Performance Tuning an Application

SunSoft
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No.: 802-3648-10
Revision A November 1995

Contents

Preface

This manual explains how to use the SPARCworks™/ProWorks™ Analyzer, a software performance-tuning tool for software application developers. It is used in conjunction with the Collector, a Debugger tool that gathers the performance data that the Analyzer processes. The Analyzer is an integrated component of the Manager, which also includes:

- Debugger
- FileMerge
- MakeTool
- SourceBrowser

Who Should Use this Book

This manual is written for devlopers who develop applications on Sun® workstations.

Before You Read this Book

This manual assumes you are familiar with the following:

- The appropriate operating system commands and concepts
- The windowing environment, particularly the use of the mouse to activate a window, select text, and click on buttons.

Note – The Analyzer and the Debugger Collector do not run under the Solaris™ 1.x environment.

From an installation point of view, each SPARCworks/ProWorks release has its own CD-ROM. Check the CD-ROM label and the *Product Notes* for release numbers.

From a usage point of view, almost all of the aspects of the SPARCworks and ProWorks Manager are the same. This includes functionality, behavior, usage, and features. For the very few details that are different, the documentation calls out those differences.

Operating Environment

See the appropriate README file (SPARCworks or ProWorks) for a description of operating environments in which the Analyzer runs.

How this Book is Organized

This manual is organized as follows:

Part 1—Overview of the Analyzer

- **Chapter 1, "Introduction to the Analyzer**," introduces the structure of the performance-tuning tools to familiarize you with the Analyzer.
- **Chapter 2, "Performance-Tuning Methods**," describes performance tuning and the performance-tuning process.

Part 2 — Basic Performance Tuning

- **Chapter 3, "Setting Up for Performance Tuning**," explains how to activate the Debugger and the tasks you must perform before you can begin collecting performance behavior.
- **Chapter 4, "Getting Started with the Collector**," describes how to activate the Collector and how to set it up for data collection.
- **Chapter 5, "Getting Started with the Analyzer**," describes how to activate the Analyzer and the displays you can use to view and examine performance data.

Part 3 — Reading the Displays

- **Chapter 6, "Overview Display**," describes the components of the Overview display.
- **Chapter 7, "Histogram Display**," describes the components of the Histogram display and the Cumulative Histogram display.
- **Chapter 8, "Address Space Display**," describes the components of the Address Space display.
- **Chapter 9, "Statistics Display**," describes the information contained in the Statistics display.

Part 4 — Advanced Performance

- **Chapter 10, "Doing More with the Collector**," describes additional and more detailed features of the Collector to use for specifying and collecting data.
- **Chapter 11, "Doing More with the Analyzer**," describes additional and more detailed features of the Analyzer to use for viewing and examining the performance data.

Part 5 — Appendixes

- **Appendix A, "Troubleshooting**," describes how to overcome problems with the Collector and the Analyzer. It also lists error messages, the possible causes for the error message, and solutions to correct the error.
- **Appendix B, "Data Collection Requirements for the Displays**," contains a table that lists all the displays and data types of the Analyzer and the type of performance data that you need to collect to view the displays.
- **Appendix C, "Experiment Record**," describes the components of the experiment record. It also describes how to move and delete files.
- **Appendix D, "Export Experiment File**," describes the contents of this file and also provides a sample file.
- **Appendix E, "Print File**," describes the contents of this file and also provides a sample file.
- **Appendix F, "Print Summary File**," describes the contents of this file.

What Typographic Changes and Symbols Mean

The following table describes the notational conventions and symbols used in this book

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `system% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`system% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*.
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.
♦	A single-step procedure	♦ **Click on the Apply button.**
Code samples are included in boxes and may display the following:		
%	C shell prompt	`system%`
$	Bourne and Korn shell prompt	`system$`
#	Superuser prompt, all shells	`system#`

How to Get Help

Four on-line help facilities are available:

- **AnswerBook Product—**is an on-line documentation tool that displays this manual in full, along with the other SPARCworks or ProWorks tools manuals. You can read this manual on-line and take advantage of dynamically linked headings and cross-references.

  To start the AnswerBook product, type: `answerbook`

- **Magnify Help™ Messages—**is a standard feature of the OpenWindows software environment. To access Magnify Help, place the pointer on the window, menu, or menu button, and then press the Help key.
- **Notices—** are a standard feature of the OPEN LOOK environment. Notices serve two functions. Some notices are prompts that inquire about whether you want to continue with a particular action. Other notices are precautionary in that they provide information about the end result of a particular action; they appear only when the end result of an action is irreversible.

Note – For detailed information about Notices, refer to the OPEN LOOK documentation listed in the Preface of this manual.

- **SunOS Manual Pages (man pages)—**provide on-line documentation about the command line utilities of the SunOS operating system. The Analyzer has the following command line utilities for batch jobs:

  er_mv(1)

  er_rm(1)

  er_print(1)

  er_export(1)

  er_mapgen(1)

  analyzer(1)

  To access the man pages, in a command shell type: `man` *utility_name*

  For example, to access the on-line documentation for `er_mapgen(1)`; type:

  ```
  man er_mapgen
  ```

  See the man pages `dbx(1)` for collecting data commands in batch jobs. See the man pages `debugger(1)`for collecting data commands using GUI.

Related Documentation

This manual is part of the SPARCworks/ProWorks document set. Other manuals in this set include:

- *Installing SunSoft Software on Solaris* (for SPARCworks)
- *SPARCworks/ProWorksTutorial*
- *Building Programs with MakeTool*
- *Debugging a Program*
- *Managing the Toolset*
- *Merging Source Files*
- *Browsing Source Code*

Part 1—Overview of the Analyzer

Introduction to the Analyzer 1

This chapter describes the Analyzer software performance-tuning tool, its components, and its graphical user interface (GUI).

This chapter is organized into the following sections:

Performance tuning applications is a two-step process that involves 1) collecting performance data using dbx or the Debugger, and 2) analyzing that data. The Analyzer is a set of performance-tuning tools that a software application developer can use for measuring, recording, understanding, and improving the performance of an application program. It provides a GUI for collecting and displaying the performance data of a target application. The Analyzer is designed for use by all software developers, regardless of whether performance tuning is the developer's main responsibility.

The Analyzer enhances the task of collecting data because different types of performance data can be collected and the data collection process can be controlled during the execution of the application.

1

Data analysis is more effective because the Analyzer collects many different types of performance data and provides displays for viewing the collected data in the analysis stage. In the data analysis stage, different hypotheses about the behavior of the application can be examined and focus given to the areas where performance problems are occurring.

In certain cases, the Analyzer assists in rebuilding a tuned application by creating a *mapfile*. The Analyzer provides an analysis feature that identifies an improved ordering for loading functions into the application address space. The application can then be relinked, creating an executable that executes with a reduced text working set size.

1.1 Solaris 2.x and 1.x

The Analyzer runs under Solaris 2.x only; it does not run under Solaris 1.x. Check the CD-ROM label for release numbers before you install.

1.2 How the Analyzer Works

The Analyzer is integrated with dbx and the Debugger. Performance data is collected using the Collector, which is a GUI tool in the Debugger. The Collector communicates with dbx which in turn collects data from the kernel under which the application is running. As the program runs, dbx writes the performance data to a file; this file is called an *experiment record*. The execution of the application while data is being collected is called the *experiment*.

When the experiment is completed, the Analyzer is used to examine the collected performance data that is contained in the *experiment record*. The data can be viewed in graphical displays on screen or as a summary report that can be sent to a printer. The `er_export` utility puts the data of the experiment record into ASCII format; the `er_print` utility prints the data of the current display to a file or to a printer. Figure 1-1 illustrates the Analyzer architecture and all its features.

If profile data is collected, then a mapfile, which has an improved ordering of functions in the application, can be generated. This mapfile can then be passed to the linker using the `-M` option. The linker then relinks the application using the ordering specified in the mapfile and produces a new executable application. For more information on mapfiles, see Chapter 11, “Doing More with the Analyzer.”

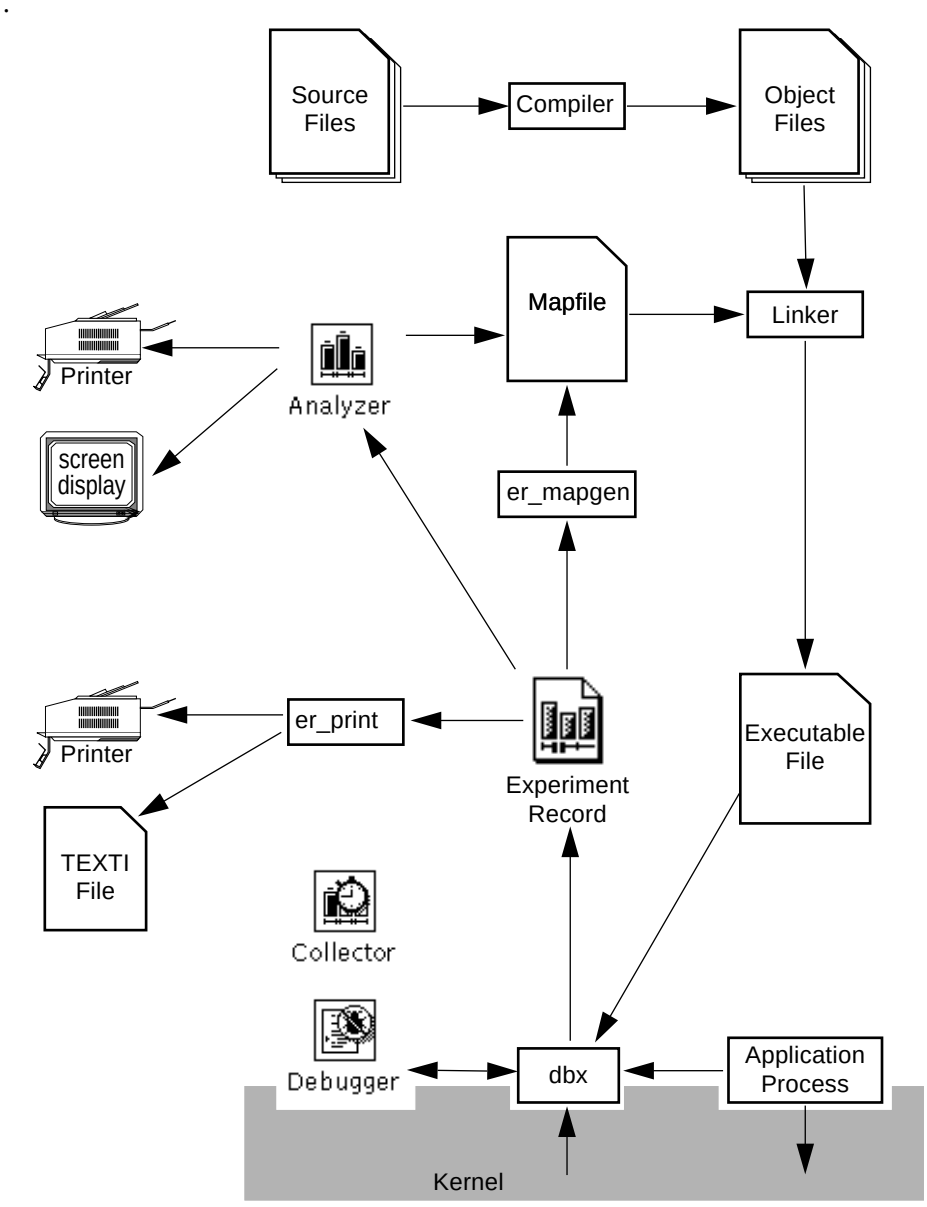


Figure 1-1 Analyzer Architecture

1.3 Graphical Overview

This section shows the icons, base windows, and menu buttons of the Analyzer. The Analyzer consists of two tools: the Collector (accessible in the Debugger) and the Analyzer.

1.3.1 Collector

The Collector is the tool you use to set up an application for data collection. The Collector gathers performance data during the execution of that application.

All data collection parameters are set in the Collector window. The Collector consists of text fields, settings, and buttons. Figure 1-2 shows the default settings of the Collector when it is first activated.

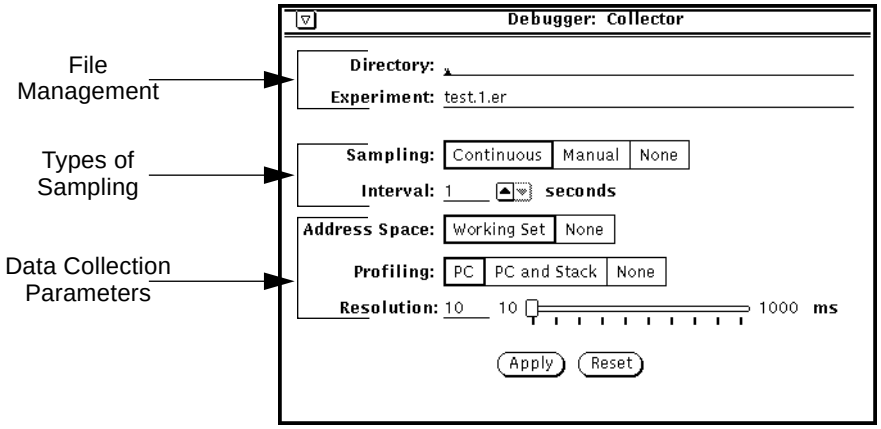


Figure 1-2 Collector Window

The Collector consists of three areas:

- **File Management**—consists of text fields where you type the experiment name and a directory name in which to store the experiment. The default directory is the current working directory.
- **Types of Sampling—**consists of settings for specifying the period of time to collect data.

- **Data Collection Parameters—**consists of settings for specifying the types of data to collect:
  - Use PC to generate performance data for histogram
  - Use PC and Stack to generate performance data for cumulative histograms

1.3.2 Analyzer

The Analyzer is the tool you use to examine the collected data. The Analyzer provides a GUI for selecting and displaying the collected data. The Analyzer processes the collected data into displays of your choice. You use these displays to examine the collected performance data.

The Analyzer consists of menu buttons, a text field, and a display pane. Figure 1-3 shows the Analyzer base window with the Overview display.

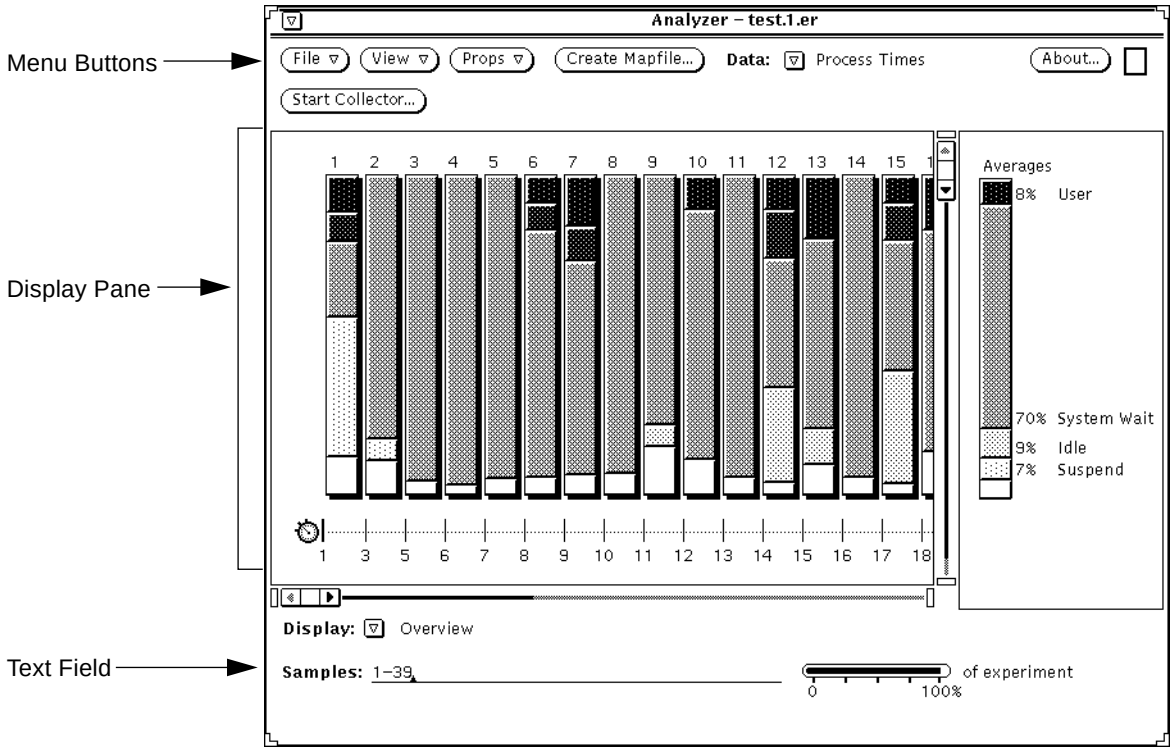
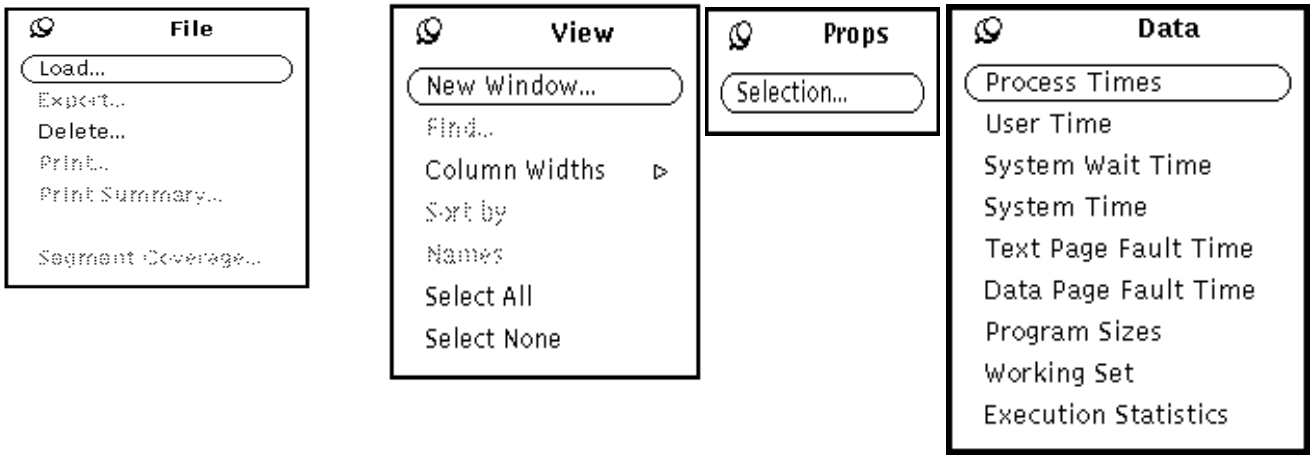


Figure 1-3 Analyzer Window

1.3.3 Menu Items

The menu buttons contain options for selecting and viewing performance data. The circled item on each menu is the default item. You do not have to open the menu to choose the default item, just click on the menu button.



1.3.4 About Button

The About button displays a pop-up window that gives the version number and the copyright information about the tool. Clicking SELECT on the Comments button in this window displays another window in which you can write any comments you may have about the tool and send the comments by email.

1.3.5 Drop Target

You can load an experiment from the FileManager into the Analyzer by selecting the experiment file, dragging it to the Analyzer's main window, and dropping it onto the Drop Target. If the file is not an experiment file, the Analyzer will not accept it. A message will appear at the bottom of the window informing you that the file is not an experiment.

1.3.6 Start Collector Button

The Start Collector button displays a dialog box, which tells you how to collect performance data by starting the Debugger and then selecting the Collector menu item from the Execution menu.

1.3.7 Displays

The Analyzer has four displays in which to view the collected performance data: Overview, Histogram, Address Space, and Statistics.

- **Overview—**is the default display of the Analyzer. It gives an overview of the performance behavior of the application. You can select individual samples and view the collected performance data that is specific to those samples.
- **Histogram and Cumulative Histogram—**provide information about the functions, modules, and segments of the application. You can alphabetically or numerically sort the functions, modules and segments, and you can search for specific functions, modules, and segments. For programs written in C++, you can display the function with or without its prototype.
- **Address Space—**shows what areas in the address space that the application occupies. The types of pages interaction shown in the Address Space display are: Modified, Referenced, and Unreferenced.
- **Statistics**—provides statistical information about the application attributes that are not displayed in any of the other displays.

1.4 Tasks Overview

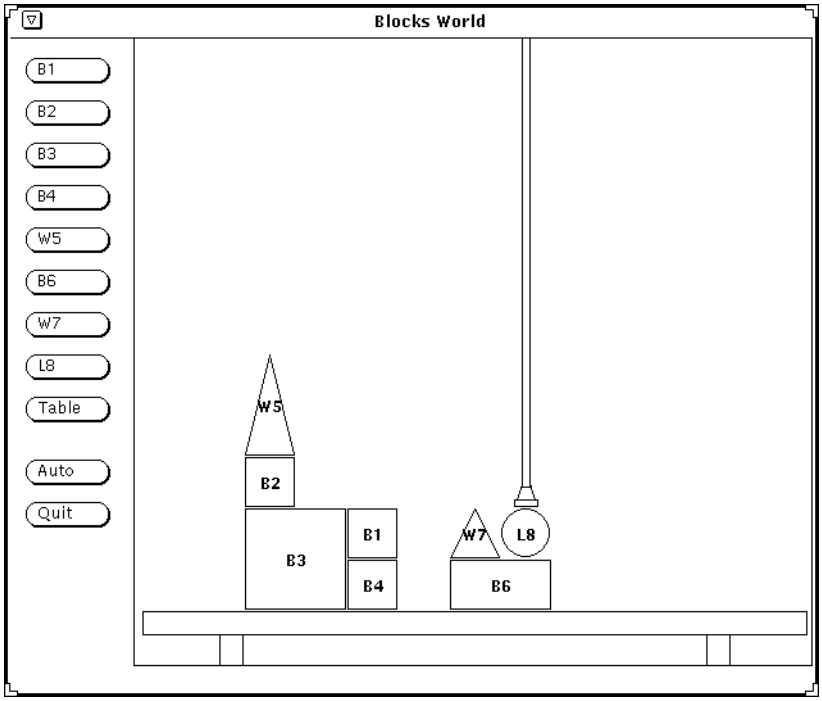
Table 1-1 describes performance tuning and the tasks for collecting and examining the performance data

Table 1-1 Performance-tuning Tasks

Task	Description
1. Develop the Model • Define performance objectives • Define application workload • Define hardware and software environment	A model is a necessary component to the performance-tuning process because it provides a basis on which to compare the behavior of the target application and also allows you to set up realistic goals for the target application to meet.
2. Build the Application • Compile application • Link application	No specific compilation procedures are required when compiling an application for performance analysis.
3. Collect the Data • Activate Manager • Activate Debugger • Activate Collector • Load the application • Set data collection parameters • Run application in Debugger	The Collector works with the Debugger. To begin data collection, first activate the Debugger from the Manager, then activate the Collector. Once the Collector window appears, load the application into the Debugger. Choose the data collection parameters in the Collector and run the application.
4. Analyze the Data • Activate Analyzer • Load experiment • View and examine data in the displays	Once you have collected performance data, you can examine the data in the displays provided by the Analyzer and identify performance problem areas. Change the application source code to eliminate the problems.
5. Evaluate Data • Compare data to model • Reorder application	Compare the results of the experiment to the model created in task one. If the results do not equal or surpass those of the model, then try reordering your application to reduce text working set size.

1.5 Example Program

This manual uses the Blocks[1] application in examples for data collection. Blocks is a C++ implementation of a Lisp application called Blocks World. The Blocks application defines several types of blocks (bricks, wedges, a ball, and a table) as subclasses of the basic class block. The Blocks application is used to move blocks on top of other blocks. The Blocks application installs automatically when you install SPARCworks.



To use the Blocks application in the examples of this manual, you need to compile the application as specified in Chapter 3, "*Building the Application*".

The Blocks application is in: `/opt/SUNspro/SW3.0.1/examples/Blocks`

1. The Blocks application used in this manual is derived from "Blocks World CLOS demo" from Chapter 21 of "Lisp", third edition, by Patrick Henry Winston and Berthold K. P. Horn, Copyright (c) 1989, 1984, 1981 by Addison-Wesley Publishing Company, Inc. and San Marco Associates.

Performance-Tuning Methods 2

This chapter describes performance tuning and the performance problem areas for which the Analyzer provides information. It also discusses the specific parameters to consider when performance tuning an application. This chapter describes how to set up a model to use for evaluating an application and introduces a specific process to follow when performance tuning

This chapter is organized into the following sections:.

2.1 What Is Performance Tuning?

A lot of time and effort goes into designing a program. A good program has been planned, well thought out, and tested. However, when the program is completed, many concerns and questions arise:

- Does the program meet its performance expectations?
- Does it perform efficiently?
- Is there any *dead* code?
- Does the application have good paging behavior?

An important concern is that the program meet its performance objectives in the most efficient manner possible. In most cases, the performance of the software is essential. For example, with a word processing application *faster is better.* A word processing application is considered to be inefficient when it does not immediately respond to keystrokes.

Many variables affect the performance of an application program; for example, the hardware and software environment on which the application runs. You may or may not have any influence in this regard, but what you are able to influence is the design of your program. Whatever the reasons for undertaking performance tuning, you want your program to be efficient, reliable, and fast. Performance tuning helps you to achieve these goals.

Note – Performance tuning and runtime checking are mutually exclusive processes. You can perform either one or the other at a time. The information you receive from tuning your application can be adversely affected if you perform runtime checking simultaneously.

Using the Analyzer to tune the performance of your application helps you to:

- Estimate the performance of your program.
- Identify the bottlenecks that are limiting performance.
- Identify where the code spends most of its time.

When you are convinced that performance tuning is a good idea, you may then ask the question *"How do I decide which application to tune?"* If a program meets its performance objectives, then it does not need to be tuned. However, most programs can benefit from at least one round of performance tuning.

2.2 The Model

Any measurement and evaluation that you perform on an application is meaningless unless you establish an ideal that you want the application to attain. This ideal is called a *model*. To begin performance tuning, any application program you must create a model. This model becomes the baseline for evaluating your application.

To create a model:

1. **Develop performance objectives for the application.**
   To develop performance objectives, outline the function(s) of the application and define the application's user.

- **Critical performance objectives—**For the application to be useful and efficient, it must meet certain *critical* objectives. One of these objectives often relates to data acquisition. For example, the purpose of the Blocks application is to move bricks, wedges, or a ball. A common, everyday task would be to place one object on top of another object. For the application to be useful, it should perform this function accurately, and in a short amount of time.
- **Non-critical performance objectives—**These objectives do not affect the quality of the program, but are necessary to promote ease of use. For example, it is convenient to have the Blocks application quickly display a message stating that the request to move an object is complete. Although this feature does not affect the qualitative behavior of the application, it is convenient for the user and contributes to the perceived performance of the application.

When setting performance objectives for any application:

- Consider the purpose of the application (what is it used for).
- Describe who will use it.
- Prioritize the functions that affect the qualitative behavior of the application and the functions that affect the user's perception of the application.

2. **Define the program workload.**

   The workload is the typical amount of time the application is used. For example, in word processing, responding to typed input is a critical feature because the application is so heavily used. For this activity, it is essential that a word processing application provides a good response time. For infrequently used features, such as converting from the Roman alphabet to the Cyrillic alphabet, you may not be so concerned with the response time of the activity.

   For a batch application, such as a compiler, you also need to understand its typical workload. For example, are the sizes of the programs to be compiled 100, 1,000, or 1,000,000 lines long? In these cases, the internal behavior and runtime of the compiler may be very different. In particular, runtime may not be a linear function of the number of lines in the source program being

compiled. Consequently, a compiler that is tuned to efficiently compile 1,000 line programs may be poorly tuned to compile 1,000,000 line programs.

3. **Define the hardware and software environment that runs the workload.**

   Hardware and software environments affect the performance of an application; a program that runs efficiently on one configuration might decrease in efficiency when run on a different configuration. When defining the hardware and software environments, the environment should be sufficient to run your application at its peak performance level. You need to consider what other applications are going to run on the workstation on which you run your application. For example, is the workstation also a print server, or is it a license server for a CAD/CAM or desktop publishing application?

After you have established these criteria, decide in what area you want to concentrate your performance-tuning efforts. The combination of steps 1, 2, and 3 assists you in deciding what to evaluate and measure when performance tuning your application.

2.3 The Process

Performance tuning is an iterative, interactive process. It is a cyclical process, whereby different performance improvements are hypothesized and tested. Performance tuning an application should be approached systematically to avoid the possibility of creating problems at the same time you are fixing problems. Setting up a systematic approach to performance tuning is easy to do with the Analyzer because it provides you with a visual means of examining areas in the application that might be creating problems. Instead of having to read many lines of code, you can view the collected data in the displays of the Analyzer. Some of the displays provide an overview of the application, while other displays focus on specific areas.

The Analyzer provides you with the resources to engage in a systematic approach when performance tuning. The following performance-tuning process is recommended. You might do some steps in parallel, others repeatedly, and others not at all. Figure 2-1 illustrates the performance-tuning process.

To conduct systematic data analysis:

1. **Compile source files and create an executable.**
2. **Set up the application executable file which also contains code for performance data collection. Run the application and collect performance data.**
3. **Analyze the data to identify the performance problem area and to hypothesize a change that solves the problem.**
4. **Rebuild the application with the hypothesized performance-improvement modifications.**
5. **Repeat steps 2, 3, and 4. The remainder of the process consists of repeating steps 2, 3 and 4 until the target application meets its performance goals.**

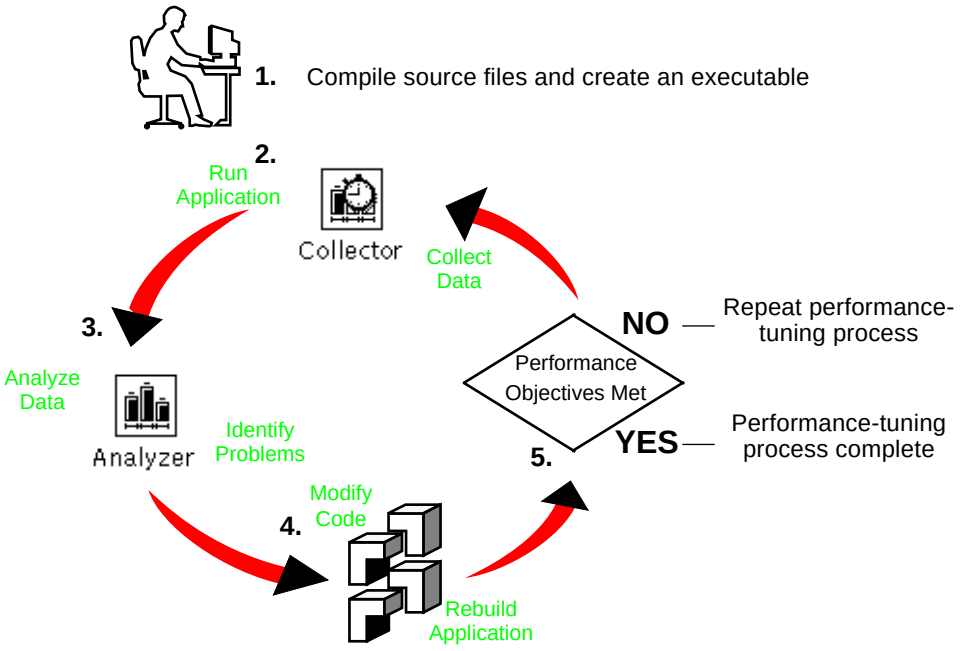


Figure 2-1 Performance-Tuning Process

2.4 Types of Performance Problems

In the most typical case, the objective of performance tuning is to make an application, or some operation performed by that application, execute faster. In other words, to reduce the elapsed time required to execute the application or operation. Note that elapsed time is the time perceived by the user. In the Analyzer, elapsed time is referred to as "process time", but other terms used to describe process time include:

- **Turnaround time—**is the complete execution time of a non-interactive or batch process.
- **Response time—**is the execution time of a single interactive operation.
- **Real time—**as opposed to "virtual time", which only includes time spent executing in the CPU.
- **Wall-clock time—**used in reference to the fact that process time is the time perceived by the user.

A secondary objective in performance tuning is to reduce the consumption of system resources, for example, CPU, memory, disk, or network bandwidth, by the application or operation. This allows the application to more easily share system resources with other applications; thus allowing both this application and the other applications to perform better. The techniques used to tune process time and process resource usage are essentially the same; only the metric used to gauge the level of success in the tuning is different.

The performance tuning methodology can be summarized as follows:

1. **Find the biggest performance bottleneck:**

   Determine what type of bottleneck it is. The most important types are CPU, I/O, and paging.

   Determine where the bottleneck exists in the application's code.

2. **Change the design or implementation to eliminate the bottleneck by changing the design:**

   to perform some operation more quickly, or

   to perform that operation less frequently.

For example, if the bottleneck is in a loop in the code, you could redesign the body of the loop to execute more quickly, or you could redesign the code to reduce the number of times the loop is executed.

3. **Repeat steps 1) and 2), fixing the biggest remaining performance bottleneck in each repetition, until you have achieved the performance objectives for your application.**

The Analyzer is designed to help you with the first of these steps, finding the performance bottleneck. Since every application is different, you must fix the bottleneck and decide when the tuning is completed.

The Analyzer's Overview display, described in Chapter 6, presents the information needed to identify the type of performance bottleneck that is affecting the application.

In this discussion, it is assumed that the application can be performing only one operation at a time. This is not always true; for example, one of the principle benefits of using threads in an application is that the process can perform multiple simultaneous operations (especially if it is running on a multiprocessor system). Another important exception is the use of asynchronous I/O, which allows multiple simultaneous I/O operations. While this description ignores these exceptions, performance tuning in the exceptional cases follows a similar methodology.

The overview display shows the different states that a process may be in at different times during its execution. The Sample Properties Window, Figure 6.4, lists nine states. In the first three states (User, System, and Trap) the process is executing in the CPU, specifically:

- **User—**The process is executing program code.
- **System—**The operating system kernel is executing a system call from the process.
- **Trap—**The operating system kernel is executing a trap, an automatic exception or memory fault. Important traps include page faults, register window overflows and underflows, arithmetic exceptions (for example, floating point overflow or divide by zero), and memory access errors (for example, accessing an unallocated memory address).

If one of these states is the dominant process state, then the application has a CPU bottleneck. See Chapter 2, "*Tuning CPU Bottlenecks*", for more information on how to handle this situation.

In the remaining states shown in the Sample Properties Window (Text Fault, Data fault, System Wait, Lock Wait, Suspend, and Idle) the process is not executing in the CPU, but is waiting for some reason. These categories are listed in the Sample Properties Window in an order approximately related to their usefulness, but it is easier to explain them here in a different order:

- **Suspend—**The process is not executing, usually because of some external activity. The most important reason for the process to be in this state is when the process is suspended by the debugger, especially so that the Collector can extract some performance data, but also when the process is suspended because it has hit a Debugger breakpoint. The process could also have been suspended by a user activity or by another process, for example, via a STOP or KILL signal. Finally, a process may also put itself in Suspended state, by executing the `lwp_stop()` system call.
- **Idle—**The process is ready to execute, but the CPU is not available. Thus, the process is in the kernel's ready queue.
- **Lock Wait—**The process is waiting for a lightweight process lock to be released. This time should be zero for any process not using the Solaris thread library. (Note: The SPARCworks/ProWorks Release 3.0.1 Analyzer does not support threaded applications.)

Suspend time and Idle time are like Trap time; they are essentially untuneable. The amount of Suspend time is usually determined by the behavior of the external activity, and cannot be reduced by tuning this application. Idle time occurs because of contention for the CPU by other processes; thus, time spent in Idle state is highly variable and cannot be reduced by tuning a single application. Thus, Suspend, Idle and Lock Wait states are presented by the Analyzer primarily for completeness.

- **Text Fault—**The process took a page fault when trying to load an instruction for execution and is waiting for that page to be loaded into memory.
- **Data Fault—**The process took a page fault when trying to load or store data and is waiting for that page to be loaded into memory.

For more information on how to handle a paging bottleneck, see Chapter 2, *"Tuning Paging Bottlenecks"*.

- **System Wait—**The process is sleeping in the kernel, but it is not in Suspend, Idle, Lock Wait, Text Fault, or Data Fault state.

While this appears to be a catchall state, System Wait state is actually the most important of the non-CPU states, because it includes time spent waiting for I/O. For more information on how to handle an I/O bottleneck, see Chapter 2, "*Tuning I/O Bottlenecks*".

As described in the sections on tuning CPU, I/O and paging bottlenecks, the Overview display is used to identify the type of performance bottleneck. The remaining Analyzer displays provide information on the location of the bottleneck in the source code or provide supplemental information to help in analyzing the bottleneck.

2.4.1 Tuning CPU Bottlenecks

The Overview display reports User, System and Trap states separately because the available performance tuning techniques vary according to the state. In the case of User CPU time, you have access to the source code to the application, so any source code modification is possible. In the case of System CPU time, the typical programmer cannot modify the design of the operating system kernel; thus you can only try to perform the operation less frequently, that is, reduce the number of system calls. This restriction is also true when the performance bottleneck is found to be in some system library or other library which you do not control. Although library execution time is included the User CPU state, you typically can only tune your application to use less of that library's services. Finally, the occurrence of a trap cannot be predicted from looking at the source code, nor can you tune the source code to deterministically avoid the occurrence of traps. Thus, time spent in the Trap state is rarely tuneable, and is presented by the Analyzer primarily for completeness.

If the performance bottleneck is User CPU time, use the User Time Histogram and/or Cumulative Histogram displays to identify the location of the bottleneck. The User Time Histogram display shows the amount of time spent executing each function in the program. The data can also be aggregated to show the time spent in different modules or segments of the program. If the Histogram display has identified a library function as the location of the bottleneck, you should use the Cumulative Histogram to identify the function(s) in your own code which have called the bottleneck function. For example, if the bottleneck were in one of the trigonometric functions, you probably do not have the option of tuning that function, but you can use the Cumulative histogram display to identify functions in your own code which

consume large amounts of User CPU time via calls other functions, including the trigonometric functions. See Chapter 7, *"Histogram Display"*, for a more complete description of these displays.

If the performance bottleneck is System CPU time, use the System Time Histogram and Cumulative Histogram displays to identify the location of the bottlenecks. In this case, the Cumulative Histogram should be used to identify the functions in the user code which are the key consumers of system time via system calls.

As described above, it is very difficult to tune Trap time. An exception is the rare case in which register window overflows and underflows are a major bottleneck. In this case, eliminating recursion and/or using more function inlining may be helpful. The Analyzer does not provide any displays for localizing a register window bottleneck. For example, suppose function A calls function B which then calls function C, and that a register window overflow occurs on the call from B to C. If this information were reported, it would appear that the way eliminate the overflow would be to inline the call from B to C. On the contrary, the overflow can be eliminated by eliminating either of the calls: in other words, by, inlining the call from A to B has the same effect on register window overflows as inlining the call from B to C.

2.4.2 Tuning I/O Bottlenecks

Other than CPU bottlenecks, I/O bottlenecks are probably the next most liekly cause of performance problems. If the Overview display shows System Wait time as the most common process state, your application may have an I/O bottleneck. Unfortunately, it is difficult for the kernel to identify the cause of these waits. While this state includes time spent waiting for I/O, it also includes other conditions which may not be tunable. An important example is waiting for user input in an interactive application; in this case, the bottleneck resource is the user, not the application.

To identify and tune an I/O bottleneck, use the System Wait Time Histogram and Cumulative Histogram displays. The Histogram will show that the time is being spent in some system library, for example, `_read()`. The Cumulative Histogram can be used to identify the function in your code which is (directly or indirectly) invoking the system service. You can usually identify whether this is a tunable I/O bottleneck from this function. For example, time spent waiting for user input in an interactive GUI-based application appears in the window system's notifier loop routines.

Please note that extensive caching and performance optimizations in Solaris often lead to I/O behavior that is different than one would casually expect. For example:

- Solaris makes extensive use of memory as a cache for disk files. If an application reads a block from a file, and that block is already resident in memory, there is no System Wait time. The `read()` system call actually consumes System CPU time copying the block into the process' address space, but in this case would not incur any System Wait time.
- When Solaris detects sequential access to a file, it may asynchronously preload blocks from that file. In this case, the first few reads from a file may incur System Wait time, but subsequent reads may not incur any.
- Solaris typically does asynchronous file writes. If an application writes a block to a file, the kernel does not write the data out immediately, so there usually is no System Wait time. Thus, if an application performs a small number of file write operations, it often incurs no System Wait time. On the other hand, if the application quickly performs a large number of writes, the system will eventually run out of free pages in memory, and a write operation will have to wait until the free page list is built up again. Viewing this behavior in the Overview display, you will initially see System CPU time, to copy the data to the kernel's buffer, but little or no System Wait time, then a sudden increase in System Wait time, even though the logic of the application may be a homogeneous series of write operations.
- Solaris provides a special file system `tmpfs`, which uses swap space as its backing storage. `tmpfs` has the characteristic that data is only written to disk when the kernel needs to free a page of memory. Thus, if there is little memory contention, I/O operations on a file in a `tmpfs` file system may incur no System Wait time at all, but the same application may suffer from an I/O bottleneck when the memory contention is high.

The net effect of these and other optimizations is that I/O performance is very dependent on external factors, including the type of file system, memory size, network load, and other activity on the host while the performance data is being collected. The environment must be carefully controlled when collecting performance data for an I/O bottleneck, and operating system optimizations must be carefully considered when interpreting the data.

2.4.3 Tuning Paging Bottlenecks

The first step in tuning a paging bottleneck is to determine whether you actually have one. If the sum of the Text Fault Time and Data Fault Time in the Overview display is large, that is a hint that you might have one. But time spent in page faults is difficult to tune because memory is shared among all processes, and the frequency of page faults in each process is dependent on amount of memory available to that process. In addition, the time required to service a page fault is affected by the workload on the disk and/or network. Therefore, looking at Text and Data Fault Time is not conclusive evidence. Ideally, you should run a series of carefully controlled experiments, varying the amount of memory available to the process in each case, and then look at the curve relating Process Time to memory size. If the Process Time for large memory sizes meets your performance objectives, but the Process Time for smaller memory sizes exceeds your objectives, then you have a paging bottleneck. Note that you should have different performance objectives for different memory sizes.

Since the process described above can be quite laborious, the Analyzer provides data which can help you estimate the relationship between Fault Time and memory size more easily. The Execution Statistics display, described in Chapter 9, shows data for the total, maximum, minimum and average working set size for your process, measured in pages. It also displays these metrics separately for text pages and non-text pages. Some of the more useful statistics are:

- **Total Working Set Size** - the number of distinct pages touched by the process during its execution. Multiply this by the page size to get the total amount of memory that would be needed to hold the entire process.
- **Total Text Working Set Size** - This statistic provides a conservative estimate for the minimum number of page faults that this process will incur, under the assumption that each text page must be brought into memory via a page fault. For small processes, Solaris preloads some pages; so in this case the process will take fewer page faults. In addition, Solaris does extensive sharing and caching of text pages, which also reduces text page faulting.
- **Maximum Working Set Size** - The working set size of a single sample is the number of pages touched during that sample. The maximum working set size is the largest working set among all samples. Thus, if the memory available to the process is at least as large as its maximum working set size, then that process' working set can always be memory resident. Note that

this does not imply that the process would not have any page faults; as the set of pages in the working set changes, new pages entering the working set will cause page faults.

- **Minimum Working Set Size** - This is the smallest working set among all samples. It gives an approximate lower bound on the minimum amount of memory needed by the process. If the process' memory allocation is lower than this minimum, then it never has enough memory for its current working set, and therefore is always page thrashing.
- **Average Text and Non-text Working Set Sizes** - These statistics indicate whether the process' working set size is dominated by text pages or non-text pages.

In addition, the Program Sizes display shows the size in bytes of the functions, modules and segments of the process.

- **Program Size Segment Histogram** - shows the size in bytes of the process' text segments. To compute the fraction of the process' text that is actually executed, divide the total text working set size by this figure.

In general, page fault time is very difficult to tune, because memory is a shared resource both with other processes and within a process. Thus, one part of the program may be making poor use of virtual memory, but the page fault may occur in a different part of the program. Here are some hints about possible tuning strategies:

- **Poor locality of text references** - If the process is spends a large part of its time in Text Fault state, it may have poor locality in its text segment. Look in the Text Page Fault Histogram display. In the typical case, you cannot modify the structure of shared libraries, so you should use the Segment unit to focus on the main program or on shared libraries that you control. If the text fault time is concentrated in one of your segments, and the segment is large, then you should follow the procedure described in Section 11.3, "*Reordering Your Application*", to reduce your application's text working set size. Note that this procedure can be followed even when the paging program primarily causes data faults, since reducing the text working set size makes more pages available for data.

You can also use the Working Set display, described in Chapter 8, to get a better understanding of the usage behavior of the process' text working set. In this display, the Page Properties window lists all functions contained in the

selected page. If a referenced page contains a set of unrelated functions, then much of the memory occupied by this page may be taken up by unused functions.

- Poor locality of data references - A poor algorithm for processing large data structures can have disastrous paging behavior. The classic example of this is to process a large Fortran array in the wrong order. For example, if A is a 10,000 by 10,000 array of integers, A(1,1) and A(2,1) are located in adjacent words of memory, but A(1,1) and A(1,2) are 40KB apart, on different pages. Problems of this sort can be identified by looking at the Data Page Fault Time Histogram or Cumulative Histogram displays. If the process spends much of its time in Data Fault state, and the time is concentrated in one or a few functions, you should examine the large data structures referenced by these functions and the algorithms used to process the data structures.

The Working Set display may also provide information which is helpful in understanding data page fault problems. For example, loads and stores to memory mapped files may cause page faults. References and modifications to memory mapped files are shown in this display.

- Heap problems can be examined by looking at the Working Set Display. The heap is typically the third segment in the display, identified by the name `<pathname>/<executable> bss` in the Page Properties or Segment Properties window. In contemporary applications it is quite common for applications to use the heap very heavily, and the heap is often a major contributor to a process' working set size. If heap size does appear to be a performance bottleneck, then you should consider modifying your data structures to be more space efficient or implementing a partitioned heap system, to cluster related heap structures together onto pages, thus improving the locality of reference to these data structures.
- Writing to text segments - The text pages of dynamically linked libraries should be referenced only (executed or read), **never** modified. Solaris allows continued execution in the face of modified text pages by doing a copy-on-write operation. This situation can be detected by looking for modified text pages in the Working Set Display. While this situation does not necessarily cause your application to execute incorrectly, it does have a performance penalty because the copy on write operation has a cost, and the copied page occupies swap space and is not sharable with other processes.

Part 2 — Basic Performance Tuning

Setting Up for Performance Tuning 3

This chapter describes the tasks you need to perform before you can begin the performance-tuning process of data collection and performance analysis. It also describes the preliminary preparation for the target application.

This chapter is organized into the following sections:

3.1 Building the Application

Compile and link your application using a SPARCcompiler for SPARCworks or a ProComplier for ProWorks.

To prepare your application for data collection:

- **Compile and link the application using a SPARC/ProCompiler.**

The following flags are recommended for building your application:

- Use the `-g` option if you want to view a module histogram. This option ensures that the data is accurately displayed. See Chapter 7, “Histogram Display,” for details on module histograms.
- Use `-xF`, `-g` options if you want to reorder your application. Reordering is done using the Create Mapfile feature of the Analyzer. See Chapter 11, “Doing More with the Analyzer,” for details on reordering.

Note – If you want to collect profile data on an application that is statically linked (`-dn`), then you need to link with the `libcollector.o` module. The `libcollector.o` module is in `/opt/SUNWspro/SW3.0.1/lib`. If you have chosen to install the software in a different location, the `libcollector.o` module is located in the location you specified during installation.

3.2 *Starting the SPARCworks/ProWorks Manager*

The SPARCworks/ProWorks Manager is a desktop tool for managing all the programming tools. It is a visual organizer that provides accessibility for starting and quitting tools. Figure 3-1 shows the Manager.

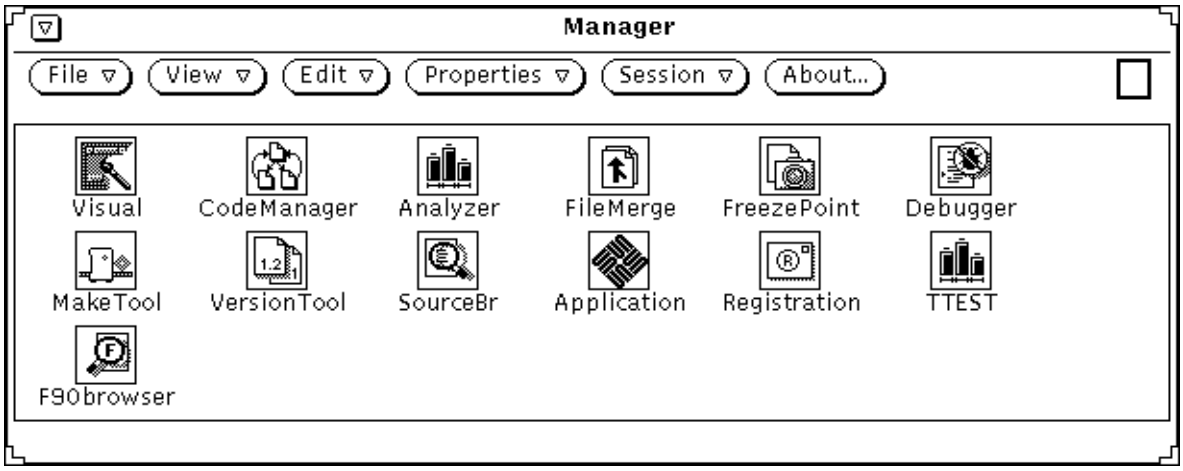


Figure 3-1 SPARCworks Manager

To start the Manager:

- **Type:** `sparcworks&`
- **Type:** `proworks&`

Refer to the manual *Managing the Toolset* for detailed information about using the Manager.

3.3 Starting the Debugger

To access Collector, you must first start the Debugger.

To start the Debugger, do one of the following:

- **From the Manager:**
  Double-click on the Debugger icon.
- **From the keyboard:**
  Change to the directory that contains the application that you want to debug and type: `debugger&`

The Debugger window appears (see Figure 3-2). You can either select commands from the Debugger window or type commands from the keyboard.

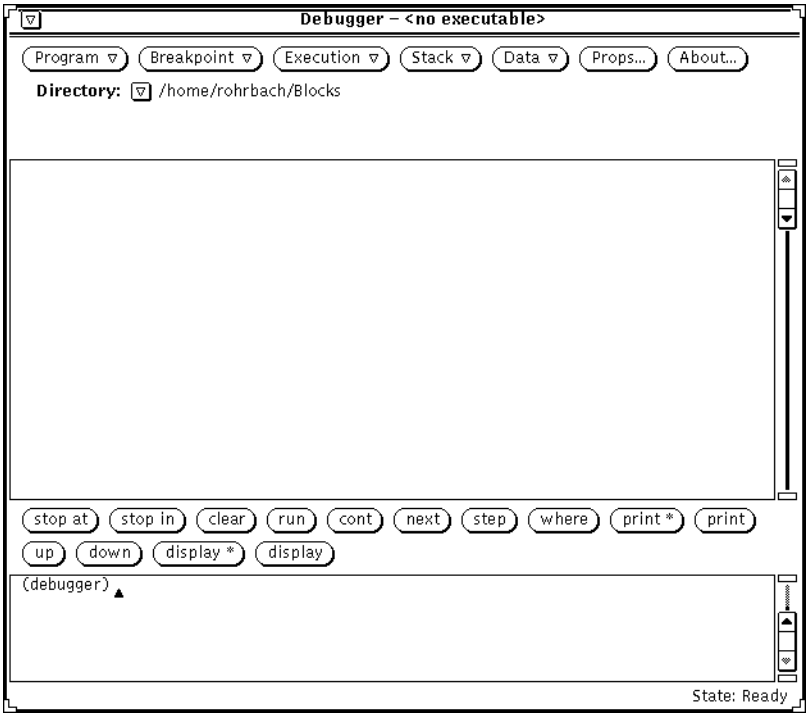


Figure 3-2 Debugger Window

3.4 Starting the Collector

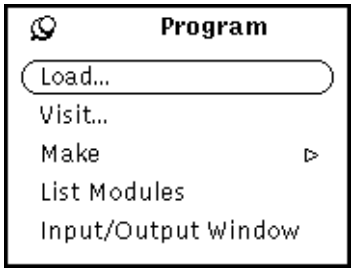
To start the Collector::

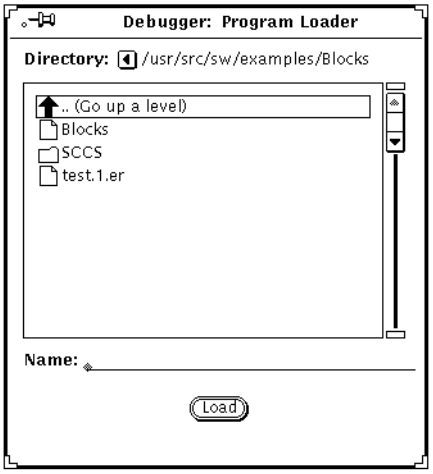
- **Choose Collector from the Execution button of the Debugger.**

3.5 Loading the Application

Once the Debugger window appears, load the source code of the application that you want to performance tune. You can load the source code in one of three ways:

- **In the Debugger window:**
  Choose Load from the Program menu button; the Program Loader window is displayed. Double-click on the directory and then double-click on the application name listed in the scrolling list. You can also type the directory and application name in the Name field and click on Load or press Return.





- **In the Debugger command line.**
  Type `debug` and the name of the executable of the application.

- **In the File Manager window:**
  Select the icon for the application and drag it into the Debugger window.

3.6 Alternative Methods

As an alternative method, you can simultaneously activate the Debugger and load an application with the following command:

- **Type:**
  `debugger` *application_name*

3

Getting Started with the Collector 4

This chapter describes the components of the Collector and how to activate this tool. It also describes the types of sampling and the data collection options offered by the Collector.

This chapter is organized into the following sections:

Note – You cannot collect profile data on a set user ID or a set group ID application. To collect data on this application, your user ID and group ID must match the ID's of the application.

For example, if you want to collect profile data on `Blocks`, then your user ID must match `evelynl`.

```
-r-sr-xr-x   1  evelynl   52952  Apr 23 10:36 Blocks
```

4.1 Activating the Collector

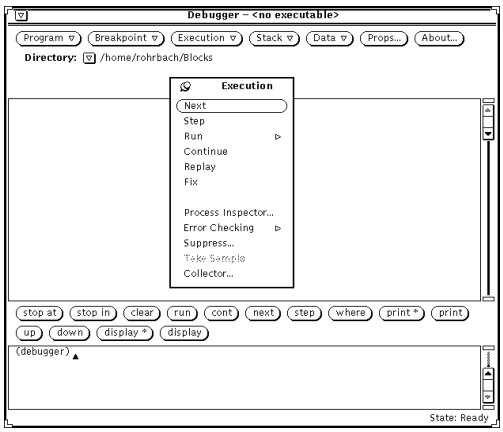
The Collector is the tool you use to prepare an application for data collection. During the execution of an application, the Collector gathers performance data and writes the data to a file. The experiment is the execution of the application while data is being collected. The collected data is called an *experiment record.*

Note – The Collector *cannot* be used to collect performance data if runtime checking (RTC) is enabled in the Debugger. RTC is enabled either by calling the Debugger with the `-C` command line option, or by setting the SW_RTC environment variable.

A single session can generate multiple experiment records; however, the *experiment record* can only contain data from one Collector session. Each experiment contains one run of an application. You cannot have multiple runs of an application in the same experiment record.

To activate the Collector:

- **Choose Collector from the Execution button menu in the Debugger window.**
  See Chapter 3, "Setting Up for Performance Tuning," for alternative methods for activating the Debugger. Note that the Collector is disabled if you are running a multithreading program.



The window opens with default settings for the experiment and samples. You do not need to change any of the settings or fields in the Collector window in order to start data collection. You can change any of these settings when the default settings do not satisfy your requirements. The settings can only be changed when the application you are debugging is stopped.

The Collector window consists of two areas (see Figure 4-1).

- Experiment
- Sample

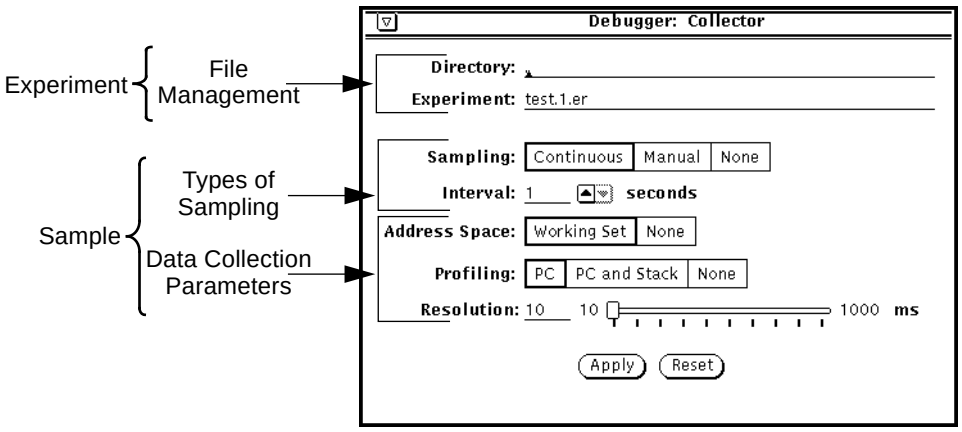


Figure 4-1 Collector Window

4.1.1 Experiment

The directory often shows ".", the current directory, as the default. An experiment consists of a set of samples taken on an application. In the experiment, you execute the application, gathering performance data with the Collector and storing the performance data in an *experiment record.* Performance data is collected in one or more samples, each of which is a measurement of a different period of time during the execution of the application. The experiment can contain one or more samples.

The experiment area of the Collector concerns itself with routine file management (Figure 4-1). You must name the experiment and specify a directory in which it is to be stored. See Section 4.5, "Storing the Collected Data," on page 4-43.

4.1.2 Sample

A sample contains data collected over a specified period of time during the execution of the application. It provides information about the performance behavior of the application. You can take one sample or multiple samples.

Multiple samples are useful when the performance of the application varies over time. Instead of examining the aggregate behavior of the entire execution of the application, you can take multiple samples to examine the performance behavior at specified intervals of the application execution.

A sample may include data describing:

- System calls
- Resource consumption
- Referenced and modified pages in the address space

The Sample consists of two areas (see Figure 4-1):

- **Types of Sampling—**consists of settings for specifying the period of time to collect data. Each sample contains information on the target application over a specified period of time. See “Choosing the Type of Sampling” on page 4-39.
- **Data Collection Parameters—**consists of settings for specifying the types of data to collect. Overview is the default data collection parameter. See “Choosing the Data Collection Parameters” on page 4-41.

4.2 Setting Up the Collector

To collect performance data, you must define the following:

- Type of sampling to perform
- Type of data to collect (data collection parameters)
- Directory and experiment name

The default settings are shown in Figure 4-1. The default settings are recommended for the first round of data collection because they provide a good overview of the application behavior. You can change the default settings, but you may want to wait until you have examined the collected data with the Analyzer, when you have more insight as to where the performance problems are occurring.

When you specify settings other than the default settings, you must click on the Apply button in the Collector window to complete and apply your settings.

To Apply or Reset the settings:

- **Click on the Apply button located at the bottom of the Collector window to apply settings.**

If you decide to use the default settings at a later point in the experiment, then you must reselect the default settings and click on the Apply button. Once the initial default settings are changed, any selection of them at a later point in the experiment are treated as new settings and they must be reapplied.

- **Click on the Reset button to restore the settings that existed at the last apply.**

4.3 Choosing the Type of Sampling

Performance data is collected through the process of sampling. Each sample contains information on the target application over a specified period of time. You can control the way in which samples are taken by choosing one of the sampling settings in the Collector window.

The default setting is Continuous samples every one second.

The sampling settings of the Collector are:

- Continuous
- Manual
- None

4.3.1 Continuous Sampling

Continuous sampling allows you to specify, in intervals of seconds, when to take samples. To specify the interval, enter the number of seconds, or use the increment / decrement buttons in the Interval numeric field. The valid range for the interval is 1-60 seconds. For example, if you choose a 2-second interval, and the application executes for 10 seconds, then you end up with a total of five samples.

Choose Continuous sampling for an overview of the application. This sampling mode provides a uniform view of the application behavior. Use it when you are:

- Unfamiliar with the behavior of the application
- Undecided about, or have not defined, specific performance concerns
- Unsure about what portions of the application execution on which to concentrate your performance-tuning efforts

After you have collected some samples, use the Analyzer to examine the data. If you need more information in your sample, then choose Manual sampling.

4.3.2 Manual Sampling

Manual sampling allows you to control the interaction between the execution of the application and the data collection. You can specify exactly when to begin a new sample. Use this sampling mode when you want to correlate the samples with real time events for feedback as to what the application is doing during that specified time period. This mode is especially useful when the application provides a visual response, either in the form of graphics or text.

Choose manual samples to collect data on a more specific aspect of a portion of the application. When you choose Manual sampling, the Take Sample button on the Debugger menu becomes selectable. Choosing the Take Sample button automatically terminates one sample and starts a new sample. The Take Sample button is a choice item located in the Execution button menu of the Debugger (Section 4-1).

4.3.3 No Sampling (None)

None stops the data collection process. This mode of sampling is useful when there is a particular period of time in the application execution about which you do not want to collect data, and you do not want any of this information in the samples.

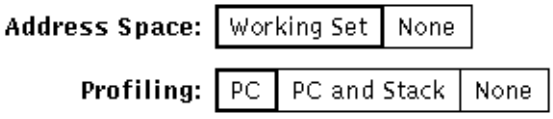
For example, suppose Blocks has a long initialization period and you are not interested in collecting performance data on this portion of the tool. If you select None during the initialization period of the application execution, then data is not collected.

When None is selected, all other panel items are not selectable. The panel items become a very light shade of gray to indicate that they are inactive. During this time, you cannot change or set any parameters.

Note – For more information on advanced methods of sampling, refer to Section 10.1, "Setting Breakpoints To Control Data Collection," on page 10-110.

4.4 Choosing the Data Collection Parameters

The Collector window provides choices for different types of performance data to collect. You control the data to collect for the experiment by choosing one of the data collection parameters in the Collector.



Default settings are None for Address Space and Profiling.

The following are the data collection parameter settings:

- Overview (default, not a selectable item)
- Working Set
- PC (program counter)
- PC and Stack

Note – Refer to Chapter 10, "Doing More with the Collector," for detailed information about data collection parameters.

4.4.1 Overview Data

Overview is the default data collection parameter of the Collector. Although it is not visible as a selectable item, Overview data is automatically collected. You can specify additional combinations of performance data to collect.

Overview is composed of resource usage and process real time performance data, which are useful for quickly identifying performance bottlenecks. It is useful for understanding time-variant performance behavior and for

understanding the types of performance bottlenecks. Although it generates the lowest amount of overhead for data collection, it provides the least information. Use only Overview data when you are:

- Unfamiliar with the behavior of the application
- Undecided about, or have not defined, specific performance concerns
- Unsure about what portions of the application execution on which to concentrate your performance-tuning efforts

Use Overview data for the first round of data collection. This choice provides a good first-time look at the application performance problem areas. After you have collected some samples, use the Analyzer to examine the data. Once you have identified where the performance problems are occurring, you can again run data collection using one of the other, more detailed modes, such as Working Set, PC, or PC and Stack. The other data collection parameters obtain more information in which to isolate the specific performance bugs in the application design.

4.4.2 Working Set Data

The Working Set data collection parameter collects data that assists you in answering questions such as:

- How does the working set change over time?
- Is the application using pages efficiently?
- Which pages are accessed during a particular operation?

Working Set data represents the process state address space as a series of segments, each of which contains a number of pages. The data collected describes the status of each page, and whether it was referenced or modified. If you select None, then Working Set data is not collected.

4.4.3 PC Data

PC (program counter) sets the data selection mode for PC profiling. This mode provides information with which to do a higher level of analysis. Profiling data represents a statistical sampling of where the application is during its execution. The data collected identifies where the application spends most of its time, and how long it spends on each function. The profiling snapshots use

a default profiling clock rate value of 10 milliseconds. The PC location is recorded in each profile snapshot**.** For detailed information about the profiling timer clock, see “The Profiling Timer” on page 10-115.

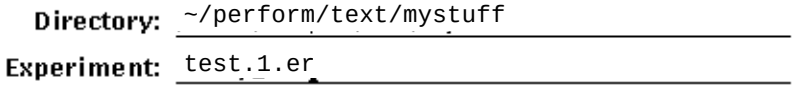
4.4.4 PC and Stack

PC and Stack sets the data selection mode to collect return function addresses on the stack. It provides a more detailed analysis than the PC mode. Use this mode when the application has a hierarchical (very layered), modular design.

For example, if function `aa` calls functions `bb` and `cc`, then collecting only PC data provides information about the execution time in function `aa`. Collecting PC and Stack data provides information about the execution time in the function `aa` as well the execution time spent by `bb` and `cc`.

4.5 Storing the Collected Data

An experiment consists of a sample or set of samples taken on the application.



To create an experiment, you must do the following:

- Name the experiment
- Specify a directory in which to store the experiment

4.5.1 Experiment Directory

The experiment is saved to the current working directory, which is the directory from which you activated the Debugger. You will see “.” when the current directory is the default.

You can use the default directory or create your own, more descriptive directory name in which to save the experiment. To specify a different directory, enter the new name in the Directory text field. In either case, whether you use the default directory or specify another directory, when the Collector is first activated, the default directory is displayed in the Directory text field.

The experiment must be saved to a directory. If you delete the default directory name and forget to enter a new name, then you are prompted to type a directory name in the directory text field. You must enter a name before you can start data collection. The default directoyr is "." (dot).

4.5.2 Experiment Name

The default experiment name is `test.1.er`. The number 1 is a version number. The version number is automatically incremented each time you create a new experiment to give the experiment a unique identifier. If you choose not to use the default experiment name and assign a different name to the experiment, then a version number is not appended to that name. In this case, you no longer have the version number to uniquely identify the experiment, so you must change the experiment name before every time you create a new experiment. If you do not change the name, then an error message is displayed.

You can also choose an experiment name that is modeled after the default name by appending `.1.er` to the name. If you append `.1.er` to the name, then the Collector increments the version number every time a new experiment is created. When the Collector increments the version number, it never overwrites an existing file. If there is another file by the same name, then the version tag is incremented each time you run the application. For example, suppose you changed the experiment name from the default name of `test.1.er` to `blocks.1.er`. If you run Blocks a total of three times, then the three experiments are named `blocks.1.er`, `blocks.2.er`, and `blocks.3.er`.

An experiment creates a *hidden directory* that contains various files. See Appendix C, "Experiment Record," for information about the *experiment record* files and how to move and delete them.

4.6 Collecting Performance Data

After you set up the Collector (choosing the type of sampling, the data collection parameters, and specifying a directory and experiment name) you can begin collecting performance data.

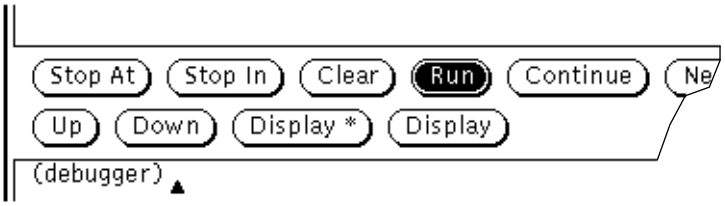
To collect performance data, you must do the following:

- Run the application in the Debugger

- Define the portion of the application to measure

To run the application, do one of the following:

- ♦ Type `run` in the command window of the Debugger.
- ♦ Click on the Run button in the Debugger window.



Note – Data collection starts after all the shared libraries are loaded. Performance data is not collected on the part of the application which loads in the shared libraries. The displays of the Analyzer do not show data for that part of the application.

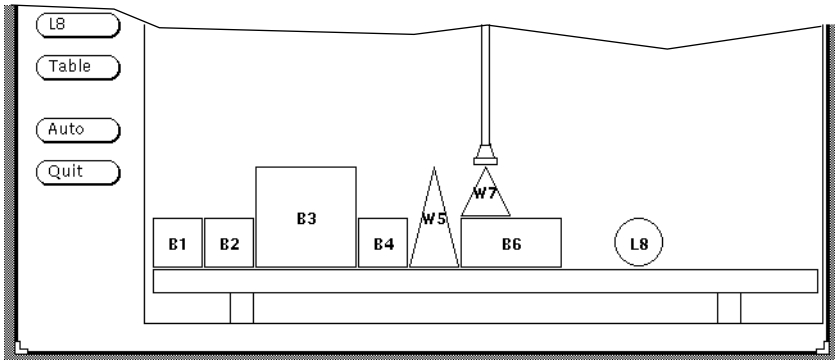
After you run the application, define the portion of the application to measure.

For example, suppose you want to measure moving a wedge on top of a brick in the Blocks application.

To start the data collection process:

1. **In the Debugger window, click on Take Sample in the Execution menu.**
2. **In the Blocks window, Click on the W7 button, followed by the B6 button, which moves wedge 7 on top of brick 6.**

3. **Click on Take Sample.**
   You get a sample that provides performance data on the move action of placing a wedge on top of a brick.



4.7 Closing and Quitting the Collector

You can close the Collector window while it is still performing data collection on an experiment, or when the data collection process is complete. The Collector icon has two modes:

- Run
- Suspend

4.7.1 Run Icon

When you close the Collector window during data collection, the icon displayed is an animated icon. The clock hands rotate continuously to indicate that data collection is in progress.

The animation of the icon can be turned off.

To turn off the animation:

- **Choose Debugger Props from the Props button of the Debugger window**
  The Properties window contains a toggle button for Collector Icon Animation (on/off).

  When animation is turned off, the hands of the clock do not rotate; however the time tick marks and the clock hands will be visible when data is being collected.

4.7.2 Suspend Icon

When you close the Collector window and the data collection process has been stopped or suspended, the icon is displayed as a clock without time tick marks or clock hands to indicate that data collection is inactive.

4.7.3 Quitting the Collector

If you decide to quit the Collector window while data collection is in progress, then the Collector stops collecting data at that point.

To quit the Collector, you can use one of the following methods:

- **From the Collector window, choose Quit.**
- **In the Debugger window, type:** `collector quit`

Getting Started with the Analyzer 5

This chapter describes the components of the Analyzer and how to activate this tool. It also describes the performance analysis options offered by the Analyzer.

This chapter is organized into the following sections:

5.1 Activating the Analyzer

The displays provided by the Analyzer allow you to view the data during the analysis stage of performance tuning. In the analysis stage, you have the opportunity to examine different hypotheses about your application behavior and to focus on the areas where performance problems are occurring.

The Analyzer provides analysis options such as:

- Viewing the data of interest in a variety of displays
- Exporting the collected data into a format that can be read by other programs
- Generating a summary report of the collected data

To activate the Analyzer:

- **Double-click on the Analyzer icon located in the Manager.**
  The Analyzer window is displayed as shown in Figure 5-1.

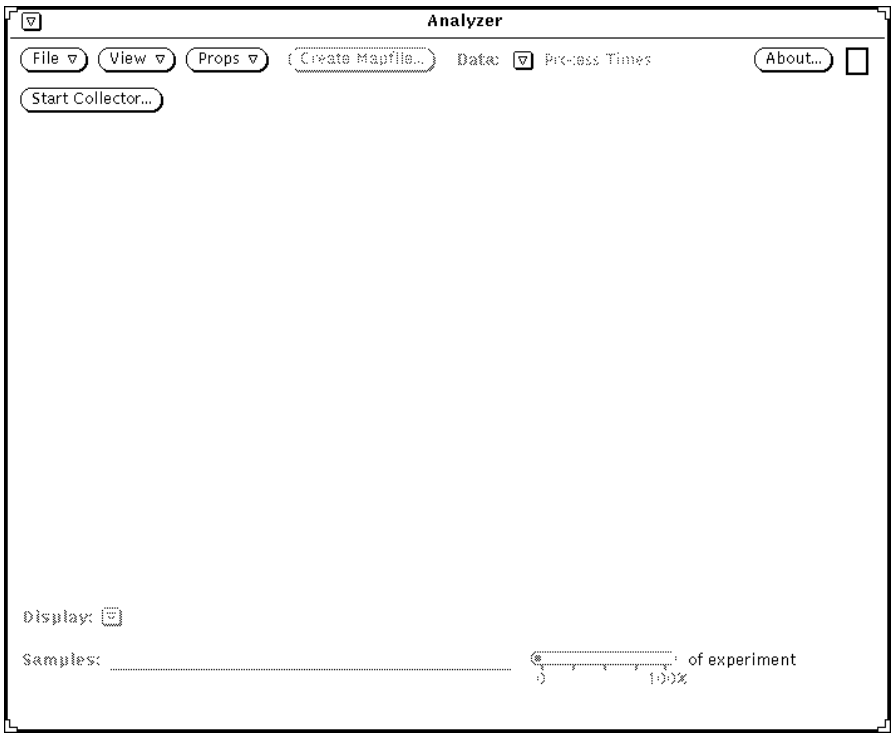


Figure 5-1 Analyzer Window

Until an experiment is loaded, some of the buttons of the Analyzer are inactive and cannot be selected. The buttons are shaded gray to indicate that they are inactive.

Note – You can also simultaneously activate the Analyzer and load an experiment. Refer to Section 5.2, “Loading an Experiment,” for details.

5.2 Loading an Experiment

You can load an experiment in three ways. One method loads the experiment, while the other methods simultaneously activate the Analyzer and load the experiment.

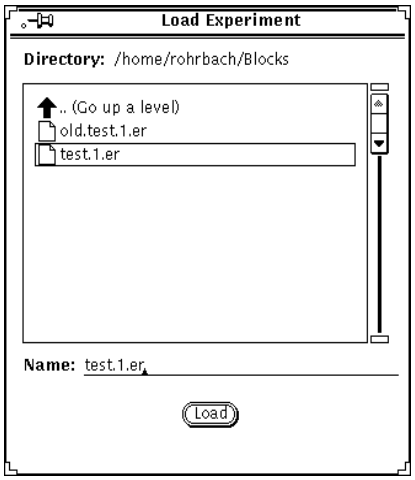
To load an experiment:

- **From the Load Experiment pop-up window:**

When you initially bring up the Analyzer, the Load Experiment popup window appears along with the Analyzer main window.

Double-click on the directory and then double-click on the experiment name listed in the scrolling list.

Alternately, you can type the directory and experiment name in the Name field and click on Load or press Return.



Other methods for loading an experiment are to simultaneously activate the Analyzer and load the experiment.

- **From the keyboard:**
  Go to the directory that contains the experiment file that you want to examine; type: **`analyzer`** *experiment_name*

- **From the File Manager:**
  Drag the icon of the experiment file and drop it into the Drop Target in the Analyzer window.

The default display of the Analyzer is the Overview display. Once the experiment is loaded, the Analyzer automatically displays the Overview display. The Overview display allows you to quickly identify overall performance problems in the application.

The other displays of the Analyzer are:

- Histogram and Cumulative Histogram
- Address Space
- Statistics

These displays provide a more detailed analysis. Use the Overview display to do your initial examination of the data, and then use the other displays after you have an idea of the performance behavior of your application.

For the other displays of the Analyzer, see the following chapters:

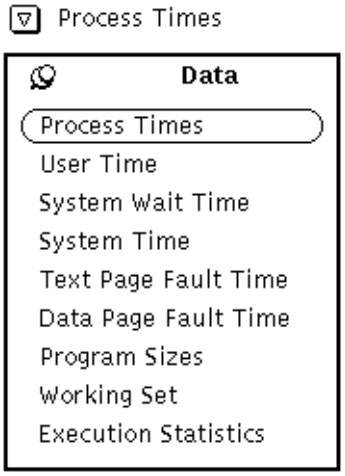
- Chapter 6, “Overview Display,” for details about Overview.
- Chapter 7, “Histogram Display,” for details about Histograms.
- Chapter 8, “Address Space Display,” for details about Address Space.
- Chapter 9, “Statistics Display,” for details about Statistics.

To delete an experiment, refer to *Deleting an Experiment*, in this Chapter.

5.3 Choosing Data Types

The Analyzer supports nine Data types in which to view, display, and analyze your application.

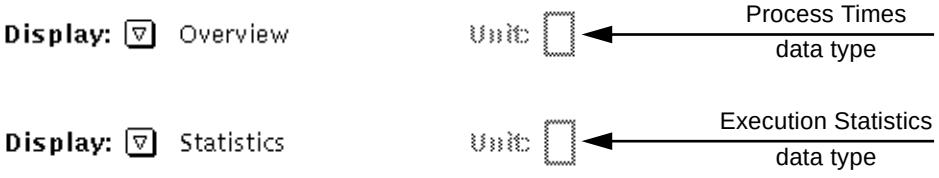
To choose a data type:



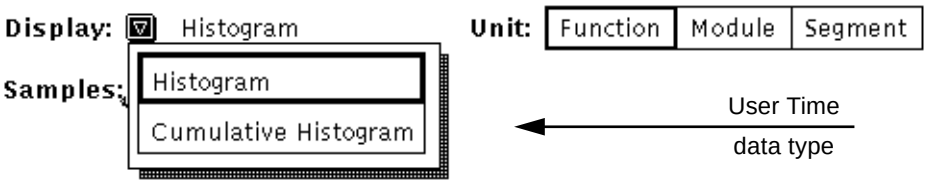
- **Click on the Data button and then choose a data type from the menu.**

Each data type has specific displays; the data type you choose determines the display you can view. Some displays can be broken down into units for viewing performance data. For some displays, certain data must be collected by the Collector. See Chapter 10, “Doing More with the Collector,” for details.

After you choose a data type to view, choose a display, and when applicable, a unit to view in the display. When a unit is not available for a particular display, the Unit button is shaded gray to denote that it is not active or selectable. In the following example, the data types selected are Process Times, which displays the Overview display and Execution Statistics, which displays the Statistics display. Notice that Unit is shaded gray because Unit types are not available with the Overview or the Statistics displays.



Some data types offer more than one display in which to view the experiment data and those displays offer more than one unit type. In the following example, the data type selected is User Time; it has several displays and unit types for viewing experiment data.



The following sections discuss the available data types, their associated displays, the units that can be viewed in the display, and the required data collection parameters of the displays.

5.3.1 Process Times

The Process Times data type is the default data type of the Analyzer; it provides information for the Overview display, which displays the occurrence of process state transitions made by the application.

When an application executes, it goes through different process states. The Process Times data type viewed in the Overview display shows the time spent by the application in process state transitions. When an application executes a system call, a transition from the user state to the system state occurs. Once the application is in the system state, it can also undergo other state transitions, such as suspend, wait, lock, and sleep, before returning to the user state.

During the process of data collection, the system records a time stamp whenever a state transition occurs. The system also computes the time difference from one state transition to another state transition and accumulates the times attributed to each state. The time difference is added to the state that has just been completed. For detailed information about the process states that you can examine in the Overview display, see Section 6.5, “Sample Properties Window,” on page 6-71.

5.3.2 User Time

The User Time data type provides information about the time spent in the user process state that occurs from the execution of instructions in the application. The displays, the units, and the required data collection parameters for the displays are shown in Table 5-1.

Table 5-1 User Time Displays

Data Type	Display Type	Unit Type	Data Collection Parameter
User	Histogram	Function	PC
		Module	PC
		Segment	PC
User	Cumulative Histogram	Function	PC and Stack
		Module	PC and Stack
		Segment	PC and Stack

5.3.3 System Wait

The process is sleeping in the kernel, but it is not in Suspend, Idle, Lock Wait, Text Fault or Data Fault state.

While this appears to be a catchall state, System Wait state is actually the most important of the non-CPU states, because it includes time spent waiting for I/O.

5.3.4 System Time

The System Time data type provides information about the time the operating system spends executing system calls. The available displays, the units, and the required data collection parameters are shown in Table 5-2.

Table 5-2 System Time Displays

Data Type	Display Type	Unit Type	Data Collection Parameter
System	Histogram	Function	PC
		Module	PC
		Segment	PC
System	Cumulative Histogram	Function	PC and Stack
		Module	PC and Stack
		Segment	PC and Stack

5.3.5 *Fault Time*

The Analyzer monitors two fault time data types:

- **Text Page Fault Time—**provides information about the time spent faulting in text pages.
- **Data Page Fault Time—**provides information about the time spent faulting in data pages.

The available displays, the units, and the required data collection parameters are shown in Table 5-3.

Table 5-3 Fault Time Displays

Data Type	Display Type	Unit Type	Data Collection Parameter
Text Page	Histogram	Function	PC
		Module	PC
		Segment	PC
Text Page	Cumulative Histogram	Function	PC and Stack
		Module	PC and Stack
		Segment	PC and Stack
Data Page	Histogram	Function	PC
		Module	PC
		Segment	PC

Data Page	Cumulative Histogram	Function	PC and Stack
		Module	PC and Stack
		Segment	PC and Stack

5.3.6 Program Sizes and Working Set

The Program Sizes and Working Set data types provide information that allows you to examine the size of your application and helps you to establish the specific memory requirements of the application.

Program Sizes describes the sizes in bytes of the functions, modules, and segments of the application.

When the working set is too large, the application might be paging text or data pages, or it might be paging both text and data pages. The Working Set data type allows you to look at the reference behavior of both text pages and data pages. The available displays, the units, and the required data collection parameters are shown in Table 5-4.

Table 5-4 Program Sizes and Working Set Data

Data Type	Display Type	Unit Type	Data Collection Parameter
Program Sizes	Histogram	Function	Overview (default)
		Module	Overview (default)
		Segment	Overview (default)
Working Set	Address Space	Page	Working Set
		Segment	Working Set

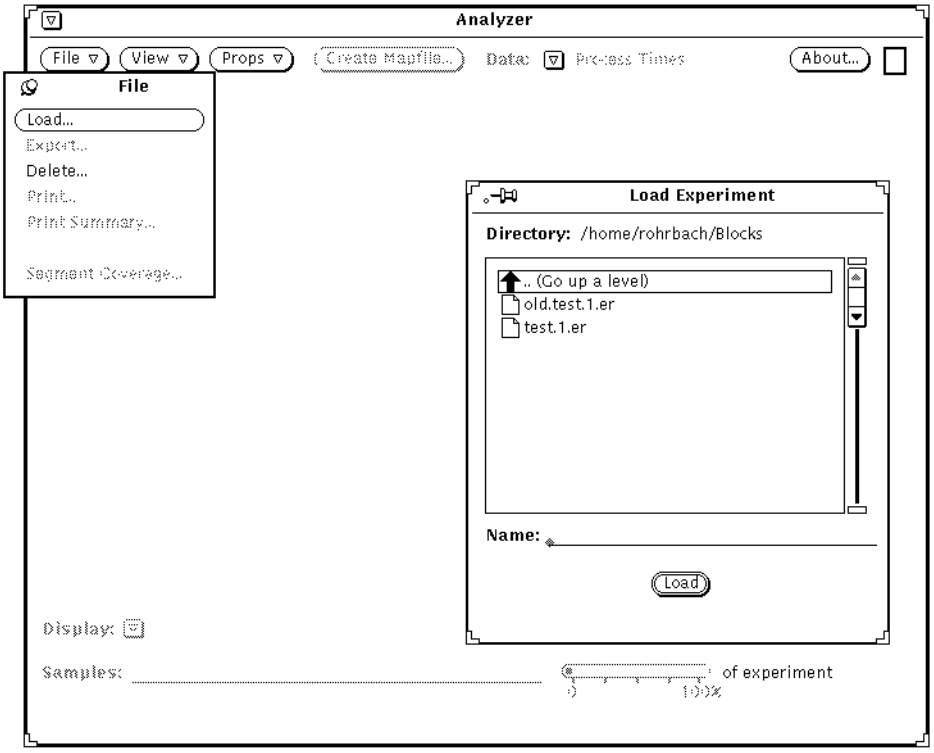
5.3.7 Execution Statistics

The Execution Statistics data type is similar to the Process Times data type in that it provides an overview of the application behavior. The only data collection parameter required to view this display is overview data, which is the default data collection parameter of the Collector. You do not need to specify a data collection parameter to view this display.

5.4 Deleting an Experiment

Use the Delete command to delete an experiment. Refer to the illustration on the next page. To delete an experiment, select the File menu in the Analyzer.

Select Delete which activates a File Chooser window. Then choose from the list of experiments. When you have selected an experiment, click on the Delete button in the File Chooser. At this point, the Analyzer asks you to confirm the deletion of the experiment. If you confirm the deletion, the experiment is deleted. Deletion of the experiment includes the pointer file, the hidden directory, and all the files associated with the experiment.


Analyzer
File
View
Props
Create Mapfile...
Data:
Process Times
About...
File
Load...
Export...
Delete...
Print...
Print Summary...
Segment Coverage...
Load Experiment
Directory: /home/rohrbach/Blocks
.. (Go up a level)
old.test.1.er
test.1.er
Name:
Load
Display:
Samples:
of experiment
0
100%

5

Part 3 — Reading the Displays

Overview Display 6

This chapter describes the components of the Overview display and the features for selecting and displaying performance data

.This chapter is organized into the following sections:

6.1 What Is the Overview Display?

The default display of the Analyzer is the Overview display. For each sample, the display shows the amount of time the application spends in different process states. The information needed to produce this display is always generated by the Collector during the data collection process.

The Overview display:

- Provides a high-level overview of the performance behavior of the application
- Provides data on how the application execution time breaks down into the different performance areas, allowing you to quickly identify CPU bottlenecks, I/O bottlenecks, or paging bottlenecks

- Shows the changes of the application performance behavior during execution. For example, the beginning part of the execution might be I/O bound and the later part of the execution might be CPU bound.

Begin your analysis with this display. Once you have an understanding of the performance behavior of your application, you can examine the application in greater detail using the other displays of the Analyzer.

The Overview display consists of two areas (see Figure 6-1):

- Overview chart, including an execution time line
- Averages legend

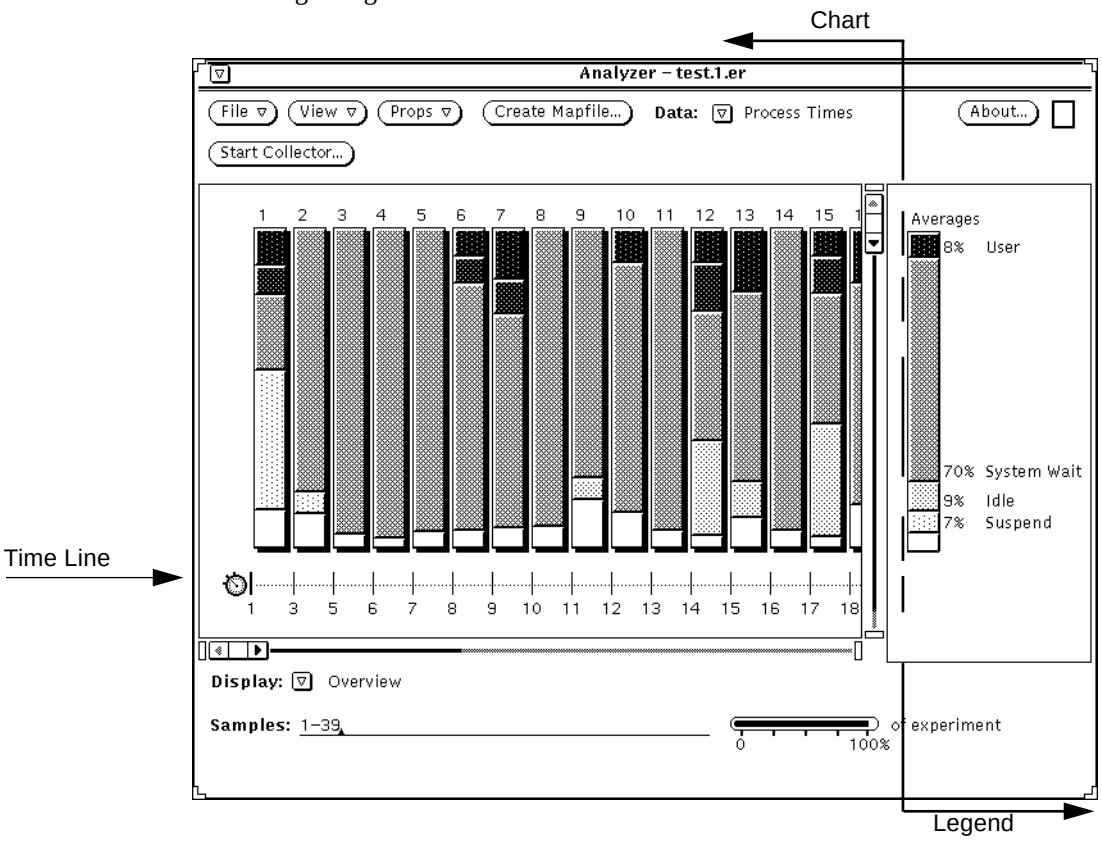
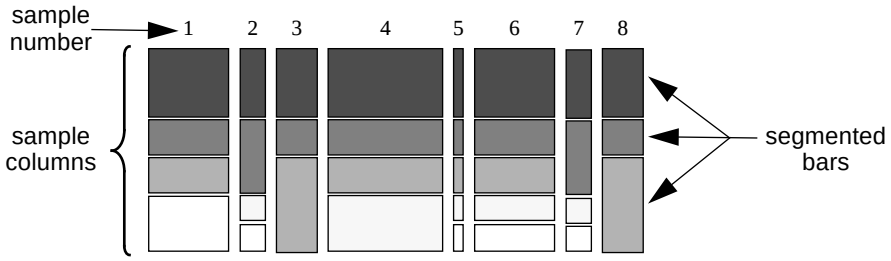


Figure 6-1 Overview Display

6.1.1 Overview Chart

The Overview chart contains numbered sample columns that are made up of segmented bars.

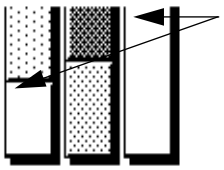


6.1.1.1 Sample Columns

The columns represent the individual samples collected during an experiment. The columns are located above the Time Line (see Figure 6-1), which represents real time of the experiment.

6.1.1.2 Segmented Bars

The bars that make up each column represent the different performance problem areas. The height of each bar is a proportional representation of the time spent in one of the performance problem areas.



A white color bar denotes the Others category. Performance problems that are too small to display are put into this category. The Analyzer determines which problems are too small to display.

To see exactly what performance problems are contained in Others, select the sample column that contains it and view it in the Sample Properties window. See Section 6.3, “Selecting and Displaying Samples,” on page 6-69.

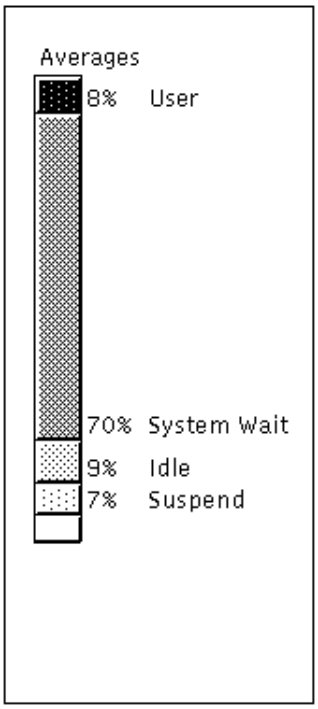
The Analyzer uses a fixed shading scheme; the shade that represents a specific performance area is consistent across all the sample columns in the experiment and is also the same shade in other experiments.

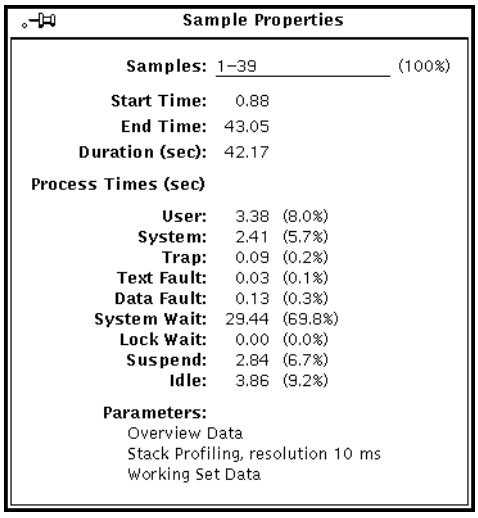
6.1.2 Averages Legend

The Averages legend displays the details of a sample or a combination of samples. In the Averages legend, you can examine the percentage of time spent in each of the performance problem areas that are contained in the samples you select from the Overview chart. You can select one sample, a combination of samples, or all the samples contained in the experiment. If you select more than one sample, then the Averages legend shows the average for the combination of samples you selected.

Note – The percentages listed in the Averages legend are rounded off to the integer; however, the exact values are shown in the Sample Properties window. For details about the Sample Properties window see "Sample Properties Window" on page 6-71.

In the following example, 39 samples are selected (1-39). Notice that the Averages legend shows System Wait as 70%, but the Sample Properties window shows the exact value, which is 69.8%.

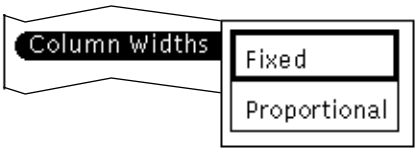




6.2 Changing the Width of the Sample Column

The Overview display provides two column width options, fixed and proportional, for displaying the individual sample columns.

To set the column width:



- **From the View button menu, choose Column widths (Fixed or Proportional).**

6.2.1 Fixed-Width Columns

If you choose Fixed, then the width of each sample column is identical in size (see Figure 6-2).

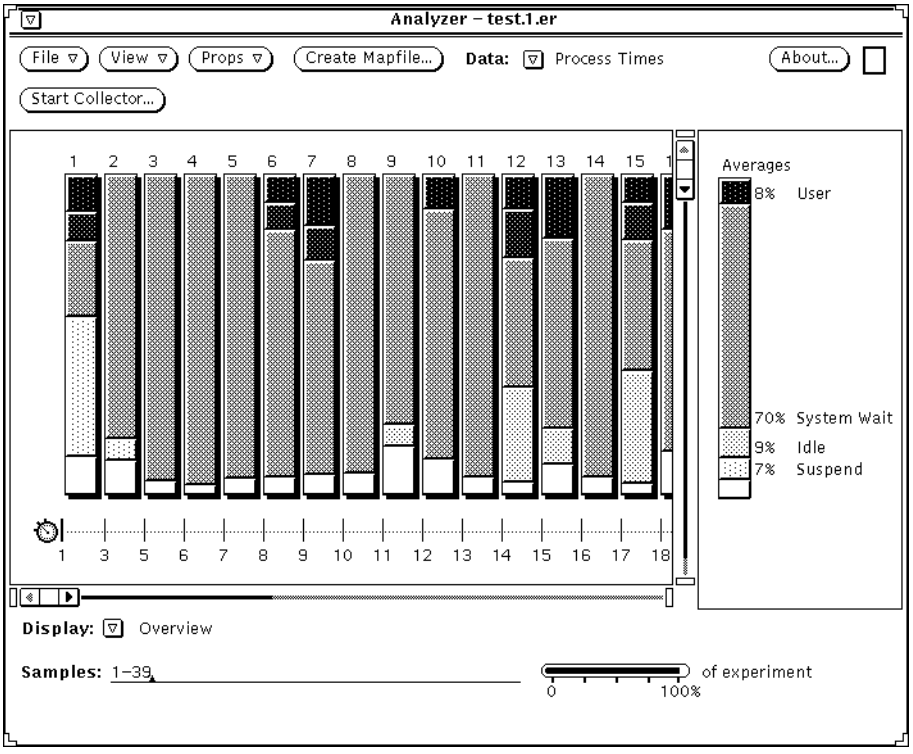


Figure 6-2 Overview with Fixed Widths

Because the size of the sample columns is identical, Fixed shows more samples in a given view of the Overview display. Also note that the Time Line is nonlinear in the fixed-width mode.

6.2.2 Proportional-Width Columns

If you choose Proportional, then the width of each sample column is proportional to the duration of the sample as it occurred during the data collection process (see Figure 6-3). The Time Line is more linear in the proportional-width mode.

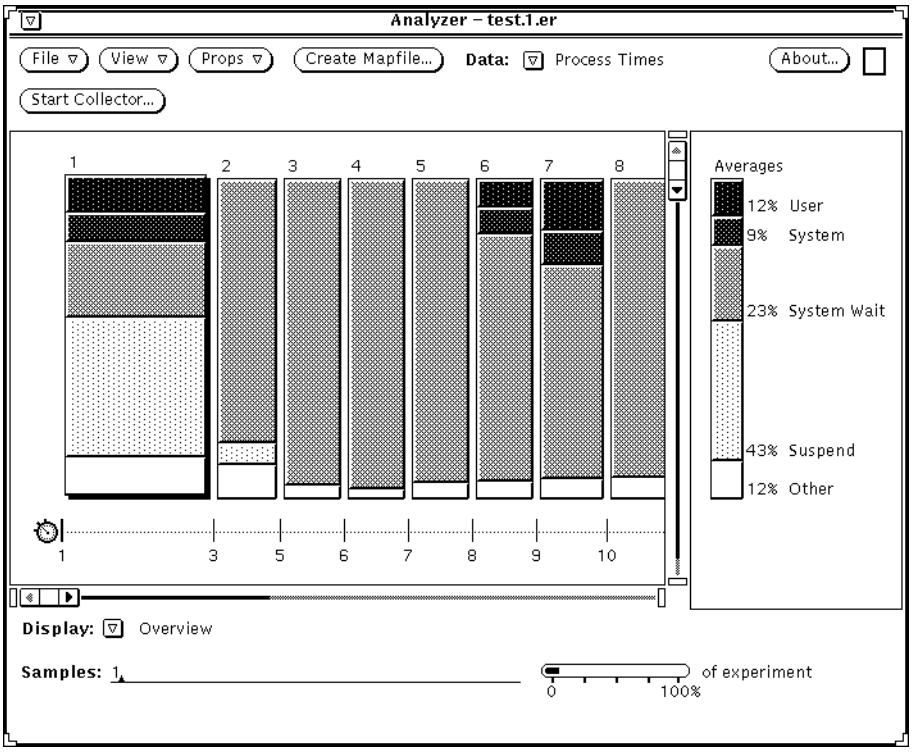


Figure 6-3 Overview with Proportional Widths

Because the size of the sample columns are unequal, proportional-width mode provides a visual comparison of the duration of the samples in relationship to each other.

Whether you are using Fixed or Proportional column width, you are not able to view all the samples of an experiment at a given time because the experiment can contain numerous samples. Use the scrollbars to bring the samples that are not displayed into the view.

You can also enlarge the window to the limits of your screen by selecting and dragging any of its corners; a larger window displays more samples.

6.3 *Selecting and Displaying Samples*

By selecting samples and viewing them in the Averages legend, you can examine an individual sample or group of samples in more detail. If you choose a sample column that contains an Others category, then you also obtain information about what performance problems are contained in that Others category.

You can view additional data about the selected samples in the following places:

- Averages Legend on page 66
- Sample Properties Window on page 71

To select samples, do one of the following:

♦ **Use the mouse to select samples from the Overview display.**
Point and click on the sample column that you want to select. You can position the pointer anywhere on the column. The mouse functions for select and adjust (toggle) operate per the OPEN LOOK GUI.

You can click on more than one column; however, if you click on one that is already selected, then that column becomes deselected.

Note – You can select one sample, or a combination of samples, but you cannot select a portion of a sample.

When the experiment is initially loaded, the Overview is displayed with all samples selected. Once you make your first sample selection, all the other samples are deselected.

- **Choose Select All or Select None from the View menu to select or deselect all samples in the experiment.**
- **Use the keyboard to type your sample selection in the Samples text field.** Type the number of the sample in the Samples text field located in the bottom left corner of the Analyzer window; press the Return key after you finish typing your sample selection.

  Sample selections can be any combination, such as those denoted in the following examples:

  Samples: 2

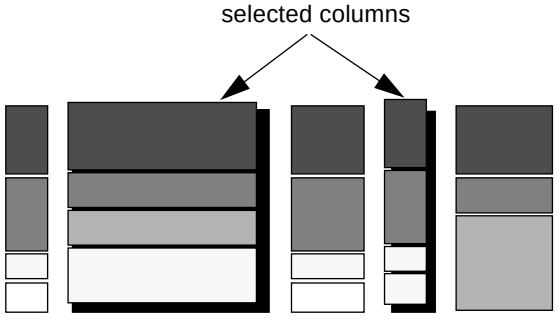
  Samples: 4,7

  Samples: 1,3,5-8

  In the first example, one sample is selected (sample 2). In the second example, two samples are selected (samples 4 and 7). In the last example, six samples are selected (samples 1, 3, 5, 6, 7, and 8).

When you select a sample by clicking on the sample in the Overview display, the number of that sample also appears in the Samples text field.

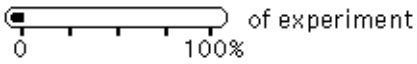
When you select a column, either by clicking on it in the Overview display or by typing its number into the Samples text field, that column moves up and over to the left of its original position and has a shadow, giving it a three-dimensional appearance to indicate that it is selected.



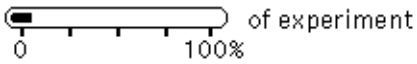
6.4 Experiment Scale

When you select a sample, or a group of samples, the Analyzer displays the time relationship of your sample selection as it relates to the entire experiment. The amount of time is displayed in the Experiment Scale located in the lower right corner of the Analyzer window (see Figure 6-1). The following are examples of sample selections along with their time displays.

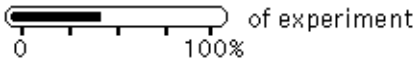
Samples: 2



Samples: 4,7,11



Samples: 1,3,5-14,22



The scale allows you to focus on the details of an experiment. Suppose for example, you have an experiment with 1000 samples, and each sample is 1 second long. You decide to examine only sample 99 of this experiment and find that System Wait time is 100% of sample 99. From this data, you might conclude that System Wait is a major bottleneck. Next, you select all the samples of the experiment to examine and find that System Wait time over the entire experiment is only 1%. From this data, you decide that System Wait is not a major bottleneck, but only a dominant factor in sample 99.

The scale is a concise reminder of the perspective that you have on the data. When you look at the aggregate data, System Wait is 1% of the experiment time; however, when you look only at sample 99, you see that I/O is 100% of the sample time.

6.5 Sample Properties Window

This window allows you to examine a sample or a combination of samples in more detail. The Sample Properties window provides information only; you cannot enter information into this window (Figure 6-4).

♦ **To activate the Sample Properties window:Click on the Props button in the Analyzer base window.**

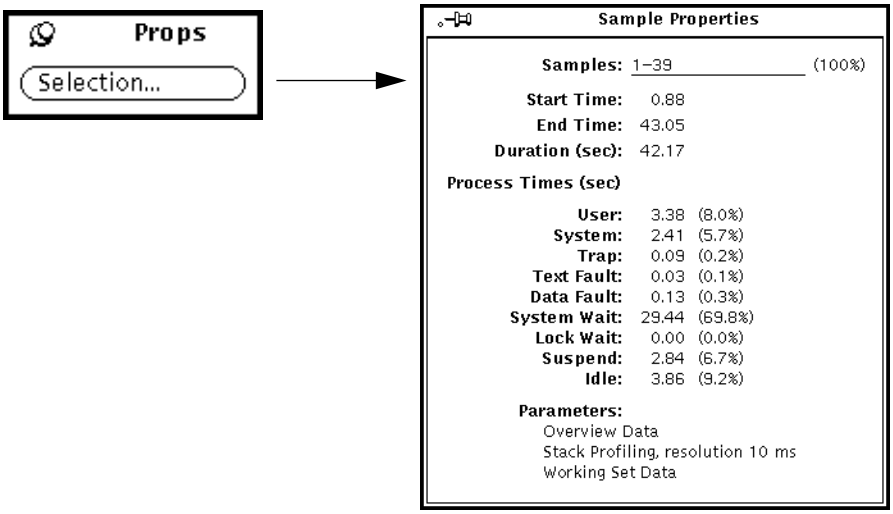


Figure 6-4 Sample Properties Window

The Sample Properties window provides detailed information about the selected sample(s), such as:

- Current sample selection and the time relationship of the sample selection as it relates to the entire experiment
- Start time of the sample
- End time of the sample
- Duration of the sample
- Process Times (see Figure 6-4). Process Times includes ten process states:

  User—the time spent executing program instructions.

  System—the time the operating system spent executing system calls.

  Trap—the time spent executing traps (automatic exceptions or memory faults).

  Text Fault—the time spent faulting in text pages.

  Data Fault—the time spent faulting in data pages.

  System Wait—the time the process is sleeping in the kernel (but not in Suspend, Idle, Lock Wait, Text Fault, or Data Fault state).

Lock Wait—the time spent waiting for lightweight process locks to be released

Sleep—the time the program spent sleeping (due to any cause other than Text Fault, Data Fault, System Wait, or Lock Wait).

Suspend—the time spent suspended (includes the time spent in the Debugger when it encounters breakpoints). A suspended state can be caused by another process that is running, job control, or execution of the suspend command, `lwp_stop`.

Idle—the time spent waiting to run while the system is busy.

- Parameters (data collection parameters of the sample)

  The Sample Properties window lists the data collection parameters that are collected for each sample. There are four data collection parameters; however, the sample example in Figure 6-4 shows three because that particular sample contains only three data collection parameters.

  Overview—(default) collects resource usage and process real time performance data.

  PC—(program counter) collects profiling data.

  PC and Stack—collects return addresses on the call stack.

  Working Set—collects memory usage data.

  See Chapter 4, “Getting Started with the Collector,” and Chapter 10, “Doing More with the Collector,” for detailed information about the data collection parameters.

Note – To obtain information about what performance problems are contained in an Others category, select a sample column that contains an Others category, then view the sample data in the Properties window.

You have two ways in which to select samples:

- Type in the number of the sample or group of samples in the Samples text field of the Analyzer window.
- Select samples by clicking on the sample column in the Overview display.

The number of the sample or samples you select is displayed in the Samples text field of the Sample Properties window. You cannot change the sample selection in the Sample Properties window.

For detailed information on selecting samples, see Section 6.3, “Selecting and Displaying Samples,” on page 6-69.

6.6 Time Line

The Time Line is the x-axis of the window display and is made up of glyphs and tick marks that specify details about the individual sample columns.

If you have the column width set for fixed width, then the tick marks are also equal, like fixed column width samples, in size and spacing.

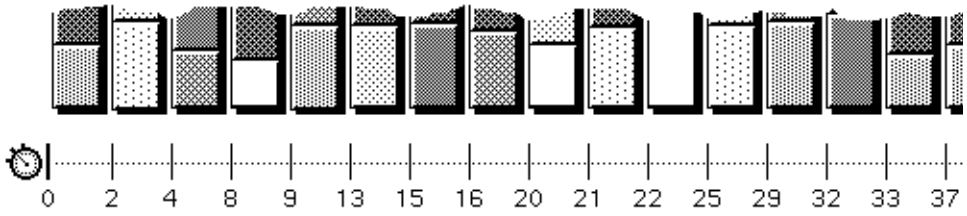


Figure 6-5 Time Line with Fixed Width

If you have the column width set for proportional width, then the tick marks are also proportional, like proportional column width samples, in size and spacing. With proportional column width, the tick marks are a proportional representation of how long it took to collect the performance data for that sample (sample duration).

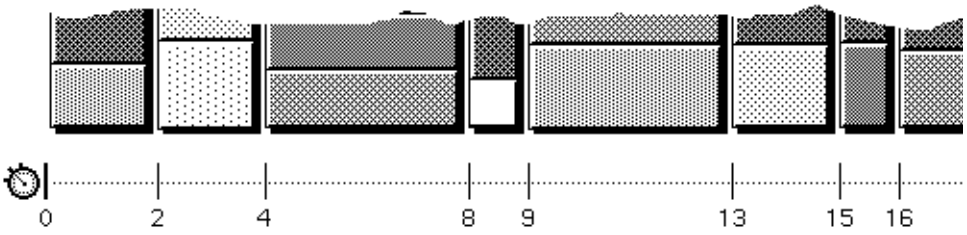


Figure 6-6 Time Line with Proportional Width

The glyphs and tick marks represent the following:

The time clock glyph is not selectable. It serves as a visual reference that the information present in this area deals with time.

A thick tick mark indicates the origin of the Time Line.

A thin tick marks denote sample boundaries, the beginning and end of a sample.

A tick mark with an arrow is placed on the line at the end of the last sample, which is also the end of the Time Line.

Note – The number that appears at the end of a tick mark is the time that elapsed since the start of the experiment. The number is rounded-off to the nearest integer.

6

Histogram Display 7

This chapter describes the components of the Histogram and the Cumulative Histogram displays, and the features for selecting and displaying the performance data.

This chapter is organized into the following sections:

7.1 What Is the Histogram Display?

The Histogram provides an overall view that allows you to visually examine the components of the application to help you detect what might be causing performance problems. In Chapter 6, “Overview Display we saw how the Overview display is used to identify the types of bottlenecks that the application might experience, for example, User Time (CPU) bottlenecks.

By using the Histogram display, you can find where the application is spending its user time. By examining the performance data of your application in a Histogram you can relate the CPU execution time to application components.

The following histograms are available in the Analyzer:

- Function
- Module
- Segment
- Cumulative

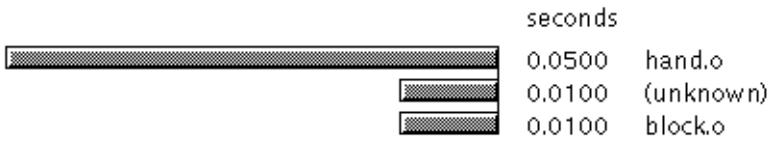
7.1.1 Histogram by Function Display

This display is the default histogram display. Use this display to find the amount of time the application spends executing functions. For example, suppose you want to examine the user time performance area of your application because you suspect a CPU bottleneck. By viewing the user time as a Histogram by Function display, you can identify the function or set of functions in which the application is spending the majority of its time executing. This function or set of functions is usually the first target to consider for tuning to increase the performance of your application.

7.1.2 Histogram by Module Display

This display profiles the amount of time spent executing modules in the application. Sometimes an application has too many functions, so its performance behavior cannot easily be understood when viewing it as a Histogram by Function. For this situation, the Analyzer provides higher levels of data aggregation. One of these levels is a Histogram by Module display. In this display, all data for a single source file (a module) is aggregated together. In programming practice, related functions are grouped to a single source file, so the module level of aggregation usually provides a more concise representation of the performance activity of the application than does the Histogram by Function display.

Note – If any part of the executable, including shared libraries, is not compiled with the `-g` option, then the Debugger may not have enough information to attribute what functions are contained in certain modules. The Debugger, by default, attributes these functions to an *unknowm* module.



Note – To ensure that the data in the module histogram is accurately displayed, we recommend that you build your application using the `-g` option See Section 3.1, “Building the Application,” on page 3-29.

7.1.3 Histogram by Segment Display

This display profiles the amount of time spent executing text segments in the application. The highest level of aggregation displayed in the Histogram is the segment. SunOS provides a feature for dynamically linking shared libraries into a process at runtime. A typical C application may have two or more segments. The first segment is the main application, which consists of application specific code. The second segment is the C library (called `libc.so`), and then any other shared libraries.

For example, the math library (`libm.so`) and the XView library (`libxview.so`) are the remaining libraries. This segment level of aggregation provides very coarse data and it is helpful and necessary to examine. It is not unusual for an application to spend the bulk of its execution time in the code of a shared library. There may be very little you can do to tune an application when this is the case; however, the information is helpful when applied to your entire performance-tuning effort.

7.1.4 Cumulative Histogram Display

The Histogram presents a general summary of the amount of time spent executing functions, modules, and segments in the application. The Cumulative Histogram shows the total amount of execution time spent by a function, module, and segment in its relationship to other functions, modules, and segments.

Cumulative Histograms are provided because a regular histogram based on PC profile data does not always give you the information you need. For example, suppose function `A` calls the `strcpy()` function 1,000,000 times for a total

execution time of 10 seconds, and function B calls `strcpy()` one time for a total execution time of 10 microseconds. In addition, the execution time in A and B, excluding the time spent in calls to `strcpy` is one second each. In a regular histogram, you would see execution times of 0.00001, 1.0, and 1.0 seconds for `strcpy()`, `A()`, and `B()`, respectively.

If you want the application used in this example to take 80 percent of its time executing in 20 percent of its code, then you would want to tune `strcpy()` for performance. However, since this time is a runtime in the standard C library, you are not able to tune `strcpy()` because it has already been tuned. If you were viewing this application in a regular histogram, then you would have no additional information indicating as to whether you should tune A or B.

In the Cumulative Histogram, all execution time accumulated in a descendant function is attributed to the parent function. In the example, `strcpy` time is attributed to A and to B. If `strcpy` also calls some other function, such as X, then the time spent in X is attributed to X, `strcpy`, A, and B. Assuming `strcpy` does not call any other functions, the Cumulative Histogram would show 11.0 seconds for A (from A itself, and calls to `strcpy`), 0.00001 seconds for `strcpy`, and 1.00001 seconds for B (from B itself, and the call to `strcpy`). The Cumulative Histogram value for `strcpy` is the same as the Histogram value for `strcpy`, but the values for A and for B are dramatically different in the Cumulative Histogram.

This information assists you in your performance-tuning efforts because you now have enough information to conclude that A is a better target to tune than is B to increase the performance of this application. You still are not able to tune `strcpy`; however, you may be able to tune the usage of `strcpy` by tuning A for performance. You might change A to call `strcpy` fewer times, or you might change the parameters of `strcpy` when you call it from A to parameter values that take less time to process.

The Histogram and Cumulative Histogram displays are available for examining the following data types:

- **User Time—**the time spent executing instructions in the application.
- **System Wait—**the time the process is sleeping in the kernel.
- **System Time—**the time spent by the operating system executing system calls.
- **Text Page Fault Time—**the time spent faulting in text pages.

- **Data Page Fault Time—**the time spent faulting in data pages.

The data that you need to collect to view Histograms and Cumulative Histograms is outlined in Table 7-1.

Table 7-1 Histogram and Cumulative Histogram Data Collection Requirements

Display Type	Unit Type	Data Collection Parameter
Histogram	Function	PC
	Module	PC
	Segment	PC
Cumulative Histogram	Function	PC and Stack

Note – *PC and Stack data must be collected to view a Cumulative Histogram.*

The Histogram display is composed of horizontal bars, numbers, and names (see Figure 7-1).

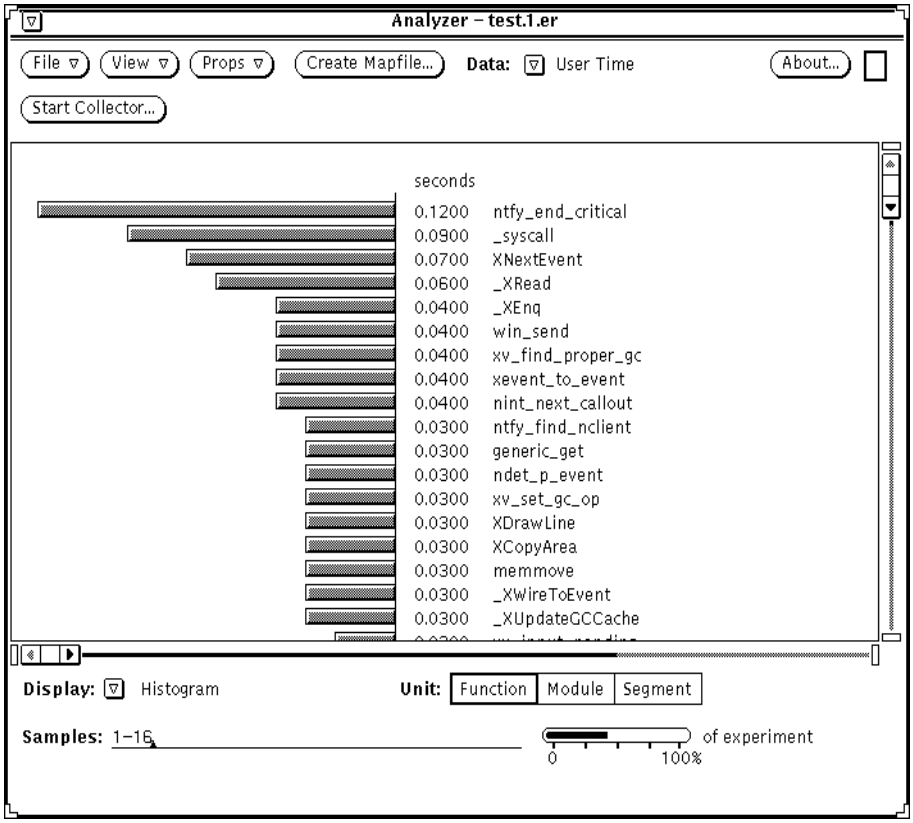


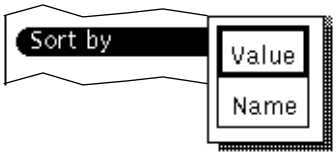
Figure 7-1 Histogram Display

- **Horizontal Bars—**are a visual representation of the time the application spent executing a function, module, or segment. In all the histograms, the bars are sorted by value (time spent) starting with the longest bar, which equals the highest value. You also have the option to display the names alphabetically. See “Sorting the Histogram” on page 7-83.
- **Numbers—**are specific to the histogram that you are viewing. For example, if you choose to display User Time as a Histogram by Function, then the numbers represent the amount of time spent executing the functions.
- **Names—**are specific to the histogram that you are viewing. For example, if you choose to display User Time as a Histogram by Function, then the names are the function names in the source code of the application.

7.2 Sorting the Histogram

The functions, modules, and segments of the application can be displayed in the Histogram and Cumulative Histogram either numerically (value) or alphabetically (name).

To sort the Histogram or Cumulative Histogram:



- **Choose Sort by from the View button menu.**

7.2.1 Sort by Value

Sort by Value is the default setting. This sorting order lists the highest value first, providing a quick visual reference that identifies the functions, modules, or segments with the largest values (see Figure 7-2). This type of sorting is useful when you are concerned with execution time.

For example, suppose you are viewing a histogram by function and you want to know which functions take the most amount of time or the least amount of time to execute. Sorting by value puts the highest value first, followed by the second most highest value, and continues to list all the values in decreasing order.

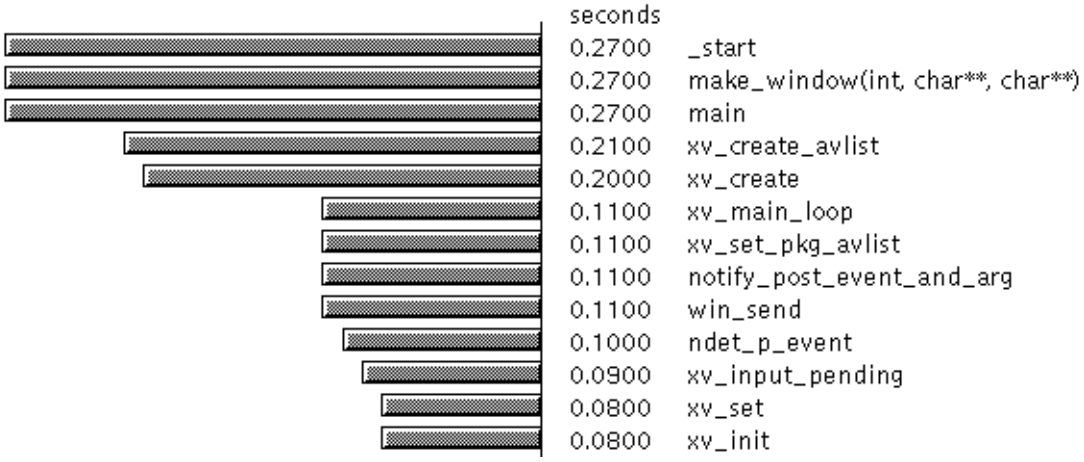


Figure 7-2 Histogram Sorted by Value

7.2.2 Sort by Name

Sort by Name alphabetizes the functions, modules, and segments, providing a visual reference that you can use when comparing samples within the experiment, or when comparing two experiments (see Figure 7-3).

For example, suppose you have two experiments and you are examining their user time performance area in a Histogram by Function. If you use Sort by Name, then you can easily compare the same functions in the two displays. Whereas, if you used Sort by Value, then functions may be listed in completely different orders, making visual comparison difficult.

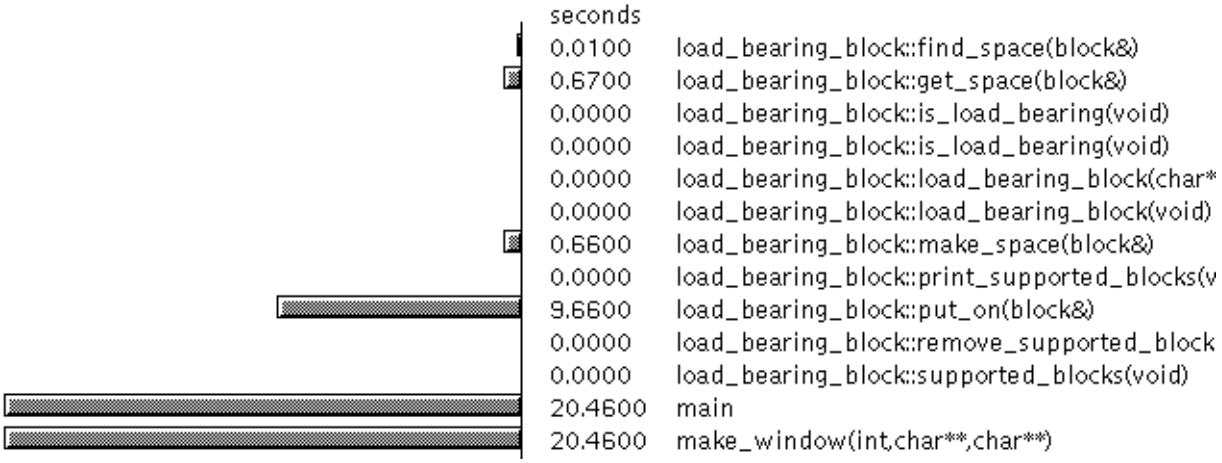
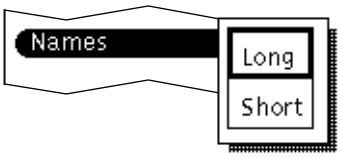


Figure 7-3 Histogram Sorted by Name

If your application is written in C++, then you can also display the Histogram and Cumulative Histogram by function name and its argument (Long) or just the function name (Short).

To display names for a C++ application:



- **Choose Names Long from the View button menu.**

Examples of Names Long and Names Short displays are shown in Figure 7-4 and Figure 7-5.

When displaying Long names, use the horizontal scroll bar to bring into view any information that exceeds the borders of the display window.

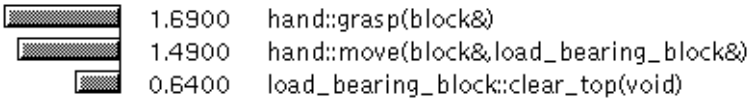


Figure 7-4 Long Names

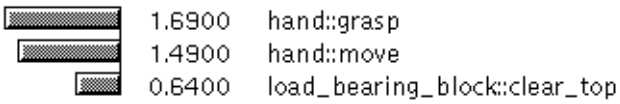


Figure 7-5 Short Names

Histograms and Cumulative Histograms, whether they are displayed as Long names or Short names, can also be sorted by Value or Name.

7.3 Searching for Specific Names

You can scroll through the information in the Histogram and Cumulative Histogram searching for a specific function, module, or segment by using the following methods:

- **Scrollbars—**move the information in the Histogram both horizontally and vertically. The horizontal scrollbar is located at the bottom of the display, and the vertical scrollbar is located on the right side of the display. See Figure 7-1 on page 82.
- **Find—**searches for a text string. Use Find in Histograms and Cumulative Histograms. The text you enter into this field can be a complete or a partial text string of a function, module, or segment name. Find searches the histogram until it finds the first occurrence of the text that you specified.

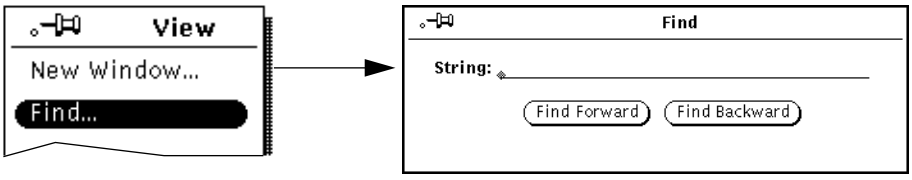
For example, if you search for `bcd`, then Find identifies every occurrence of the text string `bcd`, such as:

- bcd
- abcd
- bcdstd
- abcdstd

The Find option is case sensitive. If you search for Bcd, (upper case B) but all occurrences of bcd begin with a lower case b, then Find does not recognize or identify bcd as the text string for which you are searching.

To use Find:

- **Choose Find from the View button menu. In the Find pop-up window, type the text string of the function, module, or segment that you want to search.**



Find has two options:

Find Forward

The Find Forward option searches forward for the text string from the line of the histogram in which you are currently located. It also has a wrap-around feature so that once it reaches the end of the histogram, it continues to search for the specified text string.

Find Backward

The Find Backward option searches backward for the text string from the line of the histogram in which you are currently located. It also has a wrap-around feature, so that once it reaches the beginning of the histogram, it continues to search for the specified text string.

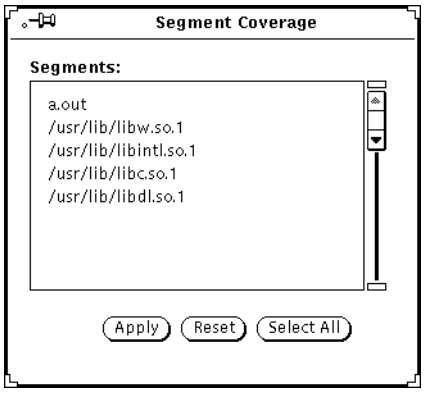
For both options, Find repaints the histogram, moving the display so that the specified function, module, or segment searched for is now at the top of the histogram display. If the first text string that Find identifies is not the text string that you want, then click on Forward or Backward to continue the search. The sorting order (by value, or number) remains the same.

7.4 Viewing Segments

You can view individual segments or a group of segments. The Segment Coverage option shows a list of all the text segments that are part of the application. The text segment of the application is one segment, and there are

additional segments for each shared library that the application links with. The default Segment Coverage is the text segment of the application and all the shared libraries.

To change the Segment Coverage:



- **Choose Segment Coverage from the File button menu in the Analyzer base window**.

  Apply—completes and applies your selection to the display.

  Reset—restores the settings that existed before the last apply.

  Select All—selects all the segments contained in the application code.

Note – The segment selection affects the display in all the windows in which you are viewing Histograms or Cumulative Histograms. The default is Select All.

Once you choose a segment or group of segments, only those functions and modules contained in those segments are displayed in the Histogram or Cumulative Histograms. The functions and modules contained in the segments you are viewing can also be sorted by value or by name using the Sort option, and you can also search for specific function and module names using the Find option.

7.5 *Selecting and Displaying Samples*

You can select a sample or a group of samples to view in a Histogram or a Cumulative Histogram display. When you select a specific sample or group of samples, the bars representing functions, modules, and segments (and their respective values) now represent the values of that particular sample or group of samples, as opposed to representing the entire experiment.

For example, suppose you have just finished viewing sample 11 in the Selected Column Legend of the Overview display, and you have also viewed sample 11 in the Sample Properties window. The examination of sample 11 shows that

most of the application time is spent in the User Time area. To examine the performance behavior of sample 11 in more detail, select sample 11 and view the System Wait as a Histogram by Function.

You may also want to view sample 11 as a Histogram by Module and as a Histogram by Segment, and also in a Cumulative Histogram. Isolating and examining sample 11 can help identify which functions, modules, or segments are using Systrem Wait time.

To select samples, do one of the following:

- **Use the mouse to select samples from the Overview display.**
  Point and click on the sample column that you want to select. You can position the pointer anywhere on the column. The mouse functions for select and adjust (toggle) operate per the OPEN LOOK GUI.

  You can click on more than one column; however, if you click on one that is already selected, then that column becomes deselected.

Note – You can select one sample, or a combination of samples, but you cannot select a portion of a sample.

When the experiment is initially loaded, the Overview is displayed with all samples selected. Once you make your first sample selection, all the other samples are deselected.

- **Choose Select All or Select None from the View menu to select or deselect all samples in the experiment.**
- **Use the keyboard to type your sample selection in the Samples text field.**
  Type the number of the sample in the Samples text field located in the bottom left corner of the Analyzer window; press the Return key after you finish typing your sample selection.

  Sample selections can be any combination, such as those denoted in the following examples:

  Samples: *2*

  Samples: *4,7*

  Samples: *1,3,5-8*

In the first example, one sample is selected (sample 2). In the second example, two samples are selected (samples 4 and 7). In the last example, six samples are selected, (samples 1, 3, 5, 6, 7, and 8).

When you select a sample or a group of samples, the Analyzer displays the time relationship of your sample selection as it relates to the entire experiment. The amount of time is displayed in the experiment scale. See Section 6.4, “Experiment Scale,” on page 6-61 for detailed information.

7

Address Space Display 8

This chapter describes the components of the Address Space display and the features for selecting and displaying the performance data.

This chapter is organized into the following sections:

8.1 What Is the Address Space Display?

The Address Space display helps you identify memory that is most valuable to the application (modified and referenced pages). This display also identifies memory that is unused because the experiment did not exercise all the application functionality or because the application has dead code or memory allocation problems. The display shows the following memory usage categories:

- **Modified—**a page that is written on during the execution of the application. Modified pages may or may not be referenced.
- **Referenced—**a page that is read by the application or contains instructions that have been executed by the application.
- **Unreferenced—**a page that is not modified and not referenced.

You must collect Working Set data to view the Address Space display.

The Address Space display consists of columns that are made up of individual squares (pages) or rectangles (segments), numbers, gaps, and a Legend (see Figure 8-1). You can view the display in Units of Page or Segment. Figure 8-1 shows Page unit and Figure 8-2 shows Segment unit.

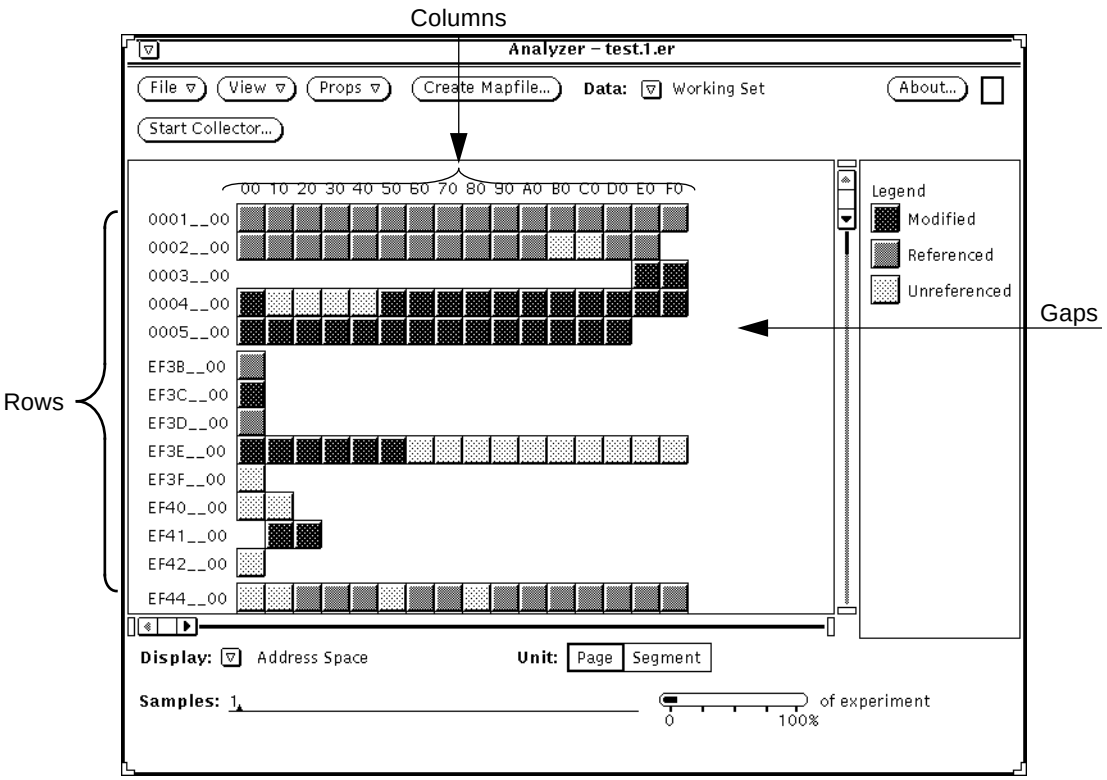


Figure 8-1 Address Space Display (Page)

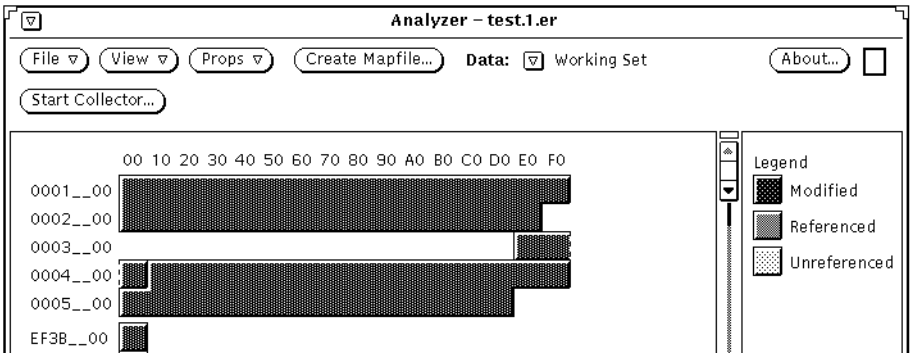


Figure 8-2 Address Space Display (Segment)

- **Individual squares or rectangles—**represent pages (squares) or segments (rectangles) in the address space.
- **Numbers—**represent the address of the space the page occupies.

  Sun systems use either 4 Kbyte or 8 Kbyte pages. The address of a page is a multiple of 0X1000 (4 Kbyte in hexadecimal) or 0X2000 (8 Kbyte in hexadecimal).

♦ **To calculate the number of pages per row divide 64 Kbyte by the page size of your system.**
If the page size is 4 Kbytes, then the number of pages per row is 16. If the page size is 8 Kbytes, then the number of pages per row is 8.

The address of a page is obtained by combining the hexadecimal values of the row and column that contains the page. For example, if the page you are examining is in the fourth row (0004_ _00), third column (20), then the address of that page is 00042000.

♦ **To verify the page size of your system, type:** `pagesize`
The page size is noted in bytes: 4096 (4 Kbyte page) and 8192 (8 Kbyte page). For detailed information about `pagesize`, refer to the man page `pagesize`.

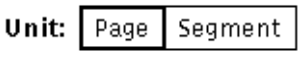
- **Gaps—**(shown as white space) represent a portion or region of the address space that was not used by the application program.

- **Legend—**is a reference that indicates the shades of the three different memory usage categories that you can view in the Address Space display. The Legend provides information only; the information contained in the Legend cannot be selected.

8.2 Selecting and Displaying Pages

The Address Space display provides two viewing modes in which you can examine the performance data: by page or by segment.

To select the viewing mode:



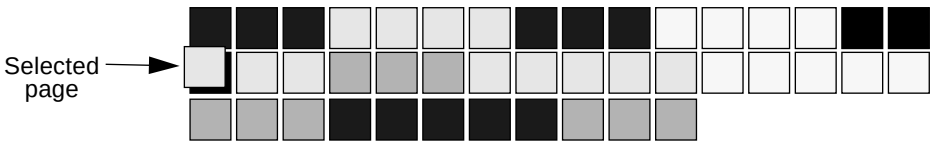
♦ **Click on Page or Segment from the Unit button.**

When viewing the Address Space as a Page or Segment, you have the following options:

- When viewing pages, you can also view the segment that contains the page.
- When viewing segments, you can also view the pages contained in that segment.

8.2.1 Viewing Pages

The pages viewing mode allows you to view an individual page and the segment that contains that page. When you select a page, the page moves up and over to the left of its original position and is shadowed, giving it a three-dimensional appearance to indicate that it is selected.

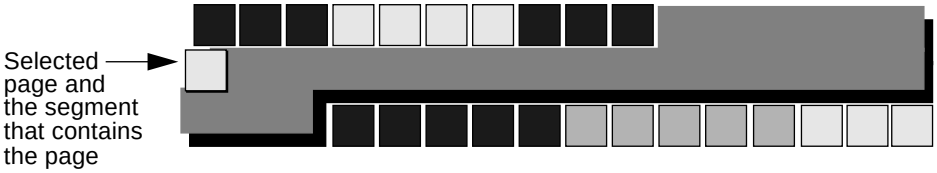


To select a page:

♦ **Position the pointer anywhere on the page that you want to select and click the left mouse button.**

Note – Only one page can be selected. You cannot select a group of pages. Once you select a page, the previous page becomes deselected.

You can also view the segment that contains a page.



To show the segment that contains the page:

- **Keep the pointer on the page and hold down the left mouse button.** The segment containing that page is highlighted and remains highlighted as long as you hold down the mouse button. When you release the mouse button, the page remains selected, all the individual pages are again displayed, and the segment is no longer visible.

8.2.2 *Viewing Segments*

The segment viewing mode allows you to view individual segments and the pages that are contained in those segments.

When you select a segment, the segment moves up and over to the left of its original position and is shadowed, giving it a 3-dimensional appearance to indicate that it is selected.

To select a segment:

- **Position the pointer anywhere on the segment that you want to select and click the left mouse button.**

Note – Only one segment can be selected. You cannot select a group of segments. Once you select a segment, the previous segment becomes deselected.

If a segment starts in one row and continues into the next row and the segment does not overlap itself, then a dotted line denotes its continuation. To denote that a segment is being continued into the next row, a dotted line is placed at the segment and a dotted line is placed at the beginning of the segment in the next row. The arrows in Figure 8-3 point to where the segment stops and where the segment is continued.

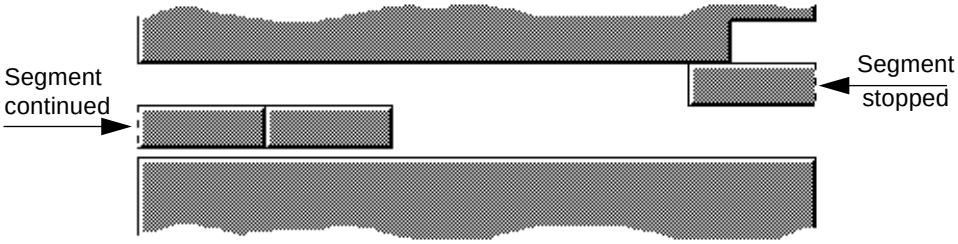
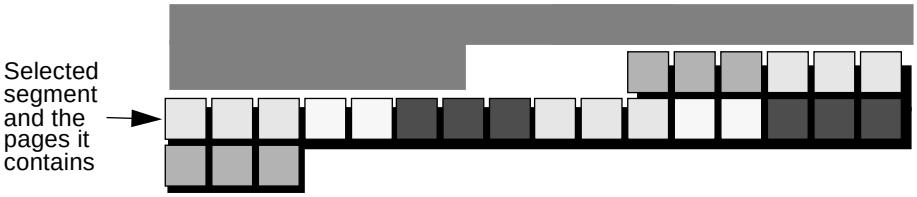


Figure 8-3 Segment Continuation

You can also view the pages contained in the segment.



To show the pages contained in the segment

- **Keep the pointer on the segment and hold down the mouse button.** The pages contained in that segment appear and remain visible as long as you hold down the mouse button. When you release the mouse button, the segment remains selected and displayed, and the individual pages are no longer visible.

8.3 *Properties Windows*

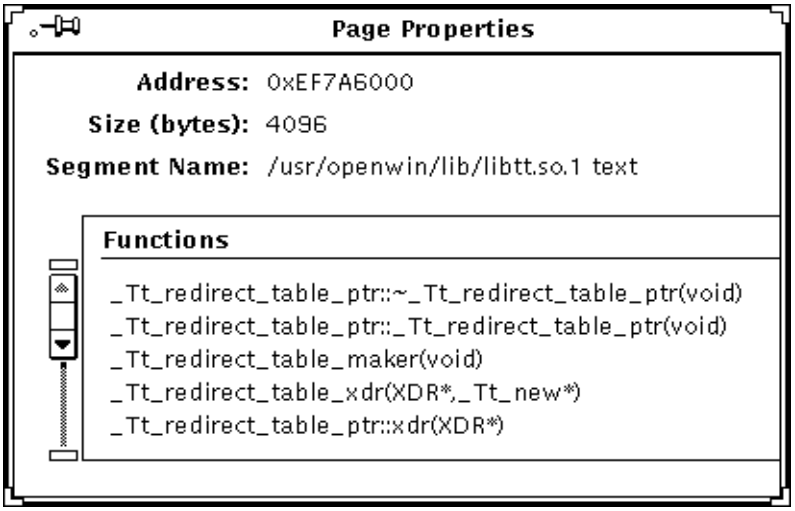
The Page (Figure 8-4) and Segment (Figure 8-5) Properties windows contain the following information:

- Address of the page or segment in hexadecimal
- Size of the page or size range of the segment in bytes
- Functions contained in the page or segment
- Name of the segment

Note – The windows provide information only; you cannot enter any information into these windows.

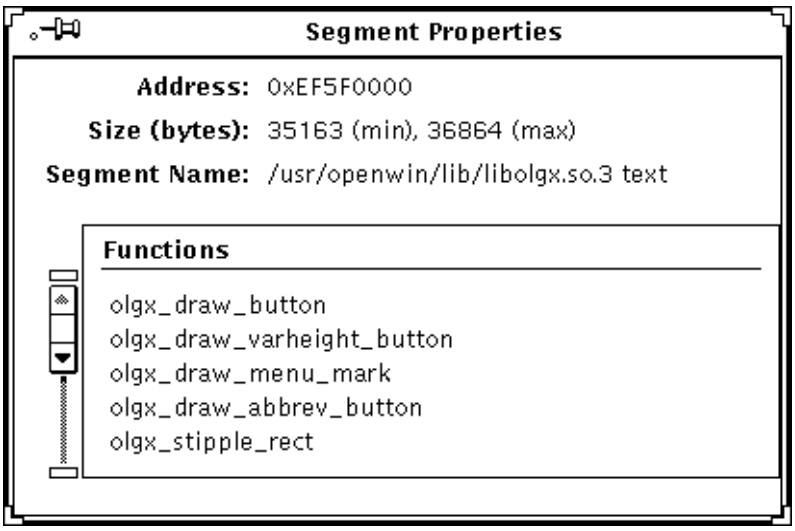
To activate the Page or Segment Properties window:

♦ **Choose Selection from the Props button menu.**



If you are viewing pages, then the Page Properties window is displayed.

Figure 8-4 Page Properties Window



If you are viewing segments, then the Segment Properties window is displayed.

Figure 8-5 Segment Properties Window

8.4 Selecting and Displaying Samples

You can examine an individual sample or a group of samples to view in the Address Space display. When you select a specific sample or group of samples, the display shows the memory usage for that particular sample or group of samples, as opposed to representing the entire experiment.

For example, suppose you have just finished viewing sample 11 in the Averages legend of the Overview display, and you have also viewed sample 11 in the Sample Properties window. The examination of sample 11 shows that most of the application time is spent in the System Wait area. You then select sample 11 and view it in the Address Space display. You might find that the functions that are heavily used are not located on the same page. Frequently called functions not located on the same page increase the size of the working set, which can slow the application. Adjusting the location of those functions so that they are located on the same page can improve the performance of the application.

To select samples, do one of the following:

- **Use the mouse to select samples from the Overview display.**
  Point and click on the sample column that you want to select. You can position the pointer anywhere on the column. The mouse functions for select and adjust (toggle) operate per the OPEN LOOK specification.

  You can click on more than one column; however, if you click on one that is already selected, then that column becomes deselected.

Note – You can select one sample or a group of samples, but you cannot select a portion of a sample.

When you initially load an experiment, the Overview is displayed with all samples selected. Once you make your first sample selection, all the other samples are deselected.

- **Choose Select All or Select None from the View menu to select or deselect all samples in the experiment.**

- **Use the keyboard to type your sample selection in the Samples text field.** Type the number of the sample in the Samples text field located in the bottom left corner of the Analyzer window; press Return after you finish typing your sample selection.

  Sample selections can be any combination, such as those denoted in the following examples:

  Samples: *2*

  Samples: *4,7*

  Samples: *1,3,5-8*

  In the first example, one sample is selected (sample 2). In the second example, two samples are selected (samples 4 and 7). In the last example, six samples are selected, (samples 1, 3, 5, 6, 7, and 8).

Note – When viewing multiple samples, segments might overlap other segments, obscuring them from view. Overlapping segments occur only when the application code contains function calls to `dlopen`(3) and `dlclose`(3), which dynamically link and unlink segments.

The Address Space display does not draw overlapping segments. If an overlap occurs, then the display draws the first segment only and does not draw the overlapping segment. If your application dynamically links and unlinks segments, then view individual samples to avoid overlapping segments.

When you select a sample or a group of samples, the Analyzer displays the time relationship of your sample selection as it relates to the entire experiment. The amount of time is displayed in the experiment scale. See Chapter 6, "*Overview Display,*" for detailed information.

Statistics Display 9

This chapter describes the information contained in the Statistics display. It also describes the features for selecting and displaying performance data.

This chapter is organized into the following sections:

9.1 What Is the Statistics Display?

The Statistics display provides aggregate data about the performance and system resource usage of the application. The Overview display also provides aggregate data, but only process times. The other displays, Histogram, Cumulative Histogram, and Address Space, show the data broken down by program components (such as functions and pages). All data that does not appear in the other displays can be examined in the Statistics display (see Figure 9-1).

This display provides information about the application attributes that are not obvious or visible in any of the other displays of the Analyzer; it is particularly useful when you have preconceived ideas about what the numerical value of these attributes should be. You can compare the actual values to your expected values. Some questions you might want to ask when comparing the values:

- Are any numbers very high or very low?
- Are any of the numbers out of the range of what you expected to see?

Some of the information provided in the Statistics display informs you about the environment of your workstation that can affect the performance of your application. However, any changes to improve performance cannot be done at the application level. For instance, in the case of `process swaps`, a high number can indicate that too many other processes are running, or that the workstation has too little memory.

These types of situations affect the performance of your application. You may or may not be able to change any of these circumstances, but you can use the information as a future reference when you might be able to influence the environment of your workstation.

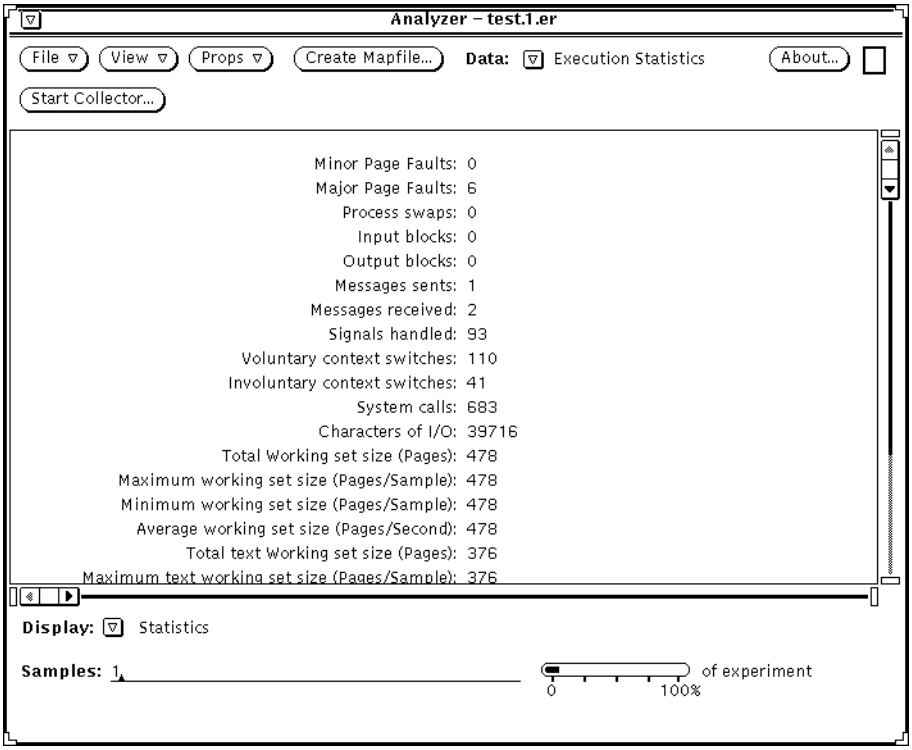


Figure 9-1 Statistics Display

The required data collection parameter for the statistics display is overview data, which is the default data. Overview data is always collected on each experiment; you do not need to set or change any of the other data collection parameters.

The following are definitions of the labels in the Statistics display:

- **Minor Page Faults—**the number of page faults serviced that do not require any physical I/O activity.
- **Major Page Faults—**the number of page faults serviced that require physical I/O activity. Pages serviced include page ahead operations by the kernel. If this number is nonzero, then the overview display shows text or data fault wait time.

For example, reclaiming the page from the free list avoids I/O and generates a minor page fault. Minor page faults occur during process startup as references to pages that are already in memory. For example, if an address space has faults on some executable or shared library, then the result is a minor page fault for the address space. Executing a read or write to a file that is in the page cache also creates a minor page fault.

If input blocks or output blocks are nonzero, then you might see I/O wait time in the Overview display. The operating system kernel does caching of data files; consequently, it is possible that all read and write system calls are serviced by data files that are resident in memory. If this situation exists, then the I/O wait time is zero.

- **Process swaps—**the number of times a process is swapped out of main memory.
- **Input blocks—**the number of times a `read()` system call is performed on a noncharacter or special file.
- **Output blocks—**the number of times a `write()` system call is performed on a noncharacter or special file.
- **Messages sent—**the number of messages sent over sockets.
- **Messages received—**the number of messages received from sockets.
- **Signals handled—**the number of signals delivered or received.

  Although signals are not visible in the other displays, the routines that handle signals are visible. If for example, the Histogram display for the routine that handles a certain signal shows a large time measurement, then you may want to examine the routine more closely and reevaluate how it is being used.

- **Voluntary context switches—**the number of times a context switch occurred because a process voluntarily gave up the processor before its allotted time is completed to wait for availability of a resource.
- **Involuntary context switches—**the number of times a context switch occurred because a higher priority process became runnable or because the current process exceeded its allotted time.
- **System calls—**the total number of system calls.
- **Characters of I/O—**the number of characters transferred in or out to a character device or file by read and write calls.
- **Working set size (Pages)—**the statistics for the total, maximum, minimum, and average size of the working set.
- **Text Working set size (Pages)—**the statistics for the total, maximum, minimum, and average size of the text working set.
- **Non-text Working set size (Pages)—**the statistics for the total, maximum, minimum, and average size of the non-text working set.

9.2 Selecting and Displaying Samples

You can select a sample or a group of samples to view in the Statistics display. When you select a specific sample or group of samples, the display now represents the values of that particular sample or group of samples instead of the entire experiment.

To select samples, do one of the following:

- **Use the mouse to select samples from the Overview display.**
  Point and click on the sample column that you want to select. You can position the pointer anywhere on the column. The mouse functions for select and adjust/toggle operate per the OPEN LOOK GUI.

  You can click on more than one column; however, if you click on one that is already selected, then that column becomes deselected.

Note – You can select one sample, or a combination of samples, but you cannot select a portion of a sample.

When the experiment is initially loaded, the Overview is displayed with all samples selected. Once you make your first sample selection, all the other samples are deselected.

- **Choose Select All or Select None from the View menu to select or deselect all samples in the experiment.**
- **Use the keyboard to type your sample selection in the Samples text field.** Type the number of the sample in the Samples text field located in the bottom left corner of the Analyzer window; press Return after you finish typing your sample selection.

  Sample selections can be any combination, such as those denoted in the following examples:

  Samples: *2*

  Samples: *4,7*

  Samples: *1,3,5-8*

  In the first example, one sample is selected (sample 2). In the second example, two samples are selected (samples 4 and 7). In the last example, six samples are selected, (samples 1, 3, 5, 6, 7, and 8).

When you select a sample or a group of samples, the Analyzer displays the time relationship of your sample selection as it relates to the entire experiment. The amount of time is displayed in the experiment scale. See Section 6.4, “Experiment Scale,” on page 6-61 for detailed information.

9

Part 4 — Advanced Performance

Doing More with the Collector 10

In Chapter 4, “Getting Started with the Collector,” you learned the basics about using the Collector, such as how to activate it and use the default settings. This chapter describes the more detailed and advanced features of the Collector that you can use for specifying and collecting data.

This chapter is organized into the following sections:

Note – You cannot collect profile data on a set user ID or a set group ID application. To collect data on this application, your user ID and group ID must match the ID’s of the application.

For example, if you want to collect profile data on `Blocks`, then your user ID must match `evelynl`.

```
-r-sr-xr-x   1  evelynl   52952  Apr 23 10:36 Blocks
```

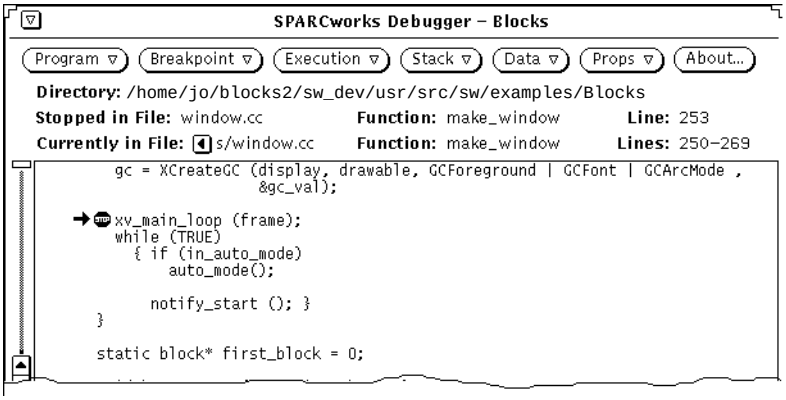
10.1 Setting Breakpoints To Control Data Collection

Using breakpoints, whether performing continuous sampling or manual sampling, allows for an even more detailed control of the information the sample takes. Setting breakpoints is a more exact way of starting and stopping a sample. This method of sampling requires more time and effort because you need to set breakpoints in the application code.

In order to use breakpoints effectively, you must be thoroughly familiar with the application code. When setting breakpoints, keep in mind the following:

- If your application is compiled using the `-g` option, then you can set breakpoints in the specific areas of the code that pertain to the information you want to collect. The breakpoints can be set at the beginning or end of a function or on any line of code.
- If the application is compiled with the `-O` option, then you can only set breakpoints on functions and not on specific source code lines. Refer to the *Debugging a Program* manual.

For example, in Blocks, you could place a breakpoint just before the call to the `xv_main_loop` function. This breakpoint allows you to bypass the initialization process. When you turn on sampling, you collect performance data on the interaction with the Blocks application that occurs between the point after the initialization of the application to the point when you quit the Blocks application.



You can set breakpoints when using either the Continuous sampling mode or the Manual sampling mode.

- **Setting breakpoints with Continuous sampling—**provides a uniform view of the application's behavior. Using breakpoints with continuous sampling provides a uniform view of a specific portion of the application behavior because breakpoints designate specific areas in the code on which to collect information.
- **Setting breakpoints with Manual sampling—**measures human interaction, for example, the time it takes to choose a command from a menu or type a command from the keyboard. Using breakpoints with manual sampling is a more exact method of collecting this information.

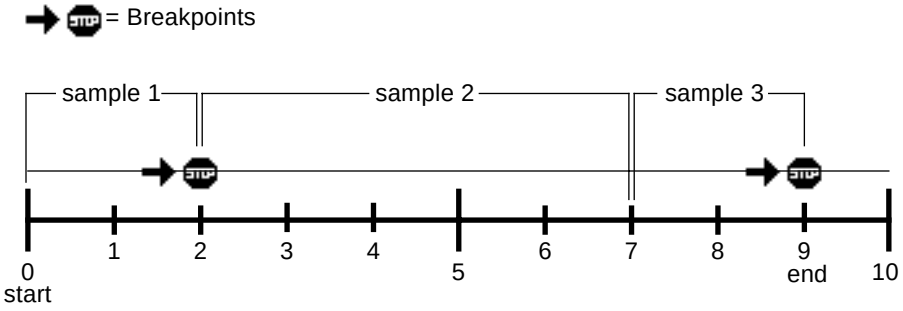
To collect performance data using breakpoints:

1. **Choose a sampling mode: Continuous or Manual.**
2. **Click on the Apply button.**
3. **Set breakpoints in the application code.**
4. **From the Execution menu of the Debugger, select Run.**
   Every time the Debugger encounters a breakpoint, the application process is suspended. To continue the process, you must select Continue, Next, or Step from the Execution menu of the Debugger.

In Manual mode, the data collected between the first breakpoint and the second breakpoint is a sample. When you select Continue, Next, or Step, a new sample is started.

In Continuous mode, samples are continuously collected at specified intervals. For example, suppose you set continuous sampling at an interval of every 5 seconds. The sample contains data for either the entire sample period of 5 seconds or it contains data for the time it takes to encounter the first breakpoint, even when that breakpoint is encountered before the sampling interval is completed.

The following diagram illustrates a continuous 5-second interval.



Using breakpoints with Continuous sampling limits the scope of the collected performance data because breakpoints encountered between the time interval of the sampling period produce small samples. In the diagram, suppose you are sampling the initialization period of the application. If you choose a 5-second interval and the application takes 10 seconds to initialize, then you end up with a total of two samples. However, if you set breakpoints and those breakpoints occur before the 5-second interval is completed, then you end up with three samples.

In the diagram shown:

- Sample 1 occurs before the 5-second interval is completed because it encounters a breakpoint.
- Sample 2 occurs during a 5-second interval because no breakpoint is encountered.
- Sample 3 occurs before the 5-second interval is completed because it encounters a breakpoint.

10.2 Setting Data Collection Parameters

Using the additional data collection parameters in the Collector gives you more control over the performance data you collect. The Histogram, Cumulative Histogram, and Address Space displays require that this additional data be collected. These displays cannot be viewed in the Analyzer unless one of the following performance data is collected:

- Working Set
- Profiling (PC and PC and Stack)

You can collect one, or a combination of the data collection parameters. Choose additional data collection parameters after you have developed more of an understanding of the performance behavior of your application.

The following sections describe the individual data collection parameters available in the Collector and also contain a table outlining the data collection parameter requirements of the displays in the Analyzer.

10.2.1 Working Set Data

The Working Set data collection parameter collects data that assists you in answering questions such as:

- How does the working set change over time?
- Is the application using pages efficiently?
- Which pages are accessed during a particular operation?

To collect Working Set data:

♦ **Choose Working Set from the Address Space setting in the Collector Window.**

Working Set data represents the process state address space as a series of segments, each of which contains a number of pages. The data collected describes the status of each page, and whether it was referenced or modified. Table 10-1 shows the displays available for analysis when you collect Working Set data.

If you select None, then Working Set data is not collected.

Table 10-1 Working Set Data Collection Parameter

Collection Parameter	Display Type	Unit Type	Data Type
Working Set	Address Space	Page Segment	Working Set Working Set

10.2.2 PC Data

The PC profiling data collection parameter is used for collecting data that assists you in answering questions such as:

- How much time is spent by the application in its execution of functions, modules, or segments?
- What functions, modules, or segments are consuming

  The most amount of time?

  The least amount of time?

To collect PC profiling data:

♦ **Choose PC from the Address Space setting in the Collector Window.**

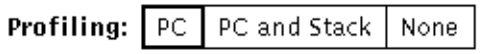


Table 10-2 shows the displays available for analysis when you collect PC profiling data.

Table 10-2 Profiling Data Collection Parameter

Collection Parameter	Display Type	Unit Type	Data Type
PC	Histogram	Function	User, System, Text and Data Page
		Module	User, System, Text and Data Page
		Segment	User, System, Text and Data Page

Note – You must collect PC profiling data to use the reordering (Create Mapfile) feature of the Analyzer. For information on reordering you application, see Section 11.3, “Reordering Your Application,” on page 11-122.

10.2.3 PC and Stack Data

PC and Stack profiling data is used for a more elaborate analysis of the application. The PC and Stack profiling data collection parameter collects data that assists you in answering questions such as *"How much time is spent by functions in the application that call other functions?"*

To collect PC and Stack profiling data:

- **Choose PC and Stack from the Address Space setting in the Collector Window.**

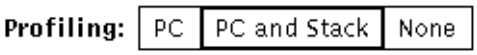


Table 10-3 shows the displays and performance problems available for analysis when you collect PC and Stack profiling data.

Table 10-3 PC and Stack Profiling Data Collection Parameter

Collection Parameter	Display Type	Unit Type	Data Type
PC and Stack	Cumulative Histogram	Function	User, System, Text and Data Page
		Module	System,, Text and Data Page
		Segment	System, Text and Data Page

10.3 The Profiling Timer

The profiling timer is displayed when you choose either PC or PC and Stack profiling data. The default value for the profiling timer is 10 milliseconds. For example, the PC value (or the set of return addresses on the stack) is recorded every 10 milliseconds, or 100 times a second. You can either use the default value or set the value to produce a more detailed sample to meet your analysis needs. See "Setting the Profiling Timer" on page 10-116. Figure 10-1 shows the default setting of the profiling time.

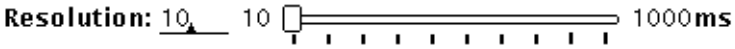


Figure 10-1 Profiling Timer

10.3.1 Displays Requiring Profiling

Certain displays in the Analyzer are only available when profiling is used during the data collection process. The following analyzer displays require profiling:

- User Time as a histogram display by function, module, or segment
- System Wait Time as a histogram display by function, module, or segment
- System Time as a histogram display by function, module, or segment
- Text Page Fault Time as a histogram display by function, module, or segment
- Data Page Fault Time as a histogram display by function, module, or segment

10.3.2 Setting the Profiling Timer

To obtain statistically significant data, you need to collect enough profile packets. Typically, at least 1000 packets should be taken, and, depending on the size and behavior of the application, a larger number may be necessary. For example, suppose in one sample you have multiple profiling packets that depend on a 1-second sample period. If the profiling timer is set to 10 milliseconds during the data collection of this sample, then you can expect the number of profile packets in the sample to be about 100.

You can increase or decrease the number of packets in a sample by increasing or decreasing the value of the profile timer.

- Increasing the profiling timer provides data that is less accurate, but decreases the overhead from data collection.
- Decreasing the profiling timer value provides data that is more accurate, but increases the overhead from data collection.

Samples that contain profiling data cause overhead in the kernel and in the application, so it is better to keep the profile timer value as large as possible while still obtaining sufficient profiling data. Stack profiling gives you more data and allows additional analysis when compared with PC profiling; however, it has a higher overhead.

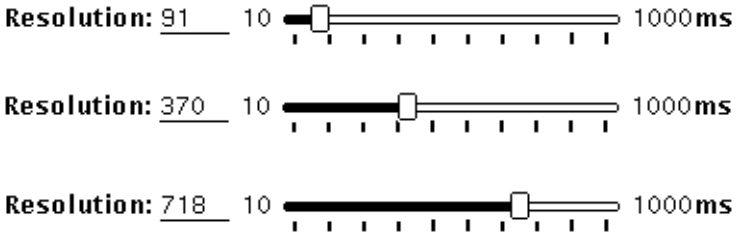
With shorter experiments, you need a smaller profiling timer value. Also, if you plan to look at the profile data from a single sample or set of samples, then the number of profile packets must be large enough for that single sample or set of samples.

Finally, the required number of profile packets is correlated to the number of program units that are being observed during profiling. The more functions, modules, or segments that are active during profiling, the more profile packets you need for statistically significant results.

You can set the profiling timer in one of two ways:

- **Type the value into the Resolution text field.**
- **Use the slide bar on the Profiling Timer.**

Whether you type in the value for the profiling timer, or set the value using the slide bar, the value and the scale always reflect the value located in the Resolution text field. The range for the value of the profiling timer is between 10 and 1000 milliseconds. The following three examples show profiling timer settings.



10

Doing More with the Analyzer 11

In Chapter 5, “Getting Started with the Analyzer,” you learned the basics about using the Analyzer, such as how to activate it, the different displays it supports for viewing performance data, and how to choose and analyze data types. This chapter describes the more detailed and advanced features of the Analyzer that you can use for viewing and examining the performance data.

This chapter is organized into the following sections:

11.1 Viewing Multiple Displays

A convenient feature of the Analyzer is that it allows you to view multiple displays simultaneously. Viewing multiple displays lets you visually compare the samples contained in the experiment. When you view multiple displays simultaneously, you can:

- View different sets of samples in multiple copies in the same type of display. For example, you can view samples 8 and 9 as Histogram by Function, and also view samples 11 and 12 as Histogram by Function.

- View the same set of samples, but in different types of displays. For example, you can view sample 8 in a Histogram display, and also view sample 8 in the Address Space display.

Note – Viewing multiple displays allows you to examine only those samples contained in the experiment that you loaded in the Analyzer. If you need to examine or compare the samples of one experiment with the samples of another experiment, then activate a second Analyzer and load a different experiment.

To activate a second Analyzer:

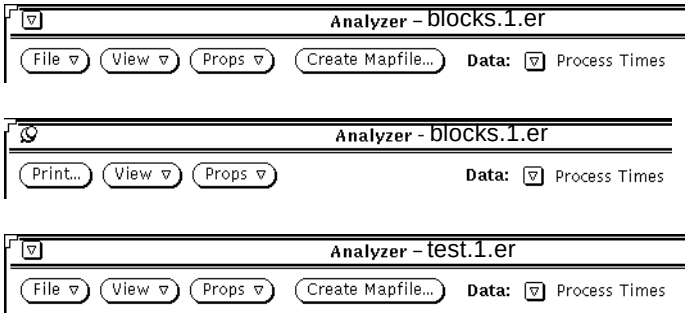
1. **Keep the current Analyzer activated.**
2. **Activate a second Analyzer by double-clicking on the Analyzer icon in the SPARCworks/ProWorks Manager.**
3. **Load the experiment you need.**

You can use multiple Analyzers to compare the performance behavior of the original application with the performance of the reordered application. See Section 11.3, “Reordering Your Application,” on page 11-122.

When viewing multiple displays, you can have several Analyzers activated and you can also have many subwindows opened. To help keep track of what experiment is being viewed, the experiment name appears in the header area of each window.

For example, suppose you have two Analyzers activated; one Analyzer contains `blocks.1.er` and the other Analyzer contains `test.1.er`. You also open a subwindow from the Analyzer containing `blocks.1.er`. You now

have a total of three windows; two of the window headers contain the experiment name `blocks.1.er`, and one window contains the experiment name `test.1.er`.



To view multiple displays:

- **Choose New Window from the View menu.** Another Analyzer window is displayed. Choose new data types and displays to examine in the new window. You can activate a new window from the Analyzer base window or from a new

---

Note – The original Analyzer window must remain open. If you close or quit the original window, then you also close or quit all subsequent windows.

The display in the new window does not inherit the settings of the display in the window from which you choose New Window; the new window contains the default display. For example, if you are viewing a cumulative histogram by module and choose New Window, the new window displays a histogram by function, which is the default Histogram display. In the new window, you can change the display and data types, and view the same samples or different samples.

Except for the File, Create Mapfile, and Start Collector buttons, the new window provides the same functions as the Analyzer base window. See Figure 11-1 and Figure 11-2.

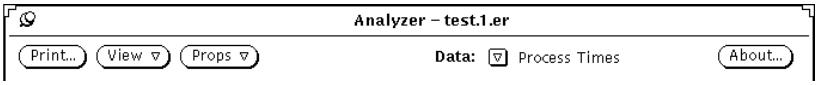


Figure 11-1 Analyzer Base Window Menu Buttons

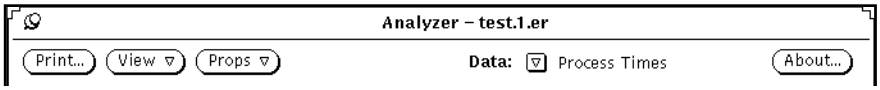


Figure 11-2 Analyzer New Window Menu Buttons

11.2 Closing and Quitting Windows

You cannot close a new window into an icon; you can only dismiss the window. However, once you dismiss the window, all the settings in that window are also dismissed and cannot be retrieved. If you wish to view the settings and display them again, then you need to activate a new window and choose the settings you want to view.

If you close the Analyzer base window, then all new windows also close. However, the settings in these windows are saved. When you reopen the Analyzer base window, all its associated windows reappear with their settings and displays unchanged.

11.3 Reordering Your Application

You can reorder an application to reduce the text working set size. You must collect profiling data during the data collection process to reorder your application. For reordering to work, files must be compiled with the `-xF` option.

Note – Reordering is useful only when the application text page fault time is consuming a large percentage of the application time. If text page fault time is not consuming a large percentage of the application, then reordering may not improve the overall performance of the application. Use the Overview display to view the percentage of text page fault time.

If you get the following warning message:

```
ld: warning: mapfile: text: .text%function_name:
object_file_name:
Entrance criteria not met
```

then this message indicates that the named file, *`function_name`*, has not been compiled with the `-xF` option.

Check any files that are statically linked, such as unshared object and library files, because these files may not be compiled with the `-xF` option. If you cannot compile a file with the `-xF` option because it is an unshared object or library file, then try to use a shared version of that object or library file. If you are unable either to obtain a shared version of the file or to compile it with the `-xF` option to statically link the file, then you can still continue with the reordering of the application; however, any functions mentioned in the warning message will not be reordered.

To reorder the application:

1. **Compile the application using the `-xF` option.**
   The `-xF` option is required for reordering. This option causes the compiler to generate functions that can be relocated independently.

   For C applications, type:
   ```
   cc -g -xF -c a.c b.c
   cc -o application_name a.o b.o
   ```

   For C++ applications, type:
   ```
   CC -g -xF -c a.cc b.cc
   CC -o application_name a.o b.o
   ```

   For Fortran applications, type:
   ```
   f77 -g -xF -c a.f b.f
   f77 -o application_name a.o b.o
   ```

   For Pascal applications, type:
   ```
   pc -g -xF -c a.p b.p
   pc -o application_name a.o b.o
   ```

2. **Load the application in the Debugger.**

3. **Activate the Collector to collect profiling data.**

4. **Run the application in the Debugger.**

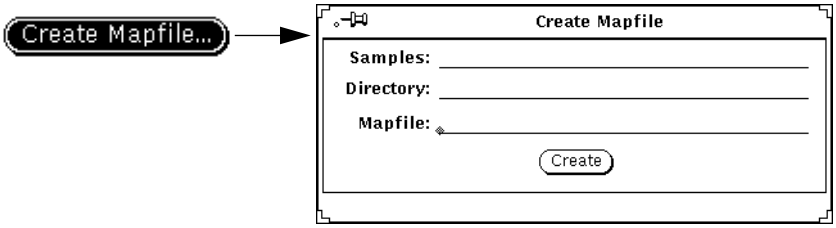
5. **Activate the Analyzer and load the specified experiment.**

6. **Click on the Create Mapfile button and enter the necessary information into the Mapfile window; click on the Create button.**
   The mapfile is dependent on the sample selection. For example, if you choose samples 17 through 31, then the mapfile contains only those functions contained in samples 17 through 31.

The mapfile contains names only of functions that have some User CPU time associated with them. The mapfile specifies a function ordering that reduces the size of the text working set. The default reordering strategy is based on profiling data and function sizes sorted in descending order. The strategy of ordering functions in descending order is to reduce the text working set size. Reduction occurs because infrequently used functions are grouped on a page and are rarely part of the text working set. All functions not listed in the mapfile are placed after the listed functions.

Once you have examined all the data and have a better understanding of what areas are performance problems, you can reorder your application. You do not need to manually change the code of your application to reorder it. The Analyzer creates a mapfile for text reordering that is specific to the experiment that is currently loaded in the Analyzer.

7. **Choose Create Mapfile to create a reordered map.**
   Create a new executable using the mapfile and linker.



   You can also use the `er_mapgen` utility to create a mapfile without using the Analyzer. See the man page for `er_mapgen`.

8. **Link the application using the new mapfile.**
   For C applications, type:
   `cc -Wl -M` *mapfile_name* `a.o b.o`

   For C++ applications, type:
   `CC -qoption ld -M`*mapfile_name* `a.o b.o`

   For C applications, the `-M` causes the compiler to pass `-M`*mapfile_name* to the linker.

   For Fortran applications, type:
   `f77 -qoption ld -M`*mapfile_name* `a.o b.o`

   For Pascal applications, type:
   `pc -qoption ld -M`*mapfile_name* `a.o b.o`

See also the man page for `er_mapgen`, which is a utility that generates a mapfile from an experiment that has been generated by the Collector.

11.4 Comparing the Reordered Application

Once you have reordered you application, you can compare the original application with the reordered application to see if there are any changes or improvements to the performance of the application.

To compare the performances:

1. **Run the reordered application.**
   Use the new executable (created in step 7 in the previous section) and collect the same data that you collected for the first experiment.
2. **Activate another Analyzer and load the new experiment.**
3. **Examine the displays.**

- Use the Overview display to compare the original application with the reordered application.
- Use the Statistics display to see if the major page faults, the average text working set size, and the total text working set size is reduced.
- Use the Address Space display to examine the memory referencing behavior of the application in more detail.

11.5 Exporting an Experiment

Export writes the data of your experiment into an ASCII format file. The data generated by Export can be used by other programs. Suppose you want to view your experiment data in a display that is not provided by the Analyzer, such as a pie chart. You can write your own application for displaying the data in a pie chart form, and use the data from the *export experiment* file in the pie chart.

To export an experiment:

- **Choose Export from the File button menu, enter a directory and a file name, and click on the Export button.**

See Appendix D, "Export Experiment File," for detailed information about the format of the Export Experiment file.

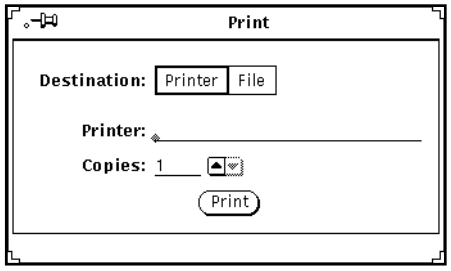
11.6 Printing Experiment Information

The Analyzer has two print options: Print and Print Summary

Print

The Print option prints an ASCII text version of the display you are currently viewing. You can print the data as a hardcopy or print the data to a file.

To print the current display:



- **Choose Print from the File button menu and enter the necessary information.**

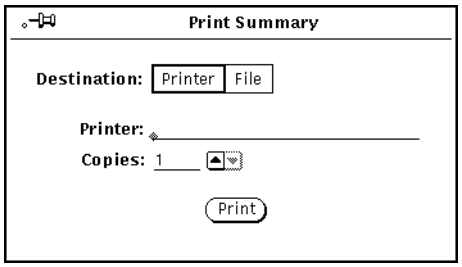
See Appendix E, "Print File," for detailed information about the format of the Print file. See also `er_print` man page.

Print Summary

The Print Summary option prints a textual overview of the experiment. This option produces an ASCII text file that has information about the experiment, such as:

- Averages of all sample time for each type of data: Working Set, PC profiling, and PC and Stack profiling
- Most frequently used functions, modules, and segments.
- Least frequently used functions, modules, and segments.

To get a Print Summary:



♦ **Choose Print Summary from the File button menu and enter the necessary information.**

See Appendix F, "Print Summary File," for detailed information about the format of the Print Summary file.

Part 5 — Appendixes

Troubleshooting A

This appendix describes how to overcome problems using the Analyzer.

This appendix is organized into the following sections:

A.1 Troubleshooting Checklist

If you are having problems using the Analyzer, then check for the following:

- ❏ Verify that the SPARCworks/ProWorks Manager is installed in the standard location. If you do not know where the SPARCworks/ProWorks Manager should be installed, contact your system administrator.
- ❏ Verify that `analyzer` is installed in the standard location. If you do not know where `analyzer` should be installed, contact your system administrator.
- ❏ Verify that `analyzer` can be found in your execution search path (as set by the `PATH` variable).

Problems specific to the Collector

- Collector menu items are disabled (shaded gray) and cannot be selected. This problem occurs when there is a version mismatch between dbx and the Debugger. Verify that the versions for dbx and the Debugger are from the same SPARCworks/ProWorks release.

- Cannot select Collector from the Execution button menu in Debugger. Your window system may be running out of resources. Kill any unnecessary processes, close or quit any unnecessary tools or windows, and try again. If you are still unable to select the Collector, then contact your system administrator. Also, verify that you have *ttsession* running on your system.

Problems specific to the Analyzer

- Cannot activate the Analyzer.
  Your window system may be running out of resources. Kill any unnecessary processes, close or quit any unnecessary tools or windows, and try again. If you are still unable to activate the Analyzer, contact your system administrator.

- Cannot load an experiment.
  The data in the experiment file may be corrupted. Redo the experiment and try again.

  If you entered the directory and experiment name from the keyboard, then verify that you have correctly typed the information.

  Verify that the *pointer file* for the experiment is pointing to the directory where the experiment is located. The name of the *pointer file* is the experiment name without a preceding dot (`.`). For example, if the experiment you want to load is `.test.1.er`, then the *pointer file* is `test.1.er`.

  Check the *pointer file* and verify that it contains the correct path for the experiment.

- You have compiled all your files with the `-xF` option, but when you reorder your application, you get the following error message:

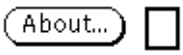
```
ld: warning: mapfile: text: .text%function_name:
object_file_name:
Entrance criteria not met
```

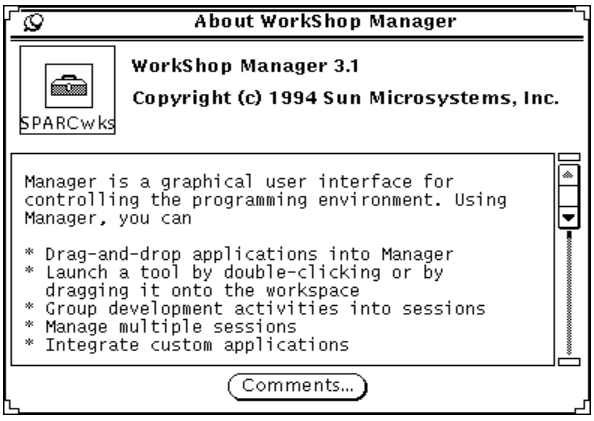
This message indicates that the named file is not compiled with the `-xF` option. Check any files that are statically linked, such as unshared object and library files, because these files may not be compiled with the `-xF` option. If you cannot compile a file with the `-xF` option because it is an unshared object or library file, then try to use a shared version of that object or library file. If you are unable to obtain a shared version of the file to compile it with the `-xF` option, then you can still continue with the reordering of the application; however, any functions mentioned in the error message are not reordered.

A.2 Reporting Problems

If you have gone through the checklist and still have problems using the Analyzer, then call Sun Microsystems at 1-800-USA-4SUN or call your local service office. You must provide the version number to the dispatcher.

To view the current version of the Analyzer, click on the About button on the Analyzer base window.





A.3 Error Messages

The Analyzer displays error messages that provide you with information or that inform you of an error. Error messages for the Analyzer appear in the footer of the window. This section lists and describes the error messages and gives suggestions about how to correct the problem.

`Cannot load experiment.`

The Analyzer is unable to load the experiment. Verify that the directory that contains the experiment is specified correctly. If the Analyzer is still unable to load the experiment, then the data in the experiment may be corrupted. Redo the experiment.

`Dlopen of collector library has failed` (Debugger message)

This error can be caused because the `libcollector.so` uses a symbol called `setitimer`. If your program makes no reference to this symbol, it is not statically linked in. As a result, when `dbx` attempts to `dlopen libcollector.so`, it cannot resolve the reference to `setitimer` and the `dlopen` fails.

To fix this problem, link in `libcollector.o` when creating the executable. The `.o` is found in the same directory as `libcollector.so`.

`Export failed.`

The Analyzer cannot find the `er_export` file in the installed `bin` directory. Verify that the `er_export` file is in the `bin` directory.
The Analyzer is unable to export the data to a file because the data is corrupted. You must redo the experiment.

`Incomplete experiment.`

The *journal file* is not complete or the experiment is corrupted. You need to recreate the experiment.

`Incomplete pointer file.`

The *pointer file* is missing one of the four fields: magic number, version, hostname, and experiment name. Verify that the *pointer file* contains the four fields. If any of the information is missing, then edit the *pointer file*. After you edit the file, load the experiment.

`Mapfile creation failed:  all zero values in profile data.`

PC (program counter) profiling data is required for reordering your application. Verify that the you have collected PC data for this experiment. If the experiment contains PC data, then verify that the samples selected contain PC data. If the experiment contains PC data, but your current sample selection does not contain PC data, then you need to choose samples that contain PC data.

`Mapfile creation failed: no profile data available.`

PC (program counter) profiling data is required for reordering your application. Verify that you have collected PC data for this experiment. If the experiment contains PC data, then verify that the samples selected contain PC data. If the experiment contains PC data, but your current sample selection does not contain PC data, then you need to choose samples that contain PC data.

`Mapfile creation failed:  unable to open file.`

Your system might not have enough disk space or swap space. Contact your system administrator. Also, check the permissions of the directory to which you are writing the mapfile; verify that the directory has write permissions.

`No directory specified.`

You did not specify a directory. You must specify the directory name that contains the experiment.

`No file specified.`

You did not enter a file name in the Export Experiment pop-up window or in the Print Summary pop-up window. You must specify a file name to which Export or Print Summary can write the data.

`No mapfile specified.`

You did not enter a name in the Create Mapfile pop-up window. You must specify a name for the mapfile that you are creating.

`No sample selected.`

You are trying to display either a Histogram, Cumulative Histogram, Address Space, or Statistics display, and did not specify a sample in the Sample text field; or, you deselected all the samples in the Overview display. You must select a sample.

`No such directory.`

The directory name you specified is either incorrect or does not exist. Verify that the directory specified exists and that you have correctly typed the name.

`No such pointer file.`

A *pointer file* does not exist for the experiment being loaded. Verify that the experiment exists and that you have correctly typed the experiment name. If the experiment does not exist, then you need to create the experiment by first collecting performance data using the Collector.

`Not an experiment.`

The Analyzer does not find a *pointer file* for the experiment. Verify that the experiment exists and that you have correctly typed the experiment name. If the experiment does not exist, then you need to create the experiment by first collecting performance data using the Collector.

`Overview data not available. Experiment must be regenerated.`

Since Overview data is the default data collection parameter, overview data is always collected. If you receive this error message, then the data in the experiment may be corrupted. Redo the experiment.

`Print failed.`

The Analyzer cannot find the `er_export` file in the installed `bin` directory. Verify that the `er_export` file is in the `bin` directory.
The Analyzer is unable to print the data to a file because the data is corrupted. You must redo the experiment.

`Profiling data not available for this sample selection.`

You are trying to display a Histogram for an experiment that does not contain profiling data. Redo the experiment and collect profiling data or choose another experiment.

You are trying to display a Histogram for a sample or sample group that does not contain profiling data. Choose another sample or sample group.

`Range out of bounds: Please enter new range.`

The sample number that you entered in the Sample text field is incorrect, or does not exist. Verify that the number you typed is a sample number that is contained in the experiment. For example, the experiment you are examining contains 53 samples and you accidently typed a sample number that is greater than 53.

`Sample selection unrecognized: Please re-enter.`

You have entered a number that is nonnumeric, such as 1a3. Only numbers, commas, and dashes are allowed for specifying samples. For example: 2,3,4 or 5-8. Correct your entry and try again.

`Sizes data not available. Experiment must be regenerated.`

To view Program Sizes as a Histogram display, Overview data must be collected. Since Overview data is the default data collection parameter, it is always collected. If you receive this error message, then the data in the experiment may be corrupted. Redo the experiment.

`Statistics data not available. Experiment must be regenerated.`

To view the Statistics display, Overview data must be collected. Since Overview data is the default data collection parameter, it is always collected. If you receive this error message, then the data in the experiment may be corrupted. Redo the experiment.

`Unable to create display.`

The display cannot be created because of lack of resources. Contact your system administrator.

`Unable to create display.  Not enough resources.`

The display cannot be created because the system ran out of swap space. Kill any unnecessary processes and try again to view the display.

`Unable to identify display.  No data available for this sample selection.`

The variable that identifies the display is corrupted or the required data was not collected to view the display. Redo the experiment.

`Unable to open journal file.`

The experiment directory does not contain a *journal file* or the *journal file* is unreadable. You need to recreate the experiment.

`Unrecognized version number; expected` *`(current version number displayed)`*`.`

The version number of the experiment is not correct. Verify that the experiment exists and that you have correctly typed the experiment name. If the experiment does not exist, then you need to create the experiment by first collecting performance data using the Collector.

`Working set data not available for this sample selection.`

The sample selection you made does not contain working set data. Choose another sample or sample group, or redo the experiment, collecting working set data.

Data Collection Requirements for the Displays

B

Each display requires the collection of specific data collection parameters. If the required data is not collected, then you cannot view that display. Table B-1 outlines the displays, and the data collection parameters required to view the displays.

Table B-1 Collection Requirements for the Analyzer Display

Data Type	**Display Type**	**Unit Type**	**Data Collection Parameter**
Process	Overview		Overview (default)
User	Histogram	Function	PC
		Module	PC
		Segment	PC
	Cumulative Histogram	Function	PC and Stack
		Module	PC and Stack
		Segment	PC and Stack
I/O	Cumulative Histogram	Function	PC and Stack
		Module	PC and Stack
		Segment	PC and Stack

Data Type	Display Type	Unit Type	Data Collection Parameter
System	Histogram	Function	PC
		Module	PC
		Segment	PC
	Cumulative Histogram	Function	PC and Stack
		Module	PC and stack
		Segment	PC and stack
Text Page	Histogram	Function	PC
		Module	PC
		Segment	PC
	Cumulative Histogram	Function	PC and Stack
		Module	PC and Stack
		Segment	PC and Stack
Data Page	Histogram	Function	PC
		Module	PC
		Segment	PC
	Cumulative Histogram	Function	PC
		Module	PC
		Segment	PC
Program Sizes	Histogram	Function	Overview (default)
		Module	Overview (default)
		Segment	Overview (default)
Working Set	Address Space	Page	Working Set
		Segment	Working Set
Execution Statistics	Statistics		Overview (default)

Experiment Record C

Each experiment has a *pointer file* and a *hidden directory*. The *pointer file* and *hidden directory* are referred to as the *experiment record*.

This Appendix is organized into the following sections:

C.1 Hidden Directory and Pointer File

The *hidden directory* contains a number of files. The directory should be accessible under a naming structure which is the same on both the monitoring machine and the machine running the target application. For example, the directory `/performtune/mystuff` contains the experiment *pointer file* `test.1.er`, and also contains the *hidden directory* `.test.1.er`. The following information is contained in *pointer file* `test.1.er`.

```
MAGIC_NUMBER
VERSION_NUMBER
HOST_NAME
~/perform/text/mystuff/.test.1.er
```

or `./.test.1.er` (the current directory)

Table C-1 lists the files that are contained in the experiment directory.

C.2 Hidden Directory Files

The files contained in the *hidden directory* are written in one of two formats and these formats can be ported across architectures.

- Binary data
- ASCII text

The following files are contained in the *hidden directory*:

- *Journal file* and its records
- Data formats
-

Table C-1 Hidden Directory Files

Name	Format	Description
journal	Text	Contains pointers to data records.
segments	Binary	Segment information from the Debugger.
modules	Binary	Module information from the Debugger.
functions	Binary	Function information from the Debugger.
lines	Binary	Line information from the Debugger.
strings	Binary	Contains the string constants for four files.
overview	Binary	Configuration and sample information.
profile	Binary	Stack back trace information.
working_set	Binary	Referenced and modified bits for all pages.

C.2.1 Journal File

The data collection process writes the *journal file*. When data is collected, entries are made into a *journal file*, which is a single straight ASCII text file. Each entry (*journal record*) has a time stamp associated with it and begins with a keyword that describes the entry type. Some records are complete and self-contained, while others are pointers to data kept in other files. For example, a *journal file* can contain the following information:

- System configuration information

- Symbol table information
- Data collection parameters
- Start time and end time of a sample, and the type of sample data collected
- Exit statement (the time that elapsed since beginning data collection)

The following is a sample of a *journal file*.

```
hostname
version 1
hostname “name”
username "user”
pagesize 4096
dbxpid 770
architecture "sun4c"
os "SunOS 5.0 jup-alpha-5.2_[ON-6.3]"
memory 0
hirestick 1000000
string 8 19193
segment 8 72
module 8 8608
function 8 52272
line 8 24
stackbase 0xf0000000
run 168585.308159000 779
profile_mode stack
profile_timer 10000
sample_mode continuous
sample_period 1000000
working_set_mode on
sample_on 0.000000000
overview 8 256
sample_new 0.000000000 0
profile 8 1528
overview 264 256
working_set 8 924 852
sample_off 0.440290000
```

```
exit 0.440290000
```

C.2.2 Data Formats

Data records are written in binary format. The following information is contained in a data record:

- `MAGIC_NUMBER`
- `VERSION_NUMBER`
- A number of data records

The first datum in any record is an integer `magic_number` that specifies the record type. This number is used for internal consistency checking and to determine the parsing of subsequent data.

C.3 File Management

The files in the *experiment record* directory can be moved, deleted, and printed. Refer to the following man pages:

- `er_mv` is a utility that moves the experiment in the file system.
- `er_rm` is a utility that removes one or more experiments.
- `er_print` is a utility that generates an ASCII version of the various displays of the Analyzer.

Note – Do not use the standard UNIX `mv` and `rm` to move and remove experiments because these commands do not move and remove the *pointer file* and *hidden directory*. Also, you cannot move experiments between file systems, you can only move experiments within the same file system.

Export Experiment File

D

This appendix describes the contents of an *export experiment* file and provides a sample of an *export experiment* file. For detailed information about the format, see Appendix B.

This appendix is organized into the following sections:

See also:

- Section 11.5, “Exporting an Experiment,” for information about the Export option.
- The man page for `er_export`.

D.1 Contents of the Export Experiment File

The Export Experiment option writes the unprocessed data into a file; this data can be used by other applications. The file contains:

- System configuration information
- Symbol table information
- Data collection parameters
- Start time and end time of sample, and the type of sample data collected
- Exit statement (the time that elapsed since beginning data collection)

D.2 Sample Export Experiment File

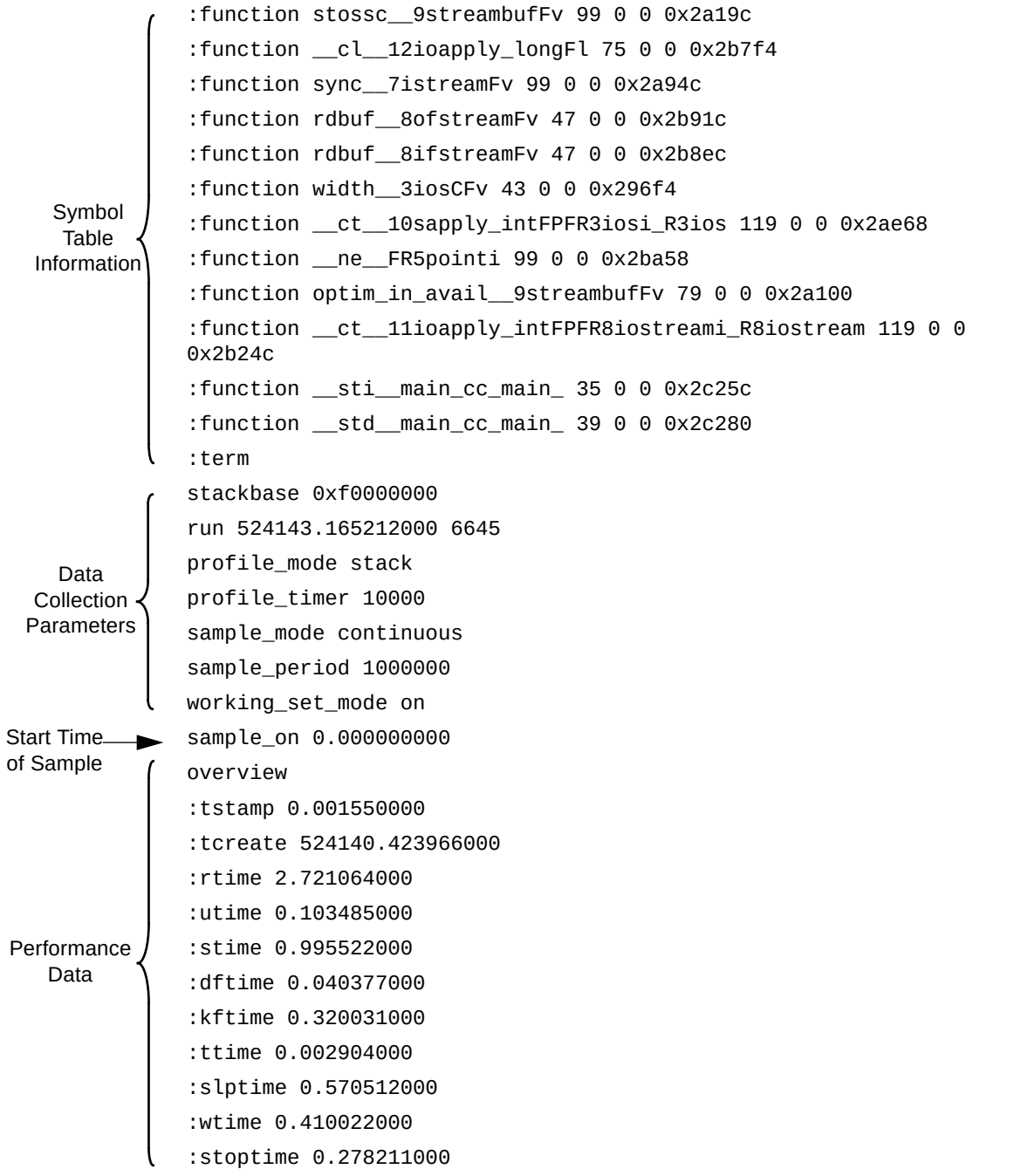
An experiment contains numerous samples and the *export experiment* file can be very lengthy. Because the *export experiment* file can be lengthy, the following sample shows the information of only one sample contained in the *export experiment record* file.

System Configuration Information

```
Experiment: blocks.1.er
sunperf
version 2
hostname "dragonfire"
username "jkchan"
pagesize 4096
dbxpid 6639
architecture "sun4c"
os "SunOS 5.0"
```

Symbol Table Information

```
:text_segment
:segment "/set/sqe/sqe6/ronnie/sophias/demo-jup/Blocks/Blocks"
0x10034 0x26b1b
:module "hand.o" "" cplus fragmented
:function __cl__10oapply_intFi 75 0 0 0x2373c
:function __cl__10iapply_intFi 75 0 0 0x235f0
:function __cl__10sapply_intFi 75 0 0 0x234a4
:function type__5wedgeFv 43 0 0 0x24438
:function type__5brickFv 43 0 0 0x2440c
:function type__5tableFv 43 0 0 0x243e0
:function type__5blockFv 43 0 0 0x240ac
:function rdstate__3iosCFv 43 0 0 0x21dac
.
.
.
:function get__7istreamFRc 163 0 0 0x22d34
:function setstate__3iosFi 83 0 0 0x22008
:function peek__7istreamFv 99 0 0 0x22e6c
```


Symbol Table Information
:function stossc__9streambufFv 99 0 0 0x2a19c
:function __cl__12ioapply_longFl 75 0 0 0x2b7f4
:function sync__7istreamFv 99 0 0 0x2a94c
:function rdbuf__8ofstreamFv 47 0 0 0x2b91c
:function rdbuf__8ifstreamFv 47 0 0 0x2b8ec
:function width__3iosCFv 43 0 0 0x296f4
:function __ct__10sapply_intFPFR3iosi_R3ios 119 0 0 0x2ae68
:function __ne__FR5pointi 99 0 0 0x2ba58
:function optim_in_avail__9streambufFv 79 0 0 0x2a100
:function __ct__11ioapply_intFPFR8iostreami_R8iostream 119 0 0 0x2b24c
:function __sti__main_cc_main_ 35 0 0 0x2c25c
:function __std__main_cc_main_ 39 0 0 0x2c280
:term
Data Collection Parameters
stackbase 0xf0000000
run 524143.165212000 6645
profile_mode stack
profile_timer 10000
sample_mode continuous
sample_period 1000000
working_set_mode on
Start Time of Sample
sample_on 0.000000000
Performance Data
overview
:tstamp 0.001550000
:tcreate 524140.423966000
:rtime 2.721064000
:utime 0.103485000
:stime 0.995522000
:dftime 0.040377000
:kftime 0.320031000
:ttime 0.002904000
:slptime 0.570512000
:wtime 0.410022000
:stoptime 0.278211000

Performance Data

```
:majf 9
:vctx 75
:ictx 67
:sysc 124
:term
sample_new 0.000000000 0
profile
:packet tstamp:524143.451984000 stoptime:4 fault:3 faddr:4018110476
:address _start+0
:packet tstamp:524143.462441000 stime:1 fault:11 faddr:4016804808
:address xv_init+176 make_window__FiPPcT2+56 main+808 _start+92
:packet tstamp:524143.470981000 stime:1 fault:11 faddr:4014922844
:address XSetIOErrorHandler+0 make_window__FiPPcT2+56 main+808
_start+92
:packet tstamp:524143.481247000 utime:1 fault:11 faddr:4015677440
:address ntfy_new_nclient+48 notify_set_destroy_func+68
generic_init+252 xv_create_avlist+496 xv_create+80
:address xv_init+2288 make_window__FiPPcT2+56 main+808 _start+92
:packet tstamp:524143.490827000 stime:1 fault:11 faddr:4014882052
:address _XrmInternalStringToQuark+0 XrmGetResource+8
defaults_get_string+48 server_init+248 xv_create_avlist+496
:address xv_create+80 xv_init+2288 make_window__FiPPcT2+56 main+808
_start+92
:packet tstamp:524143.501484000 stime:1 fault:11 faddr:4014578668
:address fopen+0 setnetconfig+40 __rpc_getconfip+72
gethostbyname+52 _XConnectDisplay+776
:address XOpenDisplay+304 server_init_x+20 server_init+456
xv_create_avlist+496 xv_create+80
:address xv_init+2288 make_window__FiPPcT2+56 main+808 _start+92
.
.
.
:packet tstamp:524150.716184000 tftime:5 wtime:10 fault:11
faddr:4016725900
:address 4016725900 server_init+3408 xv_create_avlist+496
xv_create+80 xv_init+2288
```

Performance Data

```
:address make_window__FiPPcT2+56 main+808 _start+92
:packet tstamp:524150.836189000 dftime:12 fault:11 faddr:4016783756
:address xv_set_pkg_avlist+16 xv_set+208 server_init+3408
xv_create_avlist+496 xv_create+80
:address xv_init+2288 make_window__FiPPcT2+56 main+808 _start+92
:packet tstamp:524150.905042000 dftime:7 fault:11 faddr:4016773552
:address xv_set_pkg_avlist+88 xv_set+208 server_init+3408
xv_create_avlist+496 xv_create+80
:address xv_init+2288 make_window__FiPPcT2+56 main+808 _start+92
:packet tstamp:524150.974802000 dftime:7 fault:11 faddr:300840
:address find_node+4 xv_set_pkg_avlist+40 xv_set+208
server_init+3408 xv_create_avlist+496
:address xv_create+80 xv_init+2288 make_window__FiPPcT2+56 main+808
_start+92
:term
overview
:tstamp 7.859696000
:rtime 7.859696000
:utime 0.139702000
:stime 0.290953000
:tftime 0.149058000
:dftime 0.631454000
:kftime 0.105085000
:ttime 0.014267000
:slptime 0.297467000
:wtime 0.390037000
:stoptime 5.841673000
:majf 25
:nswap 1
:msnd 4
:mrcv 9
:sigs 79
:vctx 62
:ictx 96
:sysc 364
```

Performance Data

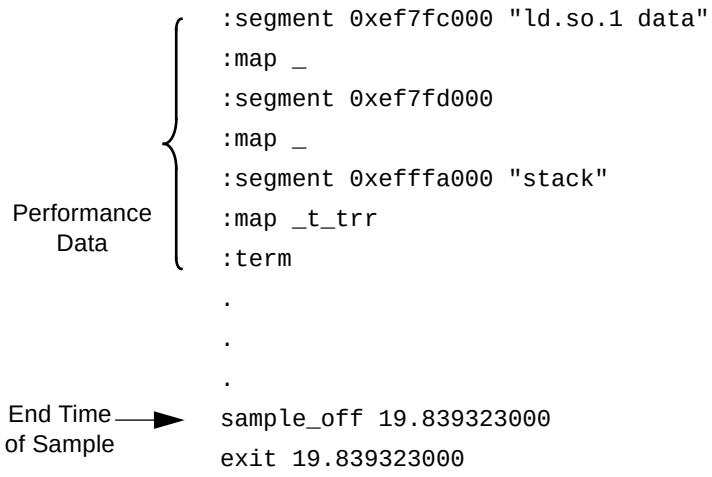
```
:ioch 7469
:term
working_set
:segment 0x10000 "/set/sqe/sqe6/ronnie/sophias/demo-
jup/Blocks/Blocks text"
:map _____________________________
:map _________
:segment 0x46000 "/set/sqe/sqe6/ronnie/sophias/demo-
jup/Blocks/Blocks data"
:map __t
:segment 0x49000 "/set/sqe/sqe6/ronnie/sophias/demo-
jup/Blocks/Blocks bss"
:map rr__t__________t_____________
:map ________________________
:segment 0xef330000 "/usr/lib/straddr.so text"
:map __
:segment 0xef341000 "/usr/lib/straddr.so data"
:map _
:segment 0xef360000 "/usr/lib/nswnis.so text"
:map _______
:segment 0xef376000 "/usr/lib/nswnis.so data"
:map _
:segment 0xef390000 "/usr/lib/switch.so text"
:map ___
:segment 0xef3a2000 "/usr/lib/switch.so data"
:map _
:segment 0xef3c0000 "/usr/lib/libw.so.1 text"
:map _____
:segment 0xef3d4000 "/usr/lib/libw.so.1 data"
:map _
:segment 0xef3f0000 "/usr/lib/libintl.so.1 text"
:map __
:segment 0xef401000 "/usr/lib/libintl.so.1 data"
:map __
```

Performance Data

```
:segment 0xef420000 "/usr/lib/libdl.so.1 text"
:map _
:segment 0xef430000 "/usr/lib/libdl.so.1 data"
:map _
:segment 0xef450000 "/usr/lib/libc.so.1 text"
:map ________________rr_r___________
:map ________rrr___________r______
:map ____________________________
:segment 0xef4b5000 "/usr/lib/libc.so.1 data"
:map r_r_r
:segment 0xef4ba000 "/usr/lib/libc.so.1 bss"
:map __
:segment 0xef4d0000 "/usr/openwin/lib/libX11.so.4 text"
:map ________________r_rr__________
:map ________________________
:segment 0xef513000 "/usr/openwin/lib/libX11.so.4 data"
:map rr
:segment 0xef515000 "/usr/openwin/lib/libX11.so.4 bss"
:map _
:segment 0xef530000 "/usr/openwin/lib/libolgx.so.3 text"
:map _________
:segment 0xef548000 "/usr/openwin/lib/libolgx.so.3 data"
:map __
:segment 0xef560000 "/usr/openwin/lib/libxview.so.3 text"
:map ____________________________
:map ____________________________
:map _________r_r__r_________r__r__
:map ____________rr_rr_____________
:map _________rr__r________________
:map ____________________________
:map ____________________________
:map ____________________________
:map ________________r_____________
:map ____________________________
```

Performance Data

```
:map ________
:segment 0xef6a3000 "/usr/openwin/lib/libxview.so.3 data"
:map rrr__________rr_r__r
:segment 0xef6b8000 "/usr/openwin/lib/libxview.so.3 bss"
:map __
:segment 0xef6d0000 "/usr/lib/libsocket.so.1 text"
:map ____________
:segment 0xef6eb000 "/usr/lib/libsocket.so.1 data"
:map __
:segment 0xef700000 "/usr/lib/libnsl.so.1 text"
:map _____________________________
:map _____________________________
:map __________________
:segment 0xef75c000 "/usr/lib/libnsl.so.1 data"
:map ______
:segment 0xef762000 "/usr/lib/libnsl.so.1 bss"
:map _____
:segment 0xef780000
"/home/tribbles6/jkchan/sparcworks/sunperf/sw_dev/Sw/sunperf/lib/l
ibsunperf/libcollector.so text"
:map r_
:segment 0xef791000
"/home/tribbles6/jkchan/sparcworks/sunperf/sw_dev/Sw/sunperf/lib/l
ibsunperf/libcollector.so data"
:map t
:segment 0xef792000
"/home/tribbles6/jkchan/sparcworks/sunperf/sw_dev/Sw/sunperf/lib/l
ibsunperf/libcollector.so bss"
:map __t_____________
:segment 0xef7e0000 "ld.so.1 text"
:map _____________
:segment 0xef7f8000
:map _
:segment 0xef7fa000
:map _
```


Performance Data
End Time of Sample
:segment 0xef7fc000 "ld.so.1 data"
:map _
:segment 0xef7fd000
:map _
:segment 0xefffa000 "stack"
:map _t_trr
:term
.
.
.
sample_off 19.839323000
exit 19.839323000

Print File

The Print option prints an ASCII text version of the display that you are currently viewing. You can print a hardcopy or print the data to a file. The `er_print` utility also provides the same information as the Print option.

This appendix contains some samples of the Print file.

- **Sample 1—**User Times data type, viewed in the Overview display with one sample selected.
- **Sample 2—**User Times data type, viewed as a Cumulative Histogram display with all samples selected.
- **Sample 3—**Working Set data type, viewed in the Address Space display as pages with one sample selected.

See also:

- Section 11.6, “Printing Experiment Information,” for information about the Print option.
- The man page for `er_print`

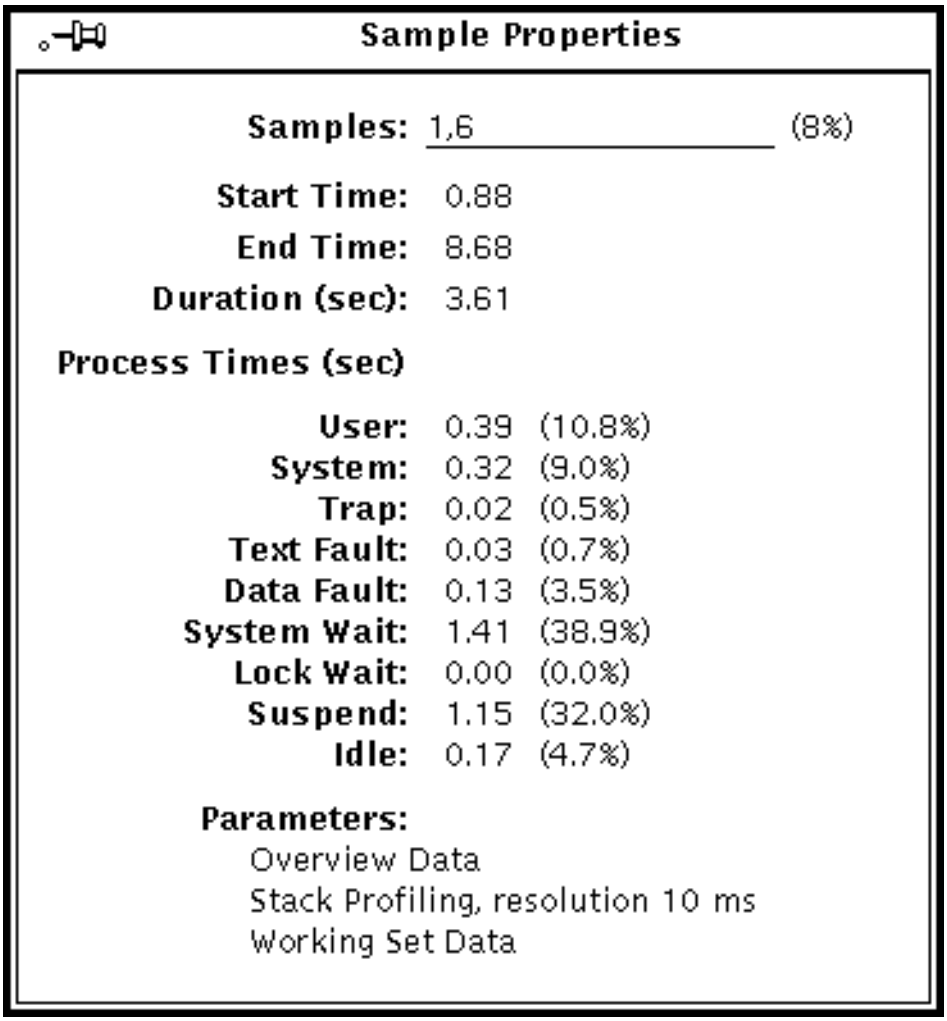
E

E.1 Sample 1

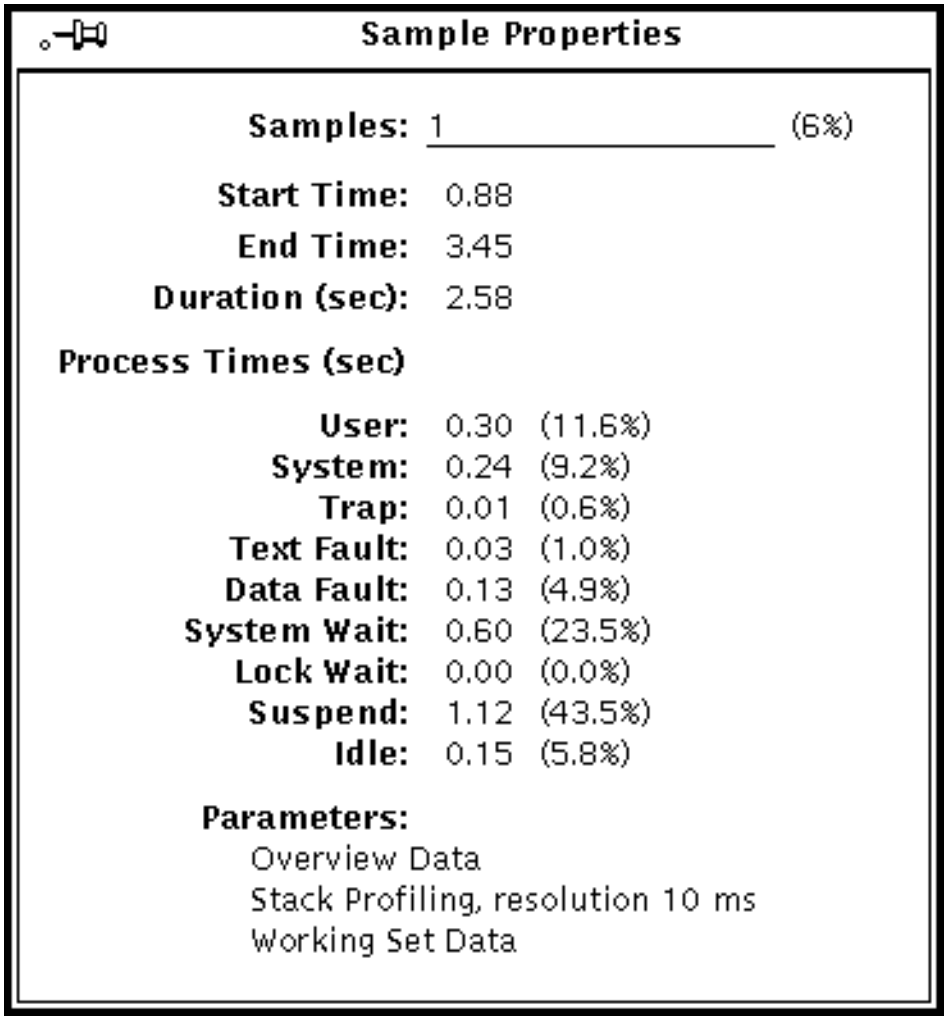
This sample is the Process Times data type, viewed in the Overview display with one sample selected. Experiment:

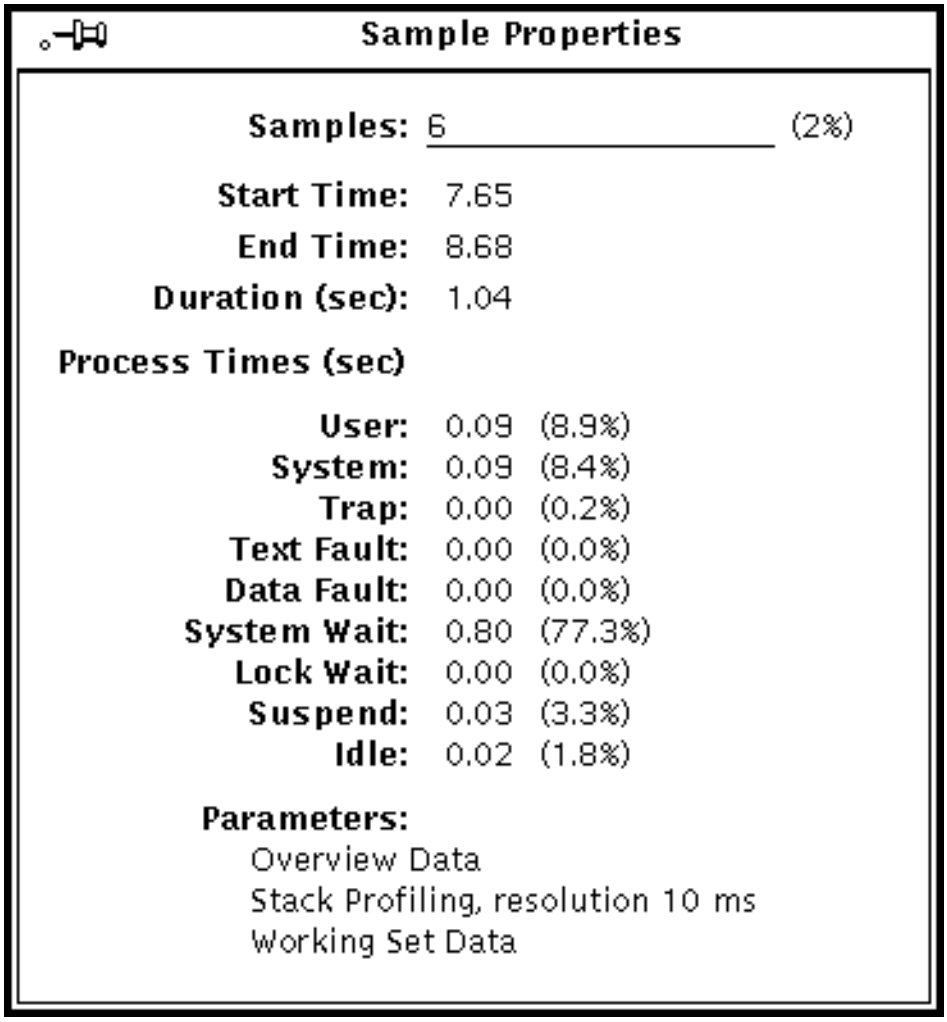
`~/sparc/blocks2/sw_dev/usr/src/sw/examples/Blocks/test.1.er`

Sample selection: 1, 6 (8%)



Individual samples





E.2 Sample 2

This sample is the User Times data type, viewed in a Cumulative Histogram display with all samples selected. Experiment:

```
~/sparc/blocks2/sw_dev/usr/src/sw/examples/Blocks/test.1.er
```

Sample selection: 1-16 (100%)

```
Segment Coverage:
~/sparc/blocks2/sw_dev/usr/src/sw/examples/ \
Blocks/Blocks text
data = user time
display = Histogram
unit = functions
cumulative = on
sort by = value
names = long
```

```
                                  seconds
################################| 20.4700 _start
################################| 20.4600 main
################################| 20.4600 make_window(int,char**,
                                  char**)
              ###############| 9.6600 load_bearing_block::put_on
                                  (block&)
                ############| 8.1900 auto_mode(void)
                   #########| 6.3600 hand::animate(point)
                    ########| 5.6200 hand::grasp(block&)
                      ######| 4.0900 button_proc(unsigned long)
                       #####| 3.3600 hand::move(block&,
                                  load_bearing_block&)
                       #####| 3.1900 get_rid_of(block&)
                        ####| 2.5300 load_bearing_block::clear_
                                  top(void)
                           #| 0.6700 load_bearing_block::get_
                                  space(block&)
                           #| 0.6600 load_bearing_block::make_
                                  space(block&)
                           #| 0.0800 unsafe_ostream::flush(void)
                          #| 0.0800 draw_cylinder(unsigned long,
                                  point,point)
```

```
#| 0.0800 ostream::flush(void)
#| 0.0800 endl(ostream&)
#| 0.0800 repaint_proc(unsigned long,
   unsigned long,rectlist*)
#| 0.0600 ostream::operator<<
   (ostream& (*)(ostream&))
#| 0.0600 table::draw(unsigned long)
#| 0.0400 auto_proc(void)
#| 0.0400 filebuf::sync(void)
#| 0.0300 move_frame_onto_screen
   (unsigned long)
#| 0.0200 operator<<(ostream&,const
   point&)
#| 0.0200 resize_proc(unsigned long,
   int,int)
#| 0.0200 brick::draw(unsigned long)
#| 0.0200 filebuf::overflow(int)
#| 0.0200 hand::ungrasp(block&)
#| 0.0200 draw_cube(unsigned long,
   point,point)
#| 0.0100 load_bearing_block::find_
   space(block&)
#| 0.0100 intersections(block&,int,
   int,list&)
#| 0.0100 unsafe_ostream::operator<<
   (long)
#| 0.0100 ostream::operator<<
   (ostream& (*)(ostream&))
#| 0.0100 ostream::operator<<(int)
#| 0.0100 list::value(void)
#| 0.0100 point::convert(int,int)
#| 0.0100 unsafe_ostream::operator<<
   (const char*)
```

.
.
.

E.3 Sample 3

This sample is the Working Set data type, viewed in the Address Space display as pages with one sample selected.

Experiment: `test.1.er`

Sample selection: 12 (1%)

Unit: PAGE

KEY
W - Modified
R - Referenced
U - Unreferenced

```
  00 10 20 30 40 50 60 70 80 90 A0 B0 C0 D0 E0 F0

0001__00  R  R  R  R  R  R  R  R  R  U  U  R  R  R  R  R
0002__00  U  R
0003__00     R  W  W  R  U  U  W  R  R  R  U  U  W  W  R
0004__00  R  R  U  U  R  U  U  U  U  U  U  U  U  U  U  U
0005__00  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
0006__00  U  U  U  U  U  U  R  W  U  R  R  W  U  U  R  U
0007__00  W  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
0008__00  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  W
0009__00  W  W  W  W  W

EF32__00  U  U
EF33__00     U
```

EF35__00 U U U U U U U U

EF36__00 U

EF38__00 U U U

EF39__00 U U

EF3B__00 U U U U

EF3C__00 U

EF3E__00 U U U

EF3F__00 U

EF41__00 U

EF42__00 U

EF44__00 R U R U U U U U U R U U U U U U

EF45__00 R U U U U U U R R R R R U U U U

EF46__00 U U U U U U U U U U U U U R R R R

EF47__00 U U U U U U U U U U U U U U R R U

EF48__00 U U R R R U U U U U U U R U R R

EF49__00 U U U U R U U U U U U U R U U

EF4A__00 R W

EF4B__00 W W R W U

EF4D__00 R U U U U U U U U U U R R U R U

EF4E__00 U R R R R U U U U U U U R U U U

EF4F__00 U U U U U U U U U U U U U U U U

EF50__00 U U U U

EF51__00 R R W

EF53__00 R R R U U U U U U

EF54__00 R U

EF56__00 R U U U R U U U U U U U U U U U

EF57__00 U U U U U U U R U U U U U U U U

EF58__00 U U U U U U U U U U R U U U U U

EF59__00 U U U U U U U U U U U U U U U U

EF5A__00 U U U U R R R R U U R U U R U U

EF5B__00 U U U U R R R R U U U U U U U U

EF5C__00 U U U R R U R R U R R U U U U U

EF5D__00 U U U U U U U U U U U U R R R R U

EF5E__00 R R R U U U U U U U U R R U U U

EF5F__00 U U U U U U U U U U U U U U U U

EF60__00 U U U R U U U R R R U U U U U U

EF61__00 U U U U U U U U R U U U U U U U

EF62__00 U U U U U U U U U U U U U U U U

EF63__00 U U R U U U U U U U U U U U U U

EF64__00 U U U U U U U U R R U U U U U U

EF65__00 U U U U U U U U U U U U U U U U

EF66__00 U U U U U U U U U U U U U U U U

EF67__00 U U U U U U U U U U U U U U U U

EF68__00 U U U U U U U U U U U U U U U U

EF69__00 U U

EF6A__00 R R R R U U U U U U U U U U R

EF6B__00 U W R R R R W W W

EF6D__00 R U U U U U U U U U U U R U

EF6E__00 R R

EF71__00 R U R U U U U U U U R U U U U U

EF72__00 U R U U U U U U U U U U U U U U

EF73__00 U U U U U U U U U U U U U U U U

EF74__00 U U U U U U U U U U U U U U U U

EF75__00 U U U U U U U U U U U U U U U U

EF76__00 U U U U U U U U

EF77__00 U U U U U U U U U

EF78__00 U U U U U

EF7A__00 R R

EF7B__00 W U U W W U U U U U U U U U U

EF7C__00 U U

EF7E__00 U U U R R U R U U R R R U

.

Print Summary File

F

This appendix describes the contents of the Print Summary file.

See Section 11.6, “Printing Experiment Information,” for information about the Print Summary option.

F.1 Contents of the Print Summary File

The Print Summary file provides a textual overview of the experiment. It is an ASCII text file that has information about the experiment.

The Print Summary file contains the following information:

- System configuration information
- Data collection parameters
- Overview summary (units are in seconds)
- Profile summary (units are in ticks). Each tick is the Profiling Timer unit that you specified in the Collector during the data collection process. See Chapter 10, “Doing More with the Collector,” for details about the Profiling Timer.
- Working set summary

The following is a list of labels (and their definitions) that are used in the Print Summary file.

See Section 6.5, “Sample Properties Window,” for more information about the following labels:

- **utime—**user time
- **stime—**system time
- **tftime—**text fault time
- **dftime—**data fault time
- **ttime—**trap time
- **slptime—**sleep time
- **ltime—**lock wait time
- **wtime—**idle time
- **stoptime—**suspend time

See Chapter 9, “Statistics Display,” for more information about the following labels:

- **minf—**minor page fault
- **majf—**major page fault
- **nswap—**process swaps
- **inblk—**input blocks
- **oublk—**output blocks
- **msnd—**message sent
- **mrcv—**message received
- **sigs—**signals handled
- **vctx—**voluntary context switches
- **ictx—**involuntary context switches
- **sysc—**system calls
- **ioch—**characters of I/O

Glossary

The *Glossary* is a collection of terms specific to the Analyzer. These terms appear throughout the manual and familiarity with their meanings will help you in using the Analyzer.

Address Space display

A display that shows the memory usage of the address space of a process. The information needed to produce this display is collected by choosing Working Set data in the Collector window. The display shows pages that are: modified, referenced, and unreferenced.

Analyzer

A tool used to examine performance data collected by the Collector. The Analyzer processes the collected data into displays of your choice. You then use these displays to examine the collected performance data.

Collector

A tool used to set up an application for performance data collection. You can collect performance data on an application of your choice.

Continuous sampling

A data collection mode that gathers performance data at specified intervals of seconds.

Control area

An area at the top of a window in the OPEN LOOK GUI that contains the buttons and menu buttons.

Cumulative histogram

A display that provides a general summary of the amount of time spent by a function, module, or segment in its relationship to other functions, modules, and segments.

Data collection parameters

A specification for sampling data in an experiment. These types of performance data are specified in the Collector. The data collection parameter settings of the Collector are: Overview, Working Set, PC (Program Counter), and PC and Stack.

Execution Statistics display

A display that provides statistical information about the application attributes that are not obvious or visible in any of the other displays. This display produces specifications for an application and is particularly useful for defining numerical values for the attributes.

Experiment

A set of samples taken on the target application program.

Experiment record

Each experiment has a *pointer file* and a *hidden directory*. The pointer file and hidden directory are referred to as the *experiment record*.

Function histogram

The default histogram display. This display shows the amount of time the application spends executing functions.

Histogram display

A display of the Analyzer used to visually compare where the application units are spending time during execution. The Analyzer supports three types of application units: function, module, and segment.

Magnify Help

On-line help provided for all tools.

Manager

A desktop tool for managing the SPARCworks programming tools. It is a visual organizer that provides easy accessibility for starting, quitting, and managing tools.

Manual sampling

A data collection mode that allows you to control the interaction between the application execution and the data collection by letting you specify exactly when to begin a new sample.

Module histogram

A display that profiles the amount of time modules are executing in the application. This display provides higher levels of data aggregation.

No samples

A data collection mode that stops data collection.

OPEN LOOK GUI

A graphical user interface that standardizes the interfaces of application programs, therefore making them easier to learn. It has a standard set of buttons, menus, controls, and operations that make it easy to learn and easy to apply across applications.

Overview data

The default data collection parameter of the Collector. It is not visible as a selectable item; it is always collected. Overview data collects resource usage and processes real-time performance data.

Overview display

The default display of the Analyzer. For each sample, the display shows the amount of time the application spends in different types of performance areas.

Pages Properties window

A pop-up window displayed from the Props button in the Analyzer. The Pages Properties window displays the address of a page, the size in bytes, and the functions associated with the page.

PC profiling data

Performance data that shows how much time is spent by the application in its execution of functions, modules, or segments. It also tells what functions, modules, or segments are consuming the most amount of time and the least amount of time.

PC and stack profiling data

Performance data that shows how much time is spent by calls in the application that address other calls. It provides a more detailed analysis than PC profiling data, yet it does include PC profiling data.

Performance tuning

The steps taken to make a program more efficient, reliable, and fast. These steps estimate the performance of a program, identify the bottlenecks limiting performance, and identify where the code spends most of its time.

Performance-tuning model

An ideal against which you compare an application when performance tuning. The model provides a baseline for evaluating your application.

Performance-tuning process

An iterative, interactive process. It is a repeated process, where performance improvements are reviewed and tested. The approach should be systematic to avoid the possibility of creating problems at the same time problems are being fixed.

Profile packet

A profile packet is a unit of profiling data that is included in a sample that has collected either PC or PC and Stack data.

Profiling timer

An adjustable timer that lets you define the number of profile packets to take. The default value of the timer is 10 milliseconds. The timer adjusts up to 1000 milliseconds to allow for more detailed samples.

Program units

Program units include program and shared libraries that are used by the application.

Reordering

An automatically invoked, multistep process that rearranges text to help reduce text working sets. The reordering strategy is based on profiling data and function size. The strategy of ordering functions is to reduce the text working set size.

Sample

A sample that contains data collected over a specified period of time during the execution of the application.

Sample Properties window

A pop-up window displayed from the Props button in the Analyzer. This window allows you to examine a sample or combination of samples in more detail. The Sample Properties window provides detailed information about selected samples such as start time of the sample, end time of the sample, and data collection parameters.

Segment histogram

A display that profiles the amount of time the application spends executing segments. The segment is the highest level of data aggregation supported by the Analyzer.

Segment Properties window

A pop-up window displayed from the Props button in the Analyzer. The Segment Properties window is for use with the segments display and shows the address, size in bytes, segment name, and a list of functions.

Target application

The application which you are tuning for performance.

Working set data

Data that represents the process state address space as a series of segments, each of which contains a number of pages. The data collected describes the status of each page, and whether it was referenced, modified, or unreferenced.

Index

A

E

F

H

T

V

W