System Activity Monitoring [7]

This section describes the design and implementation of the following system activity monitoring packages:

- Standard UNIX system activity package (`sar`, `sag`, `timex`, `mppview`, `xmppview`, `sadc`, `sa1`, and `sa2` commands)
- Cray Research system activity monitoring package (`sam`, `xsam`, `csam`, and `samdaemon` commands)
- Cray Research system activity reporting package (`sdc` and `tsar` commands)
- Disk usage monitoring

Warning: Although the standard UNIX system activity package (`sar` and related commands) is part of a Cray ML-Safe configuration of the UNICOS system, the Cray Research system activity monitoring and reporting commands described in this section are **not** part of the Cray ML-Safe configuration, including `sam`, `tsar`, and their associated commands. This section **does not** contain any further warnings or information pertaining to the use of a Cray ML-Safe configuration.

7.1 Standard UNIX System Activity Package (`sar`)

The UNICOS operating system contains a number of counters that are incremented as various system actions occur. The standard UNIX system activity package reports system-wide measurements for the UNICOS operating system, including central processing unit (CPU) utilization, disk and tape input/output (I/O) activities, terminal device activity, buffer usage, system calls, system switching and swapping, file-access activity, and queue activity. The package provides commands that generate various kinds of reports, allow you to monitor Cray MPP systems, and automatically generate daily reports.

The functions of the activity package are as follows:

Command	Function
`sag`(1)	Produces data and command files suitable for use on a front-end machine to produce graphic displays of system activity

`sar`(1)	Allows users to generate system activity reports in real time and to save system activities in a file for later use
`timex`(1)	A modified `time`(1) command that times a command and produces a report on concurrent system activity
`mppview`(8)	Displays massively parallel processing (MPP) system activity
`xmppview`(8)	Displays Cray MPP system activity through a graphic user interface

You can produce system activity daily reports automatically by using the `sadc`, `sa1`, and `sa2` commands (see `sar`(8)). Use of these commands is discussed in Section 7.1.3, page 291.

The system activity information reported by this package is derived from a set of counters in the operating system kernel, as described in Section 7.1.1, page 284.

The following sections provide information about the standard UNIX system activity package:

- System activity counters
- System activity commands
- Daily report generation
- Source files and scripts
- Derivations

7.1.1 System Activity Counters

The UNICOS operating system manages a number of counters that record various activities and provide a basis for the system activity reporting system. The data structure for most of these counters is defined in the `sysinfo` structure in the `/usr/include/sys/sysinfo.h` file. The system table overflow counters are stored in the `syserr` structure. The device activity counters are extracted from the device status tables. The I/O activity of any disk devices attached to the I/O subsystem (IOS) is recorded by the device activity counters.

The system activity counters monitored by the system activity package are described in the following sections.

7.1.1.1 CPU Time Counters

When the `sadc` program (see `sar`(8)) collects the data, it increments four standard time counters. The counters give an overview of CPU performance and record activity on each the information is also provided on a per-CPU basis. The counters are updated from the counters maintained in the processor working storage (`pws`) structure. These counters record activity on each CPU, also providing information on a per-CPU basis. Each time the UNICOS operating system executes an exchange into the kernel, it updates the counters in the `pws` table. These counters are incremented according to the mode that the CPU is in at the time of the exchange: idle, user, or kernel. I/O wait time is always 0. The `pws` counters are incremented by the number of machine cycles since the last exchange.

7.1.1.2 `lread` and `lwrite` Counters

The `lread` and `lwrite` counters record the number of logical read and write requests issued by the system block devices.

7.1.1.3 `bread` and `bwrite` Counters

The `bread` and `bwrite` counters record the number of times data is transferred between the system buffers and the block devices. The actual I/O operations are triggered by the logical I/O operations that cannot be satisfied by the current contents of the buffers. The ratio of block I/O to logical I/O is a common measurement of the effectiveness of the system buffer cache.

7.1.1.4 `phread` and `phwrite` Counters

The `phread` and `phwrite` counters record read and write requests issued by the system to raw devices.

7.1.1.5 `swapin` and `swapout` Counters

The `swapin` and `swapout` counters are incremented for each system request that initiates a transfer to or from the swap device. The amount of data transferred between the swap devices and memory is measured in blocks and counted by `bswapin` and `bswapout`.

7.1.1.6 xswapin and xswapout Counters

The xswapin counter is incremented each time a shared text segment is swapped in. The xswapout counter is incremented each time a shared text segment is freed.

7.1.1.7 switch and syscall Counters

The switch and syscall counters are related to the management of multiprogramming. The syscall counter is incremented every time a system call is invoked. In addition, the numbers of read(2), write(2), fork(2), and exec(2) system calls are kept in the sysread, syswrite, sysfork, and sysexec counters respectively.

The pswitch variable counts the times that the kernel subroutine swtch () was called to switch to another user. This occurs in the following situations:

- A system call results in a road block
- An interrupt occurs and awakens a higher-priority process
- A clock interrupt occurs

7.1.1.8 runque, runocc, swpocc, and swpque Counters

The runque, runocc, swpocc, and swpque counters record queue activities; they are implemented in the clock.c routine. At each 1-second interval, the clock routine examines the process table to see whether any processes are in core memory and are in the ready state. If so, the routine increments the runocc counter and adds the number of such processes to the runque counter. While examining the process table, the clock routine also checks to see whether any processes in the swap queue are in ready state. If the swap queue is occupied, the clock routine increments the swpocc counter and adds the number of processes in the swap queue to the swpque counter.

7.1.1.9 iget, namei, and dirblk Counters

The iget, namei, and dirblk counters apply to file-access operations. The iget and namei counters, in particular, are the names of UNICOS routines. These counters record the number of times that the respective routines are called.

The namei routine performs file path searches. It searches the various directory files to get the associated i-number (inode) of a file corresponding to the path.

The `iget` routine locates the inode entry of a particular file (i-number). It first searches the table of inodes in core memory. If the inode entry is not in the table, the `iget` routine gets the inode from the file system in which the file resides and enters the information in the table. The `iget` routine returns a pointer to this entry. The `namei` routine calls `iget`, but other file-access routines also call routine `iget`. Therefore, the `iget` counter is always greater than the `namei` counter.

The `dirblk` counter records the number of directory block reads issued by the system. The number of directory blocks read divided by the number of `namei` calls provides an estimate of the average path length of files.

7.1.1.10 `readch` and `writech` Counters

The `readch` and `writech` counters record the total number of bytes transferred by the `read`(2) and `write`(2) system calls.

7.1.1.11 `rcvint` and `xmtint` Counters

The device `rcvint` and `xmtint` counters measure T-packet activity to and from terminals attached to the IOS. The `rcvint` counter measures the number of packets flowing into the Cray Research computer system. The `xmtint` counter measures the number of packets acknowledged as received by the IOS.

7.1.1.12 `rawch`, `canch`, and `outch` Counters

The `rawch`, `canch`, and `outch` counters record the number of characters in the unbuffered queue, canonical queue, and output queue, respectively. Characters generated by terminal devices operating in unbuffered mode are counted in both `rawch` and, as they are edited, in `canch`.

7.1.1.13 `clists` Counter

The `clists` counter records the usage and overflow of `clists` so that you can carefully set the number of `clists` for terminal devices.

7.1.1.14 I/O Activities

All disk and tape I/O is monitored for each device. This counter shows the number of read and write operations on each device. Each read or write operation represents a maximum of 4096 bytes transferred. Response time and active time recorded on other UNIX systems is not useful here, because the IOS deals with the device.

7.1.1.15 Logical Device Cache

This counter records all logical device cache activity. The ratio of cache-to-user and cache-to-disk is a common measurement of the effectiveness of logical device cache.

If a user without root privileges executes the `sar`(1) command with the frequency arguments *t* and *n* specified, logical device cache data will not be available. Logical device cache statistics are obtained from `/dev/dsk`, which can only be read by users with root privileges.

7.1.1.16 Inode, File, Text, and Process Tables

The information from the inode, file, text, and process tables comes in two forms. The first shows the current utilization of table entries and the maximum configured number of table entries. The second is extracted from the `syserr` structure. When a table overflow occurs, the counter for the corresponding table is incremented.

7.1.1.17 `sysrda`, `syswra`, and `syslstio` Counters

The `sysrda` and `syswra` counters monitor `reada`(2) and `writea`(2) asynchronous I/O system call usage respectively. The `syslstio` counter counts `listio` usage.

7.1.1.18 `pkin`, `pkout`, and `pkbad` Counters

The `pkin`, `pkout`, and `pkbad` counters monitor IOS packet traffic.

7.1.1.19 `shuffle`, `textlock`, `datlock`, and `punlock` Counters

The `shuffle` counter measures the number of requests to shuffle a process into low memory. The `textlock` and `datlock` counters measure the number of text and data areas locked in memory. The `punlock` counter measures text and data area unlocks, unlocking the process.

7.1.1.20 Exchange Counts

The kernel logs user-initiated exchanges of the following error types:

Type	Description
`err`	Error exchange

`pre`	Program-range error
`ore`	Operand-range error
`fpi`	Floating-point interrupt
`dli`	Dead lock interrupt

7.1.2 System Activity Commands

The system activity package provides commands for generating various system activity reports. The `sar`(8) and `sag`(1) commands allow users to specify a sampling interval and number of intervals for examining system activity, and then to display the observed level of activity in tabular and graphic form. The `timex`(1) command reports the amount of system activity that occurred during the precise period of execution of a timed command. The `mppview`(8) command displays the system activity of a Cray MPP system. The `xmppview`(8) command displays Cray MPP system activity through a graphic user interface.

7.1.2.1 `sar`(8) Command

The `sar`(8) command reports on various types of system activity. When you specify frequency arguments *t* and *n*, `sar` invokes the `sadc` data collection program to sample the system activity counters in the operating system every *t* seconds for *n* intervals and generates system activity reports in real time. Generally, it is desirable to include the `-o` option to save the sampled data in a file for later examination. The format of the data file is shown in the file `/usr/src/prod/admin/sa/c1/sa.h` (not available on UNICOS binary-only systems). In addition to the system counters, a time stamp is included. It lists the time at which the sample was taken. If you do not supply frequency arguments, `sar` generates a system activity report for a specified time interval from an existing data file created by `sadc` (see `sar`(8)) at an earlier time. The `sar`(8) man page describes the use of the command and the various types of reports.

The `sar`(1) command extracts operating system activity information for a specified time interval. The following sections describe, by option, all displays that the `sar`(1) command can produce.

Note: By default, the `sar`(1) command runs every 10 minutes, reading data from tables in the kernel and writing to the `sa` file. It is recommended that the 10-minute default not be increased. (The overhead for `sar` is quite low.) You can decrease this interval to cause `sar`(1) to run more often, which can help solve performance problems by providing more continuous data; however, if the `sar`(1) interval is too small, this could generate too much system activity data for daily operation, increase requirements for disk space, and potentially degrade system performance.

7.1.2.2 `sag`(8) Command

The `sag`(8) command displays system activity data graphically. It relies on the data file produced by a prior run of `sar`(8); `sag` then creates plot commands and data files for any column or combination of columns of data from the `sar` report. These files are run on a machine that supports plotting. The command syntax allows you to specify either cross plots or time plotting. Select data items by using the `sar` column header names. (See the `sar`(8) man page for information about its options and usage.) The system activity graphics program cannot run on Cray Research systems.

7.1.2.3 `timex`(1) Command

The `timex`(1) command is an extension of the `time`(1) command. When you do not specify any options, `timex` behaves exactly like `time`. In addition to giving time information, it also prints a system activity report derived from the system counters. The `timex`(1) man page explains its use.

Although you, as an administrator, typically will use `timex` to measure a single command, you also can time multiple commands, either by combining them in an executable file and timing it, or, more concisely, by typing the following command:

```
timex sh -c "cmd1;cmd2; ..."
```

This use of `timex` establishes the necessary relationships between parent and child processes so that the user and system times consumed by *cmd1*, *cmd2*, and `sh` can be extracted correctly.

7.1.2.4 `mppview`(8) Command

The `mppview`(8) command displays a map of active partitions running on a Cray MPP system (see the `mppview`(8) man page). It uses the `curses`(3)

library to drive the terminal display so that many terminal types can be supported. You can specify the host system to be monitored.

Through `rpc`(3), `mppview` communicates with the system activity monitoring (`sam`) server, `samdaemon`(8), running on the host system to obtain the information that you request.

7.1.2.5 `xmppview`(8) Command

A graphical user interface also is available through the `xmppview`(8) command. It uses the X Window System X11 tool. The display graphically represents usage of processing elements according to criteria you select and gives system performance statistics in tables accessible through pull-down menus. `xmppview` also contains a tutorial. (See the `xmppview`(8) man page for set-up requirements.)

7.1.3 Daily Report Generation

It is recommended that you routinely monitor and record system activity for historical analysis. This section describes how to produce a standard daily report of system activity automatically.

7.1.3.1 Facilities

For full descriptions of the commands used in generating reports, see the `sar`(8) man page. Use the following commands to produce a system activity report:

Command	Function
`sadc`	Reads system counters from the `/dev/kmem` and `/dev/dsk` files and records them in a file. In addition to the file argument, two frequency arguments are usually specified to indicate the sampling interval and number of samples to be taken. If no frequency arguments are given, `sadc` writes a dummy record in the file to indicate a system restart.
`sa1`	Invokes the `sadc` command to write the system counters in the daily data file `/usr/adm/sa/sa` *dd*; *dd* represents the day of the month. `sa1` may be invoked with sampling interval and iterations as arguments.
`sa2`	Invokes the `sar` command to generate the daily report `/usr/adm/sa/sar` *dd* from the daily data file `/usr/adm/sa/sa` *dd*. `sa2` also removes the daily data files and report files after one

week. The starting and ending times and all the report options of `sar` are applicable to `sa2`.

7.1.3.2 Suggested Operational Setup

It is suggested that you have the `cron`(8) daemon control the normal data collection and report generation operations. For example, the following sample entries in the `crontab` file `/usr/spool/cron/crontabs/root` would cause the data collection program `sadc` (see `sar`(8)) to be invoked every 20 minutes between 8 A.M. and 5 P.M., and every hour otherwise:

```
0 8-17 * * 1-5 "/usr/lib/sa/sa1 1200 3 &"

0 18-7 * * 1-5 "/usr/lib/sa/sa1 &"
```

Data sampling is more frequent during prime time to allow more detailed analysis. The `sa1` program should be invoked hourly rather than daily to ensure that, if the system crashes, data collection will resume within 1 hour after the system is restarted.

Invoking `sadc` through the `root` crontab file ensures that all data is collected. Logical device cache statistics are read from `/dev/dsk`, which is readable only by `root`.

Because the system activity counters are reset to when the system is restarted, `sadc` writes a special record in the data file to reflect this situation. To accomplish this process, `sadc` is invoked with no frequency arguments within `/etc/rc` when going to multiuser state:

```
/usr/lib/sa/sadc /usr/adm/sa/sa‘date +%d
```

For more information on using `rc`, see General UNICOS System Administration, publication SG-2301.

The `cron` command also controls the invocation of `sar` to generate the daily report by the shell procedure `sa2`. You may choose the time period to be covered by the daily report and the groups of system activity to be reported.

For example, if the following is an entry in the `crontab` file `/usr/spool/cron/crontabs/root`, `cron` executes the `sar` command to generate daily reports from the daily data file at 20:00 on weekdays:

```
0 20 * * 1-5 su sys -c "/usr/lib/sa/sa2 -s 8:00 -e 18:00 -i3600 -A"
```

The report includes a full set of information from 08:00 to 18:00.

In case of a problem, such as disk space shortage, these data files and report files can be removed by the super user. The man page entry for `sar`(1) describes the daily report generation procedures.

7.1.4 Source Files and Scripts

(Not available on UNICOS binary-only systems.) The source files and shell programs of the system activity package are located in the `/usr/src/prod/admin/sa/c1` directory.

File or Program	Description
`sa.h`	A system activity header file that defines the structure of data file and device information for measured devices. It is included in `sadc.c`, `sar.c`, and `timex.c` records.
`sadc.c`	A data collection program that accesses the `/dev/kmem` and `/dev/dsk` files to read the system activity counters and that writes data either on standard output or on a binary data file. It is invoked when the `sar`(1) command generates a real-time report. It is also invoked indirectly by entries in the `/usr/spool/cron/crontabs/root` file to collect system activity data.
`sar.c`	A report generation program that invokes the `sadc` command to examine system activity data, generates reports in real time, and saves the data to a file for later usage. It also can generate system activity reports from an existing data file. It is invoked indirectly by the `cron`(8) daemon to generate daily reports.
`saghdr.h`	A header file for the `saga.c` and `sagb.c` programs. It contains data structures and variables used by `saga.c` and `sagb.c`.
`saga.c`, `sagb.c`	A graph generation program that first invokes `sar` to format the data of a data file in tabular form and writes a command file and data files so that the `sar` data can be displayed in graphic form on another machine.

`sa1.sh`	A shell script that invokes `sadc` to write data file records. It is activated by entries in the file `/usr/spool/cron/crontabs/root`.
`sa2.sh`	A shell script that invokes `sar` to generate the report. It also removes the daily data files and daily report files after a week. It is activated on weekdays by an entry in the `/usr/spool/cron/crontabs/root` file.
`timex.c`	A program that times a command and generates a system activity report.

7.1.5 Derivations

This section lists the derivations of reported items. Each item discussed is the data difference sampled at two distinct times, *t1* and *t2* (in seconds).

CPU utilization time is as follows:

%-of-cpu `=` `(`*cpu-util* `/` *cpu-total*`)` `*` `100`

The value of *cpu-util* is either the idle, user, or kernel (system) CPU utilization time. The value of *cpu-total* is the sum of the idle, user, and kernel (system) CPU utilization time. For a system with multiple CPUs, the information is also presented as the sum of the utilization time of each CPU, divided by the sums of the idle, user, and kernel utilization time, divided by the number of CPUs.

The cache hit rate is as follows:

%-of-cache-I/O `=` `((`*logical-I/O* `-` *block-I/O*`)` `/` *logical-I/O*`)` `*` `100`

I/O can be either a cache read or cache write operation.

Queue activity is as follows:

avg-x-queue-length `=` *x-queue* `/` *x-queue-occupied-time*`;`
%-of-x-queue-occupied-time `=` *x-queue-occupied-time* `/` `(`*t2* `-` *t1*`);`

The value of *x-queue* is the length of either a run or swap queue.

The remainder of system activity is as follows:

avg-rate-of-x `=` *x* `/` `(`*t2* `-` *t1*`)`

The rate is expressed as occurrences per second. The value of *x* can be swap or drop in/out, terminal device activities, `read/write` characters, block

`read/write`, logical `read/write`, process switch, system calls, `read/write`, `fork/exec`, `iget`, `nami`, directory blocks read, disk activities, IOS packets in and out, secondary cache read and written, `clists` active, shuffles, text/data and process unlocks, and asynchronous I/O activities.

7.2 Cray Research System Activity Monitoring (`sam`) Package

The Cray Research system activity monitor, `sam`, collects and displays system activity data from selected Cray Research computer systems. It consists of a data acquisition daemon, `samdaemon`, and two display clients, `xsam` and `csam`. Figure 6 illustrates the hardware configuration options for `sam`.

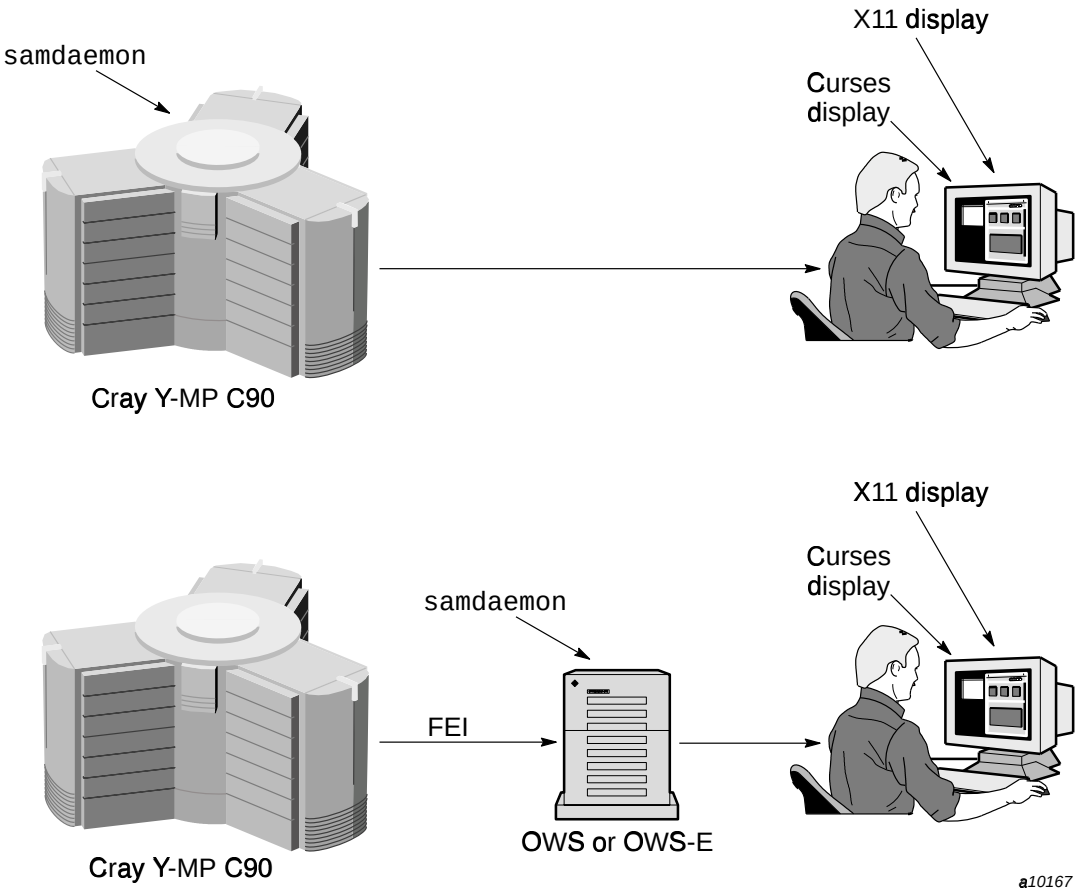

Figure 6. Hardware configuration options for sam

The `samdaemon` process can run on a Cray Research mainframe or on a connected operator workstation (OWS). When `samdaemon` is executed on a mainframe, data collection is done through the `/dev/kmem` special file using `read`(2) system calls. When `samdaemon` is executed on an operator workstation it uses the I/O path of the front-end interface to access data fields in memory, thus reducing the load on the UNICOS operating system.

The `sam`(8) command is a generic interface that invokes either the `xsam` utility (for use with terminals that run the X Window System) or the `csam` utility (for use with terminals compatible with `curses`(3)). Both utilities provide a user

interface with menus for selecting and displaying system activity data. Alternately, you can use the `xsam` or the `csam` command to enter the menus.

See the `sam`(8) man page for more information on which client `sam` invokes and on their respective displays.

The following sections discuss the daemon and commands:

Utility	Description
`samdaemon`	System activity data acquisition daemon
`csam`	Displays system activity data using the `curses`(3) library routines
`xsam`	Displays system activity data using the X Window System

7.2.1 samdaemon Process

The system activity monitor daemon, `samdaemon`(8), reads kernel data structures in memory as requested by the display clients, `xsam`(8) or `csam`(8). `samdaemon` can run on a Cray Research mainframe or on a connected operator workstation. The daemon and the clients communicate through the Remote Procedure Call (RPC) utility. When a client is not requesting data, `samdaemon` ceases to read from memory.

When started, `samdaemon` generates a private configuration database by `forking` a configuration utility, `sama`. On the Cray Research mainframe, `sama` uses the `nlist`(3) library routine to obtain the necessary data addresses. When `samdaemon` is running on the operator workstation, it generates this database by having `sama` make a request to the `samdaemon` process running on the Cray Research mainframe. Once `samdaemon` is initialized on the operator workstation, all subsequent data is obtained directly from mainframe memory through the front-end interface (FEI) connection without involvement of any processes running on the mainframe.

The `samdaemon` process provides access control to UNICOS data with the use of a validation file. If `samdaemon` finds a validation file (`/usr/lib/samdaemon.val`, by default), it will return data only for validated users.

See the `samdaemon`(8) man page in the *UNICOS Administrator Commands Reference Manual*, publication SR–2022, for more details on `samdaemon`, its options, and setup.

Note: The `samdaemon` command is not intended to be built and installed on an operator workstation by a site; it is part of the binary release for the operator workstation. `samdaemon` will not build on an operator workstation.

7.2.2 csam(8) Utility

The `csam`(8) utility displays system activity on a terminal that is compatible with the `curses`(3) library routines. `csam` can execute either on a Cray Research system or on a connected operator workstation. `csam` receives data by communicating with the `samdaemon` process on the local host, or on a remote host if one has been specified.

Because `csam` relies on the `curses` library routines, it can support a variety of terminals. `csam` is able to check the size of the screen and the `LINES` environment variable and adjust most displays to fill the screen.

You can invoke `csam` by using the `csam` command or the generic `sam` command. You can select various system performance statistics for display. See the `csam`(8) man page in the *UNICOS Administrator Commands Reference Manual*, publication SR–2022, for more detailed information on the `csam` command and its options.

7.2.2.1 csam Commands

Once `csam` is running on your terminal, you can use the following commands to change displays, move within a display, increase or decrease refresh rates, and exit the system:

Command	Description
`c` or `C`	Selects host system configuration display.
`d` or `D`	Selects disk display.
`e` or `E`	Quits program.
`f` or `F`	Selects ldcache display.
`g` or `G`	Leaves single-step mode.
`h` or `H`	Selects help screen.
`k` or `K`	Selects kernel display.
`l` or `L`	Selects logical device display.
`m` or `M`	Selects memory display.

`n` or `N`	Advances to next page (disk, ldcache, logical device, process, and tape displays). To reset these scrolled displays, use the `d`, `f`, `l`, `p`, or `t` command.
`p` or `P`	Selects process display.
`q` or `Q`	Quits program.
`r` or `R`	Resets bar graphs and ldcache display.
`s` or `S`	Selects user/kernel/wait/idle usage on kernel display.
`t` or `T`	Selects tape display. (Implementation deferred)
`u` or `U`	Selects user and system CPU usage on kernel display.
`w` or `W`	Selects swap map display.
`x` or `X`	Selects top process display.
`y` or `Y`	Selects system call display.
`z` or `Z`	Selects record/replay control panel. The `csam` record/replay function is described in detail in this section.
`+`	Increases refresh interval by 1 second.
`-`	Decreases refresh interval by 1 second. The refresh interval cannot be lowered below the refresh rates set by the server.
`.`	(Type the dot character.) Enters single-step mode. This freezes the display refresh until you press the space bar to advance it.
space bar	Advances the display in single-step mode.
numeric input	Selects a process ID for the snap display. This is only accepted in the process display. The ID must be terminated by a carriage return. The erase and kill characters are accepted during this numeric input.

7.2.2.2 Record/replay Function

The record/replay function works in a manner similar to that of an audio tape recorder. From the record/replay control screen, you can set up `csam` to record

data for replay at a later time or to replay data that was recorded at an earlier time. Enter the `csam` record/replay control screen by typing the `z` or `Z` command while the cursor is in any `csam` terminal screen.

By default, the display refresh rate of the replay function is set at 1 second, which is the minimum rate. You can adjust this rate by using the `+` and `-` commands. However, altering the playback refresh rate does not alter the refresh rates for displays obtained directly from `samdaemon`.

The following default values are displayed near the top of the record/replay control screen when it is invoked:

```
Record/Replay   File name:  client.recplay
                Mode     :  record
                State    :  not loaded
```

These lines are referred to as the status lines.

Status line	Indicates
`File name`	The name of the file to which data will be recorded or from which data will be replayed.
`Mode`	Either `record` or `replay`.

`State`	The current state of the file.
	
	The `State` status line contains two fields. The first field displays one of three values:
	
	• `not loaded`
	• `stopped`
	• `running`
	
	The second field may be blank or may display either of two values:
	
	• `rewound`
	• `end of tape`
	
	The following states are what you typically see:
	
	`not loaded`
	
	`stopped rewound`
	
	`stopped end of tape`

```
                    running                 end of tape
                    running                 rewound
```

Commonly used command keys are displayed below the status lines on the screen. While the cursor is in the record/replay control screen, you can use the following keys to set up either a record session or a replay session:

Key	Description
`s`	Stops recording or replaying. The `State` status line displays `stopped`.
`b`	Begins (starts) recording or replaying. The `State` status line displays `running`.
`r`	Rewinds the file. You must return to the beginning of the file in order to replay the data recorded there. A file is at `end of tape` position when you stop recording or when all the data has been replayed. A file is rewound automatically when it is loaded.
`f`	Fast forward to the end of file. This allows you to append additional data to the end of the file.
`l`	Load the file. This is required before recording or replaying can begin. The `State` status line displays `stopped rewound`.
`e`	Eject the file. The file is no longer loaded. The `State` status line displays `not loaded`.
`d`	Done setting up record/replay. This command returns you to the `csam Help` screen. You can use this command to exit the record/replay control screen in order to switch to one of the data display screens and begin recording or replaying data.
`m`	Switch mode. This command toggles between record mode and replay mode. The `Mode` status line displays which mode is currently active. The file is unloaded automatically when the mode is changed.
`n`	Change file. This allows you to change the file name for recording or replaying. Terminate the name with `RETURN`. The `File name` status line displays the current file name.
`q`	Quit. This command allows you to exit `csam` and return to the shell.
`t`	Truncate. This command truncates the file at the position where the command is entered. For example, if you are replaying data

and want to truncate the file, you would return to the Record/Replay Control screen by entering the `z` or `Z` command, and then pressing the `t` key. This position becomes the end of the file. The only way to overwrite the file is to use the `t` (truncate) command subsequent to pressing the `r` (rewind) key in the record/replay screen; pressing the `b` (begin) key in record mode automatically moves to the end of the file (the `end of tape` position).

Recording data. To record data, follow these steps, starting with the cursor in the record/replay control screen:

1. Press `m` to select record mode if `record` is not currently displayed on the `Mode` status line.
2. If desired, press `n` to change the file name, then press the `RETURN` key.
3. Press `l` to load the file.
4. Press `b` to start recording.
5. Press `d` to exit the record/replay control screen in order to select data for recording.
6. In the `csam` terminal screen, select the data you want to record and enter the appropriate command to display that screen. For example, to record kernel data, enter the `k` command. The data displayed on the kernel screen will be recorded to the file.
7. To stop recording data, enter the `z` or `Z` command, which returns you to the record/replay control screen, and then press the `s` key.

Replaying data. To replay data, follow these steps, starting with the cursor in the record/replay control screen:

1. If the second field of the `State` status line does not display `rewound`, assure that replay starts at the beginning of the file by pressing `r` to rewind the file.
2. To replay a file other than the one displayed on the `File name` status line, press `n` and enter a file name and press `RETURN`.
3. Press `m` to select replay mode if `replay` is not currently displayed on the `Mode` status line.
4. Press `l` to load the file.

5. Press `b` to start the replay.
6. Press `d` to exit the record/replay control screen in order to select the data to replay. For example, in the `csam` terminal screen, enter `k` to replay kernel data. This will cause `csam` to switch to that screen and immediately begin displaying the recorded data. The message, `End of file reached on playback file`, will be displayed after all data have been displayed.
7. To stop replaying data, type `z` or `Z`, which returns you to the record/replay control screen, and then type `s`.

7.2.3 `xsam` Utility

The `xsam`(8) utility allows you to use graphic displays to monitor system activity on a UNICOS system. `xsam` can execute on either a Cray Research system or on a connected operator workstation, and its display can be viewed on any terminal running the X Window System that is connected to the network. The `xsam` utility receives data by communicating with the `samdaemon` process on the local host, or on a remote host if one has been specified.

You can invoke the `xsam` utility by using the command `xsam` or the generic command `sam` with a valid `DISPLAY` shell environment variable. `xsam` accepts the following options:

Option	Description
`X11 options`	Regular X11 options to control X11 defaults.
`-f file`	Initial startup script.

7.2.3.1 X11 Window Settings

You can set your regular X11 default settings (in the `.Xdefaults` file or using the `xrdb`(1X) command) by using the application name `xsam`. For example, the following command will set your default font to `10x20`:

```
xsam*Font:      10x20
```

In addition, you can specify individual X11 defaults by including the widget name in the `.Xdefaults` statement. The following widget names are defined by `xsam`:

```
"menu bar"              : the menu bar for all windows
"menu entry"            : all pop-up menus
"console"               : the main xsam console
```

```
"setup display"         : the setup window
"host display"          : the host window
"help display"          : the help window
"config display"        : the target configuration display
"device display"        : disk and logical I/O display
"graph console"         : graph console for kernel and I/O graphs
"graph display"         : graph display for kernel and I/O graphs
"map display"           : main and swap memory display
"snapshot display"      : process snapshot display
```

The following example sets the background for `menu bar` to `lightblue` and also specifies the font for all menus. The font for the `graph console` is set to `fg-22` and to `fixed` for the `graph display` (different from the `menu bar` in the `graph display`).

```
xsam*menu bar.background:       lightblue
xsam*menu entry.font: -*-helvetica-bold-o-*-*-20-*-*-*-*-*-*-*
xsam*graph console*Font:   fg-22
xsam*graph display.font:   fixed
```

7.2.3.2 `xsam` Windows

The `xsam` utility creates six types of windows, described as follows:

Window	Description
Console window	Controls all aspects of `xsam`. It can be used to request other `xsam` displays. It also contains a message section that displays all error and informational messages.
Configuration window	Shows aspects of the hardware and software configuration of the target system.
Graph window	Monitors various kernel counters including CPU utilization, I/O statistics, and system call usages.
Map window	Displays a representation of memory. Within the map are several columns of boxes, each with its own name and each corresponding to a process in memory. The size of each box is proportional to the size of the process in main memory. As the process moves or grows, the box moves and grows.

	Command options allow you to display the process ID, view a section of memory, and view a snapshot of any process.
Snapshot window	Displays information about a specific process in text form. The data is extracted from the kernel's process structure for the target process. A snapshot can be requested from a memory map or with the snapshot command. The snapshot window contains several sets of statistics including user information (such as the size and address of the process and user and process flags), scheduling parameters, and an area devoted to user identification.
Device I/O window	Displays device configuration and I/O performance on a system-wide level, as well as on an individual device level. Possible device types are disks and logical devices.

7.2.3.3 Console Window and Available Commands

After the `xsam` utility is started, it opens a console window. This window is the primary interface for all information about `xsam` and interaction with the `xsam` utility, and is described in detail in this section. The console window, shown in Figure 7, consists of four major parts:

- Menu bar, at the top of the window
- Information area
- Command input area, in the middle of the console window
- Message output area, at the bottom of the console window

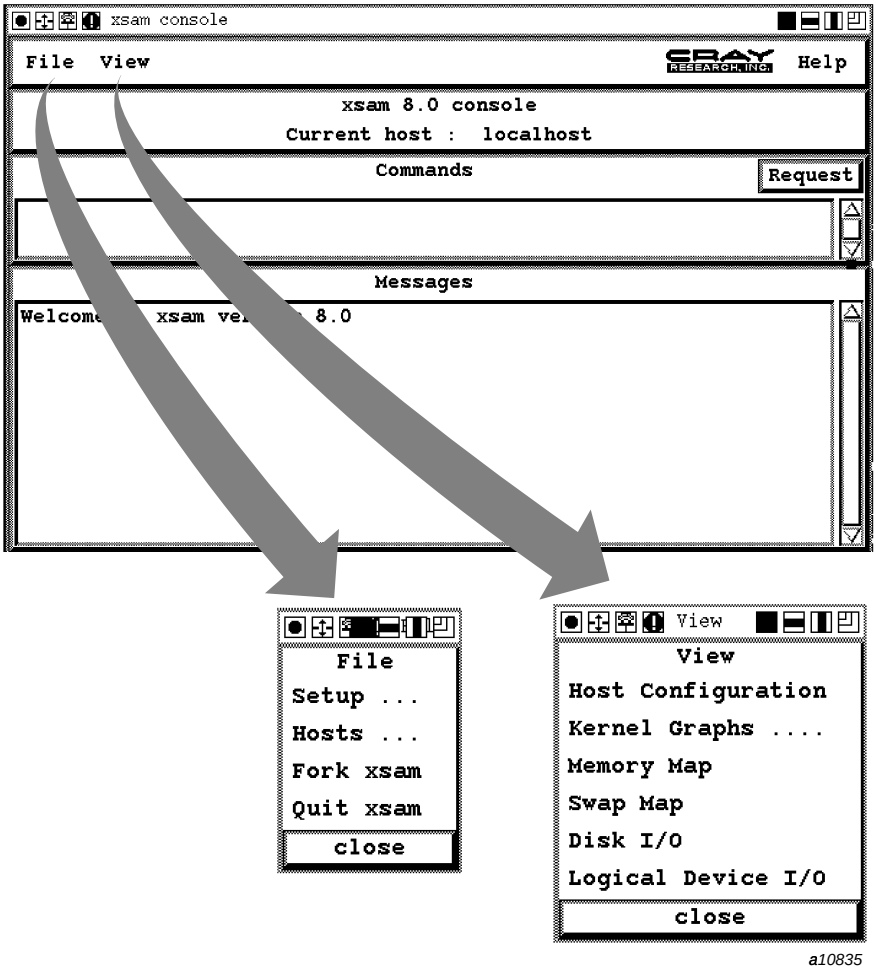

Figure 7. The `xsam` console window

To show a menu, click the left mouse button when the cursor is placed on the menu name you want to select. Most of the menu items will start a new display.

The console menu bar provides access to three menus: `File`, `View`, and `Help`. A Cray Research logo also is shown: when selected, it displays copyright information about `xsam`.

- The `File` menu has the following entries:

```
Setup ...
Hosts ...
Fork xsam
Quit xsam
```

- The `View` menu has the following entries:

```
Host Configuration
Kernel Graphs ...
Memory Map
Swap Map
Disk I/O
Logical Device I/O
```

- The `Help` menu has the following entry:

```
General Information
```

To activate the associated window for these entries and others, select the corresponding menu entry by clicking the left mouse button.

Note: Alternately, as mentioned throughout the discussion of `xsam` displays, you can enter the commands directly in the command input area of the console window to bypass the menu structure or to specify options to your command.

The information area in the console window shows some global identification, such as `xsam 9.0 console` and the name of the current host.

The `Commands` input area provides an alternative way to activate a function of `xsam`, by entering a command in the input area. With some exceptions, both the menu structure and direct command input allow access to the same functions. Whereas using `xsam` through the menu interface is self explanatory, the command interface is summarized here.

After you enter a command in the `Commands` input area, start the command by using the left mouse button to click on the `Request` button.

Note: Under normal conditions, requests also can be started by pressing the `RETURN` key while the cursor is positioned in the input area. However, the widget used to implement the input area allows some sophisticated editing (see the "Text Widget" section in the *Cray Doo-Dad Set Reference Manual*, publication SN–3098) that can disable this feature (such as using the built-in `vi` style editing). In those situations, a command must be started by using the `Request` button.

The following list shows the commands you can use in the commands input area:

`quit`
`help`
`host [`*host name*`]`
`setup` *setup options*
`script` *filename*
`config [`*X11 options*`]`
`graph [`*X11 options*`] name [name ...]`
`map [`*X11 options*`] name [map options]`
`snapshot [`*X11 options*`] pid`
`device [`*X11 options*`] name`

The menu interface allows you to issue all of these commands except the `script` command. (In addition, the `fork` and `record/playback` commands are available from the console menu.) The `script` command initiates execution of `xsam` commands from a script file. No additional support, such as flow control within the script or parameter substitution, is available. Although scripts can be nested, all scripts operate within the same `xsam` environment. (For instance, if a lower-level script changes the current host, this effects the execution of the higher-level scripts as well).

Several commands also allow you to specify X11 options. However, not all normal X11 options are available on the `xsam` command input level.

The following X11 options are available:

```
bg        color
fg        color
geometry  [[Width]x[Height]][+Xposition][+Yposition]
fn        font
```

You can also specify X11 options by using the X11 set-up tools. For more information about X11 setup, see the online help facility available in `xsam`.

Note: Some window managers vary regarding when to grant geometry requests. Check the manual for your window manager for information about setting up your window manager correctly.

The `Messages` area echos the commands issued either through the menu selections or by entering the commands directly in the `commands` area. It also displays informational and error messages.

In both the `Commands` and the `Messages` areas, you can use the scroll bar at the far right of each area to view the history of entries. The left mouse button moves the text down, while the right button moves the text up; the position of the pointer along the scroll bar controls how far the text moves with each click.

7.2.3.4 `xsam` Commands

From the console window, the `xsam` utility allows all of the commands available from the command input area, except the `script` command. In addition, the console menu makes available the `fork` and `record/playback` commands.

Command	Description
`Quit`	To terminate the entire `xsam` session, you can either select the `Quit xsam` menu entry in the `File` menu or enter `quit` in the `Commands` area. The menu entry may ask for confirmation.
`Help`	Select the `General Information` entry in the `Help` menu or enter the `help` command to bring up this display. You can click on an item in the `Help Options` area to locate it quickly in the help display. Use the `DISMISS` button to exit the help display.
`Host`	`xsam` allows you to monitor several hosts from within a single session. This is controlled by using either the `host display` menu or the `host` command. You can view multiple displays monitoring multiple hosts in a single `xsam` session. To activate the `Host` display, select the `Hosts ...` entry in the `File` menu. To enter a new host name from the `Host` display, move the cursor to the area labeled `Set Host`, enter the host name and press the `RETURN` key. The new

host becomes the current host and also is highlighted in the `Available Hosts` area. You also can change the current host by clicking on its name in the `Available Hosts` area.

A host also can be entered and made current by typing *host hostname* in the `Commands` input area of the console window. The `host` command without any arguments displays the current host in the `Messages` area of the console window.

`Fork`

Use the `Fork xsam` entry in the `File` menu to disconnect your `xsam` session from your tty session. This allows you to continue running `xsam` and to execute other programs or UNICOS commands within the same tty session. `Fork` can be executed only once during a session. Its function is disabled if logging is active and connected to your tty. `Fork` also can be started by the command:

```
setup fork
```

`Setup`

The `Setup` display provides control over resources that are local to your `xsam` session. To activate the `Setup` display, select the `Setup ...` entry in the `File` menu. Alternately, you can execute set-up commands in the console window to customize your X11 display.

The `setup` display allows you to select three options:

```
Echo Commands to Console Window
Exit from Script upon Error
Confirm Program Quit Request
```

They can be activated either from within the `setup` display or by entering the following commands in the console window:

```
setup [no]echo
setup [no]abort
setup [no]confirm
```

The left mouse button acts as a toggle on the display to turn the options on (represented by a dark box) and off (represented by a light box).

The `Setup` display area labeled `Record and Playback Control` controls the data recording and playback features. To specify the file name you want to use, move the mouse to the area labeled `Record File Name` or `Playback File Name`, enter the file name and press the `RETURN` key to activate the file. You must click on the `START` button to begin recording or playback. The record and playback feature is not accessible through the command interface. See the `Record/Playback` description in this list for further information.

The `Setup` display area labeled `Logfile Control` is used to control command and debug output logging. To activate logging, move the mouse into the area labeled `Log File Name`, enter a file name and press the `RETURN` key. Then select, by clicking, one or both of the types of logging messages available:

`Console`

or

`Debug`

An empty string entered as a file name will close the log file without opening a new one. The file names `stdout` and `stderr` correspond to the related standard UNIX files.

These logging functions also can be controlled by entering any of the following commands in the Console window:

```
setup logfile file name
setup nologfile
setup [no]debug
setup [no]conslog
```

Record/Playback

You must access `record` and `playback` sessions through the `Setup` display of the `File` menu; they are not available from the `Commands` input area of the console window. When `recording` is turned on, all incoming data is written to a file that can be used for `playback`. You can start recording at any time during a normal session.

When `playback` is turned on, data is taken from that file instead of contacting a running `samdaemon` process. Use of the `playback` feature is restricted in the following ways:

- `playback` sessions are single host sessions.
- `playback` can be activated only before connection to `samdaemon` has been established. After `playback` has been activated, no further real connections can be made within this `xsam` session.
- `playback` cannot show any data that was not recorded, but can show more information than was monitored while recording the data. What is or is not available depends on the packaging of the information that is exchanged between `samdaemon` and the `xsam` client. For example, if recording was done while a `graph user-0` was active, all information about CPU utilization is included in the data file. Thus, during playback, you may have `user-Sum`, `kernel-Sum`, and `idle-Sum` active.

  To activate either `record` or `playback`, you must first enter the name of the data file, press the `RETURN` key to select the file, and then click the `START` button. After `record` or `playback` is activated, use the associated buttons as you would on an audio tape deck, with the exception of the `TRUNC` button. This button truncates the file at the position when the button is clicked. For example, if you are playing back a file and stop the playback by clicking the `STOP` button, and then you click

the `TRUNC` button, the file will be truncated at the point currently displayed.

After activating the playback process, you must select which data to display from the file (through the `View` menu or through commands).

`Config`

The `Config` display is available through the `Host configuration` option of the `View` menu. It displays information about the hardware and software configuration of your target system. You also can activate it with the following command:

```
config
```

`Graph`

Select the `Kernel Graphs ...` entry in the `View` menu to bring up the `graph console`, which will show you a list of available graphs for that specific host. You can select graphs by clicking on their names. Selected graphs are indicated by a dark box; clicking on them again will deselect them, as indicated by a light box. Click on the `Request Selected Graphs` button to bring up the `Graph` display. Graphs are shown in alphabetical order.

If a graph name is known to the user, it also can be activated by using the following command:

`graph` *graphname1 graphname2* `...`

In addition, this allows you to specify some X11 options. Graphs are shown in the order specified in the command.

The `Graph` display shows actual performance data. It also shows the time of day of the host machine.

Within a `Graph` display, you can use the mouse to request additional displays such as time information. For information about using the mouse within a `Graph` display, see the online help facility available in `xsam`.

Two interesting types of graphs are the `-all` and `-sum` graphs for CPU utilization data. The `-all` graph shows the current utilization of all configured CPUs of the target system within a single graph (no history). This is useful for a quick overview of general system performance from hosts with many CPUs. The `-sum` graph shows the CPU utilization accumulated over all configured CPUs of the target host. These two options can be used for CPU utilization graphs `user`, `kernel`, and `idle`.

You can also display a selected graph for a specific CPU with the following `graph` command:

`graph` *graphtype-CPU*

`graphtype` may be `user`, `kernel`, or `idle`; the `-` is required (no spaces), and `CPU` is a valid CPU number.

`Map`

Select either the `Memory Map` or the `Swap Map` entries in the `View` menu to bring up the `map display`.

This display shows a box (or a line, for small entries) for every process that currently resides on the specified target memory. The size of a box is related to its actual process size, and the position of a box is related to its position within the target memory.

You can zoom into an active `Map` display by moving the cursor to the desired start position in the `Map` display, clicking the left mouse button (this will show you the actual start position), moving to the desired end position, and clicking the right mouse button. A new `Map` display will be activated.

A `Map` display also can be activated with the following command:

`map [`*map name*`][-start` *addr*`][-end` *addr*`][-id][-size]`

	map name	Either memory (the default), or swap.
	`-start` *addr*	Specifies a specific starting address of the map. Leading zeroes indicate an octal value.
	`-end` *addr*	Specifies a specific ending address of the map. Leading zeroes indicate an octal value.
	`-id`	Requests that process IDs be displayed. (Space permitting)
	`-size`	Requests that process sizes be displayed. (Space permitting)

`Snapshot`

The `Snapshot` display shows detailed information about running processes. You can activate the display directly from the `Map` display by positioning the cursor inside the box that corresponds with the desired process and pressing either the middle mouse button or the `p` key. The snapshot display can also be activated with the following command:

```
snapshot < pid >
```

pid is a valid process ID number.

`Device`

Select either the `Disk I/O` or the `Logical Device I/O` entry in the `View` menu to bring up the `Device` display.

A `Device` display can also be activated by the `device` command. This command allows you to

specify some X11 options. In addition you must specify a valid device name.

Figure 8 shows the `xsam` device display window.

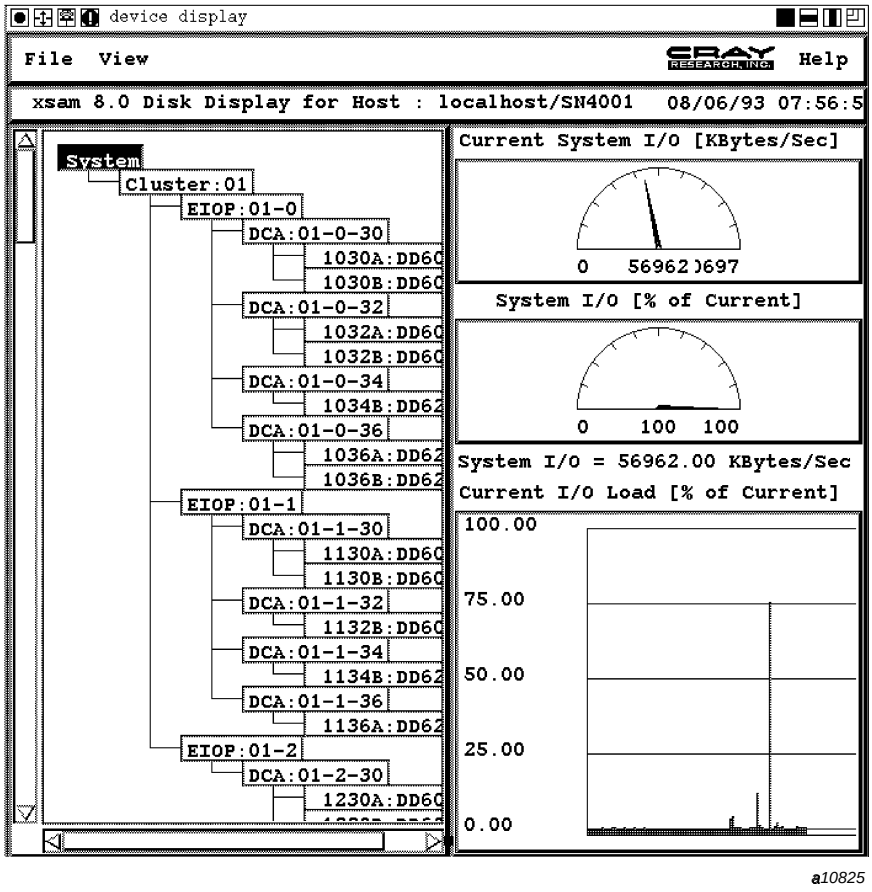

Figure 8. The `xsam` device display window

The `Device` display shows four fields of information.

- On the left side of the window is a tree of the current configuration. This tree contains all real nodes, such as physical disks or logical devices, and several configuration nodes.

- The disk display shows nodes for all `IO-Clusters`, for `EIOPs`, for disk channels (DCAs), and the global `System` node.
- The logical device display shows the global `System` node and two additional nodes for `Cached` and `Normal` (non-cached) devices.

Possible actions you can take by using the cursor inside the configuration tree are described in the "Tree Widget" section of the *Cray Doo-Dad Set Reference Manual*, publication SN–3098. The mouse action `select-node`, triggered by pressing the left button on the mouse, selects a node. This selection affects the function of the lower speedometer, which shows the I/O rate of the sub-tree starting at this node.

- The right side of the device display shows two speedometers and a bar diagram.
  - The upper speedometer (labeled `Current System I/O [KBytes/Sec]`) shows the current and maximum I/O rates in kilobytes per second. The maximum value is the highest I/O rate seen in the lifetime of this display.
  - The lower speedometer (labeled `<NAME> I/O [% of Current`) shows the I/O percentage rate of the current selected device subtree with respect to the current I/O rate (as shown in the upper speedometer). The current selected device subtree starts at the node highlighted in the configuration tree. An additional line shows the absolute value of this I/O rate.
  - The bar diagram shows the I/O percentage rate of all real devices with respect to the current I/O rate (as shown in the upper speedometer). Configuration nodes (such as `System`) are not part of this diagram.

    You can perform three actions with the cursor positioned in the bar diagram. The left and right mouse buttons change the lower and upper ends, respectively, of the nodes displayed. This allows you to zoom into the diagram for sites with a larger disk or logical device farm. The middle mouse button selects a specific disk or logical device node. This, too, affects the function of the lower speedometer. If the node is not hidden in the configuration tree, the configuration tree is changed in such a way that it is more likely that the selected node is visible; if the node is hidden, its name is printed on the main console.

The `View` menu of the `Device` display contains two entries:

```
I/O Graphs ...
Reset
```

Select the `Reset` entry to reset the internal counters (such as maximum I/O rate), clear the speedometers and the bar diagram, make the entire configuration tree visible, and select the `System` node.

Select the `I/O Graphs` entry to bring up a `graph console` that allows you to select detailed statistics about individual nodes. For disk nodes and non-cached logical devices, two graphs are available:

```
<NAME>:nrds
<NAME>:nwrts
```

For cached devices, four graphs are available:

```
<NAME>:crds
<NAME>:cwrts
<NAME>:drds
<NAME>:dwrts
```

The extensions have the following meaning:

Extension	Meaning
`nrds`	Number of blocks read
`nwrts`	Number of blocks written
`crds`	Number of blocks read from cache to user
`cwrts`	Number of blocks written from user to cache
`drds`	Number of blocks read from disk to cache
`dwrts`	Number of blocks written from cache to disk

7.3 Cray Research System Activity Reporting (`tsar`(8)) Package

You can use the `sdc`(8) and `tsar`(8) commands together to gather and to report system activity data.

Information about the system activity reporting package is divided into the following areas:

- `sdc` command
- `tsar` command
- Data collection
- Data file format

- `tsar` source scripts
- `tsar` language description
- Operational setup
- Examples
- Limitations

The functionality of `sdc` and `tsar` is similar to that of the `sadc` and `sar` commands. The `sdc` and `tsar` combination offers the following advantages:

- You can specify how much data is to be collected, either all of the `sdc` data or a subset. If you are interested only in CPU utilization, then you can direct `sdc` to gather only CPU data.
- Data can be summarized in a manner that is appropriate for your site. `tsar` report formats are not hard coded. Reports are formatted according to the directives in `tsar` source scripts, which you write.
- `sdc` data files are portable from one Cray Research platform to another. The files contain headers that describe the data records.

The current release has limitations on its implementation.

7.3.1 sdc Command

The `sdc`(8) command gathers system activity data from the kernel by generating a C program and an executable, `sdcx`. The `sdc` process creates a new session and calls the `fork`(2) system call to create a child process, which executes `sdcx`. The child process performs the actual data collection while the parent `sdc` process exits.

The `sdc` command can collect the following types of data:

- CPU utilization
- System calls
- Process management
- Memory management
- System table management
- System I/O

- General system data
- Disk activity
- Tape activity
- TCP/IP activity
- Terminal activity
- NFS activity
- Network interface activity
- IPC activity

Tables in this section list the available data items for each of these types.

For more detailed information on the command see the `sdc`(8) man page.

7.3.2 `tsar` Command

The `tsar`(8) command formats the data collected by the `sdc` command according to user-specified directives. The directives are placed in source scripts, which `tsar` processes.

The `tsar` utility is a translator that processes a subset of the `awk` language, described in *The AWK Programming Language*, by Alfred Aho, Brian Kernighan, and Peter Weinberger.

For more detailed information on the command see the `tsar`(8) man page.

7.3.3 Data Collection

By default, the `sdc` command collects data for over 100 system activity counters. You can sample a subset of these counters by invoking `sdc` with the `-R` option. The system activity data types and the data items (or counters) available for each type are described in tables in this section; the four column heads are defined as follows:

Column Head	Meaning
Name	The name that `tsar` and `sdc` use for the data item. This name should appear in the request file when you use the `sdc -R` option.

Cum — As shown in the tables, each data item is either cumulative (Y) or not (N).

- Y — The value for this item is cumulative. Thus, to determine the item's value for an interval, the value from the previous record must be subtracted from the current record. This type of counter generally is initialized at boot time.
- N — The value for this item is not cumulative.

Unit — The unit, if any, in which the data item is reported.

7.3.3.1 CPU Data

The data shown in Table 34 is available for each CPU. Each item, except `ncpus`, is an array indexed by the CPU number. Each array has `ncpus` entries.

Table 34. CPU data

Name	Cum	Unit	Description
`ncpus`	N		Number of CPUs configured
`cpuuser`	Y	clocks	CPU time in user mode
`cpuUNIX`	Y	clocks	CPU time in kernel mode
`cpuidle`	Y	clocks	CPU idle time
`cpupre`	Y		Number of program range errors
`cpuore`	Y		Number of operand range errors
`cpuerr`	Y		Number of error exchanges
`cpufpi`	Y		Number of floating point errors
`cpudli`	Y		Number of deadlock errors
`cpurpe`	Y		Number of register parity errors

7.3.3.2 System Calls

The data shown in Table 35 is available for each system call. Each item, except `nsysc`, is an array indexed by the system call name prefixed by `sc_` (such as `sc_fork`, `sc_exec`, and `sc_read`) or by 0 through (`nsysc` -1). Each array has `nsysc` entries.

Table 35. System call data

Name	Cum	Unit	Description
`nsysc`	N		Number of system calls
`sc_name`	N	ASCII	Name of system calls
`scall`	Y		Number of requests
`scalltime`	Y	clocks	Total time used by system call
`scallmax`	N	clocks	Maximum time to complete
`scallmin`	N	clocks	Minimum time to complete

7.3.3.3 Process Management

The data shown in Table 36 is available to monitor process management.

Table 36. Process management data

Name	Cum	Description
`pswitch`	Y	Number of process switches
`punlock`	Y	Number of times a process is unlocked
`runocc`	Y	Number of times `runque` was updated
`runque`	Y	Length of the run queue
`srunwait`	Y	Number of times processes were loaded and runnable but not running
`srunwaitproc`	Y	Number of processes loaded and runnable but not running

7.3.3.4 Memory Management

The data shown in Table 37 is available to monitor memory management.

Table 37. Memory management data

Name	Cum	Unit	Description
bswapin	Y		Number of blocks swapped in
bswapout	Y		Number of blocks swapped out
datlock	Y		Number of data locks
memlock	N	clicks	Amount of memory locked
shuffle	N		Number of shuffles in memory
swap	N	swap allocation	Total amount of swap space units
swapavail	N	swap allocation	Amount of available swap space units
swapin	Y		Number of swap ins
swapout	Y		Number of swap outs
swpocc	Y		Number of times swpque was updated
swpque	Y		Length of the swap queue
txtlock	Y		Number of text locks
umem	N	words	Amount of memory available to user processes
umemuse	N	clicks	Amount of memory in use by user processes
xswapin	Y		Number of times a shared-text process was swapped in
xswapout	Y		Number of times a shared-text process was swapped out

7.3.3.5 System Table Management

The data shown in Table 38 is available to monitor system table management activity.

Table 38. System table management data

Name	Cum	Description
`file`	N	Number of active entries in the file table
`file_sz`	N	Size of the file table
`file_ov`	N	Number of overflows in the file table
`nc1inode`	N	Number of active entries in the `nc1inode` table
`nc1inode_sz`	N	Size of the `nc1inode` table
`nc1inode_ov`	N	Number of overflows in the `nc1inode` table
`proc`	N	Number of active entries in the `proc` table
`proc_sz`	N	Size of the `proc` table
`proc_ov`	N	Number of overflows in the `proc` table
`text`	N	Number of active entries in the `text` table
`text_sz`	N	Size of the `text` table
`text_ov`	N	Number of overflows in the `text` table

7.3.3.6 System I/O, General

The data shown in Table 39 is available to monitor general system I/O.

Table 39. General system I/O

Name	Cum	Unit	Description - Number of
`arblks`	Y	blocks	Blocks read by `aread` ()
`aread`	Y		`aread` () calls
`awblks`	Y	blocks	Blocks written by `awrite` ()
`awrite`	Y		`awrite` () calls
`bdrblks`	Y	blocks	Buffer blocks read from devices
`bdread`	Y		`bread` () calls actually read from devices
`bdwblks`	Y	blocks	Buffer blocks written to devices

Name	Cum	Unit	Description - Number of
bdwrite	Y		bwrite () calls - buffer writes to devices
burblks	Y	blocks	lread blocks read - buffer blocks read to user
buread	Y	blocks	aread () and bread () requests - buffer reads to user
buwblks	Y		lread blocks written - buffer blocks written from user
buwrite	Y		awrite () and bwrite () requests - buffer writes from user
pktin	Y		I/O input packets processed
pktout	Y		I/O output packets sent
pktbad	Y		I/O illegal packets received
prblks	Y	blocks	Raw (physio ()) blocks read
pread	Y		Raw (physio ()) reads
pwblk	Y	blocks	Raw (physio ()) blocks written
pwrite	Y		Raw (physio ()) writes
rchar	Y	bytes	Bytes read by read(2), reada(2), listio(2)
wchar	Y	bytes	Bytes written by write(2), writea(2), listio(2)

7.3.3.7 System I/O, Caching

The data shown in Table 40 is available to monitor caching. Each item, except nldds, is an array indexed by the logical device name prefixed by ld_ (such as ld_root and ld_usr) or by through (nldds -1). Each array has nldds entries.

Table 40. System I/O - caching

Name	Cum	Unit	Description
nldds	N		Number of logical devices
ld_active	N		Number of current ldcache requests
ld_agelo	N	clocks	Trickle sync lower threshold

Name	Cum	Unit	Description
ld_ageup	N	clocks	Trickle sync upper threshold
ld_bread	Y	blocks	Number of blocks read from device
ld_bwrite	Y	blocks	Number of blocks written to device
ld_dirthi	N	cache units	Dirty unit high-water mark
ld_dirtlo	N	cache units	Dirty unit low-water mark
ld_dirtpd	N	cache units	Number of dirty units being synced
ld_dirty	N	cache units	Number of dirty units
ld_dread	Y	blocks	Number of reads from disk to cache
ld_dwrite	Y	blocks	Number of writes from cache to disk
ld_name	N	ASCII	Logical device name
ld_nch	N		Number of cache units
ld_rhit	Y		Number of read hits on ldcache
ld_rmiss	Y		Number of read misses on ldcache
ld_rreq	Y		Number of ldcache read requests
ld_sch	N	blocks	Size of each cache unit
ld_uread	Y	blocks	Number of reads from cache to user
ld_uwrite	Y	blocks	Number of writes from user to cache
ld_whit	Y		Number of write hits on ldcache
ld_wmiss	Y		Number of write misses on ldcache
ld_wreq	Y		Number of ldcache write requests

7.3.3.8 General System Data

The data shown in Table 41 is available to monitor other system activity.

Table 41. General system data

Name	Cum	Unit	Description
CLK_TCK	N	clocks	Number of clock ticks per second
HARDWARE	N	ASCII	Machine type
MACHINE	N	ASCII	Machine identification
MEMORY	N	ASCII	Memory configuration
NODENAME	N	ASCII	Network node name
RELEASE	N	ASCII	UNICOS release
SOFTWARE	N	ASCII	Software release
SYSNAME	N	ASCII	System name
VERSION	N	ASCII	UNICOS version
dirblk	Y	blocks	Number of directory blocks read by c1namei ()
iget	Y		Number of iget () function calls
ios_e	N		Set if the system has an IOS model E
namei	Y		Number of namei () function calls
nwpc	N	words	Number of words per click
semlockc	Y		Number of locked events
semlockt	Y	clocks	Time spent waiting on a semaphore
swapau	N	clicks	Swap allocation units in clicks
boottime	Y	seconds	Time at which sdc -B was executed
headertime	Y	seconds	Time at which a header record was written
shuttime	Y	seconds	Time at which sdc -S was executed
time	Y	seconds	Time at which the record was written

In this table, boottime, headertime, shuttime, and time are in seconds since 00:00:00 GMT, January 1, 1970.

7.3.3.9 Disk Data

A set of data items (system activity counters) is available for the IOS model E (IOS-E).

The amount of I/O transferred is reported in sectors, the basic I/O unit for a given disk device type. On the IOS-E, a sector is not always a block. For example, DD-60 devices have a sector size of 2048 words (4 blocks), while DD-62 devices have 512 words per sector.

The data shown in Table 42 is available for each IOS-E physical disk. Each item, except `npdds`, is an array indexed by the physical disk number prefixed by `pd_` (such as `pd_1334_03`) or by 0 through (`npdds` -1). Each array has `npdds` entries.

Table 42. IOS-E physical disk data

Name	Cum	Unit	Description
`npdds`	N		Number of physical disk devices
`pd_name`	N	ASCII	Name of disk device
`pd_ncyls`	Y		Number of cylinders crossed
`pd_nwait`	N		Current number of requests waiting
`pd_sphit`	Y		Number of spare hits
`pd_waithi`	N		Waiting high-water mark
`pd_xpart`	Y		Number of requests that cross partitions
`dr_nqreqs`	Y		Number of queued read requests
`dr_qtime`	Y	clocks	Time waiting in the read queue
`dr_nioreqs`	Y		Number of I/O read requests to the IOS-E
`dr_nblks`	Y	sectors	Number of read sectors transferred
`dr_iotime`	Y	clocks	Actual I/O read time
`dr_rerrs`	Y		Number of recovered read errors
`dr_uerrs`	Y		Number of unrecovered read errors
`dr_rectime`	Y	clocks	Read recovery time
`dw_nqreqs`	Y		Number of queued write requests

Name	Cum	Unit	Description
dw_qtime	Y	clocks	Time waiting in the write queue
dw_nioreqs	Y		Number of I/O write requests to the IOS-E
dw_nblks	Y	sectors	Number of write sectors transferred
dw_iotime	Y	clocks	Actual I/O write time
dw_rerrs	Y		Number of recovered write errors
dw_uerrs	Y		Number of unrecovered write errors
dw_rectime	Y	clocks	Write recovery time

7.3.3.10 Tape Data

The data shown in Table 43 is available for each tape drive. Each item, except ntpds, is an array indexed by the physical tape number prefixed by tp_ (such as tp_170) or by 0 through (ntpds -1). Each array has ntpds entries.

Table 43. Tape drive data

Name	Cum	Unit	Description
ntpds	N		Number of tape drives
tp_name	N	ASCII	Name of tape drive
tp_mounts	Y		Number of volumes mounted
tp_nread	Y	bytes	Number of bytes read
tp_nwrite	Y	bytes	Number of bytes written

7.3.3.11 TCP/IP Data

The data shown in Table 44 is available for analyzing TCP/IP performance.

Table 44. TCP/IP performance data

Name	Cum	Unit	Description
`tcp_hwint`	Y	clocks	Time spent on TCP/IP hardware interrupts
`tcp_swint`	Y	clocks	Time spent on TCP/IP software interrupts
`tcp_scall`	Y	clocks	Time spent on TCP/IP system calls

7.3.3.12 Terminal Data

The data shown in Table 45 is available for analyzing terminal activity.

Table 45. Terminal activity data

Name	Cum	Description
`canch`	Y	Number of canonical characters input through tty interface
`clist`	N	Number of `clist` entries in use
`clist_ov`	Y	Number of overflows in the `clist` buffer
`outch`	Y	Number of characters output through tty interface
`rawch`	Y	Number of raw characters input through tty interface
`rcvint`	Y	Number of T (terminal) packet input interrupts
`xmtint`	Y	Number of T (terminal) packet output interrupts

7.3.3.13 NFS Data

The data shown in Table 46 is available for analyzing network file system (NFS) server and client activity. Each item, except `nnfsc`, is an array indexed by the NFS call index (0 through `nnfsc` -1).

Table 46. NFS data

Name	Cum	Unit	Description
nfsc_name	N	ASCII	Name of NFS call
nfscl_calls	Y		Number of client calls
nfssv_calls	Y		Number of server calls
nnfsc	N		Number of NFS call types

7.3.3.14 Network Interface Data

The data shown in Table 47 is available for analyzing activity for each network interface that has been configured. Each item, except nnet, is an array indexed by network (0 through nnet -1).

Table 47. Network interface data

Name	Cum	Unit	Description
net_colls	Y		Number of collisions
net_ierrs	Y		Number of input errors
net_ipkts	Y		Number of input packets
net_name	N	ASCII	Internet name (address) of the network
net_oerrs	Y		Number of output errors
net_opkts	Y		Number of output packets
nnet	N		Number of configured network interfaces

7.3.3.15 IPC Data

The data shown in Table 48 is available for analyzing IPC activity.

Table 48. IPC activity data

Name	Cum	Description
`msgsend`	Y	Number of IPC messages sent
`msgrecv`	Y	Number of IPC messages received
`semops`	Y	Number of semaphore operations
`shmat`	Y	Number of shared memory attaches
`shmdt`	Y	Number of shared memory detaches

7.3.3.16 Restricted Data Collection

As mentioned in Section 7.3.3, page 320, a subset of the data in the system activity counters can be sampled by using the `sdc -R` option. You must specify the name of a request file, which contains a list of the data to be collected.

Each request file must include at least the following data items: `boottime`, `headertime`, `shuttime`, and `time`. These items are needed to determine when the data was sampled and whether there was a system boot or shutdown.

For example, only data for memory and swap usage will be gathered when `sdc` is executed with a request file that contains these items:

```
boottime
headertime
shuttime
time
HARDWARE
MEMORY
umem
nwpc
swap
swapau
swapavail
umemuse
memlock
```

The `tsar` script `/usr/src/prod/admin/uma/sar.t/M` formats the data into a report.

7.3.4 Data File Format

The system activity data file, created by the `sdc`(8) command, consists of header and data records. A data record is created for each time interval that is sampled. The data records are preceded by two ASCII header records, which define the data records.

7.3.4.1 Header Records

There must always be header records at the beginning of the data file. Two ASCII header records define the data being collected: a definitions record and a meta-data record.

The purpose of the header records is to label the data in the data records that follow. If the header records do not correspond with the data records, `tsar` will be unable to process the data. Header records must be placed at the following locations.

- The beginning of the data file.
- When the system has been reconfigured and the reconfiguration affects the data that `sdc` is sampling.
- When a subset of the data listed in Section 7.3.3, page 320, is sampled.

When `sdc` is invoked with either the `-B` or `-S` option, header records also contain the system boot time and shutdown time.

7.3.4.2 Definitions Record

The first header record written to the data file is the data definitions record. This record contains system call names, device names, kernel table sizes, and ASCII strings that describe the system configuration. When processing the data file, `tsar` 8* updates the data definitions each time it encounters a definitions record.

7.3.4.3 Meta-data Record

The second header record written to the data file is the meta-data record. This record contains the name and size of each item or array that is in the actual

data record. When processing the data file, tsar(8) updates the meta-data definitions each time it encounters a meta-data record.

7.3.4.4 Data Records

Data records are written each time sdc samples the system activity counters. Each set of data records must be preceded by a definitions and a meta-data header record.

There are two types of data records. The first type contains non-ldcache data. If ldcache statistics are requested, the second type of data record is generated. Each ldcache data record is immediately preceded by a definitions record and a meta-data record, which together define and describe the ldcache data record.

Table 49 describes which sdc, sdcx, and tsar options produce header records or data records. When the table shows that one header record is written, it means that both a definition and a meta-data record are written.

Table 49. Records written

Command	Option	Record written
sdc	-c	One header record.
sdc	-B	One header record containing the boot time.
sdc	-R	One header record. If -c, -B, and -S are not also specified, data records are written.
sdc	-S	One header record containing the shutdown time.
sdc	none of the above	One header record and data records.
sdcx	any	Data records.
tsar	-h	One header record and data records.

Caution: The tsar(8) command assumes that the header records define the data records that follow them. If a system reconfiguration occurs, and no header record is written to define the reconfiguration, then the tsar output may be erroneous or tsar may abort.

7.3.5 `tsar`(8) Modes

The `tsar`(8) command operates in one of three modes: compilation-only mode, online mode, or playback mode.

7.3.5.1 Compilation-only Mode

The `tsar -c` option places `tsar` in compilation-only mode. This mode is used to debug `tsar` source scripts. `tsar` compiles the scripts but does not execute them. The system activity data is not formatted into an ASCII report.

7.3.5.2 Online Mode

The `tsar -h` option places `tsar` in online mode. In this mode, `tsar` collects data from the host system through the `sdc`(8) command. The data can be written to a file and/or formatted into an ASCII report as it is being collected.

7.3.5.3 Playback Mode

The `tsar -p` option places `tsar` in playback mode; this also can be done by not specifying the `-h` option. In this mode, `tsar` processes data from an existing system activity data file. This is the default `tsar` mode.

7.3.6 `tsar`(8) Source Scripts

The `tsar`(8) command is a translator that formats system activity data into an ASCII report according to the directives found in a source script.

The `tsar` scripts consist of six sections, including the body, any of which may be empty or missing. Scripts can contain any of the following sections, in any order:

`BEGIN` *{ statements }*
`END` *{ statements }*
`RESTART` *{ statements }*
`RECONFIG` *{ statements }*
`function` *name (arglist) { statements }*
statements

The `tsar` command can process multiple source scripts, and one script can contain multiple `BEGIN`, `END`, `RESTART`, `RECONFIG`, `function`, or body sections. In these cases, `tsar` executes the statements for all like sections in the order that they appear in the scripts or script.

For example, all statements in the various BEGIN sections will be combined into one BEGIN section. The statements will be in the same order as they appear in the scripts or script.

7.3.6.1 BEGIN Section

The statements associated with BEGIN comprise the preamble. The preamble is executed once after the first definition and meta-data records are read. The preamble can be used to print report headings and to initialize variables used in the body.

7.3.6.2 END Section

The statements associated with END comprise the postamble. The postamble is executed once after all the records in the data file have been read. You can instruct tsar in this section to process and print summary data.

7.3.6.3 RESTART Section

The statements associated with RESTART comprise the restart section. These statements are executed each time a system boot or shutdown is found in the header record. The amount of time the system was up or down can be calculated in this section.

7.3.6.4 RECONFIG Section

The statements associated with RECONFIG comprise the reconfiguration section. These statements are executed each time a system reconfiguration or a change in which data items are sampled is detected in the header records.

Note: When a restart or reconfiguration is detected, tsar assumes that all of the system activity counters have been reset. For example, the built-in function sdiff (x) calculates the difference between the current sample of x and the first sample of x in this boot or reconfiguration period. It does not calculate the difference between the current sample and the first sample found in the data file.

7.3.6.5 function Section

The statements in the function section of tsar define functions as specified by the user. Functions always begin with the word function followed by the function name and the argument list. The *arglist* consists of names separated by commas. These argument names are the formal parameters of the function and

the variables that are local to the function. Function calls may be nested and recursive. The return statement can be used to return a value.

7.3.6.6 Body

Statements that are not in any of the preceding sections form the body of the `tsar` source script. Typically, these statements calculate the usage for each interval. This section is executed once for each data record encountered.

7.3.6.7 Example Source Scripts

Examples of `tsar` source scripts can be found in the `/usr/src/prod/admin/uma/sar.t` directory. These scripts produce output that is similar to `sar`(8) output. There is one script for each `sar` option. The name of the `tsar` script is the same as in the `sar` option.

7.3.7 `tsar`(8) Language Description

The `tsar` language is the action language of `nawk` without the string processing operations. Users familiar with `nawk` will have little difficulty writing and understanding `tsar` scripts. The pattern part of `nawk` is unnecessary in `tsar`, because the data format is defined in the data file. Users merely select the data items to process by name.

`tsar` implements a subset of the `awk` language described in *The AWK Programming Language*, by Alfred Aho, Brian Kernighan, and Peter Weinberger.

7.3.7.1 Statements

A `tsar` script can include any of the following statements:

```
if ( expression ) statement [ else statement ]
while ( expression ) statement
do statement while ( expression )
for ( expression; expression; expression ) statement
break
continue
{ [ statements ] }
expression
print expression-list [ >expression ]
printf format[, expression-list ] [ >expression ]
next
exit [ expression ]
```

`return [` *expression* `]`

The following explains further the contents of statements:

- Statement Terminators. Statements are terminated by semicolons, right braces, or newline characters.
- Statement Continuation. Statements can be continued on successive lines by using \ as the last character of the line. Statements can also be continued after the following symbols:

```
,       (comma)
{       (left brace)
&&      (logical AND)
||      (logical OR)

do
else)       (right parenthesis in an "if" or "for" statement)
```

- Comments. Nonexecutable comments begin with # and end with a newline character. They can appear anywhere in the source script.
- Expressions. Expressions include constants, variables, and operators. Parentheses can be used to control the grouping of the operations in an expression.
- Logical Expressions. Logical expressions have a value of 1 (true) and 0 (false). As in the C language, any nonzero value is taken to be true.
- Numbers. Numbers can be integers or floating points. The format is the same as that recognized by `strtod` and `strtol`: digits, decimal point, digits, `e` or `E`, signed exponent. At least one digit or a decimal point must be present; the other components are optional. Octal integers begin with 0. Hexadecimal integers begin with 0x.
- Variable Names. Variable names consist of a letter followed by a string of letters, numbers, or the character _. Variables are used to name the data items found in the data records of the system activity file.

  Some variables in the system activity data file are arrays. The elements of these arrays can be referenced by indexing. For example, the variable, `cpuuser`, is an array that contains the CPU time in user mode (see Section 7.3.3, page 320). The CPU time for CPU 0 is referenced by `cpuuser` [0].

Users can also define additional variables within the `tsar` source script; however, user-defined arrays are not supported.

7.3.7.2 Operators

Prefix, infix, and suffix operators are available for use in `tsar` scripts.

Prefix Operators

The `tsar` command applies a prefix operator immediately preceding a term and any suffix operators. It then applies any prefix operators to the left of that operator, grouping them from right to left.

Operator	Action
`++X`	Preincrement
`--X`	Predecrement
`+X`	Plus
`-X`	Minus
`!X`	Logical `NOT`

Infix Operators

The `tsar` command applies `infix` operators, in descending order of precedence, as follows:

Operator	Action
`X^Y`	Exponentiation
`X*Y`	Multiplication
`X/Y`	Division
`X%Y`	Remainder
`X+Y`	Addition
`X-Y`	Subtraction
`X<Y`	Less than
`X<=Y`	Less than or equal
`X>Y`	Greater than
`X>=Y`	Greater than or equal
`X==Y`	Equals
`X!=Y`	Not equals
`X&&Y`	Logical `AND`

`X||Y`	Logical `OR`
`Z?X:Y`	Conditional
`X=Y`	Assignment
`X*=Y`	Multiply assign
`X/=Y`	Divide assign
`X%=Y`	Remainder assign
`X+=Y`	Add assign
`X-=Y`	Subtract assign
`X,Y`	Comma

Suffix Operators

The `tsar` command applies a suffix operator immediately following a term before it applies any other operator. It then applies any suffix operators to the right of that operator, grouping them from left to right. The following list shows the suffix operators:

Operator	Action
`X++`	Postincrement
`X--`	Postdecrement
`X[Y]`	Subscript
`X(Y)`	Function call

7.3.7.3 Built-in Functions

The `tsar` command has the following built-in functions, with the function parameters (given in parentheses) defined at the end of the list:

Function name	Description
`abs`(*exp*)	Return the absolute value of *exp*.
`diff`(*term*)	Return the difference between the current and previous sample values of *term*. If *term* is an array name, then `diff` () returns the sum of the differences of the array elements.
`close`(*str*)	Close the file stream specified by *str*.
`frac`(*exp*)	Return the fractional part of *exp*.

`imax`(*arr*)	Return the index of the maximum element of array *arr*.
`imin`(*arr*)	Return the index of the minimum element of array *arr*.
`int`(*exp*)	Return the integer part of *exp*.
`isdefined`(*sym*)	Return 1 if *sym* is defined. Otherwise return 0.
`rate`(*exp*)	Return the rate of *exp* in counts per second.
`sdiff`(*term*)	Return the difference in the value of *term* between the present and first sample in this boot or reconfiguration period. If *term* is an array name, `sdiff` () returns the sum of the differences of the array elements.
`sum`(*arr*)	Return the sum of the elements in array *arr*.
`system`(*str*)	Pass *str* to the shell for execution.
`strftime`(*fmt*)	Format the time into a string according to *fmt*.
`timen`(*val*)	Return the time stamp in seconds, minutes, or hours. The time stamp is normally represented as `hh:mm:ss`. `timen` (0) returns the time stamp in seconds, which is `t = hh(3600) + mm(60) + ss.` `timen` (1) returns minutes, which is calculated as `t/60`. `timen` (2) returns hours, which is `t/3600`.

The function parameters are as follows:

arr *arr* is an array name. For example:

```
imax (ld_active)
```

exp *exp* is a variable name or a function invocation. For example:

```
abs (runque)
```

variable name

```
rate (diff (scall [ sc_reada ]))
```

function invocation

fmt *fmt* is `NULL` or a valid `strftime` format, which is enclosed in double quotes. For example:

```
strftime ()
```

`ULL` format

```
strftime (" %X ")
```

`strftime` format

str *str* is either a character string enclosed in double quotes or the name of a variable whose value is a character string. For example:

```
close (" disk.data ")
```

character string

```
command = "date; uname -a"

system(command)
```

variable which contains a character string

sym *sym* is a variable name. Names of array elements are not valid symbols. *sym* can be defined by the `tsar -D` option. For example:

```
if (isdefined (ios_e))
```

variable

```
if (isdefined (ld_active [ ld_root ]))
```

invalid

```
tsar -DCPU

if (isdefined (CPU))
```

symbol defined by the `tsar -D` option

term *term* is an expression (*exp*) or an array name. For example:

```
diff (scall [ sc_reada ])
```

expression

```
diff (tp_mounts)
```

array name

val *val* is 0, 1, or 2. For example:

`timen` (0)	Returns the time in seconds
`timen` (1)	Returns the time in minutes
`timen` (2)	Returns the time in hours

7.3.7.4 Built-in Variables

The `tsar` command has the built-in variables shown in Table 50.

Table 50. `tsar` command built-in variables

Variable	Description	Default value
`END_TIME`	Ending time of the sampling period	None
`NR`	Number of sample intervals	None
`OFMT`	Output format for printing numbers	`%.6g`
`OFS`	Output field separator	" "
`ORS`	Output record separator	`\n`
`RSIZE`	Size of the data records in bytes	None
`START_TIME`	Start time of the sampling period	None

7.3.8 Operational Setup

Information about boot and shutdown times can be collected by executing the `sdc`(8) command during the system boot and shutdown process. A `crontab`(1) or `at`(1) job can be used to gather system activity data while the system is running.

Regardless of how data is collected, the `sdc`, `sdcx`, and `tsar -h` commands must be executed by a user who has read access to the `/unicos` and `/dev/kmem` files.

7.3.8.1 Difference between `sdc`(8) and `sdcx`

As explained in Section 7.3.1, page 319, `sdc`(8) generates and compiles `sdcx` (see `sdc`(8)). This process can take a significant amount of wall-clock time on a busy system. Instead of continually generating and compiling `sdcx`, you can use the `sdc -c` option to write a header record and save the `sdcx` binary. The

sdcx binary can then be used to collect data samples, thus bypassing the need to regenerate and recompile sdcx.

The sdcx binary only writes data records and not header records, as mentioned in Section 7.3.4.1, page 333. Thus, when a system reconfiguration affects the sdcx data samples, a new header record must be written and a new sdcx must be generated. Failure to do this can cause tsar to create erroneous reports or to be unable to read the data files.

If no system reconfigurations occur during the sampling interval, sdc can be executed once to write a header record and to save the sdcx binary. sdcx can then be used to collect the data samples.

If reconfigurations are likely during the sampling interval, it is always best to execute sdc. Each time sdc is executed, a header record is written. Data samples also can be collected.

It is important to understand when sdc writes the header and data records. Section 7.3.4, page 333, and Section 7.3.8.4, page 345, describe which command options write the various records. Both sections show examples.

7.3.8.2 Boot Time Data

The sdc command with the -B option must be executed during the system boot process in order for sdc to record the boot time. One way of doing this is to invoke sdc from the /etc/rc.pst. Code similar to the following can be used in /etc/rc.pst:

```
if [ -x /usr/bin/sdc ]
then
        /usr/bin/sdc -B /usr/adm/tsar/dcf.‘date +%m%d‘
fi
```

This will write a header record, which contains the boot time, to the file /usr/adm/tsar/dcf.*MMDD*, where *MMDD* is the current month and day. No data will be sampled.

If no system reconfigurations are going to occur while the system is running, the sdc -c option can also be used in /etc/rc.pst. The -c option will save the sdcx binary in the specified directory. This binary can be executed from a crontab file.

7.3.8.3 Shutdown Data

The system shutdown time is recorded by the `sdc -S` option. This command can be invoked from the `/etc/shutdown.pst` file. Code similar to the following can be put in the file:

```
if [ -x /usr/bin/sdc ]
then
        /usr/bin/sdc -S /usr/adm/tsar/dcf.‘date +%m%d‘
fi
```

This will write a header record, which contains the shutdown time, to the file `/usr/adm/tsar/dcf.`*MMDD*, where *MMDD* is the current month and day. No data will be sampled.

The `sdc -B` and `-S` options may take significant wall-clock time to execute on busy systems, as noted in Section 7.3.8.1, page 343, and under the Section 7.3.10, page 349.

7.3.8.4 `crontab`(1) Entries

System activity data can be collected periodically by `sdc`, `sdcx`, or `tsar -h`, through the `crontab`(1) command. The following examples show how this can be done.

Example 1: This example writes a header record (definitions and meta-data records) followed by three data records each hour. Data is sampled at a rate of one sample every 20 minutes. A header and data record is written on the hour. Additional data records are written at 20 and 40 minutes after the hour.

If a system reconfiguration occurs at 09:25, then `tsar` may interpret the 09:40 data record written incorrectly. `tsar` assumes that the 09:40 record has the format described in the 09:00 header record.

```
0 8-17 * * 1-5 /usr/bin/sdc -i 20m -n 3 /usr/adm/tsar/dcf.‘date +%m%d
```

Example 2: This example writes a header and data record every 20 minutes. Because each data record is preceded by its own header record, system reconfigurations will be detected by `tsar`.

```
0,20,40 8-17 * * 1-5 /usr/bin/sdc -n 1 /usr/adm/tsar/dcf.‘date +%m%d
```

Example 3: This example writes a data record every 20 minutes. No header records are written. It is assumed that both a header record and the `sdcx` binary were generated at boot time. (See Section 7.3.8.2, page 344.) System

reconfigurations will not be detected by `tsar`, as the data file contains only one header record.

```
0 8-17 * * 1-5 /usr/adm/tsar/sdcx -i 20m -n 3 >> /usr/adm/tsar/dcf.‘date+%m%d
```

7.3.9 Examples

System boot, shutdown, and system activity data can be collected by invoking `sdc` as described in Section 7.3.8.2, page 344, Section 7.3.8.3, page 345, and Section 7.3.8.4, page 345. Data gathering and reporting, however, do not have to be scheduled by `cron`(8).

7.3.9.1 `sdc`(8) Data Collection

Like `sadc` (see `sar`(8)), `sdc` can be executed at any time to collect data. For example, to sample data in the background at 10 minute intervals for 2 hours, you can execute the following:

```
nohup sdc -i 10m -n 12 dcf.‘date +%m%d‘&
```

The equivalent `sadc` command is as follows:

```
nohup sadc 600 12 sa.‘date +%m%d‘&
```

`sdc` data is written to the file `dcf.`*MMDD*, and `sadc` output is directed to `sa.`*MMDD*, where *MMDD* is the current month and day.

The startup time for `sdc` is longer than that for `sadc`, because `sdc` must generate and compile the `sdcx.c` code.

7.3.9.2 `tsar`(8) Data Collection

Like `sar`(8), `tsar` can be used to simultaneously sample and format data. For example, to sample data at 5-minute intervals for 1 hour and report swap activity on a machine named `gust`, you can execute the following:

```
tsar -h gust -i 5m -n 12 -r dcf.‘date +%m%d‘ M
```

In this example, `M` is a copy of a script that is found in the `/usr/src/prod/admin/uma/sar.t` directory.

The equivalent `sar` command is as follows:

```
sar -o sa.‘date +%m%d‘ -M 300 12
```

`sdc` data is written to the file `dcf.`*MMDD*, and `sadc` data is directed to `sa.`*MMDD*, where *MMDD* is the current month and day. The ASCII reports are written to `stdout`.

7.3.9.3 `tsar`(8) Report Formatting

The `tsar` utility allows users to generate ASCII reports in a variety of formats. Users specify the report format in scripts described in Section 7.3.6, page 335, and Section 7.3.7, page 337. This differs from `sar`, in that `sar` report formats are hard coded in the source code and cannot be readily changed by the user.

The `/usr/src/prod/admin/uma/sar.t` directory contains `tsar` scripts which correspond to the various `sar` reports. In addition to mimicking `sar` reports, users can write `tsar` scripts that create reports in a site-specific format or graph the data.

The following `tsar` script graphs the read and write activity of a specified device against the sample time. If the script is named `disk_plot`, and the sample data is in the file `dcf.1007`, then the plot for device `1130_00` is generated by the following command:

```
tsar -Ddisk=\"1130_00\" -p dcf.1007 disk_plot
```

```
#
#       tsar script to plot the total disk i/o of a specified device via
#       xgraph.  The i/o is plotted as the number of sectors read and
#       written vs. the time expressed as hours.
#
#       The tsar "-D" option is used to specify the device.
#       For example, to graph the i/o for a disk device named 1130_00,
#       you would execute something similar to:
#               tsar -Ddisk=\"1130_00\" -p dcf.1007 disk_plot

BEGIN {
F_ALL_DISKS = "disk.data"  # file with i/o stats for all disk devices
F_XDATA = "xgraph.data"    # file with info needed to plot the graph
F_CMD = "disk.cmd"         # file with grep command to extract specified i/o stats
system("rm -f disk.data xgraph.data disk.cmd")

if (ios_e) {
        nitem = npdds
} else {
        nitem = ndsds
}
```

```
first_time = 1
#
#       Setup the titles and limits for the graph.
#
printf("TitleText: %s Total Disk IO on %s  %s\n",
   disk, NODENAME, strftime("%D")) > F_XDATA
print("XUnitText: Time of Day")  > F_XDATA
print("YUnitText: Sectors") > F_XDATA
print("YLowLimit: 0") > F_XDATA
}

#
#       For each sample interval, write the i/o statistics for
#       busy devices to file F_ALL_DISKS
#
if (first_time == 1) {
        printf("XLowLimit: %2.4f\n", timen(2)) > F_XDATA
        first_time = 0
}

for (n = 0; n < nitem; n++) {
        if (diff(dr_nblks[n]) + diff(dw_nblks[n]) != 0) {
                printf("%2.4f %9.0f  ", timen(2),
                   diff(dr_nblks[n]) + diff(dw_nblks[n])) >> F_ALL_DISKS
                if (ios_e) {
                        printf("%s\n", pd_name[n]) >> F_ALL_DISKS
                } else {
                        printf("%s\n", dd_name[n]) >> F_ALL_DISKS
                }
        }
}

END {
#
#       Extract the i/o statistics for the specified device and
#       write them to file F_XDATA.
#
printf("XHighLimit: %2.4f\n", timen(2)) > F_XDATA
print("\"\"") > F_XDATA
print("  ") > F_XDATA
printf("grep '%s' %s >> %s\n", disk, F_ALL_DISKS, F_XDATA) > F_CMD

close(F_ALL_DISKS)
```

```
close(F_XDATA)
close(F_CMD)
system("chmod 755 disk.cmd")
system("./disk.cmd")

#
#       Plot the i/o via xgraph.
#
system("/usr/bin/X11/xgraph -title 'Total Disk IO' -tk xgraph.data")
system("rm -f disk.data xgraph.data disk.cmd")
}
```

7.3.10 Limitations

The following are descriptions of limitations to use of the `sdc` and `tsar` commands.

Large system activity data file

The `sdc` output file (system activity data file) may be much larger than the equivalent `sar`(8) data file. The size of the `sdc` output file can be controlled by sampling a subset of the system activity counters through the `sdc -R` option. Ldcache and disk data can require an enormous amount of disk space, so sample them only when necessary.

Undetected ldcache or disk reconfigurations

If either ldcache or disks have been reconfigured, a header record must be written to the system activity file. If a header record is not written, `tsar` may report erroneous ldcache or disk information, or `tsar` may abort.

Long `sdc`(8) execution time

The `sdc` command always generates and compiles `sdcx`. Thus, when you specify the `-B` or `-S` options to `sdc`, it may take a while for them to complete on a busy system. This can be particularly true when `sdc -S` is executed from the system shutdown script.

`imin`() examines unused array entries

Arrays such as `dr_nblks`, `ld_active`, and `tp_mounts` often contain unused entries. `imin`() examines these entries and may return an index to them, because unused array entries contain 0.

No user-defined arrays

`tsar` does not support user-defined arrays. Therefore, it may be difficult to save or sum array elements across restarts or configurations.

No string comparison functions

Built-in `tsar` string comparison functions are not currently available.

Limited access to the first data record

Data from the first record in each boot or reconfiguration period cannot be accessed except through some of the built-in functions.

No `sdcx`(8) record locking (see `sdc`(8))

The `sdcx` binary does not lock records when writing output. Thus, if multiple `sdcx` or `sdc` processes write to the same output file, the output could become corrupt.

No support for foreign file systems

The `sdc` command locks the records in the output file before writing the output. Because record locking is not supported on foreign file systems (such as network file systems), `sdc` cannot write to a file that is on such a file system.

No support for `tsar`(8) running on the SunOS operating system

The `tsar` command may not run on the SunOS Release 4.1 operating system. The system must have an ANSI C compiler (`acc`) to run `tsar`. This feature is not supported in the current release.

Disk I/O units may differ

The `sdc` command records disk I/O in units of sectors, but the size of a sector differs according to the disk device type. (See

Section 7.3.3.9, page 328.) The device type is not available through `sdc`, so `tsar` is unable to adjust the data to a common unit.

7.4 Disk Usage Monitoring

One of the most important responsibilities of the system administrator is to monitor system disk usage and to ensure that users have sufficient free space on their file systems to accomplish their work.

The following commands are useful in monitoring disk use:

Command	Description
`diskusg`(8)	Summarizes the disk usage on a file system by file ownership and identifies users who are using most of the space on a file system. The `/usr/lib/acct/diskusg` command is the preferred command for summarizing disk use. It is faster and more accurate than `du`(1) because it bypasses the file system software in the kernel.

`du`(1)	Provides a summary of the disk use on a file system, by directory structure. The `-s` option provides the total number of disk blocks used under each directory (or file) named. For example, if user accounts are stored on the `/u` file system, the following lists the total number of blocks used by each account on the file system:
	`cd /u`
	`du -s *`
	The output of `du`(1) may be piped into `sort`(1), as follows, providing a sorted list of directories that use the most disk space (allowing you to identify users who are using the most disk space):
	`du -s * | sort +nr`
	This command is useful for locating the points at which users are using most of the file system's disk space and, in conjunction with `diskusg`, identifying users who may own files located in directory subtrees other than their home directories. Results from the `du`(1) program can be affected by changes to the file system during execution of `du`.

`find`(1) Identifies large files in an emergency when disk space is scarce; however, `find` does not identify temporary files whose directory links are removed. The following command line, when executed from the topmost directory of a file system, lists all files on file systems that are larger than 1 Mbyte, sorting them in order of size:

```
find . -size +1000000c -print |
  xargs ls -li | sort +5nr
```

With such a list, you can look for a few large files that are likely candidates for deletion because they have not been modified or used for a specified length of time (this value is site-dependent, but is usually a month), because the owner no longer needs the file, or because the file is unreasonably large.

Similarly, the `find` command can look for all files that have not been modified or used within a certain span of time (for example, the last 3 months).

To free disk space, you may find it appropriate to perform one of the following functions:

- Archive such files to tape for permanent storage.
- Archive such files on tape for eventual deletion, perhaps using a set of four tapes rotated on a weekly basis. This would allow a user 1 month to ask for retrieval of an archived file before its tape is reused.
- Delete such files.
- You can use the `crontab`(1) command to execute `find` regularly during off-peak hours in order to generate a daily list of large, old files that can be archived to tape by operators.