicator must participate.

command alias. When using the PE command line debugger, *pdbx*, you can create abbreviations for existing commands using the ***pdbx alias*** command. These abbreviations are known as *command aliases*.

Communication Subsystem (CSS). A component of the IBM Parallel System Support Programs for AIX that provides software support for the High Performance Switch. It provides two protocols; IP (Internet Protocol) for LAN based communication and US (user space) as a message passing interface that is optimized for performance over the switch. See also *Internet Protocol* and *User Space*.

communicator. An MPI object that describes the communication context and an associated group of processes.

compile. To translate a source program into an executable program.

condition. One of a set of specified values that a data item can assume.

control workstation. A workstation attached to the RS/6000 SP that serves as a single point of control allowing the administrator or operator to monitor and manage the system using IBM Parallel System Support Programs for AIX.

core dump. A process by which the current state of a program is preserved in a file. Core dumps are usually associated with programs that have encountered an unexpected, system-detected fault, such as a Segmentation Fault, or severe user error. The current program state is needed for the programmer to diagnose and correct the problem.

core file. A file which preserves the state of a program, usually just before a program is terminated for an unexpected error. See also *core dump*.

current context. When using either of the PE parallel debuggers, control of the parallel program and the display of its data can be limited to a subset of the tasks that belong to that program. This subset of tasks is called the *current context*. You can set the current context to be a single task, multiple tasks, or all the tasks in the program.

D

data decomposition. A method of breaking up (or decomposing) a program into smaller parts to exploit parallelism. One divides the program by dividing the data (usually arrays) into smaller parts and operating on each part independently.

data parallelism. Refers to situations where parallel tasks perform the same computation on different sets of data.

dbx. A symbolic command line debugger that is often provided with UNIX systems. The PE command line debugger, ***pdbx***, is based on the ***dbx*** debugger.

debugger. A debugger provides an environment in which you can manually control the execution of a program. It also provides the ability to display the program's data and operation.

distributed shell (dsh). An IBM Parallel System Support Programs for AIX command that lets you issue commands to a group of hosts in parallel. See the *IBM*

RISC System/6000 Scalable POWERparallel Systems: Command and Technical Reference (GC23-3900-00) for details.

domain name. The hierarchical identification of a host system (in a network), consisting of human-readable labels, separated by decimals.

E

environment variable. 1. A variable that describes the operating environment of the process. Common environment variables describe the home directory, command search path, and the current time zone. 2. A variable that is included in the current software environment and is therefore available to any called program that requests it.

event. An occurrence of significance to a task; for example, the completion of an asynchronous operation such as an input/output operation.

Ethernet. Ethernet is the standard hardware for TCP/IP LANs in the UNIX marketplace. It is a 10 megabit per second baseband type network that uses the contention based CSMA/CD (collision detect) media access method.

executable. A program that has been link-edited and therefore can be run in a processor.

execution. To perform the actions specified by a program or a portion of a program.

expression. In programming languages, a language construct for computing a value from one or more operands.

F

fairness. A policy in which tasks, threads, or processes must be allowed eventual access to a resource for which they are competing. For example, if multiple threads are simultaneously seeking a lock, then no set of circumstances can cause any thread to wait indefinitely for access to the lock.

FDDI. Fiber distributed data interface (100 Mbit/s fiber optic LAN).

file system. In the AIX operating system, the collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

fileset. 1) An individually installable option or update. Options provide specific function while updates correct an error in, or enhance, a previously installed product. 2) One or more separately installable, logically grouped units in an installation package. See also *Licensed Program Product* and *package*.

foreign host. See *remote host*.

Fortran. One of the oldest of the modern programming languages, and the most popular language for scientific and engineering computations. It's name is a contraction of *FORmula TRANslation*. The two most common Fortran versions are Fortran 77, originally standardized in 1978, and Fortran 90. Fortran 77 is a proper subset of Fortran 90.

function call tree. A graphical representation of all the functions and calls within an application, which appears in the Xprofiler main window. The functions are represented by green, solid-filled rectangles called function boxes. The size and shape of each function box indicates its CPU usage. Calls between functions are represented by blue arrows, called call arcs, drawn between the function boxes. See also *call arcs*.

function cycle. A chain of calls in which the first caller is also the last to be called. A function that calls itself recursively is not considered a function cycle.

functional decomposition. A method of dividing the work in a program to exploit parallelism. One divides the program into independent pieces of functionality which are distributed to independent processors. This is in contrast to data decomposition which distributes the same work over different data to independent processors.

functional parallelism. Refers to situations where parallel tasks specialize in particular work.

G

Gauss-Seidel. An iterative relaxation method for solving Laplace's equation. It calculates the general solution by finding particular solutions to a set of discrete points distributed throughout the area in question. The values of the individual points are obtained by averaging the values of nearby points. Gauss-Seidel differs from Jacobi-Seidel in that for the $i+1$ st iteration Jacobi-Seidel uses only values calculated in the i th iteration. Gauss-Seidel uses a mixture of values calculated in the i th and $i+1$ st iterations.

global max. The maximum value across all processors for a given variable. It is global in the sense that it is global to the available processors.

global variable. A variable defined in one portion of a computer program and used in at least one other portion of the computer program.

gprof. A UNIX command that produces an execution profile of C, Pascal, Fortran, or COBOL programs. The execution profile is in a textual and tabular format. It is useful for identifying which routines use the most CPU time. See the man page on **gprof**.

GUI (Graphical User Interface). A type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device.

H

High Performance Switch. The high-performance message passing network, of the RS/6000 SP(SP) machine, that connects all processor nodes.

HIPPI. High performance parallel interface.

hook. **hook** is a **pdbx** command that allows you to re-establish control over all task(s) in the current context that were previously unhooked with this command.

home node. The node from which an application developer compiles and runs his program. The home node can be any workstation on the LAN.

host. A computer connected to a network, and providing an access method to that network. A host provides end-user services.

host list file. A file that contains a list of host names, and possibly other information, that was defined by the application which reads it.

host name. The name used to uniquely identify any computer on a network.

hot spot. A memory location or synchronization resource for which multiple processors compete excessively. This competition can cause a disproportionately large performance degradation when one processor that seeks the resource blocks, preventing many other processors from having it, thereby forcing them to become idle.

I

IBM Parallel Environment for AIX. A program product that provides an execution and development environment for parallel Fortran, C, or C++ programs. It also includes tools for debugging, profiling, and tuning parallel programs.

installation image. A file or collection of files that are required in order to install a software product on a RS/6000 workstation or on SP system nodes. These files are in a form that allows them to be installed or removed with the AIX **installp** command. See also *fileset*, *Licensed Program Product*, and *package*.

Internet. The collection of worldwide networks and gateways which function as a single, cooperative virtual network.

Internet Protocol (IP). 1) The TCP/IP protocol that provides packet delivery between the hardware and

user processes. 2) The High Performance Switch library, provided with the IBM Parallel System Support Programs for AIX, that follows the IP protocol of TCP/IP.

IP. See *Internet Protocol*.

J

Jacobi-Seidel. See *Gauss-Seidel*.

job management system.

The software you use to manage the jobs across your system, based on the availability and state of system resources.

K

Kerberos. A publicly available security and authentication product that works with the IBM Parallel System Support Programs for AIX software to authenticate the execution of remote commands.

kernel. The core portion of the UNIX operating system which controls the resources of the CPU and allocates them to the users. The kernel is memory-resident, is said to run in *kernel mode* (in other words, at higher execution priority level than *user mode*) and is protected from user tampering by the hardware.

L

Laplace's equation. A homogeneous partial differential equation used to describe heat transfer, electric fields, and many other applications.

latency. The time interval between the instant at which an instruction control unit initiates a call for data transmission, and the instant at which the actual transfer of data (or receipt of data at the remote end) begins. Latency is related to the hardware characteristics of the system and to the different layers of software that are involved in initiating the task of packing and transmitting the data.

Licensed Program Product (LPP). A collection of software packages, sold as a product, that customers pay for to license. It can consist of packages and filesets a customer would install. These packages and filesets bear a copyright and are offered under the terms and conditions of a licensing agreement. See also *fileset* and *package*.

LoadLeveler. A job management system that works with POE to allow users to run jobs and match processing needs with system resources, in order to better utilize the system.

local variable. A variable that is defined and used only in one specified portion of a computer program.

loop unrolling. A program transformation which makes multiple copies of the body of a loop, placing the copies also within the body of the loop. The loop trip count and index are adjusted appropriately so the new loop computes the same values as the original. This transformation makes it possible for a compiler to take additional advantage of instruction pipelining, data cache effects, and software pipelining.

See also *optimization*.

M

menu. A list of options displayed to the user by a data processing system, from which the user can select an action to be initiated.

message catalog. A file created using the AIX Message Facility from a message source file that contains application error and other messages, which can later be translated into other languages without having to recompile the application source code.

message passing. Refers to the process by which parallel tasks explicitly exchange program data.

MIMD (Multiple Instruction Multiple Data). A parallel programming model in which different processors perform different instructions on different sets of data.

MPMD (Multiple Program Multiple Data). A parallel programming model in which different, but related, programs are run on different sets of data.

MPI. Message Passing Interface; a standardized API for implementing the message passing model.

N

network. An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

node. (1) In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network. (2) In terms of the RS/6000 SP, a single location or workstation in a network. An SP node is a physical entity (a processor).

node ID. A string of unique characters that identifies the node on a network.

nonblocking operation. An operation, such as sending or receiving a message, which returns immediately whether or not the operation was completed. For example, a nonblocking receive will not wait until a message is sent, but a blocking receive will wait. A nonblocking receive will return a status value that indicates whether or not a message was received.

O

object code. The result of translating a computer program to a relocatable, low-level form. Object code contains machine instructions, but symbol names (such as array, scalar, and procedure names), are not yet given a location in memory.

optimization. A not strictly accurate but widely used term for program performance improvement, especially for performance improvement done by a compiler or other program translation software. An optimizing compiler is one that performs extensive code transformations in order to obtain an executable that runs faster but gives the same answer as the original. Such code transformations, however, can make code debugging and performance analysis very difficult because complex code transformations obscure the correspondence between compiled and original source code.

option flag. Arguments or any other additional information that a user specifies with a program name. Also referred to as *parameters* or *command line options*.

P

package. A number of filesets that have been collected into a single installable image of program products, or LPPs. Multiple filesets can be bundled together for installing groups of software together. See also *fileset* and *Licensed Program Product*.

parallelism. The degree to which parts of a program may be concurrently executed.

parallelize. To convert a serial program for parallel execution.

Parallel Operating Environment (POE). An execution environment that smooths the differences between serial and parallel execution. It lets you submit and manage parallel jobs. It is abbreviated and commonly known as POE.

parameter. * (1) In Fortran, a symbol that is given a constant value for a specified application. (2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. (3) A name in a procedure that is used to refer to an argument that is passed to the procedure. (4) A particular piece of information that a system or application program needs to process a request.

partition. (1) A fixed-size division of storage. (2) In terms of the RS/6000 SP, a logical definition of nodes to be viewed as one system or domain. System partitioning is a method of organizing the SP into groups of nodes for testing or running different levels of software of product environments.

Partition Manager. The component of the Parallel Operating Environment (POE) that allocates nodes, sets up the execution environment for remote tasks, and manages distribution or collection of standard input (STDIN), standard output (STDOUT), and standard error (STDERR).

pdbx. **pdbx** is the parallel, symbolic command line debugging facility of PE. **pdbx** is based on the **dbx** debugger and has a similar interface.

PE. The IBM Parallel Environment for AIX program product.

performance monitor. A utility which displays how effectively a system is being used by programs.

POE. See Parallel Operating Environment.

pool. Groups of nodes on an SP that are known to the Resource Manager, and are identified by a number.

point-to-point communication. A communication operation which involves exactly two processes or tasks. One process initiates the communication through a *send* operation. The partner process issues a *receive* operation to accept the data being sent.

procedure. (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a procedure call. (2) A set of related control statements that cause one or more programs to be performed.

process. A program or command that is actually running the computer. It consists of a loaded version of the executable file, its data, its stack, and its kernel data structures that represent the process's state within a multitasking environment. The executable file contains the machine instructions (and any calls to shared objects) that will be executed by the hardware. A process can contain multiple threads of execution.

The process is created via a **fork()** system call and ends using an **exit()** system call. Between **fork** and **exit**, the process is known to the system by a unique process identifier (pid).

Each process has its own virtual memory space and cannot access another process's memory directly. Communication methods across processes include pipes, sockets, shared memory, and message passing.

prof. A utility which produces an execution profile of an application or program. It is useful to identifying which routines use the most CPU time. See the man page for **prof**.

profiling. The act of determining how much CPU time is used by each function or subroutine in a program. The histogram or table produced is called the execution profile.

Program Marker Array. An X-Windows run time monitor tool provided with Parallel Operating Environment, used to provide immediate visual feedback on a program's execution.

pthread. A thread that conforms to the POSIX Threads Programming Model.

R

reduction operation. An operation, usually mathematical, which reduces a collection of data by one or more dimensions. For example, the arithmetic SUM operation is a reduction operation which reduces an array to a scalar value. Other reduction operations include MAXVAL and MINVAL.

remote host. Any host on a network except the one at which a particular operator is working.

remote shell (rsh). A command supplied with both AIX and the IBM Parallel System Support Programs for AIX that lets you issue commands on a remote host.

Report. In Xprofiler, a tabular listing of performance data that is derived from the gmon.out files of an application. There are five types of reports that are generated by Xprofiler, and each one presents different statistical information for an application.

Resource Manager. A server that runs on one of the nodes of an RS/6000 SP (SP) machine. It prevents parallel jobs from interfering with each other, and reports job-related node information.

RISC. Reduced Instruction Set Computing (RISC), the technology for today's high performance personal computers and workstations, was invented in 1975.

S

shell script. A sequence of commands that are to be executed by a shell interpreter such as C shell, korn shell, or Bourne shell. Script commands are stored in a file in the same form as if they were typed at a terminal.

segmentation fault. A system-detected error, usually caused by referencing an invalid memory address.

server. A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server.

signal handling. A type of communication that is used by message passing libraries. Signal handling involves using AIX signals as an asynchronous way to move data in and out of message buffers.

source line. A line of source code.

source code. The input to a compiler or assembler, written in a source language. Contrast with object code.

SP. RS/6000 SP; a scalable system from two to 128 processor nodes, arranged in various physical configurations, that provides a high powered computing environment.

SPMD (Single Program Multiple Data). A parallel programming model in which different processors execute the same program on different sets of data.

standard input (STDIN). In the AIX operating system, the primary source of data entered into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output (STDOUT). In the AIX operating system, the primary destination of data produced by a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can go to a file or to another command.

stencil. A pattern of memory references used for averaging. A 4-point stencil in two dimensions for a given array cell, $x(i,j)$, uses the four adjacent cells, $x(i-1,j)$, $x(i+1,j)$, $x(i,j-1)$, and $x(i,j+1)$.

subroutine. (1) A sequence of instructions whose execution is invoked by a call. (2) A sequenced set of instructions or statements that may be used in one or more computer programs and at one or more points in a computer program. (3) A group of instructions that can be part of another routine or can be called by another program or routine.

synchronization. The action of forcing certain points in the execution sequences of two or more asynchronous procedures to coincide in time.

system administrator. (1) The person at a computer installation who designs, controls, and manages the use of the computer system. (2) The person who is responsible for setting up, modifying, and maintaining the Parallel Environment.

System Data Repository. A component of the IBM Parallel System Support Programs for AIX software that provides configuration management for the SP system. It manages the storage and retrieval of system data across the control workstation, file servers, and nodes.

System Status Array. An X-Windows run time monitor tool, provided with the Parallel Operating Environment, that lets you quickly survey the utilization of processor nodes.

T

task. A unit of computation analogous to an AIX process.

thread. A single, separately dispatchable, unit of execution. There may be one or more threads in a process, and each thread is executed by the operating system concurrently.

tracing. In PE, the collection of data for the Visualization Tool (VT). The program is *traced* by collecting information about the execution of the program in trace records. These records are then accumulated into a trace file which a user visualizes with VT.

tracepoint. Tracepoints are places in the program that, when reached during execution, cause the debugger to print information about the state of the program.

trace record. In PE, a collection of information about a specific event that occurred during the execution of your program. For example, a trace record is created for each send and receive operation that occurs in your program (this is optional and may not be appropriate). These records are then accumulated into a trace file which allows the Visualization Tool to visually display the communications patterns from the program.

U

unrolling loops. See *loop unrolling*.

US. See *user space*.

user. (1) A person who requires the services of a computing system. (2) Any person or any thing that may issue or receive commands and message to or from the information processing system.

user space (US). A version of the message passing library that is optimized for direct access to the SP High Performance Switch, that maximizes the performance capabilities of the SP hardware.

utility program. A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program.

utility routine. A routine in general support of the processes of a computer; for example, an input routine.

V

variable. (1) In programming languages, a named object that may take different values, one at a time. The values of a variable are usually restricted to one data type. (2) A quantity that can assume any of a given set of values. (3) A name used to represent a data item whose value can be changed while the program is running. (4) A name used to represent data whose value can be changed, while the program is running, by referring to the name of the variable.

view. (1) In an information resource directory, the combination of a variation name and revision number that is used as a component of an access name or of a descriptive name.

Visualization Tool. The PE Visualization Tool. This tool uses information that is captured as your parallel program executes, and presents a graphical display of the program execution. For more information, see *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

VT. See *Visualization Tool*.

X

X Window System. The UNIX industry's graphics windowing standard that provides simultaneous views of several executing programs or processes on high resolution graphics displays.

xpdbx. This is the former name of the PE graphical interface debugging facility, which is now called **pedb**.

Xprofiler. An AIX tool that is used to analyze the performance of both serial and parallel applications, via a graphical user interface. Xprofiler provides quick access to the profiled data, so that the functions that are the most CPU-intensive can be easily identified.

Bibliography

This bibliography helps you find product documentation related to the RS/6000 SP hardware and software products.

You can find most of the IBM product information for RS/6000 SP products on the World Wide Web. Formats for both viewing and downloading are available.

PE documentation is shipped with the PE licensed program in a variety of formats and can be installed on your system. See “Accessing PE Documentation Online” and “Parallel Environment (PE) Publications” on page 174 for more information.

This bibliography also contains a list of non-IBM publications that discuss parallel computing and other topics related to the RS/6000 SP.

Information Formats

Documentation supporting RS/6000 SP software licensed programs is no longer available from IBM in hardcopy format. However, you can view, search, and print documentation in the following ways:

- On the World Wide Web
- Online (on the product media and via the SP Resource Center)

Finding Documentation on the World Wide Web

Most of the RS/6000 SP hardware and software books are available from the IBM RS/6000 Web site at:

<http://www.rs6000.ibm.com>

The serial and parallel programs that you find in the *IBM Parallel Environment for AIX: Hitchhiker's Guide* are also available from the IBM RS/6000 Web site, in the same location as the PE online library.

You can view a book, download a Portable Document Format (PDF) version of it, or download the sample programs from the *IBM Parallel Environment for AIX: Hitchhiker's Guide*.

At the time this manual was published, the Web address of the “RS/6000 SP Product Documentation Library” page was:

http://www.rs6000.ibm.com/resource/aix_resource/sp_books

However, the structure of the RS/6000 Web site may change over time.

Accessing PE Documentation Online

On the same medium as the PE product code, IBM ships PE man pages, HTML files, and PDF files. To use the PE online documentation, you must first install these filesets:

- **ppe.html**
- **ppe.man**
- **ppe.pdf**

To view the PE HTML publications, you need access to an HTML document browser such as Netscape. The HTML files and an index that links to them are installed in the `/usr/lpp/ppe.html` directory. Once the HTML files are installed, you can also view them from the RS/6000 SP Resource Center.

If you have installed the SP Resource Center on your SP system, you can access it by entering this command:

```
/usr/lpp/ssp/bin/resource_center
```

If you have the SP Resource Center on CD-ROM, see the `readme.txt` file for information about how to run it.

To view the PE PDF publications, you need access to the Adobe Acrobat Reader 3.0 or later. The Acrobat Reader is shipped with the AIX Version 4.3 Bonus Pack and is also freely available for downloading from the Adobe web site at:

<http://www.adobe.com>

To successfully print a large PDF file (approximately 300 or more pages) from the Adobe Acrobat reader, you may need to select the "Download Fonts Once" button on the Print window.

RS/6000 SP Publications

SP Hardware and Planning Publications

The following publications are related to this book only if you run parallel programs on the RS/6000 SP. These books are not related if you use an IBM RS/6000 network cluster.

- *IBM RS/6000 SP: Planning, Volume 1, Hardware and Physical Environment*, GA22-7280
- *IBM RS/6000 SP: Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281

SP Software Publications

LoadLeveler Publications

- *LoadLeveler Diagnosis and Messages Guide*, GA22-7277
- *Using and Administering LoadLeveler*, SA22-7311

Parallel Environment (PE) Publications

- *IBM Parallel Environment for AIX: DPCL Class Reference*, SA22-7421
- *IBM Parallel Environment for AIX: DPCL Programming Guide*, SA22-7420
- *IBM Parallel Environment for AIX: Hitchhiker's Guide*, SA22-7424
- *IBM Parallel Environment for AIX: Installation*, GA22-7418
- *IBM Parallel Environment for AIX: Messages*, GA22-7419
- *IBM Parallel Environment for AIX: MPI Programming Guide*, SA22-7422
- *IBM Parallel Environment for AIX: MPI Subroutine Reference*, SA22-7423
- *IBM Parallel Environment for AIX: MPL Programming and Subroutine Reference*, GC23-3893
- *IBM Parallel Environment for AIX: Operation and Use, Volume 1*, SA22-7425
- *IBM Parallel Environment for AIX: Operation and Use, Volume 2*, SA22-7426

PSSP Publications

The following publications are related to this book only if you run parallel programs on the RS/6000 SP. These books are not related if you use an IBM RS/6000 network cluster.

- *IBM Parallel System Support Programs for AIX: Administration Guide*, SA22-7348
- *IBM Parallel System Support Programs for AIX: Command and Technical Reference*, GA22-7351
- *IBM Parallel System Support Programs for AIX: Diagnosis Guide*, GA22-7350
- *IBM Parallel System Support Programs for AIX: Installation and Migration Guide*, GA22-7347
- *IBM Parallel System Support Programs for AIX: Messages Reference*, GA22-7352

AIX and Related Product Publications

For the latest information on AIX and related products, including RS/6000 hardware products, see *AIX and Related Products Documentation Overview*, SC23-2456. You can order a printed copy of the book from IBM. You can also view it online from the "AIX Online Publications and Books" page of the RS/6000 Web site at:

http://www.rs6000.ibm.com/resource/aix_resource/Pubs

DCE Publications

You can view a DCE book or download a PDF version of it from the IBM DCE Web site at:

<http://www.ibm.com/software/network/dce/library>

Red Books

IBM's International Technical Support Organization (ITSO) has published a number of redbooks related to the RS/6000 SP. For a current list, see the ITSO Web site at:

<http://www.redbooks.ibm.com>

Non-IBM Publications

Here are some non-IBM publications that you may find helpful.

- Almasi, G. and A. Gottlieb. *Highly Parallel Computing*, Benjamin-Cummings Publishing Company, Inc., 1989.
- Bergmark, D., and M. Pottle. *Optimization and Parallelization of a Commodity Trade Model for the SP1*. Cornell Theory Center, Cornell University, June 1994.
- Foster, I. *Designing and Building Parallel Programs*, Addison-Wesley, 1995.
- Gropp, William, Ewing Lusk, and Anthony Skjellum. *Using MPI*, The MIT Press, 1994.

As an alternative, you can use SR28-5757 to order this book through your IBM representative or IBM branch office serving your locality.

- Koelbel, Charles H., David B. Loveman, Robert S. Schreiber, Guy L. Steele Jr., and Mary E. Zosel. *The High Performance FORTRAN Handbook*, The MIT Press, 1993.
- Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 1.1*, University of Tennessee, Knoxville, Tennessee, June 6, 1995.

- Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface, Version 2.0*, University of Tennessee, Knoxville, Tennessee, July 18, 1997.
- Pfister, Gregory, F. *In Search of Clusters*, Prentice Hall, 1998.
- Snir, M., Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The Complete Reference* The MIT Press, 1996.
- Spiegel, Murray R. *Vector Analysis* McGraw-Hill, 1959.

Permission to copy without fee all or part of Message Passing Interface Forum material is granted, provided the University of Tennessee copyright notice and the title of the document appear, and notice is given that copying is by permission of the University of Tennessee. ©1993, 1997 University of Tennessee, Knoxville, Tennessee.

For more information about the Message Passing Interface Forum and the MPI standards documents, see:

<http://www.mpi-forum.org>

Index

T

trademarks 162



Program Number: 5765-D93

IBM Confidential, Limited Rights Data



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.