

U  
S  
E  
U NIX  
USENIX

CONFERENCE  
PROCEEDINGS

USENIX Systems Administration  
(LISA VII) Conference

November 1-5, 1993  
Monterey, California

For additional copies of these proceedings contact

USENIX Association  
2560 Ninth Street, Suite 215  
Berkeley, CA 94710 USA  
Telephone: 510-528-8649

The price is \$25 for members and \$33 for nonmembers.

Outside the U.S.A. and Canada, please add  
\$12 per copy for postage (via air printed matter).

Past USENIX Large Installation Systems Administration Workshop  
and Conference Proceedings (price: member/nonmember)

Large Installation Systems Admin. I Workshop	1987	Philadelphia, PA	\$4/\$4
Large Installation Systems Admin. II Workshop	1988	Monterey, CA	\$8/\$8
Large Installation Systems Admin. III Workshop	1989	Austin, TX	\$13/\$13
Large Installation Systems Admin. IV Conference	1990	Colorado Spgs, CO	\$15/\$18
Large Installation Systems Admin. V Conference	1991	San Diego, CA	\$20/\$23
Large Installation Systems Admin. VI Conference	1992	Long Beach, CA	\$23/\$30

Outside the U.S.A. and Canada, please add \$8  
per copy for postage (via air printed matter).

Copyright 1993 by The USENIX Association.  
All rights reserved.

This volume is published as a collective work.

ISBN 1-880446-56-1

System 7 and Macintosh are trademarks of Apple Computer Corp.  
Auspex NetServer NS5000 is a trademark of Auspex Systems, Inc.

Amiga is a trademark of Commodore, Inc.

GIF may be a trademark of CompuServe.

Wren might be a trademark of Control Data Corporation.

Ultrix, DEC, VAX, and VMS trademarks of Digital Equipment Corp.

FrameBuilder and FrameMaker are trademarks of Frame Technology Corp.

MPE, HP, HP-UX, and HPDesk are trademarks of Hewlett-Packard Company.

Interleaf is a trademark of Interleaf, Inc.

MVS/TSO, IBM, AIX, VM, Sysctl, OS/2, DOS are trademarks of International Business Machines

cc:Mail and Lotus 1-2-3 are trademark of Lotus Development Corp.

Hesiod and Kerberos are trademarks of Massachussets Institute of Technology.

Windows and MS Windows are trademarks of Microsoft.

System V, Unix, and Novell are registered trademarks of Novell, Inc.

DFS and OSF/DCE is a trademark of Open Software Foundation.

IRIX is a trademark of Silicon Graphics Corp.

Catalyst CDWare, NFS, SunOS, Answerbook, OpenWindows, and Solaris are trademarks of Sun Microsystems.

AFS is a trademark of Transarc Corp.

USENIX acknowledges all trademarks appearing herein.

Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste. 

# **USENIX Association**

## **Proceedings of the Seventh Systems Administration Conference (LISA VII)**

**November 1-5, 1993  
Monterey, CA, USA**

# TABLE OF CONTENTS

Acknowledgments .....	iv
Author Index .....	v

## Plenary Session

**Wednesday (8:30-10:00)**

**Chair: Bjorn Satdeva**

Opening Remarks and Announcements <i>Bjorn Satdeva, /sys/admin, inc.</i>	
Keynote Address: A Management View of Systems Administration <i>John Black, Oracle</i>	

## Vis Á Vis

**Wednesday (10:30-12:00)**

Collaborative Networked Communication: MUDs as Systems Tools .....	1
<i>Rémy Evard, Northeastern University</i>	
Horses and Barn Doors: Evolution of Corporate Guidelines for Internet Usage .....	9
<i>Sally Hambridge &amp; Jeffrey C. Sedayao, Intel Corp.</i>	
Our Users Have Root! .....	17
<i>Laura de Leon, Mike Rodriquez, &amp; Brent Thompson, Hewlett-Packard Company</i>	

## Software Installation and Customization

**Wednesday (1:30-3:00)**

LUDE: A Distributed Software Library .....	25
<i>Michel Dagenais, Ecole Polytechnique de Montréal; Stéphane Boucher, Bell-Northern Research; Robert Gérin-Lajoie, Université de Montréal; Pierre Laplante, Centre de Recherche Informatique de Montréal; Pierre Mailhot, Université de Montréal</i>	
The Corporate Software Bank .....	33
<i>Steven W. Lodin, Delco Electronics Corporation</i>	
Customization in a UNIX Computing Environment .....	43
<i>Craig E. Wills, Kirstin Cadwell, &amp; William Marrs, Worcester Polytechnic Institute</i>	

## Source Control

### Wednesday (3:30-5:00)

Local Disk Depot – Customizing the Software Environment .....	51
<i>Walter C. Wong, Carnegie Mellon University</i>	
Methods for Maintaining One Source Tree in a Heterogeneous Environment .....	57
<i>Bjorn Satdeva, /sys/admin, inc.</i>	
Towards a POSIX Standard for Software Administration .....	67
<i>Barrie Archer, ICL</i>	

## Getting Down To Busines

### Thursday (8:30-10:00)

Managing the Mission Critical Environment .....	81
<i>E. Scott Menter, Enterprise Systems Management Corporation</i>	
Open Systems Formal Evaluation Process .....	87
<i>Brian William Keves, Systems And Network Management</i>	
A Case Study on Moves and Mergers .....	93
<i>John Schimmel, Silicon Graphics Inc.</i>	

## System Administration by Delegation

### Thursday (10:30-12:00)

Guerrilla System Administration: Scaling Small Group Systems Administration To a Larger Installed Base .....	99
<i>Tim Hunter &amp; Scott Watanabe, University of Colorado, Boulder</i>	
Role-based System Administration or Who, What, Where, and How .....	107
<i>Dinah McNutt, Tivoli Systems</i>	
Delegation: Uniformity in Heterogeneous Distributed Administration .....	113
<i>Jean-Charles Grégoire, INRS-Télécommunications</i>	

## System Administration Tools

### Thursday (1:30-3:00)

satool – A System Administrator's Cockpit, An Implementation .....	119
<i>Todd Miller, Christopher Stirlen, Evi Nemeth, University of Colorado, Boulder</i>	
Sysctl: A Distributed System Control Package .....	131
<i>Salvatore DeSimone &amp; Christine Lombardi, Project Agora, IBM T. J. Watson Research Center</i>	
Automated System Monitoring and Notification with Swatch .....	145
<i>Stephen E. Hansen &amp; E. Todd Atkins, Stanford University</i>	

## Disk Management

### Thursday (3:30-5:00)

Where Did All The Bytes Go? .....	157
<i>Dinah McNutt, Tivoli Systems; Michael Pearlman, Rice University</i>	
A Practical Approach to NFS Response Time Monitoring .....	165
<i>Gary L. Schaps &amp; Peter Bishop, Cirrus Logic, Inc.</i>	
The Amanda Network Backup Manager .....	171
<i>James da Silva &amp; Ólafur Guðmundsson, Department of Computer Science, University of Maryland</i>	

## Information Tools

### Friday (8:30-10:00)

PLOD: Keep Track of What You're Doing .....	183
<i>Hal Pomeranz, QMS, Inc.</i>	
How to Keep Track of Your Network Configuration .....	189
<i>J. Schönwälder &amp; H. Langendörfer, TU Braunschweig, Germany</i>	
Forecasting Disk Resource Requirements for a Usenet Server .....	195
<i>Karl L. Swartz, Stanford Linear Accelerator Center</i>	

## The Human Behind The Root

### Friday (10:30-12:00)

Simplifying System Administration Tasks: The UAMS Approach .....	203
<i>Roland J. Stolfa, Oklahoma State University</i>	
System Administration as a User Interface: An Extended Metaphor .....	209
<i>Wilson H. Bent, Jr., University of Southern California</i>	
The System Administration Maturity Model – SAMM .....	213
<i>Carol Kubicki, Motorola Cellular Infrastructure Group</i>	

## Potpourri

### Friday (1:30-3:00)

Establishing and Administering a Public Access Internet facility .....	227
<i>Sheri Byrne, Gemini Learning Center</i>	
Implementing Execution Controls in Unix .....	237
<i>Todd Gamble, WilTel Network Services</i>	
HLFSD: Delivering Email to Your \$HOME .....	243
<i>Erez Zadok, Columbia University; Alexander Dupuy, System Management ARTS</i>	

# ACKNOWLEDGMENTS

## GENERAL CHAIR

Bjorn Satdeva, */sys/admin, inc.*

## PROGRAM COMMITTEE

Brent Chapman, *Great Circle Associates*  
Lee Daemon, *Castle PAUS*  
Tina M. Darmohray, *Lawrence Livermore National Labs*  
Janet Frazer, *UNIX System Laboratories*  
Helen Harrison, *SAS Institute*  
Dinah McNutt, *Tivoli Systems*  
Bryan McDonald, *SRI International*  
Arch Mott, *cisco Systems, Inc.*  
Jeff Polk, *Berkeley Software Design, Inc.*  
Greg Rose, *Australian Computing & Comm. Inst.*  
Peg Schafer, *Bolt, Beranek, & Newman, Inc.*  
Steve Simmons, *Inland Sea*  
Lisa Y. Weissler, *RAND Corporation*  
Pat Wilson, *Dartmouth College*  
Elizabeth Zwicky, *SRI International*

## TUTORIAL COORDINATOR

Daniel V. Klein, *USENIX Association*

## TERMINAL ROOM COORDINATOR

Arch Mott, *cisco Systems, Inc.*

## PROCEEDINGS PRODUCTION

Rob Kolstad, *BSDI*  
Carolyn Carr, *USENIX*  
*Malloy Lithographing, Inc.*

## USENIX MEETING PLANNER

Judith F. Desharnais, *USENIX Association*

## USENIX EXECUTIVE DIRECTOR

Ellie Young, *USENIX Association*

## USENIX SUPPORT STAFF

Diane DeMartini, *USENIX Association*  
Patrick Dunne, *USENIX Association*  
Toni Veglia, *USENIX Association*

## VENDOR DISPLAY COORDINATOR

Cynthia Deno, *USENIX Consultant*

# AUTHOR INDEX

Barrie Archer .....	67
E. Todd Atkins .....	145
Wilson H. Bent, Jr. ....	209
Peter Bishop .....	165
Stéphane Boucher .....	25
Sheri Byrne .....	227
Kirstin Cadwell .....	43
Michel Dagenais .....	25
Salvatore DeSimone .....	131
Alexander Dupuy .....	243
Rémy Evard .....	1
Robert Gérin-Lajoie .....	25
Todd Gamble .....	237
Jean-Charles Grégoire .....	113
Ólafur Guðmundsson .....	171
Sally Hambridge .....	9
Stephen E. Hansen .....	145
Tim Hunter .....	99
Brian William Keves .....	87
Carol Kubicki .....	213
H. Langendörfer .....	189
Pierre Laplante .....	25
Laura de Leon .....	17
Steven W. Lodin .....	33
Christine Lombardi .....	131
Pierre Mailhot .....	25
William Marrs .....	43
Dinah McNutt .....	107
Dinah McNutt .....	157
E. Scott Menter .....	81
Todd Miller .....	119
Evi Nemeth .....	119
Michael Pearlman .....	157
Hal Pomeranz .....	183
Mike Rodriguez .....	17
Bjorn Satdeva .....	57
J. Schönwälder .....	189
Gary L. Schaps .....	165
John Schimmel .....	93
Jeffrey C. Sedayao .....	9
James da Silva .....	171
Christopher Stirlen .....	119
Roland J. Stolfa .....	203
Karl L. Swartz .....	195
Brent Thompson .....	17
Scott Watanabe .....	99
Craig E. Wills .....	43
Walter C. Wong .....	51
Erez Zadok .....	243



# satool – A System Administrator's Cockpit, An Implementation

Todd Miller, Christopher Stirlen, Evi Nemeth  
– University of Colorado, Boulder

## ABSTRACT

Monitoring a large number of machines in a distributed environment can be time consuming and inefficient. Often a system administrator discovers there is a problem with a machine or network link when a frustrated user calls. **satool** provides a way to efficiently monitor groups of machines and identify problems and potential problems quickly; it's sort of an early warning system for sysadmins.

**satool** is composed of three independent parts: an SNMP (Simple Network Management Protocol) agent that runs on each machine being monitored, a database that collects data from each client machine, and a graphical user interface (GUI) that acts as the interface between the user and the database.

## Introduction

With the advent of fast, inexpensive workstations, the standard computing environment has changed from a few Vaxen to a distributed network that has become increasingly complex and difficult to administer. Monitoring large numbers of machines in a distributed environment can be time consuming and inefficient. In such an environment, it is usually not possible to see the warning signs that point to imminent disaster for a machine or network. Instead, system administrators find themselves fighting fires when their time would be better spent elsewhere. **satool** provides a way to efficiently monitor groups of machines and find problems and potential problems quickly in a straightforward manner.

The **satool** server maintains a database of client machines and the values of supported variables gathered during the last poll. Normally, a client is added to the database when the server receives a message from the SNMP trap daemon that the client's SNMP agent is running. However, the server also maintains an on-disk copy of the database that is read in on invocation of the server. Because of this, the database will survive system crashes. Client machines can also be specified in the server's configuration file to "prime the cache" of machines to monitor. The time between polls is configurable on a per-machine basis (overriding a stated default). The server listens for database requests from the display system on port 0xFB1 (that's a one not an I).

The **satool** display system includes an X based GUI (Graphical User Interface) built using **tcl/tk**. It supports a hierarchical top level view of the machines being monitored. Machines can be grouped to allow the screen area to scale with the number of machines being watched. For example, a sysadmin can configure his workstation to display

the machines he is directly responsible for, the machines that his colleague on vacation is responsible for, and his network gateway to the outside world. After traversing the hierarchy to an individual machine, the display contains visual widgets representing the health of that machine: disk space free, memory statistics, cpu activity, mail queue length, nfs statistics, etc. The user can choose the form of the widget (currently text, a thermometer display, or a sliding window histogram) to display each quantity.

Alarm conditions for each variable can be expressed in the configuration files as well. If the value of a variable crosses the alarm threshold, a special action (blinking, beeping, reverse video, digital pager, etc.) is taken on the screen to indicate it. Alarms are propagated to the top level display, thus if /var/tmp fills up on host *heineken* on the *beers* subnet of the *cs* domain and if the groups are set appropriately, the alarm condition would be noted in the widgets representing *heineken*, *beers*, and the *cs* domain. A user would see or hear the alarm independent of which level he was actively displaying at the time.

A working prototype of **satool** exists. It will be used this fall by the Computer Science Department's undergrad lab sysadmin group and graduate/faculty research sysadmin group to monitor about 300 machines. We expect to use feedback from these groups to expand the sysadmin MIB and to build more display widgets. The code will be freely available with a Berkeley style copyright notice.

## Design Goals

**satool** was designed to be *flexible*, *scalable*, *easy to manage*, and to *reuse* existing tools.

*Flexibility* is achieved through the use of configuration files and **satool's** modular design. On the data gathering side, configurable items include: the actual data to be collected and how often to collect it. From within the display system the user can also configure threshold values that indicate a problem and the action to take if a threshold is exceeded. Display system configuration also selects the hosts or devices to display, the type of widget to represent a variable, and personal preferences for items like the arrival of an alarm condition (for example color, sound, blinking, etc).

We wanted **satool** to be used on both small and large networks, thus *scalability* was important. Data collection has a low impact on the host and the networks on which **satool** is running. Allowing the user to interact with hosts in a hierarchical manner also increases scalability.

When dealing with large numbers of hosts *manageability* is essential. **satool** provides sensible defaults for parameters in configuration files. No changes are required to monitor a new host; it will be added when it sends a trap to the server. It is important to note that if the correct site-specific values are compiled into **satool**, there is only one configuration file to maintain, the server's.

In designing **satool** we also wanted to *reuse* tools where possible to avoid reinventing the wheel. To this end **satool** takes advantage of many standard UNIX utilities to gather data. We also use the **tcl** and **tk** languages from John Ousterhout at the University of California, Berkeley for the display and SNMP 1.1b from Carnegie Mellon University. An SNMP agent and sysadmin MIB (Management Information Base) provide for communication between the data collection and data gathering activities. The actual data gathering is done by an SNMP agent running on the hosts being monitored. We have extended the CMU SNMP 1.1b agent and MIB to include variables of interest to UNIX system administrators not included in the standard network or host MIBs.

### Components

**satool** is made up of three distinct components: a data gathering agent, a data collecting server, and a display system. The agent is an SNMP agent extended to use the **satool** MIB. It uses "helper scripts" to massage data from standard UNIX commands. The server polls agents at set intervals and stores the resulting data in an *ndbm(3)* database. It also services requests from the display system using an SMTP-like protocol. The display system (written in **tcl** and **tk**) interacts with the user and initiates data transfers from the server.

#### satool Daemon (satoold)

**satoold** has three main functions: gather data from its clients, store it in a database, and service

requests to access that data from the GUI. It also writes its process id to a file (*/etc/satoold.pid* by default) for convenience.

### Data Gathering

**satoold** polls clients for data via SNMP at configurable intervals. All polling is done by a forked process which passes data back to the parent via a UNIX domain socket. Polling times for clients are kept in a linked list (called the "timer queue" although it is not truly a queue) sorted by time to poll (in UNIX time format). To allow concurrent polls, a compiler-time variable specifies the number of simultaneous polling processes allowed. A counter keeps track of the number of polling children along with an array of their process ids.

The flow of control is as follows:

- **satoold** is notified by the SNMP trap daemon that a client has come up.
- The client is inserted into the timer queue if it is not already present there.
- If the client is not already in the database, it is added. Otherwise, the client's current database entry is updated to reflect the fact that the client is now up.
- An alarm goes off, signifying that it is time to poll a client.
- A signal handler is called, and a child is forked to poll the client.
- The child completes the poll and sends the data back to its parent via a UNIX domain socket.
- If the connection to the client times out (the timeout is defined at compile time), that machine is now marked as down and its polling frequency is reduced (however, the polling frequency never reaches zero).

### Data Storage

Client data is stored in an *ndbm(3)* database, keyed on the fully qualified hostname. The use of *ndbm(3)* allows for some basic crash recovery since a copy of the database is kept on disk. As such it can survive system downtime.

- The on-disk database is read on invocation of **satoold** and a timer queue is created based on the information in the database.
- Obviously bogus (empty) keys are discarded (this is the most common cause of database corruption we have seen).

### Servicing GUI Requests

**satoold** accepts connections on port 4017 (0xFB1 in hex). Connections timeout after five minutes of inactivity (configurable at compile time). The protocol used is similar to *SMTP* (Simple Mail Transport Protocol). The protocol commands are:  
HELO Say hello to the daemon;  
HELP Prints a short help message;  
LIST Lists the all clients in the database;  
GET Gets the data for a particular machine.

**satoold** responds to each command with a three digit completion code and a status/error message (in text) before returning the requested data (also like *sendmail*). The breakdown of the three digits is as follows:

**First Digit:**

- 2 Command completed
- 5 Command failed with a fatal error

**Second Digit:**

- 0 Syntax
- 1 Information
- 2 Connection
- 3 Host
- 5 Data

**Third Digit:**

The third digit differentiates between codes that have the same first two digits. For instance, code 220 is the "connection established" greeting, and 221 is the "connection closed" message.

**satoold Configuration File**

Parameters to **satoold** can be configured through its configuration file (*/usr/local/etc/satoold.conf* by default). The polling interval for most hosts is specified by the *interval default* entry. It defaults to 300 seconds if not specified. Individual host values specified using *interval hostname* entries override the default. Hosts to preload automatically, without receiving a trap from the host, are specified with a *preload hostname* entry. Anticipated use of the preload feature is to include main servers in the config file to prime the cache. By using host-specific intervals one could also poll those hosts more frequently.

A sample *satoold.conf* file follows:

```
# example satoold.conf
#
# interval default seconds
# interval hostname seconds
# preload hostname
#
interval default 300
preload hazelrah.cs.colorado.edu
preload alta.cs.colorado.edu
preload kinglear-gw.cs.colorado.edu
#preload fiver.cs.colorado.edu
interval romeo.cs.colorado.edu 600
interval alta.cs.colorado.edu 200
interval pipkin.cs.colorado.edu 400
```

**SNMP Agent**

The SNMP (Simple Network Management Protocol) agent used in **satool** is based on *snmp1.1b* from Carnegie-Mellon University. This release is MIB-I compliant. There are two major differences between CMU and **satool** versions: the

configuration file and support for **satool** variables in the MIB (Management Information Base).

**Config File**

The **satool** SNMP agent's configuration file (*/usr/local/etc/satool-agent.conf* by default) is currently only used to specify the host to send coldstart traps to (in the absence of a configuration file a default host is used that is specified at compile time). On invocation, the agent will send a coldstart trap to the host listed in the config file (if that file exists) to announce its presence. The SNMP agent runs on the hosts being monitored and does the actual data gathering. We have extended the CMU SNMP 1.1b agent and MIB to include variables of interest to UNIX system administrators not included in the standard network or host MIBs.

**satool Variables**

The agent now supports variables defined by the **satool** MIB. The values for most of the variables are obtained via "helper scripts" that run standard UNIX commands and parse the output into a form that the agent can use. Their are two major reasons to use these "helper scripts." First is the increased portability and flexibility scripts provide. Second is our desire to use existing UNIX tools where available.

There is, however, a problem with the approach described above. It is extremely slow for a large number of variables because the agent does a *popen*(3) call which forks and execs the script for each variable. A solution is to have the script output all the variables we might be interested in at once and cache the values. For example, instead of calling a *vmstat*(1) helper script eighteen times, we call it once and cache the values for ten seconds (configurable at compile time). Subsequent requests for any of the variables will get the cached values. This speeds things up considerably and the ten second granularity is considered acceptable. The full **satool** MIB can be found in appendix A of this paper.

**Sample MIB Entry**

The following MIB entry describes the load average to an SNMP agent.

```
satoolLoadAve OBJECT-TYPE
SYNTAX INTEGER
ACCESS read-only
STATUS optional
::= { satool 1 }
```

The first line defines an object called *satoolLoadAve*. It is of type *INTEGER* (SNMP doesn't have floats so we multiply the load average by 100 and then truncate it). The *ACCESS* line indicates that *satoolLoadAve* is read-only (wouldn't we all like to see writable load averages!). The status of the object is "optional" because it is not part of the

standard MIB. The last line describes where the object "fits in" to the MIB hierarchy (which is visually a tree). In this case `satoolLoadAve` is the first leaf in the `satool` branch. Its full path is `.iso.org.dod.internet.private.enterprises.cu.satool.satoolLoadAve.0`. The terminating zero indicates that `satoolLoadAve` is a leaf node.

### Display System

The display system provides an X-based Graphical User Interface (GUI) for viewing the contents of the `satool` database. The GUI, written with `Tcl` and `Tk`, uses configuration files to setup `satool's` hierarchical view of the hosts being monitored, to specify thresholds for data values, and to set the type of display objects. The display system also checks for data values out of bounds and notifies the user when an alarm threshold has been crossed. It is intended to be run on a sysadmin's workstation or dedicated management station sitting quietly in the background until an alarm is triggered when it will notify the sysadmin of impending trouble.

### Tcl and Tk

`Tcl` (pronounced "tickle"), which stands for "tool command language", is an interpretive programming language built from a library of C procedures. `Tk` is an X11 toolkit that is accessible from `Tcl`. They were developed by John K. Ousterhout at the University of California, Berkeley.

`Tcl/Tk's` strengths are its portability, extensibility, and communication. `Tcl` and `Tk` have been ported to most UNIX based platforms that support X11R4 or higher. Scripts written in `Tcl/Tk` can be extended and modified at run time without recompilation. `Tcl/Tk` provides a powerful communication command called `send`, which allows different `Tcl/Tk` processes to communicate with each other.

`Tcl/Tk` was chosen for the `satool` project (over `Interviews` or `Suit`) because it is mature and easy to use. Chris, who did the GUI part of `satool`, took the manual home one evening for bed-time reading. By afternoon the following day, after only four hours playing with it, he had a small application built. After this experience other windowing toolkits were not seriously considered.

### Configuring the Display

`satool` represents groups of hosts hierarchically, very much like the `netgroup` concept defined by Sun. The groups can be nested; there is only a practical limit to the depth of the hierarchy. Clicking on the icon representing a group zooms you to the members of that group.

Groups are defined by the configuration file `satool-display.conf` using the syntax of the `/etc/aliases` file, namely:

```
groupname: member,member, ...
```

A member can be either an individual host or a group, for example:

```
ugradlab: kinglear, kinglear_clients
kinglear_clients: hamlet, ophelia, \
juliet, caesar, romeo
```

The alarm thresholds for each variable monitored are also specified in this file on a per host basis. Macros are supported, so that a group of machines with the same values can be configured in a single line. The syntax is:

```
macro = variable value, variable ...
machine: variable value, variable ...
machine: macro
groupname: macro
```

For example, to configure all the `kinglear` clients to use the same alarm threshold values:

```
hp_common = load_avg 5,num_procs 100
kinglear_clients: hp_common
kinglear: load_avg 10,num_procs 200
```

If a machine object is not mentioned in the configuration file, it will be assigned default values which are set in the GUI code. There are two additional entries: the domain designation and the server designation. Specifying a domain allows all hostnames following it to use short names, rather than fully qualified names. It is in effect until another domain entry occurs. The syntax is:

```
domain: domain-name
```

The server entry specifies the hostname of the machine that contains the database, for example:

```
server: kinglear
```

### Display Objects

`satool's` display objects, currently a sliding histogram, a thermometer, and a text object, are used to view the data in the database. The frequency at which a display widget is updated can be changed by selecting a new value from the Frequency menu at the lower left corner of the window. The user cannot choose an update rate that is more frequent than the database polling rate. The default value for the update frequency is 5 minutes; it is set in the `satool-display.conf` file but can be overridden by the user.

We call a histogram which displays the values from the last  $n$  (10 by default) time intervals a sliding histogram. This type of widget is appropriate for variables like the load average and gives a sense not only of the current value but of its first derivative. The minimum, maximum, and running average are also shown. The histogram will scale as appropriate.

The thermometer widget is used to display one dimensional data. The current implementation

uses a percentage value instead of scaling the data. The thermometer displays the current data value, the maximum and minimum running values, and the running average. Figure 1 shows a typical thermometer.

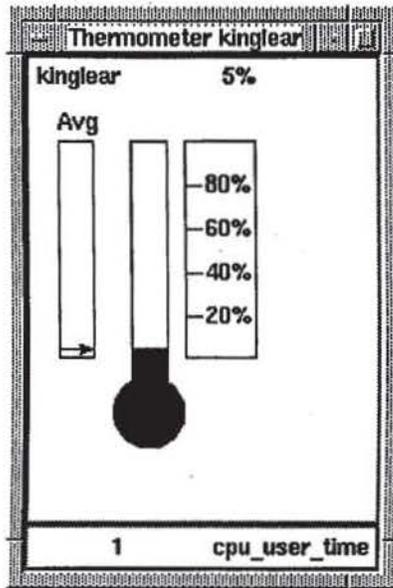


Figure 1: Thermometer

satool's text object displays the name of the machine being monitored and the requested data items. Four data values: average, minimum, maximum and threshold are shown.

### Configuration

The sysadmin can configure their satool display to select the hosts whose data will be displayed, types of widgets to use, alarm thresholds and actions, and screen layout. The users configuration file is called .satoolrc and should be in his home directory. Examples of each configuration primitive are listed below:

- configure display object for each data item
 

```
display: load_avg H, \
          disk_usage T
```
- configure alert colors and icons
 

```
alert1: green client1
alert2: yellow client2
alert3: red client3
alert4: black client4
```

(above syntax: alert\_name: color icon)
- select groups from satool-display.conf
 

```
groups: kinglear_clients, alta_clients
```
- select macros from satool-display.conf
 

```
kinglear_clients: hp_common
```
- set thresholds
 

```
alta_clients: load_ave 3
```
- define groups
 

```
my_group: kinglear_clients, kinglear
```
- save state of display objects
 

```
Histogram kinglear.cs.colorado.edu \
          load_ave +20+20
```

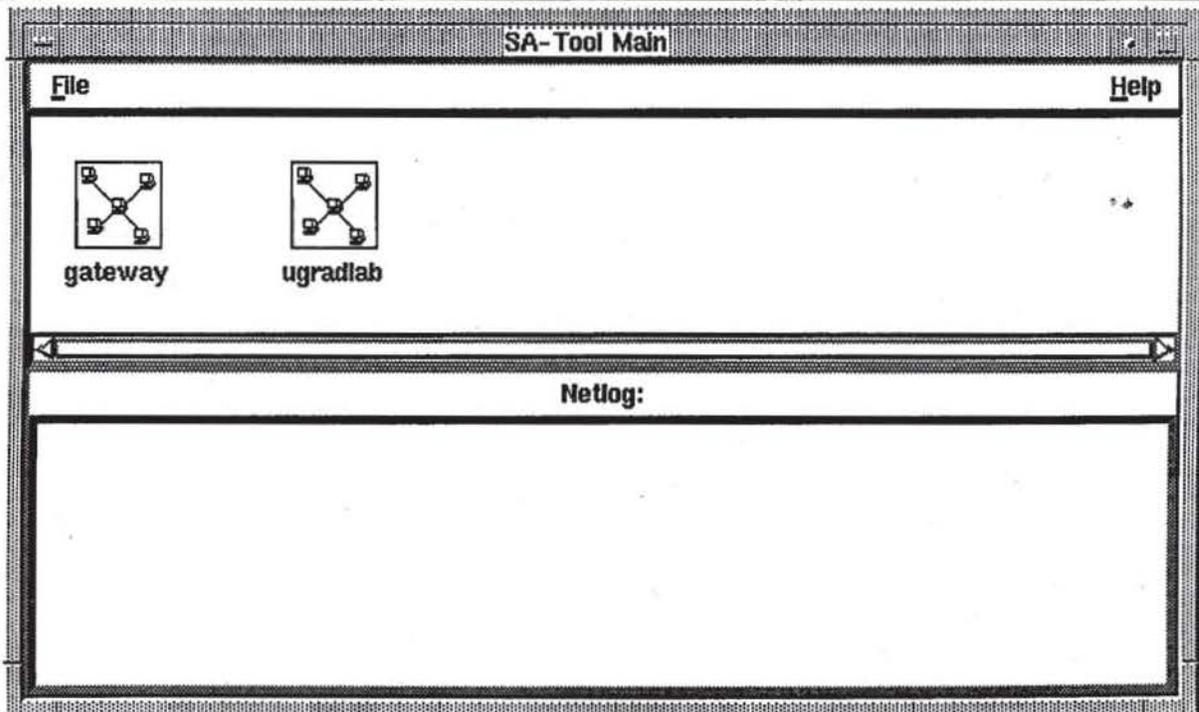


Figure 2: Main Window

- save state of main window  
Main +274+245
- set default display object  
default: X
- set frequency default (in minutes)  
frequency: 10
- set number of histogram bars  
h\_bars: 15
- set pager phone number (also in satool-display.conf)  
pager: 303-555-1212

**Windows and Menus**

In addition to the data display objects, **satool** has four other windows: a main window, viewer windows, help and message windows.

The main window contains the root of the host hierarchy. From it you can traverse the hierarchy via viewer windows. It also provides an interface to netlog, a local tool for displaying system events sent via the 4.3 BSD *syslog*(3) facility. Figure 2 shows a typical main window.

The main window's File menu has three options: Exit, Save Config, and Save Config on Exit. Saving the configuration writes the current screen layout, display options, and window locations to the users .satoolrc startup file. To configure a netlog section of the main window include the following line in your .satoolrc file:

```
netlog: loghost.domain
```

for example:

```
netlog: kinglear.cs.colorado.edu
```

Viewer windows show group membership and give the user access to the data **satool** monitors. The Data menu, which is opened by selecting a host, lists the data items available. The option All will display a text object showing all the data items for the selected host. The viewer window's Display menu allows the user to override the type of display object that will appear. Once the user has selected an option from this menu, it will remain selected until the window is closed or another option is chosen. Figure 3 shows a typical viewer window.

Help on the use of the GUI is available through the Help Menu.

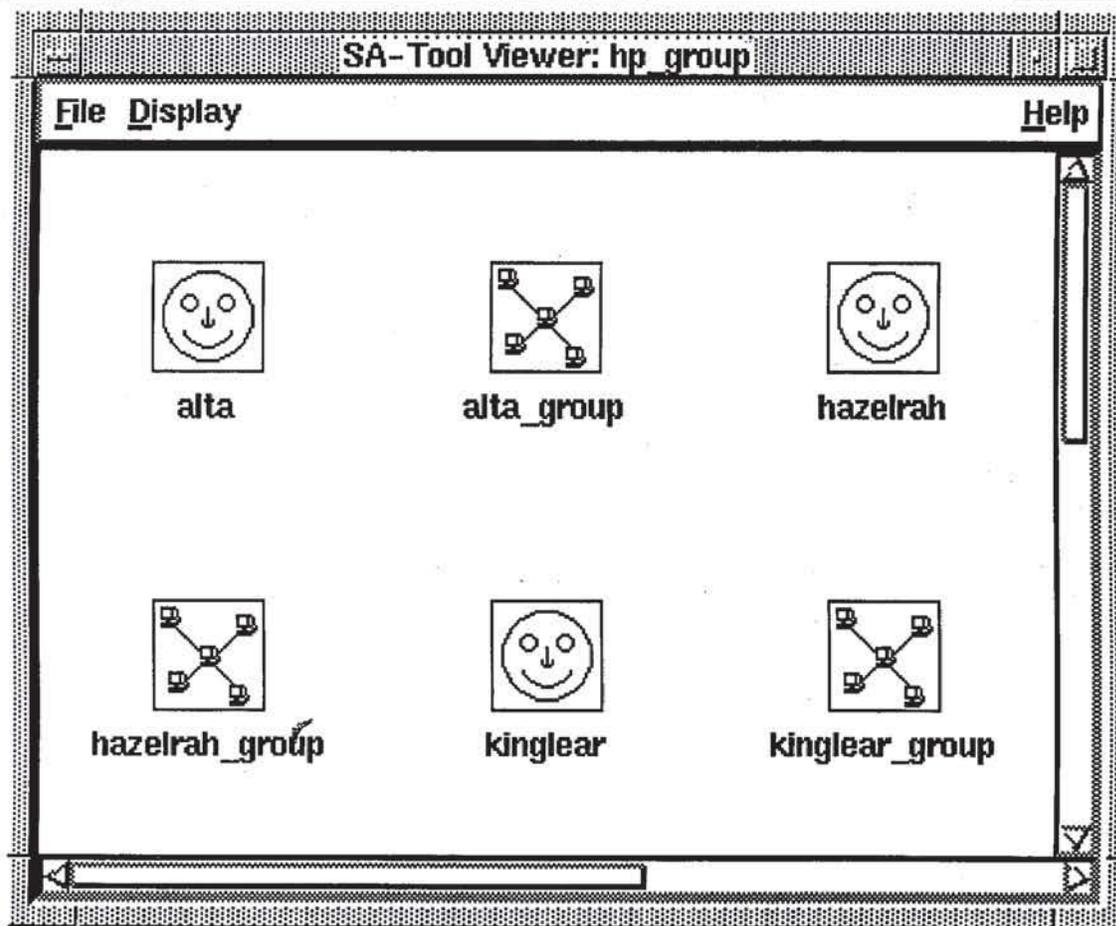


Figure 3: Viewer window

satool's alert system notifies the user when data values cross stated thresholds. The data object causing the alert will change appearance (color, reverse video, new icon, etc.); the alert will cascade up through the hierarchy.

In addition, the user can configure additional notification methods on a per machine basis. The current list of alarms implemented are:

- alert\_alarm: sends annoying beeps
- alert\_mtf: moves the offending object to the front of the screen
- alert\_flash: flashes the offending object
- alert\_mesg: creates the message window
- alert\_pager: page a human being

The message window receiving an alert will report the host, data item, current value and threshold value. It also displays error messages including configuration errors, start-up errors and errors in accessing the database.

### Extending satool

Adding a new variable to monitor is fairly easy. Following is a list of the steps we took to include a new variable that does a *traceroute* (8) to an outside machine to make sure gatewaying is working correctly. In each case, the area where code needs to be added is marked with `EXTEND_HERE` in a comment.

#### Agent

The first step is to write a helper script that the SNMP agent will call. In this case, it is a *perl* (1) script called *traceroute-helper*. The script simply does a *traceroute* (8) to the machine `enss.ucar.edu` and prints 1 if it was successful and 0 if there was a problem (routing loop or host unreachable). This script must be placed in a directory in the agents path. The next step is to add the new variable to the MIB.

```
satoolGatewayOk OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 11 }
```

It is now necessary to modify the SNMP agent to run *traceroute-helper* when appropriate and update *satoold* to use the new variable. There are three places to change in the agent. The first step is to assign the new variable a NUMBER in the *satool* header file. This is used to identify the variable internally. The last number used is 48 so we can do:

```
#define SATOOLGATEWAYOK 49
```

Next, we need to tell the agent about the variable's existence. This is done in *snmp\_vars.c* in the `variables[ ]` array. We just need to add a line like the following:

```
{ENTERPRISES, CU_ENTERPRISE,
 SATOOL_NUM, 11, 0}, 10, INTEGER,
 SATOOLGATEWAYOK, RONLY,
 var_satool }
```

after the last *satool* variable. Between the first set of braces is the numeric representation of the variable name, separated by commas. `ENTERPRISES` represents `iso.org.dod.internet.private.enterprises` and `CU_ENTERPRISE` is the enterprise number assigned to the University of Colorado. `SATOOL_NUM` is *satool*'s number in the *cu* hierarchy, and 11 is our place in the *satool* hierarchy (the eleventh branch). The zero is used as a terminator. The "10" is a count of the elements in the first set of braces. `INTEGER` denotes the type of the data and `SATOOLGATEWAYOK` is our variable number. `RONLY` signifies that this variable is read-only. `var_satool` is the function that is to be called to resolve the variable. Now that the agent knows about the variable and how to resolve it, we need to write a resolution function. In this case, we just put the new variable in an existing function, `var_satool()`. This function is in *satool.c* and currently resolves the load average and mail queue length. To add our new variable we just add the following code to the switch statement in `var_satool()`.

```
case SATOOLGATEWAYOK:
    fildes = popen("traceroute-helper",
        "r");
    if (!fildes) return(NULL);
    if (fscanf(fildes, "%d", &n) < 0)
    {
        pclose(fildes);
        return(NULL);
    }
    long_return = (long)n;
    pclose(fildes);
    return (u_char *) &long_return;
```

All we are really doing here is running our helper script and passing what it prints back as an int.

#### Server

We still need to tell *satoold* to monitor and store this variable. To do this we need to update three files. The first step is to add the new variable to the *satool\_variables* struct in *satool\_db.h*. We can add something like:

```
int gateway_ok;
```

Next we need to have this updated when the server polls its clients. The place to do this is in `update_data()` in *load\_values.c*. We need to add two lines at the end of the if statements.

```
else if (!strcmp("satoolGatewayOk.0",
    name))
    sat_var_ptr -> gateway_ok =
        snmp_var_to_int(buf, variable,
            subtree->enums);
```

This converts the snmp variable into an int and stores it in the correct field of the struct. The next step is to add code to print the variable and value when the display system requests a host's data. We just need to add one line in the get\_command() function in parse\_client\_request.c.

```
printf((char *)ret + strlen(ret),
      "%s %d ", "gateway_ok",
      sat_var -> gateway_ok);
```

This just adds a "variable value" pair to the return buffer. That's all it takes. If you want to add variables that are outside the satool part of the MIB you will need to add extra code to update\_data(). This is not necessary from within the satool hierarchy because the SNMP getnext operator is used to get the variables (as such the server need not know them by name when requesting them).

### GUI

The final step is to add the new variable to the GUI. It must be added to the default list in procedure BuildList, file IO.tcl:

```
gateway_ok 0 X
```

The variable name is followed by a threshold value and its default display object. A threshold value of 0 for a boolean variable triggers the alarm when the gateway is down; a text display object is specified. Note that the text display is the default and so would not actually need to be listed.

The new variable must also be added to the Data menu. This is done in the mkDataMenu procedure in Menu.tcl. The new option can be set as follows:

```
$parent.menu.data.misc add command \
-label "Gateway Ok?" \
-command "RunDisplay gateway_ok"
```

Since tcl/tk is interpreted there is no need to recompile; just restart satool and the gateway\_ok data object will be immediately available.

### Supported Architectures

Currently, the SNMP agent is only known to work under 4.3 BSD. The satool elements of the agent are known to also work under Ultrix 4.2/4.3 and should be easily portable to most versions of UNIX. The satool GUI will run on any system capable of running tcl/tk. satool should run on any version of UNIX that supports Berkeley sockets and ndbm(3).

### Future Enhancements

satool is currently deployed in the Computer Science Department's research and instructional networks. As experience is gained in this "pseudo-real-world" environment, we expect to continue improving it. Current plans include:

- MIB-II support
- Support for more architectures in the SNMP agent
- Extra modules to alert sysadmins of pending and existing problems (email and pager).
- Extra widgets for the display system
- More flexible alarms, including special handling based on the time of day and a muting option.
- Scalable alert thresholds including support for a bottom threshold as well as the upper limit.

### Author Information

Todd Miller is a recent graduate of the University of Colorado, Boulder where he received a Bachelors Degree in Computer Science and served as a systems administrator for the last two years he spent there. You can reach him electronically at millert@alumni.cs.colorado.edu.

Christopher Stirlen is a masters student in Computer Science at the University of Colorado, Boulder. He currently works for XVT Software in Boulder as a Software Test Engineer. Chris has a strong interest in User Interface design and visual programming. Reach him electronically at stirlen@cs.colorado.edu.

Evi Nemeth is an Associate Professor of Computer Science at the University of Colorado, Boulder. Reach her electronically at evi@cs.colorado.edu.

### References

Case, et al., *RFC 1067*, DDN Network Information Center, SRI International, 1988.

Hardy, Darren and Morreale, Herb, *buzzerd: Automated Systems Monitoring with Notification in a Network Environment*, LISA VI Conference Proceedings, 1992.

Nemeth, Evi, *SA-Tool, A System Administrator's Cockpit*, AUUG Conference Proceedings, 1991.

Ousterhout, John K., *An Introduction To Tcl and Tk*, 1993, available via anonymous ftp on sprite.berkeley.edu:tcl/bookps.Z.

Rose, Marshall, *The Simple Book*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1991.

Rose, Marshall and McCloghrie, Keith, *RFC 1065*, DDN Network Information Center, SRI International, 1988.

Rose, Marshall and McCloghrie, Keith, *RFC 1066*, DDN Network Information Center, SRI International, 1988.

Appendix A: satool MIB

```

-- sa tool
cu OBJECT IDENTIFIER
    ::= { enterprises 632 }
satool OBJECT IDENTIFIER ::= { cu 1 }
-- to get real load average divide
-- the int by 100
satoolLoadAve OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 1 }
-- number of entries in the mail queue
satoolMailQueueLen OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 2 }
-- number of system calls / sec.
satoolNumSysCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 3 }
-- process information
processes OBJECT IDENTIFIER
    ::= { satool 4 }
-- number of processes
satoolNumProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 1 }
-- number of processes in disk wait
satoolNumWaitingProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 2 }
-- number of zombied processes
satoolNumZombieProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 3 }
-- number of processes in the run queue
satoolRunQueueLen OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 4 }
-- number of processes blocked for
-- resources
satoolNumBlockedProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 5 }
-- number of processes runnable
-- but swapped
satoolNumRunnableButSwapped OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 6 }
-- vm information
vm OBJECT IDENTIFIER ::= { satool 5 }
-- number of context switches / sec
satoolNumContextSwitches OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 1 }
-- number of active virtual pages
satoolActivePages OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 2 }
-- number of free virtual pages
satoolFreePages OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 3 }
-- number of page reclaims / sec
satoolPageReclaims OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 4 }
-- number of pages attached / sec
satoolPagesAttached OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 5 }
-- number of pages paged in / sec
satoolPageIns OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 6 }
-- number of pages paged out / sec
satoolPageOuts OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 7 }
-- number of pages freed / sec
satoolPagesFreed OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 8 }
-- anticipated short term memory
-- shortfall
satoolMemLow OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 9 }
-- number of pages scanned clock
-- algorithm / sec
satoolPagesScanned OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 10 }
-- io stuff
io OBJECT IDENTIFIER ::= { satool 6 }

```

satool - A System Administrator's Cockpit, An Implementation

```

-- number of device interrupts / sec.
satoolNumInterrupts OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { io 1 }

-- rpc stuff
rpc OBJECT IDENTIFIER ::= { satool 7 }

-- number of rpc calls (server)
satoolServerRpcCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 1 }

-- number of bad rpc calls (server)
satoolServerRpcBadCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 2 }

-- number of empty rpc calls (server)
satoolServerRpcNullRecv OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 3 }

-- number of rpc calls with too small
-- a body (server)
satoolServerRpcBadLen OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 4 }

-- number of rpc calls that failed to
-- decode into xdr (server)
satoolServerRpcXdrCall OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 5 }

-- number of rpc calls (client)
satoolClientRpcCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 6 }

-- number of bad rpc calls (client)
satoolClientRpcBadCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 7 }

-- number of retransmitted rpc calls
-- (client)
satoolClientRpcRetrans OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 8 }

-- number of rpc calls where the reply
-- transaction ID did not match the
-- request transaction ID (client)
satoolClientRpcBadXid OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 9 }

-- number of rpc calls that timed out
-- (client)
satoolClientRpcTimeout OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 10 }

-- number of times the client had
-- to sleep
satoolClientRpcWait OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 11 }

-- nfs stuff
nfs OBJECT IDENTIFIER ::= { satool 8 }

-- number of nfs calls (server)
satoolServerNfsCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 1 }

-- number of bad nfs calls (server)
satoolServerNfsBadCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 2 }

-- number of nfs calls (client)
satoolClientNfsCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 3 }

-- number of bad nfs calls (client)
satoolClientNfsBadCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 4 }

-- number times a client structure
-- was successfully gotten
satoolClientNfsNclGet OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 5 }

-- number times all client structures
-- were busy
satoolClientNfsNclSleep OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 6 }

-- cpu stuff
cpu OBJECT IDENTIFIER ::= { satool 9 }

-- percent of cpu in user time
satoolCpuUserTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { cpu 1 }

-- percent of cpu in system time
satoolCpuSysTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional

```

```

 ::= { cpu 2 }
-- percent of cpu in idle time
satoolCpuIdleTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { cpu 3 }

-- percent of cpu spent running niced
-- processes
satoolCpuNiceTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { cpu 4 }

-- table from df
satoolDfTable OBJECT-TYPE
    SYNTAX SEQUENCE OF DfEntry
    ACCESS read-write
    STATUS optional
    ::= { satool 10 }

dfEntry OBJECT-TYPE
    SYNTAX DfEntry
    ACCESS read-write
    STATUS optional
    ::= { satoolDfTable 1 }

DfEntry ::= SEQUENCE {
    dfIndex
        INTEGER,
    dfDevice
        OCTET STRING,
    dfMountPoint
        OCTET STRING,
    dfTotalKb
        INTEGER,
    dfUsedKb
        INTEGER,
    dfAvailKb
        INTEGER,
    dfCapacity
        INTEGER
}

dfIndex OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { dfEntry 1 }

dfDevice OBJECT-TYPE
    SYNTAX OCTET STRING
    ACCESS read-only
    STATUS mandatory
    ::= { dfEntry 2 }

dfMountPoint OBJECT-TYPE
    SYNTAX OCTET STRING
    ACCESS read-only
    STATUS mandatory
    ::= { dfEntry 3 }

dfTotalKb OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { dfEntry 4 }

dfUsedKb OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { dfEntry 5 }

dfAvailKb OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { dfEntry 6 }

dfCapacity OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { dfEntry 7 }

```

# The USENIX Association

The UNIX and Advanced Computing Systems Professional and Technical Association

The USENIX Association is a not-for-profit membership organization of those individuals and institutions with an interest in UNIX and UNIX-like systems and, by extension, C++, X windows, and other programming tools. It is dedicated to:

- sharing ideas and experience relevant to UNIX or UNIX inspired and advanced computing systems,
- fostering innovation and communicating both research and technological developments,
- providing a neutral forum for the exercise of critical thought and airing of technical issues.

Founded in 1975, USENIX sponsors twice yearly general conferences accompanied by vendor displays and frequent single-topic conferences and symposia. USENIX publishes proceedings of its meetings, the bi-monthly newsletter ;login:, the refereed technical quarterly Computing Systems. (published with the University of California Press), and is expanding its publishing role in cooperation with MIT Press with a book series on advanced computing systems. The Association actively participates in various ANSI, IEEE and ISO standards efforts with a paid representative attending selected meetings. News of standards efforts and reports of many meetings are reported in ;login:.

## SAGE, the Systems Administrators' Guild

USENIX has recently launched its first Special Technical Groups (STGs), the Systems Administrators' Guild (SAGE). SAGE is devoted to the advancement of systems administration as a profession. It will recruit talented individuals to the profession, develop guidelines for the education of members of the profession, establish standards of professional excellence and provide recognition for those who attain them, and promote work that advances the state of the art and propagates knowledge of good practice in the profession.

USENIX and SAGE will work together to publish technical information and sponsor conferences, symposia, tutorials and local groups in the field of systems administration. Currently USENIX and SAGE jointly sponsor the annual Systems Administration Conference and they, together with FedUNIX, are sponsoring the 1993 World Conference on Tools and Techniques for Systems Administration, Networking and Security (SANS-II). SAGE News and other items of interest to systems administrators are found in each issue of the USENIX newsletter ;login:.

There are four classes of membership in the Association, differentiated primarily by the fees paid and services provided.

USENIX Association services include:

- Subscription to login:, a bi-monthly newsletter;
- Computing Systems, a refereed technical quarterly;
- Discounts on various UNIX and technical publications for purchase;
- Technical conference and tutorial program twice a year and single-topic symposia periodically;
- A discount on technical conference and workshop registration fees;
- The right to vote on matters affecting the Association, its bylaws, election of its directors and officers.
- Right to join Special Technical Groups such as SAGE

The supporting members of the USENIX Association are:

ASANTÉ Technologies, Inc.	OTA Limited Partnership
ANDATACO	Quality Micro Systems
Frame Technology, Inc.	Sybase, Inc.
Matsushita Electrical Industrial Co., Ltd.	UNIX System Laboratories, Inc.
Network Computing Devices, Inc.	UUNET Technologies, Inc.

For further information about membership, conferences or publications, contact:

USENIX Association  
2560 Ninth Street, Suite 215  
Berkeley, CA 94710  
Telephone: 510/528-8649  
Email: office@usenix.org  
FAX: 510/548-5738