The IRIXpro Administrator’s Guide

Document Number: 007-2446-001

CONTRIBUTORS

Written by Jeffrey B. Zurschmeide
Edited by Christina Cary
Production by Lorrie Williams
Written Contributions by Susan Ellis, Margaret-Anne Halse, John C. Stearns, Jean-Dominique Gascuel and Jan Newmarch.
Cover design and illustration by Rob Aguilar, Rikk Carey, Dean Hodgkinson, Erik Lindholm, and Kay Maitz
Engineering contributions by Kevin Conry, Victor Mitnick, Anurag Narula, Paul Robins, and John Schimmel

The IRIXpro Administrator's Guide
Document Number: 007-2446-001

Contents

List of Figures

List of Tables

About This Guide

This guide describes the *IRIXpro* software package of advanced system administration applications for the Silicon Graphics family of graphical workstations and servers. These applications have been designed and developed to provide the large-site administrator with tools to automate many tasks that have previously been performed by hand on each individual host system, and to maintain records of the tasks performed. Below is a description of each chapter and its contents.

Contents of This Guide

- Chapter 1, "The IRIXpro System Administration Tools" gives a brief overview of IRIXpro and its component applications.
- Chapter 2, "Software Distribution, User and Host Management With propel," provides information about the propel software distribution and host database management application. Using propel, you can keep all your host systems installed with the correct software in a dynamic environment.
- Chapter 3, "Dynamic Network Configuration With proclaim," describes the proclaim automatic network configuration application. With proclaim, you can automatically assign hostnames and IP addresses to new systems as they are installed on your network.
- Chapter 4, "Distributed System Monitoring With provision," discusses the provision distributed system monitoring system. The provision application allows you to monitor all your systems' running statistics from a single workstation.
- Chapter 5, "System Administration Request Management With problema," provides a mechanism to automate the system administration request process. This application provides a standard form to all your users and automatically tracks the status of each request.

- Appendix A, “The hp-ux_sgi MIB,” is a reprinting of the default MIB shipped with IRIXpro.

Audience For This Guide

This guide is written for the system administrator who will be directly using and administering the IRIXpro applications. It is assumed that you have installed InSight or have access to the *IRIX Advanced Site and Server Administration Guide* and the *Personal System Administration Guide* and are familiar with their contents.

Additional Resources

The primary resources for system administrators are the *IRIX Advanced Site and Server Administration Guide* and the *Personal System Administration Guide*. These guides explain the basic tasks and responsibilities of system administrators. Also, the IRIX Reference Pages, available online through the *man*(1) command, are an important resource for system administrators.

Typographical Conventions

As you read this guide, you will notice that special fonts are used for certain words.

`typewriter font`
: indicates system output, such as responses to commands that you enter and the text of messages that appear in Warning and other informational windows. This font is also used for examples of the contents of files, filters and filter components, examples of network addresses, Management Information Base (MIB) object names, and example workstation and network names and addresses.

`typewriter bold font`
: indicates text you must enter, such as command lines and filter expressions. Names of nonprinting keys on the keyboard, such as the **`<Enter>`** key, also appear in typewriter bold and are surrounded by angle brackets.

bold font	designates literal options to commands.
italic font	indicates filenames, command names, and reference page names. Lowercase italic words also represent variables—text strings that you must specify. References to other documents, button names, *inst*(1M) subsystem names, user IDs, and group names are also in *italics*.

Product Support

Silicon Graphics provides a comprehensive product support and maintenance program for its products.

If you are in the U.S. or Canada and would like support for your Silicon Graphics-supported products, contact the Technical Assistance Center at 1-800-800-4SGI.

If you are outside of the U.S. or Canada, contact the Silicon Graphics subsidiary or authorized distributor in your country.

Chapter 1

The IRIXpro System Administration Tools

This chapter provides a brief overview of the individual software components of IRIXpro, and other information to help you use this guide. The following sections are provided in this chapter:

- "Introduction to IRIXpro" on page 2 provides a general overview of the IRIXpro component applications.
- "The IRIXpro Databases" on page 4 describes the databases used in common by the IRIXpro applications.

Introduction to IRIXpro

IRIXpro has been designed to be used in the Indigo Magic™ desktop environment provided on all Silicon Graphics workstations and graphics servers. The standard directory view of the */usr/IRIXpro* directory, where the tools, applications, and many of the configuration files are installed, is available by issuing the command:

```
irixpro
```

The directory view looks similar to this:

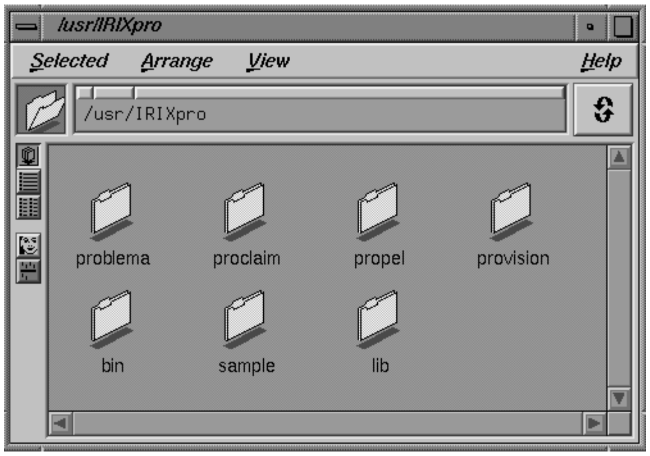


Figure 1-1 The IRIXpro Directory View

For information on using the directory view and the Indigo Magic environment, see the *IRIS Essentials* guide in your InSight™ online documentation, or “User Interface Terminology” on page 8 in this chapter.

The following sections provide a short introduction to the components of IRIXpro. Each piece of component software is described completely in a separate chapter in this book.

The IRIXpro Components

IRIXpro is made up of four related and interacting applications, and the SQL database query language. All of IRIXpro was written using the sgitcl programming language, and IRIXpro is completely configurable and extensible by the user.

Below is a brief description of the four IRIXpro applications.

propel

The *propel* distributed system management application provides a convenient graphical interface to several databases, including the Host and Collection database described in this chapter. Support is also included for using these databases to disseminate software and system configuration file changes across your network. Using *propel*, you can automatically keep host systems updated to the correct revision level of applications or system software, and manage your system configuration files with ease. The *propel* application is described in Chapter 2, “Software Distribution, User and Host Management With propel.”

proclaim

The *proclaim* network host ID management system allows you to dynamically assign and reclaim network IP addresses on an as-needed basis for your entire network and to configure a new system’s networking parameters. The *proclaim* application is described in Chapter 3, “Dynamic Network Configuration With proclaim.”

provision

The *provision* distributed system monitoring tools allow you to monitor the essential statistics and system states of each machine on your network from a central monitoring server. The *provision* application is described in Chapter 4, “Distributed System Monitoring With provision.”

problema

The *problema* system administration request management application allows your users to submit requests for system administration services, and allows you to track each request. The *problema* application is described in Chapter 5, “System Administration Request Management With problema.”

The IRIXpro Databases

The applications that compose IRIXpro each maintain various databases to hold and query the information specific to the application’s needs. The Host and Collection databases, however, are used by all the IRIXpro applications.

The Host Database

The Host database holds the following information about each host on your network:

Hostname	This field holds the name that uniquely identifies a system within an administrative domain. For example, the name of the system might be *Wilmer*. This should match the system hostname in */etc/sys_id*. The system named in this field is called the *named system* in all descriptions of other fields in the IRIXpro databases.
Domain	This field contains the NIS/BIND domain name of the named system.
Aliases	This field contains a list of alternative names for the named system. These names must each also be unique within the administrative domain. For example, an alias for the named system might be *labfileserver*.
IP Addresses	This field holds a list of software IP addresses for the named system. For example, software IP addresses have the form *123.45.67.890*. If the named system has more than one network connection (as in the case of a network gateway or hub), then this field should contain all the addresses used by the machine.
Lease	This field is not used by the editor, but exists for the use of the *proclaim* application. The field holds information about the expiration time of the system’s IP address provided by *proclaim*.
Location	This field holds the location of the named system. For example, the location of a system might be “Upstairs Lab.”

Mail Exchangers	This field holds the list of system names that can receive mail and hold it for the named system. For example, the list of mail exchanger systems might be: *stuart.eng*, *gordon.eng*
MAC Addresses	This field holds the network hardware addresses of the host. For example, this field might contain information similar to 00:00:A7:10:91:03. As with IP addresses, there is a different network hardware address for each interface on the machine.
Manufacturer	This field describes the named system by the name of the hardware manufacturer. For example: *SGI*. The purpose of this field is to allow you to query the database and locate all machines by a given manufacturer for software update or inventory.
Model	This field contains the manufacturer's model designation of the named system. For example: *Indy*. The purpose for this field is to allow you to query the database and locate all machines of a given model.
OS Release	This field specifies the operating system software revision level on the named system. For example: *IRIX 5.1*. The purpose of this field is to allow you to locate all machines (perhaps of a given manufacturer and model) at the same operating system software revision level. You do not necessarily have to enter the operating system revision level in this field; you could also use it to track the revision level of any software you choose.
Owner	This field names the person who is responsible for the named system. For workstations, this person is usually the primary user, and for servers the usual person is the administrator who acts as a point of contact for the server users (for example, *Wilmer McLean*).
Server	This field identifies the slave distribution server system that distributes new software to the named system with the *propel* software distribution facilities. (For example, the slave server might be called *distmachine*.)

System ID	This field contains the serial number of the system hardware.
Comment	This is an open field that can be used for any purpose. It can be useful in queries to further categorize systems that are administered in a similar manner, such as development or test systems.

The standard system host databases, such as */etc/hosts*, */etc/ethers*, and the *named* maps in */var/named* can all be generated using the Host database as input. Under *IRIXpro*, administrators need only maintain one list of host information for the entire network. Individual system data files can be generated from this list, thus simplifying large domain host administration. See “The propel File Generation System” on page 69 for complete instructions.

The Collection Database

The Collection database holds records defining collections of hosts from your Host database. These can be arbitrary groupings, or groups generated by querying the Host database. The following fields are used in the Collection database:

name	The name of the collection of hosts. Make this name unique from any hostname. For example, if you wish to make a collection describing all Silicon Graphics machines on your network, you might select the name *sgihosts*.
collection	The query parameters to define the collection. You can query on any field or fields in the Host database. For a complete set of instructions on forming a database query, see the section titled “Additional Database Query Techniques” in Chapter 2.
comment	Any notes you wish to make about the collection. This is an open field that can be used for any purpose.

Information on creating, modifying, and querying entries in the Host and Collection databases is found in “The Host Database” on page 29 and in “The Collection Database” on page 33.

Running snmpd on Your Network

In order to use many of the utilities and features of IRIXpro, each system on your network should be running the *snmpd* daemon. This daemon provides support for SNMP (Simple Network Management Protocol) and allows other systems to query the host for information about its configuration. This daemon is described in the section titled "The MIB Browser" in Chapter 4.

To obtain SNMP support, the IRIXpro distribution packages to install on each station are:

```
snmpd          01/04/95 SNMP Daemon with HP MIB Support
snmpd.sw       01/04/95 SNMP Daemon with HP MIB Support
snmpd.sw.agent  01/04/95 SNMP Daemon with HP MIB Support
```

To configure a workstation so that *snmpd* is started automatically when the system is rebooted, install the packages listed above, and enter this command on the system while logged in as *root*:

```
chkconfig -f snmpd on
```

To check to see if the daemon is already running, enter this command:

```
ps -e | grep snmpd
```

If there is no output from this command, *snmpd* is not running. Become *root* and enter this command to start *snmpd*:

```
snmpd &
```

User Interface Terminology

Figure 1-2 and Figure 1-3 show examples of windows, with the window terms used in this guide noted.

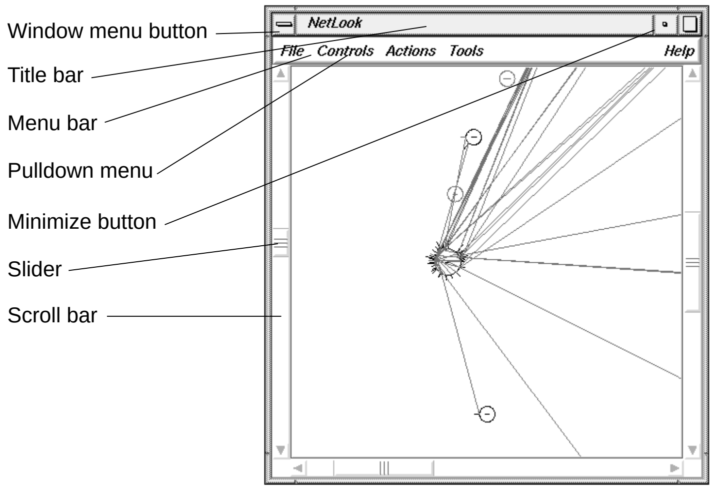
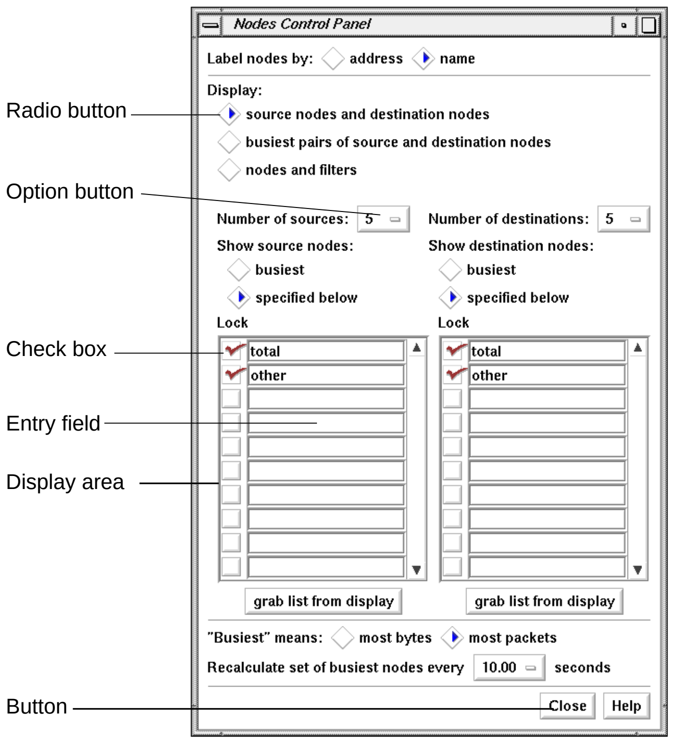


Figure 1-2 Window Terms

The mouse buttons have these functions:

left	Perform most basic tasks: click buttons, select an entry field to type into, select menu choices, select items in a display, select text to modify, and so on.
middle	Reposition windows and icons.
right	Access pop up menus. Pop up menus appear when you press the right mouse button in certain locations on the screen.



Nodes Control Panel

Label nodes by: address [✓ name]

Display:
[✓ source nodes and destination nodes]
busiest pairs of source and destination nodes
nodes and filters

Number of sources: 5 Number of destinations: 5

Show source nodes: busiest [✓ specified below]
Show destination nodes: busiest [✓ specified below]

Lock ✓ total ✓ other
Lock ✓ total ✓ other

grab list from display grab list from display

"Busiest" means: most bytes [✓ most packets]

Recalculate set of busiest nodes every 10.00 seconds

Close Help

Figure 1-3 More Window Terms

This guide uses the following terms to describe the use of the mouse:

press	Hold down a mouse button.
drag	Move the mouse while a mouse button is pressed.
click	Press a mouse button and immediately release it without moving the mouse.
double-click	Press and release a button twice in quick succession without moving the mouse.
select	The term “select” is used in the following ways: • Click the left mouse button on an item to highlight it.

- Press the left mouse button in an entry field, drag the cursor across some or all of the text, and release the mouse button. The text becomes highlighted.
- Press the left mouse button on a menu title in a menu bar, move the cursor to a menu choice, and release the mouse button while a menu choice is highlighted.
- To select a traffic line, node, or network in the NetLook main window, double-click it.

deselect	Click a highlighted item to turn off the highlighting.

Common User Interface Operations

The graphical IRIXpro tools have a common look and feel for consistent operation and easy switching between tools. This guide assumes that you are familiar with using the mouse, working with windows, and using pulldown and rollover menus. These operations are described in the *IRIS Essentials* manual.

The sections below explain how to use additional components of the user interface that are common to several of the tools.

Using Scroll Bars

You can use scroll bars (see Figure 1-4) to change the area and scale of a viewing area and to display different lines or portions of lines in a display area. The size of the slider is proportional to the amount of the total that you are viewing. You operate scroll bars by pressing the left or middle mouse button when the cursor is in the scroll bar. There are several ways to operate the scroll bar:

- Press the left mouse button on the slider, drag the cursor to a new slider position, and release the button.
- Move the slider incrementally by clicking the triangles at each end of the scroll bar.
- Move the slider up or down by positioning the cursor in the trough above or below the slider and clicking the left mouse button.

- Move the slider to a specific position by positioning the cursor at that position and clicking the middle mouse button.

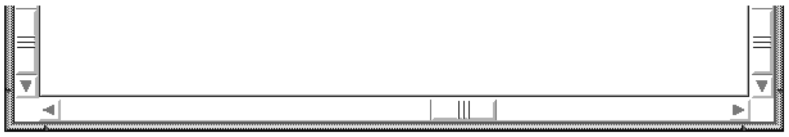


Figure 1-4 A Horizontal Scroll Bar

Entering and Removing Text in a Field

Editing text in the entry fields (see Figure 1-5) is the same as editing text in the entry fields of other applications:

- Position the text insertion point by moving the mouse to the entry field and clicking the left mouse button.
- Select (highlight) text by pressing the left mouse button at one end of the text that you want to select and dragging to the other end.
- Select a word including a space or punctuation-delimited characters by moving the cursor to the word and double-clicking the left mouse button.
- Select the entire contents of an entry field by moving the cursor over the entry field and triple-clicking the left mouse button.
- Delete selected (highlighted) text by pressing the **`<Backspace>`** key.
- Delete the character to the left of the insertion point by pressing the **`<Backspace>`** key.

Figure 1-5 An Entry Field

Using Option Buttons

Option buttons (on the left in Figure 1-6) let you select a numeric value from among a predefined set of choices. To use an option button, first press the option button with the left mouse button. A menu pops up (on the right in Figure 1-6). Move the cursor to your selection and release the mouse button.

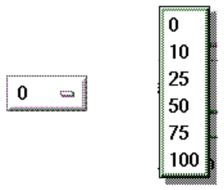


Figure 1-6 An Option Button and an Option Button Menu

Using a File Prompter

File prompter windows (like the one in Figure 1-7) are used to specify filenames. You can choose a filename by double-clicking a name in the display area. You can also type the name into the filename entry field and press **`<Enter>`** or click the *Accept* button to complete your filename selection. You can change directories to the parent of the current directory by clicking the *Up* button, or return to the directory where you started the tool by clicking the *Original* button.

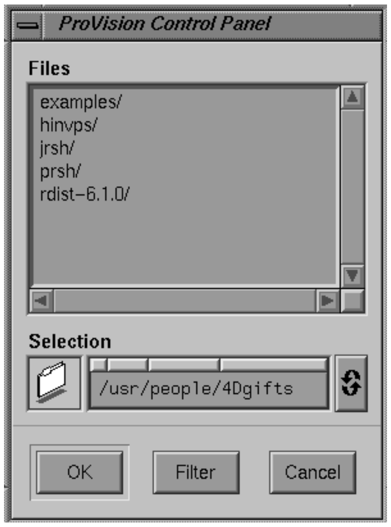


Figure 1-7 A File Prompter Window

Using Online Help

IRIXpro provides many online help files to help you as you learn to use the tools. You access these files from the Help menu in the menu bar of many IRIXpro windows (shown in Figure 1-8 on top) and from the *Help* button that appears in some IRIXpro windows (on the bottom in Figure 1-8).

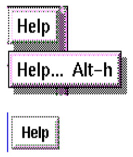


Figure 1-8 A Help Menu and a Help Button

When you choose “Help...” from a menu or click a *Help* button, a Showcase window appears and displays the first help card.

Some help files contain several cards. Page through these cards using the **`<Page Up>`** and **`<Page Down>`** keys in the cluster of six keys just to the right of the **`<Backspace>`** key or click the left mouse button on the arrows at the bottom of the pages. Make sure the cursor is in the Help window when you press these keys.

When you're finished reading a help file, you can close the Help window just as you close any other window, for instance, by double-clicking the Window menu button in the upper left corner of the window or by selecting "Quit" from the Window menu.

Chapter 2

Software Distribution, User and Host Management With propel

The *propel* distributed system management application provides a convenient graphical interface to several system information databases. Support is also included for using these databases to disseminate software and system configuration file changes across your network. Using *propel*, you can automatically keep host systems updated to the correct revision level of applications or system software, and manage your system configuration files with ease.

This chapter describes the *propel* software package. The following topics are covered:

- Installation tips for *propel*. See “propel Installation Instructions” on page 16.
- The *propel* system overview. See “The propel Management System” on page 22.
- An overview of the way *propel* operates. See “How propel Works” on page 24.
- The *propel* databases, which hold information about the systems that receive files and the files themselves, as well as the rules that govern the distribution. See “Using the propel Databases” on page 29.
- The Host database editor. See “The Host Database” on page 29.
- The Collection manager. See “The Collection Database” on page 33.
- The File database editor. See “The File Database” on page 41.
- The Distribution rule editor. See “The Distribution Rule Database” on page 45.
- The User database editor. See “The User Database” on page 49.
- The User Group database editor. See “The Group Database” on page 53.

- Using advanced techniques to query the *propel* databases. See “Additional Database Query Techniques” on page 60.
- Using the *propel* databases from a command line program. See “Using the dbredit Utility” on page 65.
- Information necessary to run the *propel* program automatically or from the command line. See “Running propel” on page 68.
- The *propel* file generation system to dynamically generate files for distribution. See “The propel File Generation System” on page 69.

The purpose of *propel* is to allow the site administrator to maintain a master set of system configuration and software distribution files on a single server (or graphics workstation), and to distribute those files to other systems throughout a heterogeneous network. The *propel* package consists of two main parts:

- A set of databases that hold information about users, network host machines, files that are to be distributed, and the set of rules that governs the distribution.
- A set of commands to maintain and modify the *propel* databases and execute the distribution.

propel Installation Instructions

Use the standard Silicon Graphics installation utility *inst*(1M) to install the IRIXpro software, including the *propel* application. Your system must be running IRIX version 5.3 or later for the IRIXpro installation to succeed.

The *propel* application includes a database and programs to edit the database. Plus, a number of sample scripts are included to load the databases by parsing system files. Individual sites will want to change these scripts to fit their configuration.

Initializing the Host Database

When you install *propel*, the host database is initialized automatically. If you wish to reinitialize the database at any time, enter the following command while logged in as **root**:

```
dbredit host init
```

Remember that this command will eliminate all current entries in your Host database.

Automatically Loading the Host Database

All systems using the *propel* software must have a record in the *propel* Host database. The easiest way to place all your systems in the database is to parse the standard system files */etc/hosts* and */etc/ethers*, and load the system names directly into the database. There are a pair of scripts provided with the *propel* software that parse these files and load the database. The first is a *perl* script, */usr/IRIXpro/sample/propel/host_parser* that reads through these system files, and outputs one line for each system name that it finds. The output format is that of a TCL keyed list, which can be inserted into the host database using the *dbredit* command. Running the script */usr/IRIXpro/sample/propel/add_hosts* works for many sites. You may edit the *host_parser* script to make changes to fit your local needs.

The *propel* database has one entry for each "host," but each host can have multiple network interfaces, and therefore multiple entries in the *hosts* file. The standard *host_parser* script recognizes the extra interfaces of hosts if they have a comment of the format *GATE(hostname)* or if the primary name for these interfaces is of the form *gate3-hostname*. The test for such names is clearly marked within the script and can be changed to fit local conventions.

The most useful script available for populating your *propel* Host database is */usr/IRIXpro/sample/propel/find_hosts*. Invoked with an IP subnet number (in the form *123.45.67*) as an argument, this script sends an ICMP echo (*ping*) request to each possible node (0-255) on the specified subnet. When a remote host responds, it sends an SNMP request to the remote host and creates a record in the Host database with any data that is returned.

Once a record exists in the Host database for each host, it is a good idea to fill in the remaining information for each host. This can be done by completing each entry in an editor, or as a batch update. There is a very simple script that attempts to attach to each machine using the *rsh*(1) command, and reading information from the *uname*(1) command to glean the standard information. This script is */usr/IRIXpro/sample/propel/host_info*, and it outputs a line for each machine that successfully completes the process. There is a simple *perl* wrapper program, */usr/IRIXpro/sample/propel/batch_update*, that takes a file of *host_info* output lines and performs the updating in the Host database. A more powerful script, */usr/IRIXpro/sample/propel/update_hosts*, updates the records of each host in the database by parsing the result of an SNMP request for the *sysDescr* variable. If you are running SNMP on the client hosts then this script should provide the best automatic updating.

If you are administering a network of Silicon Graphics systems, the following steps will initialize and populate your Host database. If you are administering a heterogeneous network, not all hosts may be located and entered in the database through this process. If this happens, you may need to edit the databases with the *hostedit* editor, as described in “Editing the Host Database.” Follow these steps to populate your database:

1. Certify that the entire IRIXpro distribution has been installed on your master server system and log in as **root** or as a member of the user group **irixpro**.
2. Obtain a basic listing of the hosts on your network from the */etc/hosts* and */etc/ethers* files with the command:

   ```
   /usr/IRIXpro/sample/propel/add_hosts
   ```
3. Next, use the command

   ```
   hostedit
   ```

   to view and edit the Host database entries for completeness and correctness. You may wish to create a collection (see “The Collection Database” on page 33 for more information about creating collections) of all your Silicon Graphics systems at this time if you need to distribute the SNMP agent software to them.
4. Certify that an SNMP agent (*snmpd*) is running on each host on the network. On each host, enter the command:

   ```
   ps -ef | grep snmpd
   ```
5. If Silicon Graphics systems need to have *snmpd* installed, use the */usr/IRIXpro/sample/propel/snmp_push* script. This script adds File database records for all files needed to run *snmpd* and *hpsnmp* on a host.

   After running the *snmp_push* script, you can enter the command

   ```
   runpropel snmp-5.3 ‘os=”IRIX 5.3”’
   ```

to push SNMP to all Silicon Graphics systems running IRIX 5.3 or later. You can use any host query to select the correct hosts in your domain.

For non-Silicon Graphics systems, you must make sure that a compatible SNMP daemon is running. Consult your documentation for each system to determine how to obtain an SNMP daemon.

6. Finally, use the command

   ```
   update_hosts
   ```

   to use SNMP to obtain any remaining fields from your SNMP hosts.

You can also use the */usr/IRIXpro/sample/propel/find_hosts* script, which performs many of the same functions. This script searches your network for other hosts and populates the database using SNMP responses. Obviously this command is only useful when the SNMP daemon is running on all your systems. The command has the syntax:

```
find_hosts subnet
```

The script pings each possible address on the given subnet. For example, if given the subnet argument 127.0.0, *find_hosts* returns the following:

```
Trying: 127.0.0.1
   Address 127.0.0.1 pings, but is not in database.
   Does not answer SNMP request.
Trying: 127.0.0.2
Trying: 127.0.0.3
Trying: 127.0.0.4
```

And so on up to address 127.0.0.255. IP address 127.0.0.1 is the standard loopback interface. The *ping* response received was from the local host. When this command is used on a valid subnetwork, the database is populated using the SNMP responses from the hosts. You must be logged in as **root** to use *find_hosts*.

Creating the Collection Database

Once you have populated your Host database, you should create your first set of collections of hosts. Instructions for creating and modifying collections are found in the section titled “The Collection Database” on page 33.

Populating the User and Group Databases

The script used to automatically populate both the User and Group databases at once is */usr/IRIXpro/sample/propel/add_users*. This script parses the */etc/passwd* and */etc/group* files to build these databases. The script is sophisticated enough to bypass NIS entries in these files, but does generate entries for accounts such as *uucp* and demonstration users.

As **root**, enter the command:

```
/usr/IRIXpro/sample/propel/add_users
```

Loading the File Database

Initially, you must create a set of files that can be distributed to client hosts. The creation process includes putting the files into the */var/IRIXpro/propel/data* directory, and creating a File database entry for each file. Files can also be automatically generated before they are distributed by setting a generator command in the File database record. Scripts are included in your propel application for generating the files */etc/hosts*, */etc/ethers*, and the DNS maps, using information from the Host database. Each of these sample scripts must be edited to deal with local naming conventions at your site.

Files can be distributed from locations other than the */var/IRIXpro/propel/data* directory if slave servers are not used, that is, only if files are distributed from the original server system directly to the final client systems. By default, only the *data* directory can be distributed to slave servers, which subsequently distribute the files to client systems. This behavior can be changed, however, by editing the */usr/IRIXpro/bin/runpropel* script.

The propel License Manager (lm_tcp)

The *lm_tcp* daemon is the *propel* license manager, and is controlled by *chkconfig*(1M) and the */etc/init.d/propel* script. When *propel* is installed, the value of the *chkconfig* variable *propel* is set to *on* and the license manager daemon (*lm_tcp*) is invoked.

Note: If you have trouble starting the database editors once you have installed *propel*, you may need to explicitly start the license manager daemon */usr/IRIXpro/bin/lm_tcp* by issuing the commands:

```
/etc/init.d/propel stop
/etc/init.d/propel start
```

You must be a member of the user group **irixpro**. For complete information on user groups, see the *IRIX Advanced Site and Server Administration Guide.*

propel and rdist

The *runpropel* program uses the industry standard utility *rdist* to distribute files from master servers to slave servers, and from slave servers to client hosts. The *rdist* utility originates from the BSD releases and is a standard utility under most UNIX® operating systems. It is also freely available on the Internet in source form so that it can be ported to most other systems. The purpose in using a standard file distribution program is to make *propel* useful in a heterogeneous environment. The *rdist* utility is distributed with IRIXpro in the *IRIXpro.sw.gifts* installation package. When installed, the source is placed in the */usr/people/4Dgifts/rdist-6.1.0* directory

The *rdist* program pushes files using a generalized configuration file to control the distribution. The *runpropel* program generates the necessary *rdist* configuration files (known as *Distfiles*), and then runs *rdist* on the Distfiles.

All *propel* servers and clients must have the 6.1 version of the *rdistd* daemon installed, at a minimum. The *propel* software works with any version of *rdist*, but the version shipped with *propel* is *rdist* 6.1. The use of *rdist* 6.1 is strongly recommended for its extended logging, improved security, and improved performance. There are, however, some sites that cannot build the new *rdist* software but have the BSD software running. These sites must use the old distribution software on the master, and a modified master *runpropel* script. If you would like to use *propel* with an older version of *rdist*, you must first unset the LOGERRS variable in the */usr/IRIXpro/bin/runpropel* command script. To set the LOGERRS variable, edit the *runpropel* command script and change the line that reads:

```
LOGERRS=”-l syslog=warning,nerror,ferror:file=/usr/propel/
rdist/log.$FREQUENCY=all:stdout=warning,nerror,ferror -L
syslog=all”
```

to read as follows:

```
LOGERRS=""
```

Server/Client Security

The *propel* application currently uses *rdist* for its distribution mechanism, which means that every client system must trust at least one other system on the network as its server. If files are to be pushed directly from the master server to client systems, then each client system must place the master server's hostname in its */.rhosts* file. If a hierarchical push strategy is used then each client must have the hostname of the slave server from which the client will receive files in the client's */.rhosts* file, and the slave server must have the hostname of the master server in its */.rhosts* file. Files may be pushed with the UID of a user other than **root**, in which case the *.rhosts* file in that user's home directory needs to include the server's system name.

How to Use propel

The following sections explain how to get the most out of *propel* for various tasks. This is not an exhaustive list. The configurability and extensibility of IRIXpro allow you to modify and extend the software to meet the individual needs of your installation.

The propel Management System

There are six databases that hold the information needed by *propel*. The databases hold information about:

- Collections of related hosts
- Information about the files and file packages being sent and received
- Groups of users
- Hosts that send and receive files
- Information about the rules under which files and packages of files are distributed to hosts and collections of hosts.
- User accounts

A graphical tool and command-line interface is provided for editing each database, as well as an example script that shows how to build the initial Host database from the standard system files. The command

```
propel
```

brings up a standard desktop view of the *propel* tools. The desktop view appears similar to this, shown in Figure 2-1:

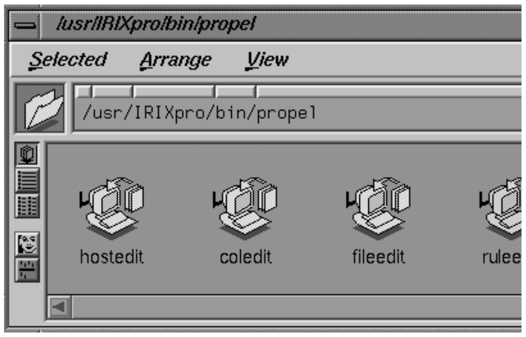


Figure 2-1 The propel Directory View

Alternately, you can invoke each tool on its own with one of the following commands:

```
/usr/IRIXpro/bin/coledit
/usr/IRIXpro/bin/fileedit
/usr/IRIXpro/bin/groupedit
/usr/IRIXpro/bin/hostedit
/usr/IRIXpro/bin/ruleedit
/usr/IRIXpro/bin/useredit
```

These commands bring up the graphical tools for the Collection, File, Group, Host, Rule, and User databases, respectively.

Icons in any *propel* tool window can be dragged and dropped on other *propel* tool windows to fill needed fields, or to other IRIXpro windows as appropriate. For example, you may populate a collection by dragging host icons from the Host database editor and dropping them in the appropriate window of the Collection manager.

The command line interface for editing databases is *dbredit*(1M). This command allows you to edit the database of your choice. The *dbredit* utility is completely described in the *dbredit*(1M) reference page, but an overview is also provided in this Guide in the section titled “Using the dbredit Utility” on page 65.

How propel Works

The *propel* software package is highly flexible. In fact, the individual administrator who uses *propel* defines virtually every parameter of the operation. As an example, *propel* does not come with a preconfigured set of rules concerning the frequency with which files are propagated. The terms “Daily” and “Weekly” mean nothing to *propel* until you define them. You enter terms as parameters in the *propel* databases, and then use different commands to query the databases and execute the distributions based on those terms. It is therefore vitally important that you plan your *propel* strategy in advance to avoid confusion.

It is this flexibility that makes *propel* a powerful tool. Because there is not a predefined set of rules about concerns such as the frequency of the distribution and the naming conventions of files and packages of files, you are not restricted in your use of *propel* to simple hourly, daily, or weekly updates, though these might form the bulk of your usage.

You will probably start your *propel* databases by listing the host machines to which you will distribute files. This is a straightforward process of listing the names of the machines and their network addresses and other pertinent information about them. This is where you start defining your *propel* environment. The fields in the Host database that describe the manufacturer, OS revision level, and other such information can be vitally important when you wish to target a very small subset of a great number of hosts for distribution. There are no preset entries for these fields, so you can use them (especially the comment field) as you wish. The *Fill* button on the *hostedit* window is provided to assist you in populating your database. If you list the hostname or IP address of a workstation on the network and use the mouse to click the *Fill* button, *propel* uses SNMP to query the host itself for answers to as many of the other fields as possible.

The Collection database is used to make collections of hosts from the Host database. This is done through making queries to the Host database. All hosts that match the query parameters you select will compose the collection. To prepare for these queries, it is recommended that you fill in all fields in each Host database record.

The User database allows you to list all the users at your site and manage that information from a central server. Since all user information is managed at one place, you can be sure that user names and UID numbers are not duplicated, and that departing users will be removed from all machines at the site. You can also use this database to provide access to new systems for existing users. The */etc/passwd* and */etc/group* files can be generated for and distributed to any system on your network by this database.

The Group database performs the same function for users that the Collection database provides for hosts. This database allows you to make groups of users with related needs or other common factors. The *groupedit* tool allows you to edit and visualize your user groups.

The File database allows you to list the files that are to be distributed in much the same manner that you listed your host systems and users. There is a field in each file entry called “Package” that lists the name of a file package (a group of related files.) There can be more than one entry in the File database for each file you wish to distribute. You may wish to place a file in more than one package, or you may wish to have a file generated by one program for some hosts and another program for other hosts. In these cases, you make two entries in the File database that both point to the same file, but with different file-generating programs or in different packages.

The most flexible of all the tools is the Rule database. By itself, the Rule database takes no action. The *runpropel* command searches the Rule database for the parameters given on the *runpropel* command line and executes any rules that match those command line parameters. Thus, you may create a rule that says that files A and B and file package C are to be distributed to host collection “engr” and to any client host with a hostname beginning with the letter “Q,” and the frequency of the distribution is to be “walnuts.” That frequency has no meaning until you assign it a meaning by the way you use *runpropel*.

Typically, distributions of a regular frequency are managed through the *cron*(1) facility of your workstation or server. The *cron* utility does define regular time frequencies, so it is possible to simply tell *cron* to execute the command

```
runpropel -f walnuts
```

every day at exactly 2 PM. Thus, “walnuts” means “daily at 2 PM” because you set it up to have that meaning. This may seem like a lot of work to perform a simple task, but the advantage comes when you want to change the meaning of “walnuts.” You then have only to change your *cron* command, and not the database entries of every rule that runs with a frequency of “walnuts.”

Once you become comfortable with the concepts behind *propel*, you can configure the software to suit your needs exactly. The graphical tools make this configuration easy and intuitive. There is a command line interface available through the *dbredit*(1M) command for server users with no graphics capability.

Using propel to Manage a Domain

This section contains some information on using *propel* to simplify the management of a large network domain.

Distributing Silicon Graphics Software With propel

Silicon Graphics currently releases all software in a proprietary format. This format can only be installed using the *inst*(1) program. In the past it has been very difficult to upgrade a large number of hosts with *inst* because it required that the host be running from the *miniroot*, and required a great deal of interaction with the administrator. New versions of *inst* support live installations, and installations from a configuration file of preselected options. This allows the administrator to ‘‘walk through’’ an installation on one host, saving his or her choices into a configuration file, and then performing identical installations on other hosts while the hosts are running, with no interaction. If the configuration file is checked into the *propel* File database and distributed to a group of machines with the *inst* command specified as the exit operation, an entire network can easily be upgraded from a central host with *propel* invoking *inst* on several hosts at the same time, and logging errors for any installation that fails. To perform a mass installation, distribute the *inst* configuration file, and specify a command such as:

```
inst -a -f machine:/location_of_package -F config-file
```

For information on creating an *inst* configuration file, see the *Software Installation Administrator's Guide*.

Internal Data Distribution With propel

For sites that develop and use in-house programs or data, *propel* is an obvious choice for keeping all the hosts on a network at the same software revision level. Since *rdist* is supported on nearly every version of UNIX in existence, *propel* can instantly be installed and used to manage large networks of heterogeneous systems.

Installing Free Software Packages With propel

Typically, packages obtained from public networks have the ability to install themselves into the non-standard directory of your choice. For instance, all of the tools from the Free Software Foundation have a DISTDIR variable in the *Makefile*, which specifies the top level directory in which the software is to be installed. For example, if you install the GNU C compiler into the directory */var/IRIXpro/propel/data/gcc*, you can add a file database record for the directory, then distribute the directory to client systems. For example, you could install the *gcc* directory onto all of your Indigo and Indy systems by giving the command:

```
runpropel gcc ‘model~Ind*’
```

You can also keep your client systems up to date with the *gcc* software by adding a distribution rule to update them weekly, and then by occasionally deleting the old *gcc* package, and by installing a new version of *gcc*.

Host Information Management With propel

The *propel* application uses a full-featured database to maintain information about hosts in your domain. The Host database is automatically locked and unlocked during operations so that many users can update the information concurrently, and the database deals with complicated queries quickly. Enough information can be kept in the *propel* Host database that all of the standard host files can be generated using the database, centralizing all host management into a single location. Scripts are included to create the system files */etc/hosts*, */etc/ethers*, and all of the DNS databases. The Host database can also be used to generate and distribute the DBM databases for the optional NIS software.

User Information Management With propel

The User database performs the same function for your network's users that the Host database performs for your hosts. Each user has an entry that can be copied to any new system to which that user needs access. Furthermore, you can load the User and Group databases, you can generate the */etc/passwd* and */etc/group* files, and you can create user accounts with the following scripts found in */usr/IRIXpro/sample/propel*.

account	Generates accounts for a list of users by using *propel* to push a copy of the local *guest* account to the remote host, and changing the permissions after the transfer is complete.
add_users	Loads the User and Group databases by parsing the local host's */etc/passwd* and */etc/group* files.
make_group	Generates a file in standard */etc/group* format from the *propel* Group database.
make_passwd	Generates a a file in standard */etc/passwd* format from the *propel* User database. It takes a user query as an argument so you can include part or all of the User database in the generated file.

Localizing propel

The *propel* application is written primarily using human-readable scripts. The editors are all created with the sgitcl programming language and an SQL database. The user interface routines are all created using the Bourne Shell (*/bin/sh*), and the sample scripts are a mixture of sgitcl, *sh*, and *perl*. The only portions of the software that cannot be changed easily are the query parsers, which were built with *lex* and *yacc*, and the sgitcl interpreter, which requires a database development license.

Fields can be added easily to any of the databases, and to the editing tools by adding a small block of code. Tools can be created to maintain system files other than those already supported by existing *propel* scripts.

Using the propel Databases

The following sections explain the fields in each of the *propel* databases and offer specific instructions in the use of the various database editing utilities. A section on formulating queries is also provided. A final section offers instruction in the use of the *dbredit*(1M) command-line database editor.

The Host Database

The *Host* database contains information about all of the machines in an administrative domain. Each record in this database contains the following fields:

Hostname	This field holds the name that uniquely identifies a system within an administrative domain. This should match the system hostname in */etc/sys_id*. The system named in this field is the named system in all subsequent fields. (For example, the name of the system might be *Wilmer*.)
Domain	This field holds the network domain name of the system, for example, *lab.eng.com*.
Aliases	This field contains a list of alternative alias names for the named system. These names must each also be unique within the administrative domain. (For example, an alias for the named system might be *labmachine*.)
IP Addresses	This field holds a list of software IP addresses for the named system. (For example, software IP addresses have the form *123.45.67.890*)
Lease	This field is not used by the editor, but exists for the use of the *proclaim* application. The field holds information about the expiration time of the system's IP address provided by *proclaim*.
Location	This field holds the location of the named system. For example, the location of a system might be "Upstairs Lab."
Mail Exchangers	This field holds the list of system names that can receive mail and hold it for this host. For example, the list of systems might be *stuart.eng*, *gordon.eng*.)

MAC Addresses	This field holds the network hardware address of the host. For example, this field might contain information similar to: `00:00:A7:10:91:03`
Manufacturer	This field describes the named system by the name of the hardware manufacturer. (For example: *SGI*.)
Model	This field contains the manufacturer's model designation of the named system. (For example: *Indigo*.)
OS Release	This field specifies the operating system software revision level on the named system. (For example: *IRIX 5.1*.)
Owner	This field names the person who should be held responsible for the named system. For workstations, this person is usually the primary user, and for servers the usual person is the administrator who acts as a point of contact for the server users. (For example, *Wilmer McLean*.)
Server	This field identifies the slave distribution server system that should update the named system. (For example, the slave server might be called *distmachine.eng*.)
System ID	This field contains the serial number of the system hardware.
Comment	This is an open field that can be used for any purpose.

The standard system host databases, such as */etc/hosts*, */etc/ethers*, and the *named* maps in */var/named* can all be generated using this database as input. Under the *propel* distribution system, administrators need maintain only one list of host information. System data files can be generated from this list, thus simplifying large domain host administration. For more information on file generation using *propel*, see "The propel File Generation System" on page 69.

Editing the Host Database

To edit the Host database, enter the command:

```
/usr/IRIXpro/bin/hostedit
```

You see the following initial screen, shown in Figure 2-2:

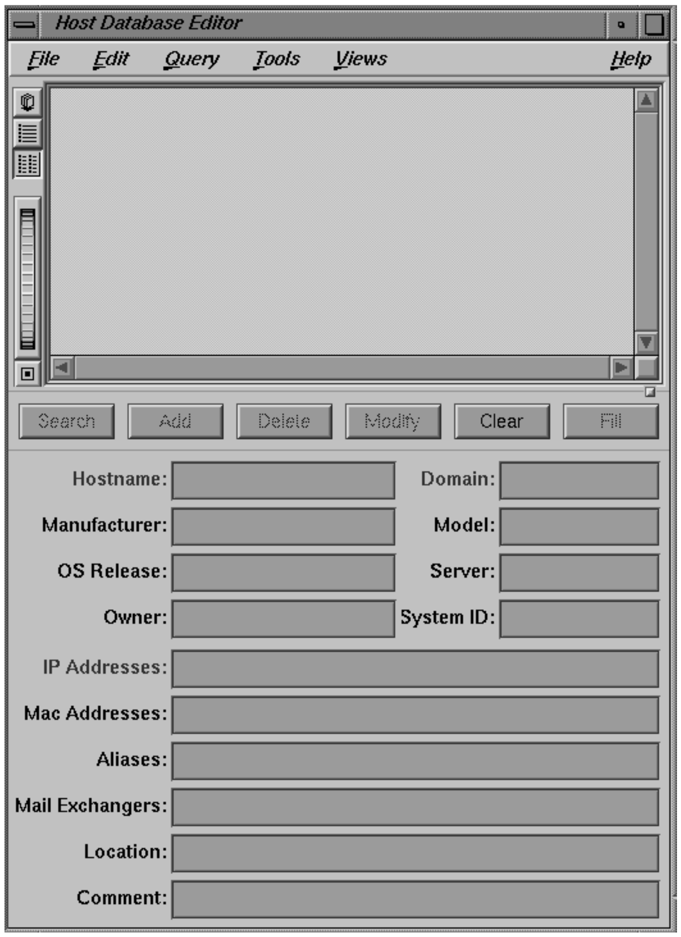


Figure 2-2 The Host Database Editor

When you first see this screen, there are no hosts listed and only the *clear* button is functional.

To add a host to your database, you must fill in the following fields (which appear in red in the hostedit window):

- Hostname
- IP address (at least one)
- Domain

When there are entries in these three fields, the *add*, *search*, and *fill* buttons become functional. You can see a change in their visual presentation. If possible, you should enter all the fields. Once entered, you can use each field as a query parameter for that record in the Host database.

Inserting a Host Entry

Once you have entered the host's information, press the *add* button by placing the cursor over the button and pressing the left mouse button. You see the information appear in the window in a slightly different format. This tells you your entry has been accepted into the database. If you do not know all the information, you can also use the *fill* button to automatically query the system itself for various fields, such as the MAC address and OS version fields, before you add the entry. The *fill* button works only if the remote system is running the *snmpd* daemon. See "Running snmpd on Your Network" in Chapter 1 for information on starting *snmpd* on a remote system.

Updating a Host Entry

To change this entry, click the entry in the window with the left mouse button to highlight it. The information you entered appears in the fields in the lower part of the screen. Change these entries as you wish and then press the *modify* button.

Querying the Host Database

For an example of using the Host database fields as query points, you can make a query to the database to find all machines that are owned by user *Kate* and are at OS Release level 5.3. Begin by using the mouse to press the *Clear* button. This clears the fields of all information. Then enter your query parameters in the appropriate fields. In this case, enter "*Kate*" in the *Owner* field and *5.3* in the *OS Release* field. You can use glob expressions as wild cards to search in any field. For more information on using glob expressions in queries, see the section titled "Additional Database Query Techniques."

When you have made your query, use the mouse to press the *search* button. All hosts in your database that match the specified parameters will be displayed in the window. If you know that there is only one system owned by any individual, you can use the *fill* button to display the remaining fields.

Parsing System Files Into the Host Database

A tool is provided to allow you to quickly parse your standard system files into the Host database. This tool is *add_hosts*(1) and is found in the directory */usr/IRIXpro/propel/sample*. The *add_hosts* command parses your system's existing */etc/hosts* file and inserts a basic record for each host found into your *propel* Host database. The *update_hosts* utility uses *snmpd* to fill in the remaining fields for each host running *snmpd*. For step-by-step instructions on filling your Host database using these programs, see "Automatically Loading the Host Database" on page 17.

For more information on running *snmpd* on all your hosts, see the section titled "Running snmpd on Your Network" in Chapter 1.

The Collection Database

Records in the Collection database have the following fields:

name	The name of the collection of hosts. Make this name unique from any hostname.
collection	The query parameters to define the collection. You can query on any field in the Host database. The field is made up of a list of typed fields separated by newlines (**`<Enter>`** keystrokes). The first word on each line is the type of query that the line represents. The available types are *hostname*, *hostquery*, and *collection*.
comment	Any notes you wish to make about the collection.

Editing the Collection Database

The *coledit* utility allows you to define collections of hosts for *propel*. A collection of hosts is simply a group of hosts that you have defined. Usually, collections are made of hosts with some common factor, such as manufacturer or usage. Since you specify the attributes of a host by using the *hostedit* utility, you can prepare hosts to be part of a collection when you enter them in your Host database. For example, you can enter several hosts in your Host database with the comment “engineering lab” in the comment field. Then, when you use *coledit* to create a collection, you specify the “engineering lab” comment as the query field when creating the collection. The syntax for these queries is described in “Additional Database Query Techniques” on page 60. All hosts that you entered with that comment becomes part of the collection. Then you name the collection, and use the Rule database to specify how and when the collection should receive files. The names of collections and the Host database queries that define the collection are stored in the Collection database.

Once defined, the *coledit* utility stores the parameters of the collection in a database. Enter the command:

```
/usr/IRIXpro/bin/coledit
```

and you see the following window, shown in Figure 2-15:

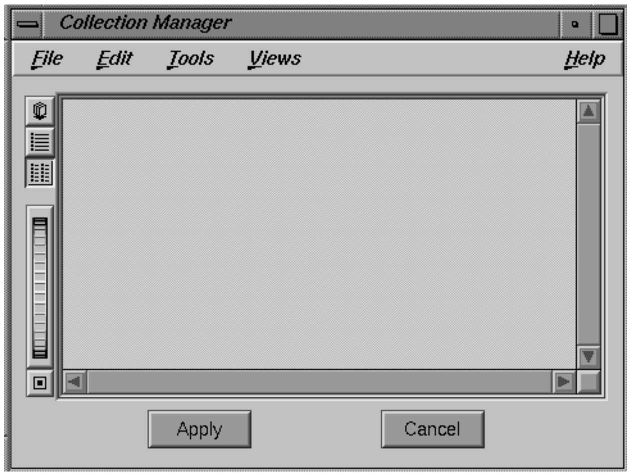


Figure 2-3 The Collection Manager

When you bring up the Collection Manager window after your database is populated, you see an icon and a name for each collection in the database.

Adding a Collection of Hosts to the Collection Database

To use the *coledit* tool to add a host collection to your Collection database, place the mouse cursor inside the *coledit* window and use the right button of the mouse to bring up the popup menu. There are several options:

Help	Brings up a help card.
Add Collection	Brings up the Add Collection window.
Add Host	Adds a host to an existing collection.
Add Query	Brings up the Add Host Query window, and allows you to add an additional query parameter to select hosts for an existing collection.
Remove Selected	Removes the selected collection.
Edit Selected Hosts	Brings up the *hostedit* tool with the selected host's information displayed. To select a host, you must have a collection displayed in a collection display window.
Preview	Shows you the list of hosts currently selected in a collection.

Select the Add Collection option to create a new collection of hosts. When you select this option, you see the new window, shown in Figure 2-16.

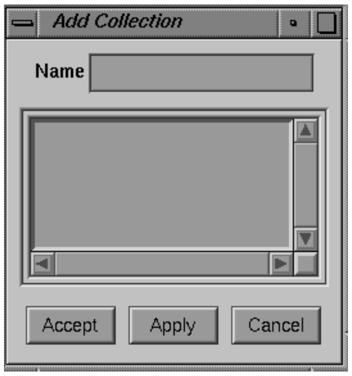


Figure 2-4 The Add Collection Window

You must enter a name for the collection, then use the left mouse button to press the Accept or Apply buttons to create your new collection. The Apply button creates your new collection, and retains the Add Collection window, so you can continue to create more collections conveniently. The Accept button creates your collection and removes the Add Collection window. The Cancel button cancels the operation without creating the new collection.

Once you have created the collection, you must specify hosts to be part of the collection. You should notice that an icon has appeared in the *coledit* window. If you made two collections and called one “first collection,” and the other “second collection,” the window now looks like Figure 2-17.

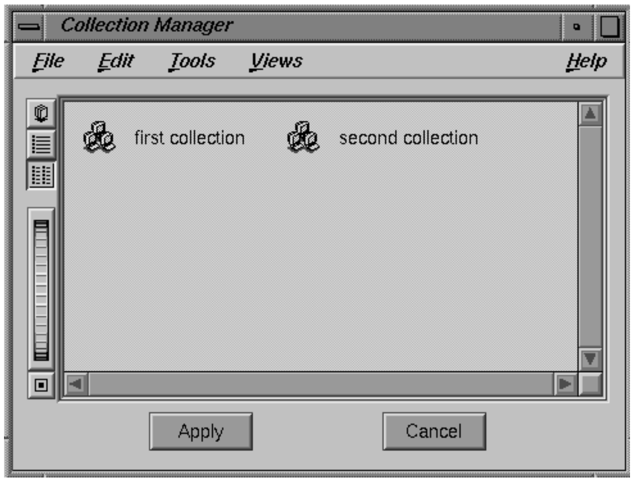


Figure 2-5 A Populated Collection Manager Window

To add a list of hosts, a list of collections, a host query, or any combination of these things to your new collection, double-click the left mouse button with the mouse cursor over the icon for the new collection. You see a new window that looks something like Figure 2-18.

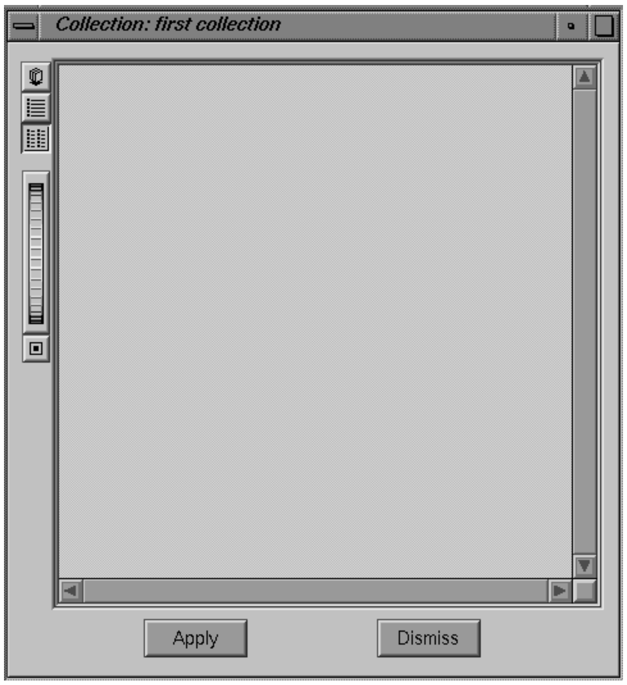


Figure 2-6 A Collection Display Window

Now use the right mouse button to bring up the popup menu, and select either the Add Host, Add Collection, or Add Query options to populate your collection.

The Add Host option brings up the Add Host window, shown in Figure 2-19, with the entire list of hosts from your Host database.

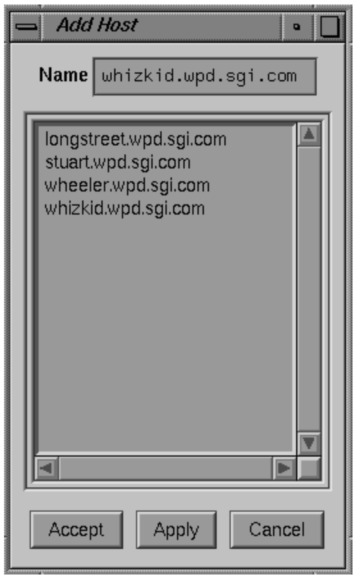


Figure 2-7 The Add Host Window

To place a host in your new collection, place the mouse cursor over the name of the host you wish to select and double-click the left mouse button. When you are finished selecting individual hosts, click the Accept button to secure your changes and remove the Add Hosts window.

The Add Collection option brings up the Add Collection window, previously shown in Figure 2-16. All your currently configured collections are shown in this window (although in this example, you are creating your first collection, so no collections may appear at this time). As with the Add Host window, double-click the name of a collection you wish to add to your new collection and complete your changes by pressing the Accept button.

The Add Query option brings up the Add Host Query window, shown in Figure 2-8.

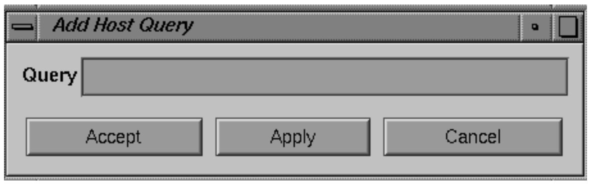


Figure 2-8 The Add Host Query Window

This window allows you to enter a query to the Host database. You must formulate your query according to the rules outlined in “Additional Database Query Techniques” on page 60.

When you have used these various windows to populate your collection, the collection display window looks something like the window shown in Figure 2-9.

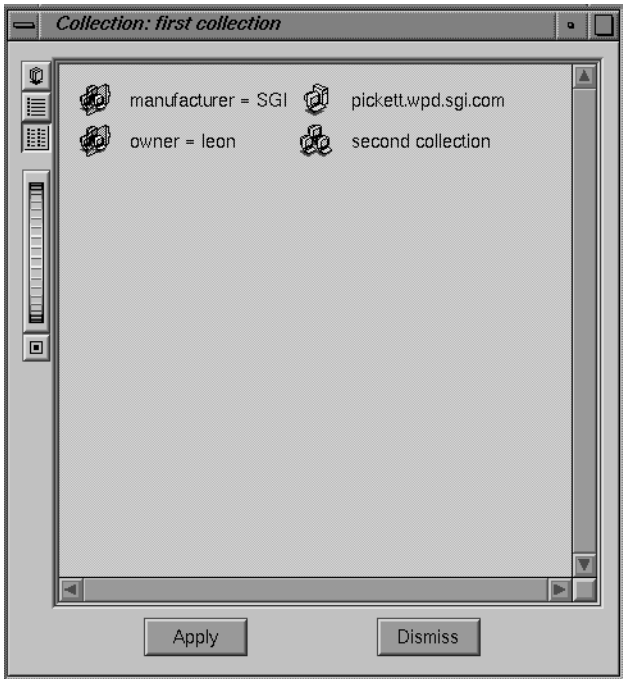


Figure 2-9 A Populated Collection

The collection shown in Figure 2-9 has a single host, a pair of host queries, and another collection as its elements. This is to demonstrate the various items that may go into a collection.

Once you have entered all the information and created your collections, press the *apply* button in the Collection Manager window to place your collections in the Collection database.

Updating a Collection Database Entry

To change an entry, click the icon with the left mouse button to highlight it, and the Collection Display window for that collection appears. Change these entries as you would if you were making a new collection, then click the *Apply* button to finalize your changes.

The File Database

The File database contains all the necessary information about the files that are to be distributed to systems on your network.

Records in the File database have the following fields:

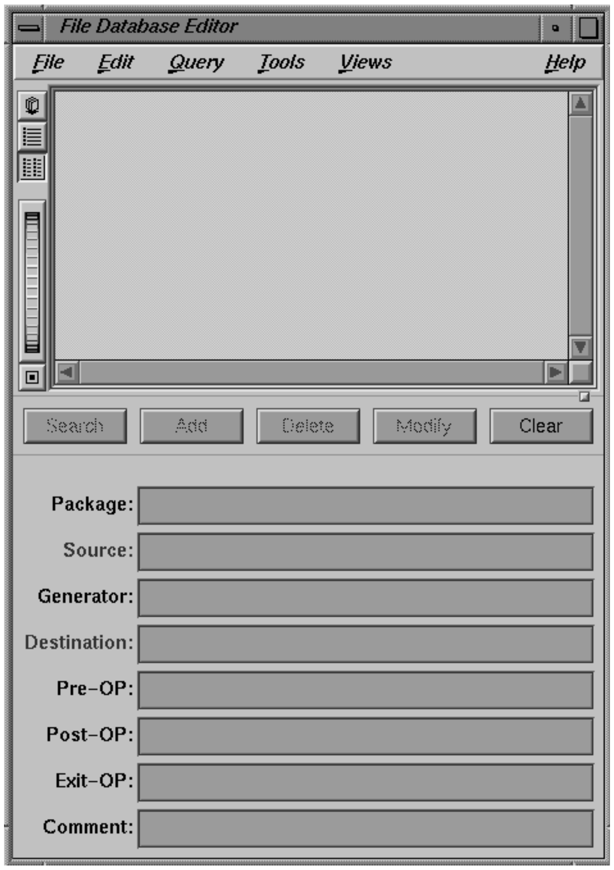
Source	The field usually contains the name of a file in the */var/IRIXpro/propel/data* directory. The file specified in this field becomes the named file in all subsequent fields in the database entry. (For example, the file named might be: */var/IRIXpro/propel/data/usr/local/bin/progname*). Any file on your system can be specified as a source file to be distributed directly to clients, but only files from the */var/IRIXpro/propel/data* directory structure can be distributed to slave distribution servers for further sub-distribution to clients. Always specify the full (absolute) pathname of each source file in this database.
Generator	If this file is generated, the name of the program that generates the file goes here.
Destination	This field specifies the pathname on the remote system where the named file will be placed (for example, this field might contain the pathname: */usr/local/bin/progname*).
Pre-OP	This field specifies any commands to be run on the remote system before the named file is installed (for example, this field might contain the command: */etc/killall progname*).
Post-OP	This field specifies any commands to be run on the remote system after the named file is installed (for example, this field might contain the command: */usr/local/bin/progname*).

Exit-OP	This field contains any commands to be run on the remote system after the last file in the distribution rule is distributed. For example, if certain daemons need to be restarted to read distributed configuration files, this field should contain the command to restart them.
Package	This field holds the name of the software package that the named file belongs to, if such a package exists (for example, the named file could be part of a package called: *pkg.1*).
Comment	This field is available for any purpose.

Editing the File Database

This database contains the names of the files that are propagated to systems in your *propel* network. The File database also contains instructions for commands to be run on the client system before and after the file is transmitted, and information about file packages of which this file is a member.

The command */usr/IRIXpro/bin/fileedit* invokes this tool, and you see the following screen, shown in Figure 2-10.



Package:	
Source:	
Generator:	
Destination:	
Pre–OP:	
Post–OP:	
Exit–OP:	
Comment:	

Figure 2-10 The File Database Editor

Adding a File to the File Database

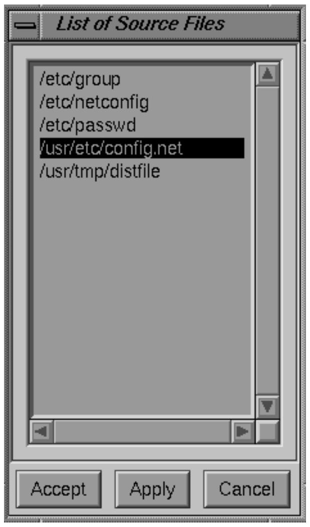
To use the *fileedit* tool to add a file to your File database, you must enter the following information in the fields provided:

- Source pathname
- Destination pathname

The remaining fields are optional, but you should fill them in, if at all possible, for use in file queries. The source pathname field is where you list the location of the file to be propagated. Typically, all files are located in the directory */var/IRIXpro/propel/data*. If you have many files and file packages to propagate, this directory may have many subdirectories or may be linked to a separate directory altogether. Only source files in this directory structure may be distributed to slave distribution servers for further distribution to client hosts. If you are distributing a file directly to the final client host, you need not have the source file in this directory structure, though it is recommended for convenience and organization.

Once you have entered the file's information, press the *add* button by placing the cursor over the button and pressing the left mouse button. You see the information appear in the window in a slightly different format. This tells you that your entry has been accepted into the database.

To obtain a listing of all source files, packages, or destination locations in your database, use the right mouse button to bring up a popup window with the cursor over the appropriate field. For example, if you choose the List option from the popup menu over the Source field of the File database editor, you see a window very much like that shown in Figure 2-11.



/etc/group
/etc/netconfig
/etc/passwd
/usr/etc/config.net
/usr/tmp/distfile

Figure 2-11 A List of Source Files in the File database

Updating a File Database Entry

To change this entry, click the entry in the window with the left mouse button to highlight it, and the information you entered appears in the fields in the lower part of the screen. Change these entries as you wish and then press the *modify* button.

Querying the File Database

For an example of using the File database fields as query points, you can make a query to the database to find all files that are part of the package ‘’network configuration’’ and have the comment ‘’fully propagated.’’ Begin by using the mouse to press the *Clear* button. This clears the fields of all information. Then enter your query parameters in the appropriate fields. Then use the mouse to press the *search* button. All files in your database that match those two parameters will be displayed in the window.

The Distribution Rule Database

This database holds a set of rules that control the distribution of files and packages to host systems. This file uses a very simple rule set to describe which files should be pushed to which systems, and under what conditions the push takes place.

The fields in this database are:

Frequency	This field holds a simple term of your choice to describe how often files and packages with this frequency are pushed. For example, you may set your frequencies to Daily and Weekly, and run them accordingly with *cron*, or you can simply call the frequencies A, B, and C and select your own definitions of those terms through *cron* or manual use of the *runpropel* command. The *runpropel* command line that you issue determines the actual operation; this field is merely your tag for a frequency group that you define.
Name	This field contains a name for the distribution rule.
File Query	This field contains a file attribute query. Use this field to enter keywords to find in the File database.

Host Query	This field contains a host attribute query. Use this field to enter keywords to find in the Host database.
Comment	This is an unused text string that can be used for noting information about a rule.

Editing the Rule Database

The *ruleedit* command allows you to specify which files or packages of files are pushed out to a list of client hosts. The Rule database stores the rules you have created with the *ruleedit* command. Enter the command:

```
ruleedit
```

You see the following screen shown in Figure 2-12.

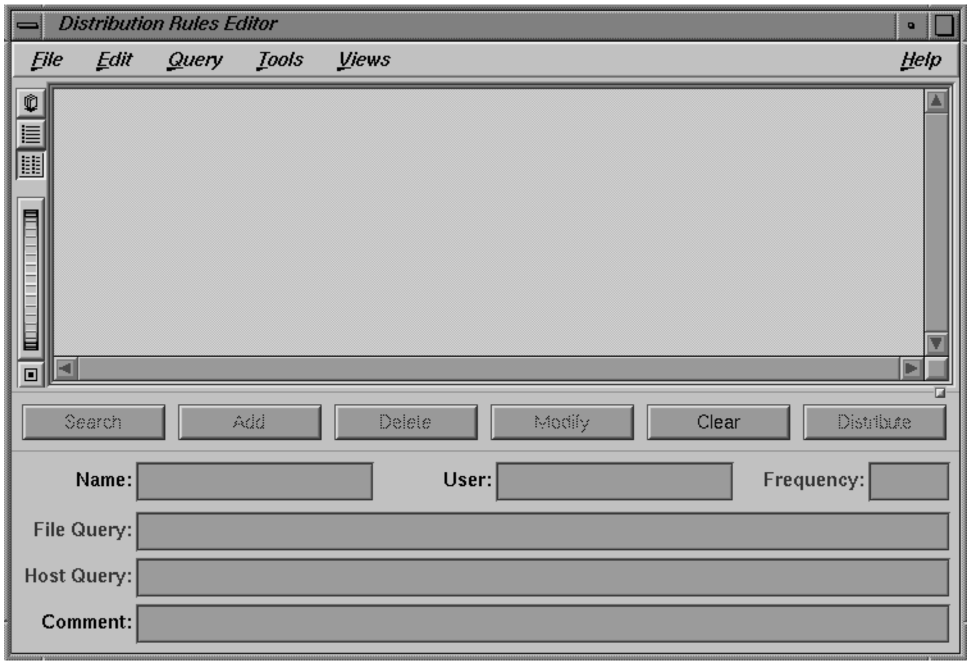


Figure 2-12 The Distribution Rules Editor

Adding a Distribution Rule to the Rule Database

To use the *ruleedit* tool to add a distribution instruction to your Rule database, you must enter the following information in the fields provided:

Frequency of the distribution
: This is a frequency parameter of your choice, as discussed in "How propel Works" on page 24.

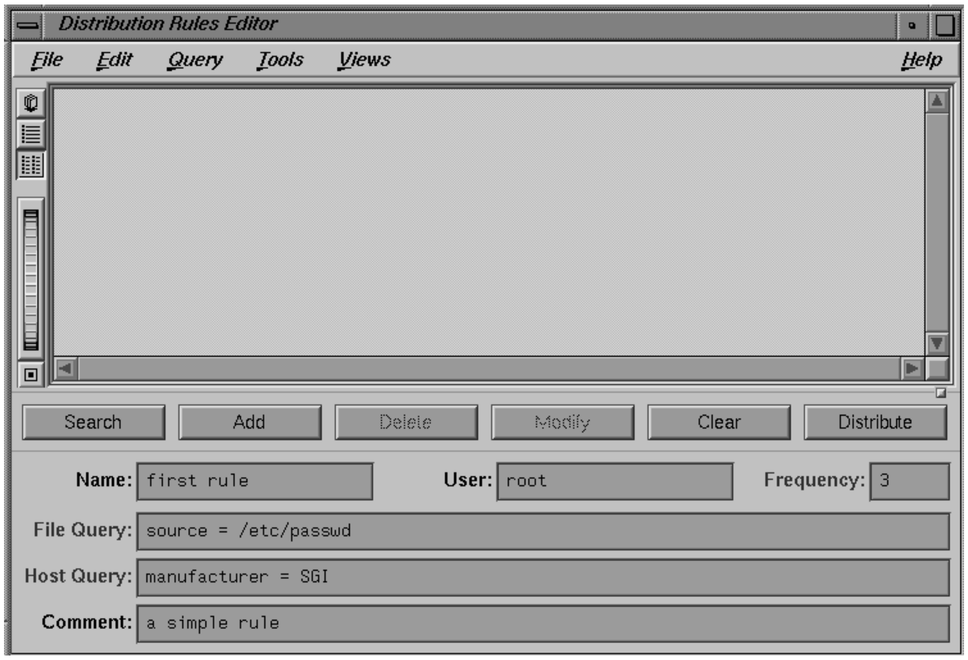
A Name for the rule
: Every rule must have a name.

File query parameters
: This must be a query formed according to the rules described in "Additional Database Query Techniques" on page 60.

Host query parameters
: This must be a query formed according to the rules described in “Additional Database Query Techniques” on page 60.

Once you have entered the rule’s information, press the *add* button by placing the cursor over the button and pressing the left mouse button. You see the information appear in the window in a slightly different format. This tells you that your entry has been accepted into the database.

The *ruleedit* window with a prepared entry looks something like the window in Figure 2-13.



Name:	first rule
User:	root
Frequency:	3
File Query:	source = /etc/passwd
Host Query:	manufacturer = SGI
Comment:	a simple rule

Figure 2-13 An Prepared Distribution Rule

Updating a Rule Database Entry

To change an entry, click the entry in the window with the left mouse button to highlight it. The information you entered appears in the fields in the lower part of the screen. Change these entries as you wish and then press the *modify* button.

Querying the Rule Database

To query the Rule database, begin by using the mouse to press the *Clear* button. This clears the fields of all information. Then enter your query words any field or combination of fields. Then use the mouse to press the *search* button or press **`<Enter>`**. All rules in your database that match the query parameters are displayed in the window.

The User Database

The User database contains all the necessary information about the users on your network. For more information on users and groups and the contents of these fields, see the *IRIX Advanced Site and Server Administration Guide.*

Records in the User database have the following fields:

User ID	This field contains the user's unique UID number. UID numbers are integers from 0 to 65535. This field must be filled for an entry to be accepted. See the *IRIX Advanced Site and Server Administration Guide* for more information on assigning User ID numbers if you are unsure about this field.
Login	This field contains the login name of the user. This login name must be unique within your network domain.
Password	This field contains the user's encrypted password. By using the right mouse button, you can select an option to allow you to set the password in clear text. The password you set will be encrypted and the encrypted text will be placed in the field. You can also use the *passwd*(1) command and copy the encrypted string from the */etc/passwd* file. Be certain to enter the encryption string exactly, or the password will not function correctly.

Shell	This field contains the user's default login shell. This is typically */bin/sh* or */bin/csh*.
Server	This field contains the name of the user's home system. This home system should also be the user's mail server.
Name	This field contains the user's full name. This field is required for an entry to be accepted.
Groups	This field contains the names of any user groups to which the user belongs. Multiple groups may be specified in a space-separated list. The first group listed in the user's primary group. This field is required for an entry to be accepted.
Home	This field contains the pathname of a user's home directory. (For example, this field might contain the pathname: */usr/people/armistead*).
Projects	This field contains the names of any projects the user may be associated with. Multiple projects may be specified in a space-separated list. This field is provided as a convenient mechanism for making collections of users through queries. You may place any group-identifying tags in this field if you choose.
Comment	This field is available for any purpose.

Editing the User Database

This database contains the names and vital information of the users on your network. The command */usr/IRIXpro/bin/useredit* invokes this tool, and you see the following screen.

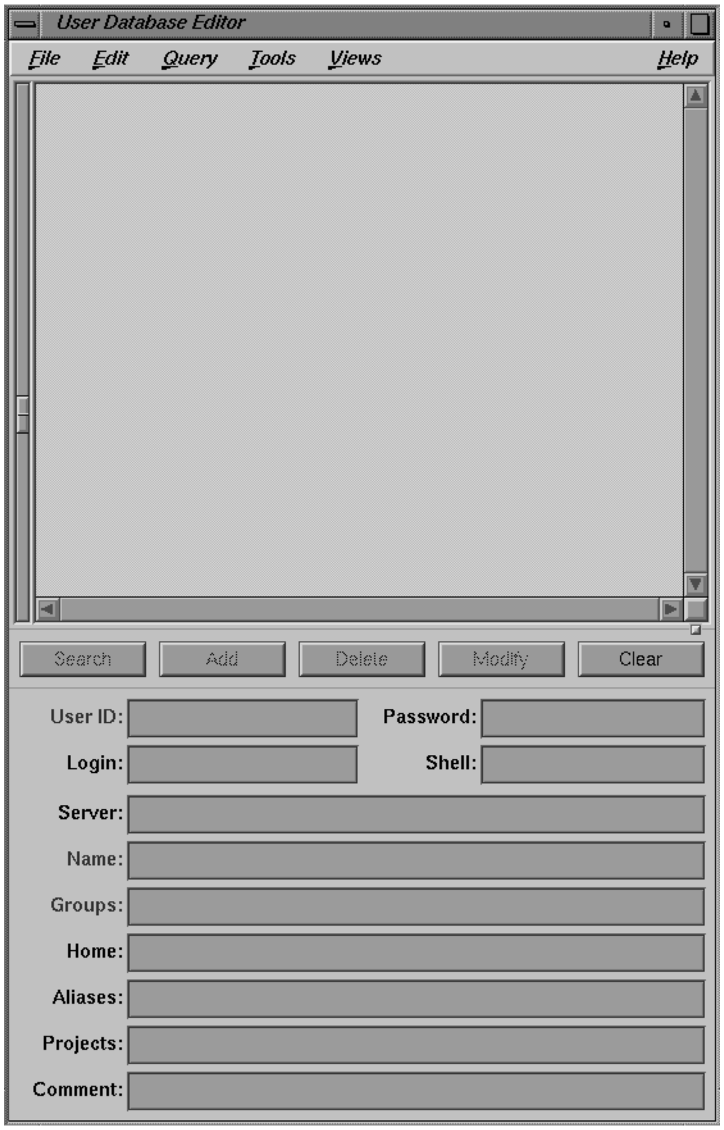


Figure 2-14 The User Database Editor

Adding a User to the User Database

To use the *useredit* tool to add a user to your User database, you must enter the following information in the fields provided:

- User ID
- Name
- Group (at least one)

The remaining fields are optional, but you should fill them in if at all possible for use in user queries. Any record associated with an actual user who will be logging in should be fully filled out. The minimal requirements allow you to reserve UID numbers without actually allocating them to specific users. It is sometimes useful to reserve blocks of UID numbers in advance if you wish to create future users in a certain group with consecutive UID numbers.

Once you have entered the user's information, press the *add* button by placing the cursor over the button and pressing the left mouse button. You see the information appear in the display window in a slightly different format. This tells you your entry has been accepted into the database.

Updating a User Entry

To change this entry, clicking on the entry in the display window with the left mouse button highlights it, and the information you entered appears in the fields in the lower part of the screen. Change these entries as you wish and then press the *modify* button.

Querying the User Database

For an example of using the User database fields as query points, you can make a query to the database to find all users that are part of the group ''networking'' and which have the comment ''member of technical staff'' Begin by using the mouse to press the *Clear* button. This clears the fields of all information. Then enter your query parameters in the appropriate fields. Then use the mouse to press the *Search* button. Records of all users in your database who match those two parameters will be displayed in the window.

The Group Database

Records in the Group database have the following fields:

Name	The name of the group of users. Make this name unique from any user or host name.
Group ID Number	The unique GID number associated with this group.
Query	The query parameters to define the group. You can query on any field in the User database. The field is made up of a list of typed fields separated by newlines (**`<Enter>`** keystrokes). The first word on each line is the type of query that the line represents. The available types are *username*, *userquery*, and *groupname*.
Comment	Any notes you wish to make about the group.

Editing the Group Database

The *groupedit* utility allows you to define groups of users for *propel*. A group of users is simply a selection of users that you have defined. Usually, groups are made of users with some common factor, such as a common project or organization. Since you specify the attributes of a user by using the *useredit* utility, you can prepare users to be part of a group when you enter them in your User database. For example, you can enter several users in your User database with the comment “advanced engineering lab” in the comment field. Then, when you use *groupedit* to create a group, you specify the “advanced engineering lab” comment as the query field when creating the group. The syntax for these queries is described in “Additional Database Query Techniques” on page 60. All users with that comment become part of the group. The names of individual users, User database queries, and other groups that define the new group are stored in the Group database.

Once defined, the *groupedit* utility stores the parameters of the group in a database. Enter the command:

`/usr/IRIXpro/bin/groupedit`

you see the following window, shown in Figure 2-15:

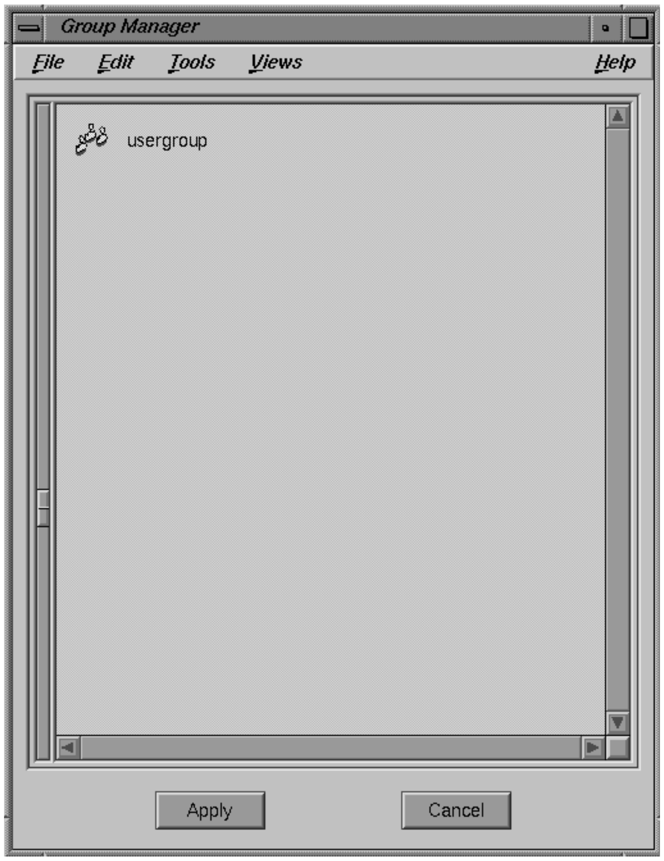


Figure 2-15 The Group Manager

When you bring up the Group Manager window after your database is populated, you see an icon and a name for each group in the database.

Adding a Group of Users to the Group Database

To use the *groupedit* tool to add a user group to your Group database, place the mouse cursor inside the *groupedit* window and use the right button of the mouse to bring up the popup menu. There are several options:

Help	Brings up a help card.
Add Group	Brings up the Add Group window.
Add User	Adds a user to an existing group.
Remove Selected	Removes the selected group.
Edit Selected Users	Brings up the *useredit* tool. To select a user, you must have a group displayed in a group display window.
Preview	Shows you the list of users currently in a group. All User database queries and other groups that are part of the current group are fully expanded.

Select the Add Group option to create a new group of users. When you select this option, you see the new window, shown in Figure 2-16.

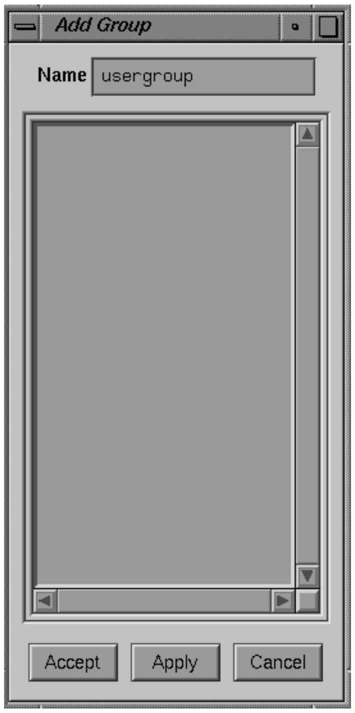


Figure 2-16 The Add Group Window

You must enter a name for the group, then use the left mouse button to press the Accept or Apply buttons to create your new group. The Apply button creates your new group, and retains the Add Group window, so you can continue to create more groups conveniently. The Accept button creates your group and removes the Add Group window. The Cancel button cancels the operation without creating the new group.

Once you have created the group, you must specify users to be part of the group. You should notice that a new icon has appeared in the *groupedit* window. If you have previously used the *add_user* script, (described in “Populating the User and Group Databases”) your Group Manager window will already show many icons representing your existing user group. If you make two new groups and call one “irixpro” and the other “user” the window looks similar to Figure 2-17.

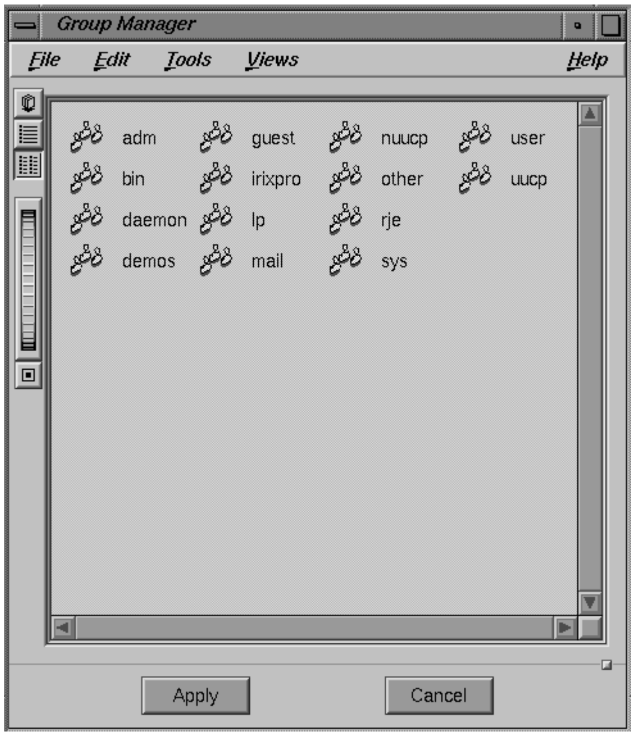
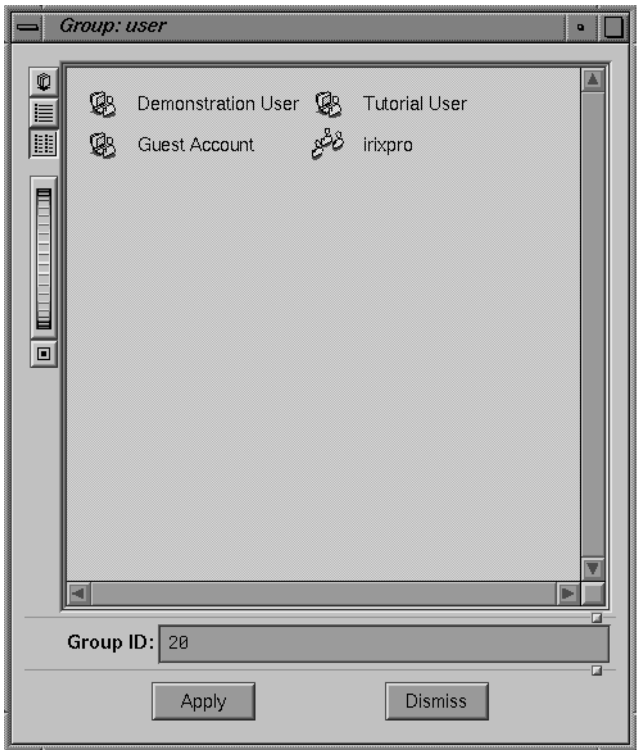


Figure 2-17 A Populated Group Manager Window

To add a list of users, a list of groups, a User database query, or any combination of these things to your new group, double-click the left mouse button with the mouse cursor over the icon for the new group. You see a new window that looks something like Figure 2-18.

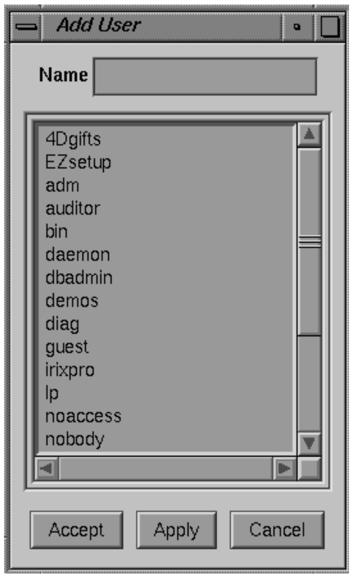


Group ID: 20

Figure 2-18 A Group Display Window

Now use the right mouse button to bring up the popup menu, and select either the Add User or Add Group options to populate your group.

The Add User option brings up the Add User window, shown in Figure 2-19, with the entire list of users from your User database.



4Dgifts
EZsetup
adm
auditor
bin
daemon
dbadmin
demos
diag
guest
irixpro
lp
noaccess
nobody

Figure 2-19 The Add User Window

To place a user in your new group, place the mouse cursor over the name of the user you wish to select and double-click the left mouse button. When you are finished selecting individual users, click the Accept button to secure your changes and remove the Add User window.

The Add Group option brings up the Add Group window, previously shown in Figure 2-16. All your currently configured groups are shown in this window. As with the Add User window, double-click the name of a group you wish to add to your new group and complete your changes by pressing the Accept button.

Once you have entered all the information and created your groups, press the *apply* button in the Group Manager window to place your groups in the Group database.

Updating a Group Database Entry

To change a Group database entry, click the icon with the left mouse button to highlight it, and the Group Display window for that group appears. Change these entries as you would if you were making a new group, then click the *Apply* button to finalize your changes.

Additional Database Query Techniques

The *propel* databases can be queried using query language rules and glob expressions. The rules regarding glob expressions are based on the Bourne shell glob expressions. See the section of the *sh*(1) reference page titled *File Name Generation* for additional documentation on glob expressions.

There are two basic ways to query from the graphical utilities: direct and indirect queries.

Direct queries are queries to the current database made through the editor for that database. For example, to query the Host database through the *hostedit* utility, you clear the screen with the *Clear* button on the utility, then enter your query parameters in the appropriate fields and press the *Query* button using the mouse.

Indirect queries are queries to a database other than the current database. For example, an indirect query would happen if you are editing the Rule database and you wish to query the Host database for a list of hosts to use in the rule you are creating.

The principal difference between the two forms of database query is that in the direct query, you simply fill in the desired parameter in the provided field and press the *Query* button using your mouse. To make an indirect query, you must fill in the field name from the database you are querying and also fill in the desired parameter value. For example, a simple indirect query in the *host query* field of the Rule database has the form:

fieldname = value

Any propel query menu allows you to form a query in this manner, and all query fields have an option available from the popup menu (brought up by pressing the rightmost button on the mouse while the cursor is within the query window) to expand the query immediately.

If you use this form to query for a host named *pickett*, you would form the actual query like this:

```
hostname = pickett
```

You cannot immediately see the results of your query on an indirect query unless you explicitly expand it. The results are not printed in the display window of the database editor. The query takes place only when *propel* executes and performs the distribution based on your query.

The advantage to this method is that you do not need to rebuild your databases if you add a new file or host that will be subject to a query. For example, if you have set a distribution to be sent to all hosts with the model of “Indy” on a regular basis, you do not need to remake the distribution rule if you add another “Indy” host. The new host is automatically selected by the existing query. If you wish to check to see if your query parameters will produce the desired results, use the editor for the database you are querying indirectly or the *dbredit* command to make a test query using the same parameters.

A glob expression in this type of query has the format:

```
fieldname ~ expression
```

For example, an actual query would be similar to this:

```
model ~ indigo*
```

or similar to this:

```
os ~ “IRIX 5.*”
```

You can use several query parameters at once on both direct and indirect queries. In a direct query, simply fill in as many fields as you like before you press the *Query* button on your database editor. For an indirect query, you must form the query according to query language rules. For example, to query for all hosts whose manufacturer is “SGI” or whose model name begins with the characters “Ind” use the following query:

```
manufacturer = SGI, model ~ Ind*
```

With the above query, you have selected all hosts whose manufacturer is listed as “SGI” as well as all hosts whose model name begins with “Ind.” This is known as a logical ‘‘OR’’ query.

To select only those hosts whose manufacturer is listed as “SGI” and whose model is listed as “Indy,” use the following logical ‘‘AND’’ query:

```
manufacturer = SGI && model = Indy
```

The difference is that the second query selects only those hosts that fulfill both requirements, while the first query selects those hosts that fulfill either requirement.

You can further expand your queries by incorporating more parameters. For example, to select those hosts whose manufacturer is “SGI” and whose model is “Indy,” and also the host named “callahan,” and any host with the comment “Universal,” use the following query:

```
manufacturer = SGI && model = Indy, hostname = callahan,
comment = universal
```

Query Keywords and Glob Expressions

The query rules follow simple glob expression syntax. The following keywords and operators are recognized:

all The **all** keyword returns all elements in the database. For example, the Host database query

```
all && !labmachine
```

returns all the hostnames in the database except *labmachine*.

to The **to** keyword allows you to specify a range of records. For example, the Host database query

```
iplist = 192.27.3.0 to 192.27.3.255
```

returns all addresses in the specified range. In the above example, all hosts on the 3 subnet would be returned.

= The equal sign operator is the most basic operator. Given alone, it indicates that the expression following describes the desired query parameters.

!	The exclamation point operator indicates a logical “NOT” operation. This operator negates the usual meaning of a query parameter. For example, if you want to query for all SGI manufactured hosts except the host “bragg,” use the query:

```
manufacturer = SGI && ! hostname = bragg
```

!=	The exclamation point negation operater followed by an equal sign indicates “everything not equal to the following expression.” For example, the query

```
manufacturer != SGI
```

selects all manufacturers except SGI.

>	The Greater-Than operator selects all values greater than the given expression.
>=	The Greater-Than or Equal To operator selects all values greater than or equal to the given expression.
<	The Less-Than operator selects all values less than the given expression.
<=	The Less-Than or Equal To operator selects all values less than or equal to the given expression.
&&	The double ampersand operator indicates a logical “AND” operation. It is used to indicate that both parameters in a query must be satisfied to create a match. For example, to select those hosts whose manufacturer is “SGI” and whose model is “Onyx,” use the following query:

```
manufacturer = SGI && model = Onyx
```

You may use as many logical “AND” operators together as you wish, and the meaning remains the same. All query parameters put together with “AND” operators must be satisfied to create a match.

||	The double pipe operator indicates a logical “OR” operation. This means that only one query parameter needs to be satisfied to create a match. For example, the query:

```
manufacturer = SGI || owner = armistead
```

matches all hosts whose manufacturer is “SGI” or that are owned by “armistead” regardless of their manufacturer. You may use as many logical “OR” operators together as you wish and the meaning remains the same. Only one query parameter in an “OR” string must be satisfied to create a match.

( ) Parentheses enclose expressions that need to be evaluated separately. For example, suppose you wish to evaluate a query to find all hosts whose model is “Indy” except those whose OS Release is “5.2,” and then add a query for all hosts whose model is “Indigo.” In this case, you would use parentheses to force the first parameters to be evaluated before adding the last parameter. The syntax for this query would be:

```
(model = Indy && ! os = 5.2), model = indigo
```

Query Shortcuts

Query shortcuts exist for each database. When querying the Host database, the following shortcut rules apply:

- A single word, such as “**upstairs**,” is assumed to indicate the name of a collection. Thus, “**upstairs**” is interpreted as equal to the query “**collect:name = upstairs**.”
- A string of letter characters with periods interspersed is assumed to indicate an individual hostname. Thus, “**gordon.eng.biz.com**” is interpreted as equal to the query ‘’**hostname = gordon && domain = eng.biz.com**.”
- A string of numeral characters with periods interspersed is assumed to indicate an IP address. Thus, “**192.27.0.1**” is interpreted as equal to the query “**iplist = 192.27.0.1**.”
- A field name listed by itself matches all records where that field has an entry. Thus, “**owner**” returns all records where an owner is listed.

When querying the File database, the following shortcut rules apply:

- A single word, such as “**newfiles**,” is assumed to indicate the name of a package of files. Thus, “**newfiles**” is interpreted as equal to the query “**package = newfiles**.”

- A string of letter characters with slashes (/) interspersed is assumed to indicate a source file pathname. Thus, “**`/var/IRIXpro/propel/data/etc/hosts`**” is interpreted as equal to the query ‘’**`source = /var/IRIXpro/propel/data/etc/hosts`**.”
- A field name listed by itself matches all records where that field has an entry. Thus, “**`source`**” returns all records where a source filename is listed.

When querying the Collection database, the following shortcut rules apply:

- A single word, such as “**`upstairs`**,” is assumed to indicate the name of a collection. Thus, “**`upstairs`**” is interpreted as equal to the query “**`name = upstairs`**.”
- A field name listed by itself matches all records where that field has an entry. Thus, “**`name`**” returns all records where a name is listed.

When querying the Rule database, the following shortcut rules apply:

- A single word, such as “**`walnuts`**,” is assumed to indicate the name of a rule. Thus, “**`walnuts`**” is interpreted as equal to the query “**`name = walnuts`**.”
- A field name listed by itself matches all records where that field has an entry. Thus, “**`frequency`**” returns all records where a frequency is listed.

Using the dbredit Utility

The *dbredit*(1) utility is a command-line interface to the *propel* databases designed for use in server environments where the graphical utilities cannot be used. All functions that are performed by the graphical utilities can also be performed with *dbredit*. Complete information on the *dbredit* command is provided in the *dbredit*(1) reference page. The basic syntax of *dbredit* is:

```
dbredit database command information
```

where *database* is the name of the *propel* database to be manipulated or queried, *command* is an operation corresponding to the operations performed through the graphical editors, and *information* is the information to be entered into the database or the query parameters.

The *database* variable can be one of *host, file*, *rules*, or *collect*. The *command* variable can be one of *classes, init*, *list*, *insert*, *query, delete*, or *update*. The *information* variable is a set of arguments specific to each *command*. The arguments for each *command* are listed below:

classes	This command prints information about all databases currently configured.
init	This command initializes the named database.

list

Takes no arguments. This command returns all records in the database. For example, to see your entire Host database, use the command:

```
dbredit host list
```

You see output similar to the following:

```
{hostname {sherman}} {manufacturer {SGI}} {os
{IRIX 5.3}} {domain {engr.com}} {model {IRIS
Indigo2}} {iplist {127.0.0.1 192.999.888.7}}
{maclist {{} 08:00:69:07:94:D7}} {mxlist {{}}}

{hostname {maguire}} {manufacturer {SGI}} {os
{IRIX 5.3}} {domain {wpd.sgi.com}} {model {IRIS
Indy}} {iplist {192.555.444.2}} {maclist
{08:00:69:06:C1:95}} {mxlist {{}}}
```

There is one output entry per entry in your database, and each field in the entry is listed along with its value, if any. For example, in the above listed output, the first field reads:

```
{hostname {sherman}}
```

The first word is the field name, and the second word (inside braces) is the field value. The whole field statement is further enclosed in braces. Where there is no value for the field, only the field name and empty braces are displayed, as follows:

```
{mxlist {{}}}
```

add

Takes a list of “attribute = value” arguments that correspond to the fields in the appropriate database and returns the entityID number of the new record. For example, to add a typical file record, you might use the following command:

```
dbredit file add source=/tmp/localfile \
destination=/tmp/remotefile
```

search Takes a list of queries or entityID numbers, and returns the matching records. For example, to search for all machines manufactured by Silicon Graphics, use the command:

```
dbredit host query manufacturer=SGI
```

delete Takes a list of query strings or entityID numbers, and returns the records that have been deleted. For example, to delete the records for all machines manufactured by "Tredegar," use the command:

```
dbredit host delete manufacturer=Tredegar
```

modify Takes a list of queries or entityID numbers, and returns the entityID numbers of the updated records. For example, to change all of the operating system version strings for machines running "IRIX 5.2" to "IRIX 5.3" use the command:

```
dbredit host modify os="IRIX 5.2" os="IRIX 5.3"
```

destroy Removes all values of a named field or database. For example, the command

```
dbredit host destroy owner
```

removes all values of the ‘‘owner’’ field from the Host database. The command

```
dbredit host destroy
```

removes the entire Host database. After a destroy operation, the database may be reinitialized or simply repopulated.

names Returns the complete list of fields for the given database. For example, the command

```
dbredit host names
```

returns all the field names in the Host database. Given with no database name, the command returns a list of the known databases. For example, the command:

```
dbredit names
```

returns the names of all known databases.

All rules about query structure and glob expressions that apply to the graphical database editing utilities also apply to *dbredit*. If you have trouble creating the correct syntax, try issuing a **`dbredit names`** command and entering the field names exactly as they are displayed.

Running propel

The *propel* program can be executed automatically by the *cron* process, or from a shell command at your convenience. The following sections describe the two ways of running *propel*.

Running propel From a Shell Command

There is a shell command, *runpropel*(1M), available to force a distribution to a host (or collection of hosts) immediately. Command line arguments are available to specify a direct push or a periodic push of any standard frequency. For complete information on *runpropel*(1M), see the *runpropel*(1M) reference page.

Running propel Through cron(1M)

The *runpropel* command can be run by *cron*(1M) regularly. Each time *cron* executes *runpropel*, the program reads and interprets the various databases and then generates or modifies all necessary files.

An example entry from a distribution system's *crontabs* file would be:

```
runpropel -f week
```

The frequency of the push is configurable at any time by changing the command in your *crontabs* file and restarting *cron*. For complete information on *cron*, see the *cron*(1M) reference page, as well as the section titled "Automating Tasks with at(1), batch(1), and cron(1M)" in the *IRIX Advanced Site and Server Administration Guide*.

The propel File Generation System

The file generation system consists of a directory of scripts that are run by *propel*, if needed, before the actual distribution takes place. If an entry in the File database has a program listed in the “generator” field, *propel* runs the generation program and takes the output from that program to replace the contents of the existing file. For example, you can create an entry in the File database in which the destination file is */etc/hosts*, and the “generator” field lists the */usr/IRIXpro/sample/propel/make_hosts* command. In this case, the *make_hosts* command builds an */etc/hosts* file from the entries of the *propel* Host database, and then distributes the resulting file to all client hosts called for in the distribution rule, installing the new file as */etc/hosts*.

Some utilities to generate files are provided with *propel*, though you can always create your own file-generation utilities. By using the utilities provided with *propel*, and by propagating the resulting files, you can be sure that your entire network maintains consistent system configuration files. There are several file-generation scripts shipped with *propel* in the */usr/IRIXpro/sample/propel* directory:

make_hosts	Generates the */etc/hosts* file from the *Host database*.
*make_ether*s	Generates the */etc/ethers* file from the *Host database*.
make_fwd_dns	Generates the forward maps for *named*(1M) to be distributed to the *named* master.
make_rev_dns	Generates the reverse maps for *named*(1M) to be distributed to the *named* master.

Chapter 3

Dynamic Network Configuration With proclaim

This chapter describes the *proclaim* dynamic network configuration software package. The following topics are covered:

- The *proclaim* system overview. See "The proclaim Dynamic Network Configuration System" on page 71.
- Installing *proclaim* on a server. See "Installing proclaim on a Server" on page 72.
- Installing *proclaim* on a client system. See "Installing proclaim on a Client" on page 74.
- Configuring *proclaim* on a server system. See "Configuring proclaim" on page 74.
- Sample entries in *proclaim* configuration files. See "Sample Configuration File" on page 78.
- Information on limitations of this implementation with respect to the DHCP software standard. See "Limitations and Restrictions of This Release" on page 80.

The proclaim Dynamic Network Configuration System

The purpose of *proclaim* is to allow the site administrator to set up one or more server systems that dynamically distribute network IP addresses and site configuration parameters to new and requesting client systems. In this way, a site with only a few available addresses can serve a large number of hosts that connect to the network only occasionally, or a very large site can manage the permanent assignment of addresses with a minimum of human attention. The *proclaim* application is based on the Dynamic Host Configuration Protocol described in IETF RFC 1541.

The *proclaim* application consists of a daemon that runs on the server or servers, a set of configuration files for each server. The client process is distributed with the standard IRIX operating system and can be set up to be run through the bootup process or it can be run by hand. For more information, see the *proclaim*(1) reference page.

Installing proclaim

The following sections provide information on the installation of *proclaim* on servers and clients.

Installing proclaim on a Server

Like all of IRIXpro, *proclaim* is installed using *inst*(1). When you install *proclaim*, you are installing the server daemon and the initialized configuration files. (There is template information in the configuration file.) Files are also added to the */etc/init.d* directory and an option is added to your *chkconfig* list.

The following list shows each file installed on your server:

/etc/config/dhcp_bootp.options
: The configuration file described in "The dhcp_bootp.options File" on page 79.

/etc/config/proclaim_server
: The *chkconfig*(1M) file for *proclaim*.

/usr/IRIXpro/proclaim/config/config.Default
: The default configuration file for *proclaim*, described in "The Standard Configuration File" on page 74 and in the section titled "Sample Configuration File" on page 78.

/usr/etc/dhcp_bootp
: The special *bootp* program used by *proclaim*. This program replaces the standard *bootp* and serves both the standard *bootp* and dhcp clients.

After installing the *proclaim* subsystem of the *IRIXpro* product, you need to perform the following steps in order:

1. Enter this command:

   ```
   chkconfig proclaim_server on
   ```

2. Next, you must modify the */etc/inetd.conf* file to use *dhcp_bootp* rather than the standard *bootp* protocols. Place a hashmark (#) at the beginning of the *bootp* entry to make that line a comment, and open a new line below that entry. Enter the following text all on one line:

   ```
   bootp dgram udp wait root /usr/etc/dhcp_bootp
   dhcp_bootp -P -o /etc/config/dhcp_bootp.options
   ```

   In the above example, the **-P** option signifies that this is a dhcp bootp server. The **-o** option is used to specify a *dhcp_bootp* configuration file for specifying any additional *proclaim* specific options.

   Save and exit the *inetd.conf* file when you have made this entry.

3. Reboot your system for these changes to take effect.

Files Modified by proclaim

The *proclaim* server may modify the following files on your system:

- */etc/hosts*
- */etc/ethers*
- */usr/IRIXpro/proclaim/etherToIP*

Starting proclaim

To start the *proclaim* daemon at any time on a server, log in as **root** and enter this command

```
chkconfig proclaim_server on
```

Then reboot your system for the change to take effect.

Stopping proclaim

To stop the *proclaim* daemon on the server, log in as **root** and enter this command

```
chkconfig proclaim_server off
```

Then reboot your system for the change to take effect.

Installing proclaim on a Client

The *proclaim* client software is included in the standard distribution of IRIX versions 5.3 and later. Administrators of heterogeneous networks can find the source code to the client daemon in */usr/people/4Dgifts/dhcp* and must port and compile the code as needed for other brands of computers.

Configuring proclaim

The following sections provide information on configuring *proclaim* on servers, and on the various configuration files, and limitations of this implementation of DHCP. If you are not comfortable with the basic concepts of networking, IP addresses, and netmasks, read Chapters 15 through 18 of the *IRIX Advanced Site and Server Administration Guide* before you continue reading this section.

Configuring proclaim on the Server

The *proclaim* server implementation uses three levels of configuration parameters based upon the subnet number of the originating client request. The configuration files are all placed in the directory */usr/IRIXpro/proclaim/config* and are named in the form *config.netnumbers*. For example, the configuration files for configuring clients on the 192.26.61 network are named *config.192.26.61.0*. If the configuration file for a client request originating on a particular subnet is not found, then the next level of configuration is supplied in the file *config.Default* in the same directory. If the default configuration file is not present or is unreadable, all clients are supplied the same configuration as the *proclaim* server itself.

The Standard Configuration File

The following configuration parameters can be supplied in the standard configuration file for each network, or the default configuration file. Host address specification can either be in standard IP address dot notation or as a hex number prefixed with a 0x. Most of the fields may be left blank to render them non-applicable.

pro_address_counter
: This integer field specifies the host number for the next IP address. The next address will be constructed using the counter and checked through the range of the assignable addresses. The first available address in the range will be assigned.

pro_host_pfx_counter
: This integer field specifies the starting number that will be appended to the *pro_host_prefix* to generate a new hostname. This counter will be incremented and a new hostname generated until a unique unused name is found.

pro_netmask
: This field takes a netmask in address form (*xx.xx.xx.xx*). For more information on netmasks, see the IRIX Advanced Site and Server Administration Guide. This field specifies the subnetmask that will be used by the client systems.

pro_lease
: This unsigned integer field specifies the client address lease time in seconds. This implementation of the DHCP software assigns only infinite leases, and thus the leases expire only when explicitly surrendered by the client.

pro_propel_server
: This field takes an IP address in address form (*xxx.xxx.xxx.xxx*) and specifies the IP address of the *propel* server that will be serving the clients on this subnet. Additional information about *propel* can be found in Chapter 2, “Software Distribution, User and Host Management With propel.”

pro_host_prefix
: This string field specifies the default text prefix for generating client hostnames. For example, the prefix “iris” directs *proclaim* to generate hostnames of the form *iris1*, *iris2*, *iris3*, and so on.

pro_choose_name
: This boolean (true or false) flag specifies whether the client systems are allowed to choose their own hostname or whether they must be assigned the name given to them by

the server. A value of 1 (true) in this field brings up a dialog box on the client system giving the user the option of either taking the name offered by the server or entering a hostname of the user's choice. If the user selects a name, the server will allow this name if it passes basic tests for syntax and uniqueness, otherwise the server/client dialogue will continue until a mutually acceptable name is submitted. A value of 0 (false) in this field indicates that the user on the client system must accept the name provided by the server.

pro_ipaddress_range

This field takes an entry of integers using standard numeric range rules. The entry defines the range of host number addresses assignable by this server. For example, the form is:

```
1-3, 5-7, 9
```

In the above example, the server would issue IP addresses with the base address specified in the configuration filename (such as *config.192.26.61.0*). Each client is issued an IP address matching the name of the configuration file, suffixed with the numbers 1 through 3, and 5 through 7, and 9, but not 4 or 8.

This option is used to restrict the IP addresses offered by a given server. This option is very useful if the administrator wants to assign only certain block(s) of addresses using *proclaim*, or in the absence of a server to server protocol, wishes to have multiple servers serve clients on the same subnetwork.

pro_router_addr

This field of comma-separated IP addresses specifies a list of addresses for network routers on the client's subnet. Routers should be listed in the order of preference for their use.

pro_timeserver_addr

This field of comma-separated IP addresses specifies a list of addresses for time servers available to the client. Addresses should be listed in the order of preference for their use.

pro_dnsserver_addr
This field of comma-separated IP addresses specifies a list of addresses for Domain Name System servers available to the client. Servers should be listed in the order of preference for their use.

pro_nisserver_addr
This field of comma-separated IP addresses specifies a list of addresses indicating NIS servers available to the client. Servers should be listed in the order of preference for their use.

pro_dns_domain
This text field specifies the domain name that client should use when resolving hostnames using DNS.

pro_nis_domain
This text field specifies the name of the client's NIS domain.

pro_mtu This unsigned short integer field specifies the MTU (maximum transmission unit) to use on the network interface configured in this file. The minimum legal value for the MTU is 68.

pro_allnets_local
This Boolean (*true* / *false*) field specifies whether or not the client may assume that all other subnets of the IP network to which the client is connected use the same MTU as the subnet to which the client is directly connected. A value of 1 (*true*) indicates that all subnets share the same MTU. A value of 0 (*false*) means that the client should assume that some other subnets may have smaller MTUs.

pro_broadcast
This IP address field specifies the broadcast address in use on the client's subnet.

pro_domask_disc
This Boolean (*true* / *false*) "Perform Mask Discovery" field specifies whether or not the client should perform subnet mask discovery using ICMP. A value of 1 (*true*) means that the client should perform mask discovery, while a value of 0 (*false*) indicates that the client should not perform mask discovery.

pro_resp_mask_req

This Boolean (*true* / *false*) "Mask Supplier" field specifies whether or not the client should respond to subnet mask requests using ICMP. A value of 1 (*true*) means that the client should respond. A value of 0 (*false*) in this field means that the client should not respond.

pro_static_route

This field takes a comma-separated list of routes in the following form:

*dest_address - router_address, dest_address2 - router_address*2

The static route field specifies a list of static routes that the client should install in its routing cache. If multiple routes to the same destination are specified, they should be listed in descending order of priority. The routes consist of a list of IP address pairs. The first address is the destination address; its counterpart address, separated by a dash (**-**), is the address of the router to the destination. The default route (0.0.0.0) is an illegal destination for a static route.

Sample Configuration File

The following are the contents of a sample *config.150.166.61.0* configuration file.

```
pro_address_counter: 25
pro_host_pfx_counter: 5
pro_netmask: 255.255.255.0
pro_lease: 100000
pro_propel_server: 150.166.75.20
pro_host_prefix: irixpro
pro_choose_name: 0
pro_ipaddress_range: 3, 9-11, 40-75, 200-254
pro_router_addr: 150.166.61.19
pro_timeserver_addr: 150.166.61.27
pro_dnsserver_addr: 192.26.61.24
pro_nisserver_addr: 192.48.150.150
pro_dns_domain: sgi.com
pro_nis_domain: engr.sgi.com
pro_mtu: 1600
pro_allnets_local: 1
pro_broadcast: 150.166.61.255
```

```
pro_domask_disc: 0
pro_resp_mask_req: 0
pro_static_routes: 192.26.80.118 - 192.26.80.10,
192.26.80.118 - 150.166.61.33
```

The dhcp_bootp.options File

The */etc/config/dhcp_bootp.options* file may specify the following additional options. The options may be specified on individual lines or on the same line separated by white spaces.

-s *propel_database*	Directs the *proclaim* server to use the specified *propel* database for hostname, IP address, and network hardware address resolution. For more information on *propel*, see Chapter 2, "Software Distribution, User and Host Management With propel."
-y	The *proclaim* server uses the existing NIS maps for hostname, IP address, and network hardware address resolution. This host is required to be the NIS Master.
-h *hostname*	Specifies the name of the host where the *propel* lock manager is running if it is not running on this system.
-w *hosts_map*	Specifies the optional location of the *hosts* map. Not valid with the **-s** option. The default is */etc/hosts*.
-e *ethers_map*	Specifies the optional location of the *ethers* map. Not valid with the **-s** option. The default is */etc/ethers*.
-u *sysname*	Specifies the name for an optional *sysname* file. The default is */unix*.
-c *proclaim config dir*	This flag specifies an optional *proclaim* server configuration directory. The default directory is */usr/IRIXpro/proclaim/config*.

Limitations and Restrictions of This Release

The following restrictions and limitations are present in *proclaim* in this release:

- The server must be the NIS Master if it is using NIS for hostname, IP address, or network hardware address validation and mapping. Note that NIS is an optional software product, and not all systems and networks use it.
- This release of the server software assigns only infinite (meaning roughly 10-year) address leases. A client requesting a shorter lease must actually surrender the lease in order for the lease to be considered expired.
- The server will not serve *proclaim* and standard bootp clients in the same request packet.
- Clients requesting additional configuration parameters that are part of RFC1533 but are not supported in this release are not served by the *proclaim* server.

Chapter 4

Distributed System Monitoring With provision

This chapter describes the *provision* application and its use in monitoring the status of the host systems on your network. The following sections are provided:

- “The provision Monitoring System” provides an overview of the nature and purposes of the *provision* application.
- “Installing and Configuring provision” provides detailed instructions on installing *provision* and configuring it to your specific needs.
- “Using provision” provides general information on the interfaces and different programs that make up *provision*.
- “Using pvcontrolpanel” provides detailed information on the *pvcontrolpanel* logging and notification tool.
- “Using pvcontrol” provides detailed information on the *pvcontrol* text-based notification and logging tool.
- “Using pvgraph” provides detailed information on the dynamic *pvgraph* graphing tool.
- “The MIB Browser” provides detailed information on the MIB browser and MIB tools in general.

The provision Monitoring System

The *provision* application allows you (the administrator) to keep track of the running statistics of each system in your heterogeneous network from a single location. This location may be a host workstation or server console, or even a text-based terminal. The data provided about each system on the network can be displayed graphically, if the administrator’s host system allows it, or in text, or data can be stored for later analysis. Error messages from each system can be displayed immediately on the administrator’s console.

The *provision* application provides three basic utilities:

- *pvcontrolpanel* is a graphical notification and logging utility for use on systems with graphics capabilities, such as graphical workstations and X-terminals.
- *pvcontrol* is a text-based utility for use on non-graphics servers and ASCII terminals.
- *pvgraph* is a graphical tool to dynamically graph system performance and view logs of statistics made with *pvcontrol* and *pvcontrolpanel*.

The graphical user interface provides the full power of *provision* to the administrator. Information is updated in real time, and you can add or delete variables as you wish.

The text-based user interface is a subset of the graphical interface, and is provided for those administrators without access to graphics capability. The text-based interface does not provide the real-time updating of information that is featured in the graphical interface, but an interactive mode is available to change the collection instructions.

You may also want to coordinate the use of the standard IRIX features *sysmon*(1M), and *syserrpanel*(1M) with *provision*. These standard IRIX utilities use the system log daemon (*syslogd*) to monitor the system status. Complete information on *sysmon* and *syserrpanel* is available through the IRIX reference pages.

The *provision* application collects its information according to programs provided as part of IRIXpro, but you can write your own instruction sets in the programming language of your choice to customize *provision*. The *provision* application uses SNMP to collect information over the network.

SNMP stands for Simple Network Management Protocol. SNMP is used to communicate with other systems that also run SNMP. The other system can be a workstation, a router, a bridge, a hub, or a gateway—any host that has an IP address and implements the SNMP protocol and agent. SNMP implements an "agent." An agent is an SNMP program that exchanges information with a remote host. *snmpd*(1M) is the Silicon Graphics SNMP agent. Agents for other types of nodes may be implemented in software or firmware and are vendor-specific. There is a reference book for SNMP called *The Simple Book, An Introduction to Management of TCP/IP-Based Internets*, by

Marshall T. Rose. The book was published in 1991 by Prentice-Hall, of Englewood Cliffs, New Jersey, USA 07632. The ISBN number of this book is 0-13-812611-9.

SNMP relays basic system information about each host to the other hosts, on request. The information relayed comes from the Management Information Bases (MIBs) for that host. An MIB is the specification for the virtual store of the information supported by an agent. The standard IRIXpro MIBs are the *hp-ux_sgi* MIB and the *mib2* MIB, both distributed with IRIXpro in the */usr/lib/netvis/mib* directory. The *hp-ux_sgi* MIB is reprinted in Appendix A, "The hp-ux_sgi MIB," of this guide. Further information about MIBs and the MIB browser is available in the section titled "The MIB Browser." The browser is designed to be used by network managers experienced in managing various devices on the network.

You can create and add your own MIBs to your systems, or you can use MIBs obtained from other vendors with IRIXpro. MIB textual descriptions should be placed in the directory */usr/lib/netvis/mibs* to be accessed through IRIXpro.

The reason for creating and developing this software is to allow the system administrator of a large site with many different brands of hardware an extensible system to monitor many heterogeneous hosts from a single station.

Installing and Configuring provision

When you install IRIXpro, *provision* is automatically installed in the */usr/IRIXpro* directory structure. You must add */usr/IRIXpro/bin* to your PATH environment variable to easily use *provision*.

To use *provision* on your network, you must first propagate the *snmp* daemon to all systems that may be monitored.

On the monitoring system, the following requirements must be met before *provision* can run successfully. These requirements assume that you also wish to monitor the monitoring system itself:

- IRIXpro must be correctly installed.

- The *provisiond* daemon must be running. Place the following lines in the following files to cause *provisiond* to run automatically on the monitoring machine:

  To */etc/services*, add this line:

  ```
  provisiond    5299/udp   # IRIXpro provision daemon
  ```

  To */etc/inetd.conf*, add this entry (all on the same line):

  ```
  provisiond dgram udp wait root /usr/IRIXpro/bin/
  provisiond provisiond
  ```

  Then enter the command

  ```
  killall -HUP inetd
  ```

  to cause *inetd* to restart and run the *provision* daemon.

- The SNMP daemon (*snmpd*) must be running. Enter the following commands as **root** to cause *snmpd* to run automatically on the monitoring machine:

  ```
  chkconfig network on
  /etc/init.d/network start
  chkconfig snmpd on
  /etc/init.d/snmp start
  ```

- The hp-ux_sgi MIB must be installed. This is installed by default with the *snmpd* package of IRIXpro in */usr/lib/netvis/mibs/hp-ux_sgi.mib*.

On all Silicon Graphics systems to be monitored, the following requirements must be met before *provision* will successfully monitor their status:

- The SNMP daemon (*snmpd*) must be running. First, install the following package from your IRIXpro distribution on each Silicon Graphics system to be monitored:

  ```
  snmpd          01/04/95 SNMP Daemon with HP MIB Support
  ```

  Next, enter the following commands as **root** on the monitored machine to cause *snmpd* to run automatically:

  ```
  chkconfig snmpd on
  /etc/init.d/snmp start
  chkconfig network on
  /etc/init.d/network start
  ```

- The hp-ux_sgi MIB is installed by default with the *snmpd* package of IRIXpro. This MIB must be installed in order for the system to be monitored.

On all systems not manufactured by Silicon Graphics, the following requirements must be met before *provision* will operate correctly. Note that if another manufacturer's system MIB and SNMP daemon are not fully compatible with the distributed MIB and SNMP daemon, some scripts and MIB variables distributed with *provision* may not function for those systems. However, new MIBs and variables may be created for any or all systems:

- An SNMP agent (daemon) must be running on the system.
- An MIB must be installed on the system.

Consult your system manufacturer's documentation for information on fulfilling these requirements.

Using provision

There are two interfaces provided for *provision*, the graphical and the text interface. The graphical interface is the primary interface, since *provision* is designed to provide graphical information about your systems.

The provision Graphical Interface

When you first invoke *provision*, you see the window shown in Figure 4-1, in the standard Silicon Graphics desktop format. (See the section titled "Managing Windows" in the *IRIS Essentials* guide for complete information on the facilities of desktop windows.)

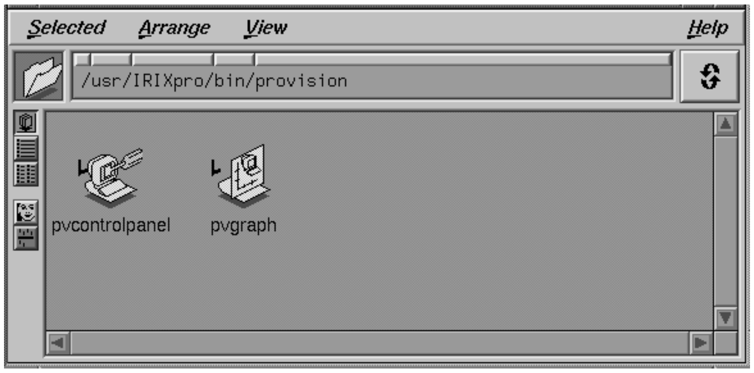


Figure 4-1 The provision Window

There are two main tools you can select from this window, *pvcontrolpanel* and *pvgraph*. These tools and their subordinate tools are discussed in the sections titled "Using pvcontrolpanel," "Using pvcontrol," and "The MIB Browser." To invoke a tool, place the cursor over the desired icon and double-click the left mouse button. The icon changes color when you select it, and the ''carpet'' underneath the icon moves up to show that the invocation was successful and a new tool window appears on your screen. Each of these tools is detailed in its own section below.

Using pvcontrolpanel

This tool is the main controlling panel for *provision*. From this panel, you can set up monitors on all systems on your network, and receive error messages and notifications. When you invoke *pvcontrolpanel*, the window shown in Figure 4-2 appears on your screen

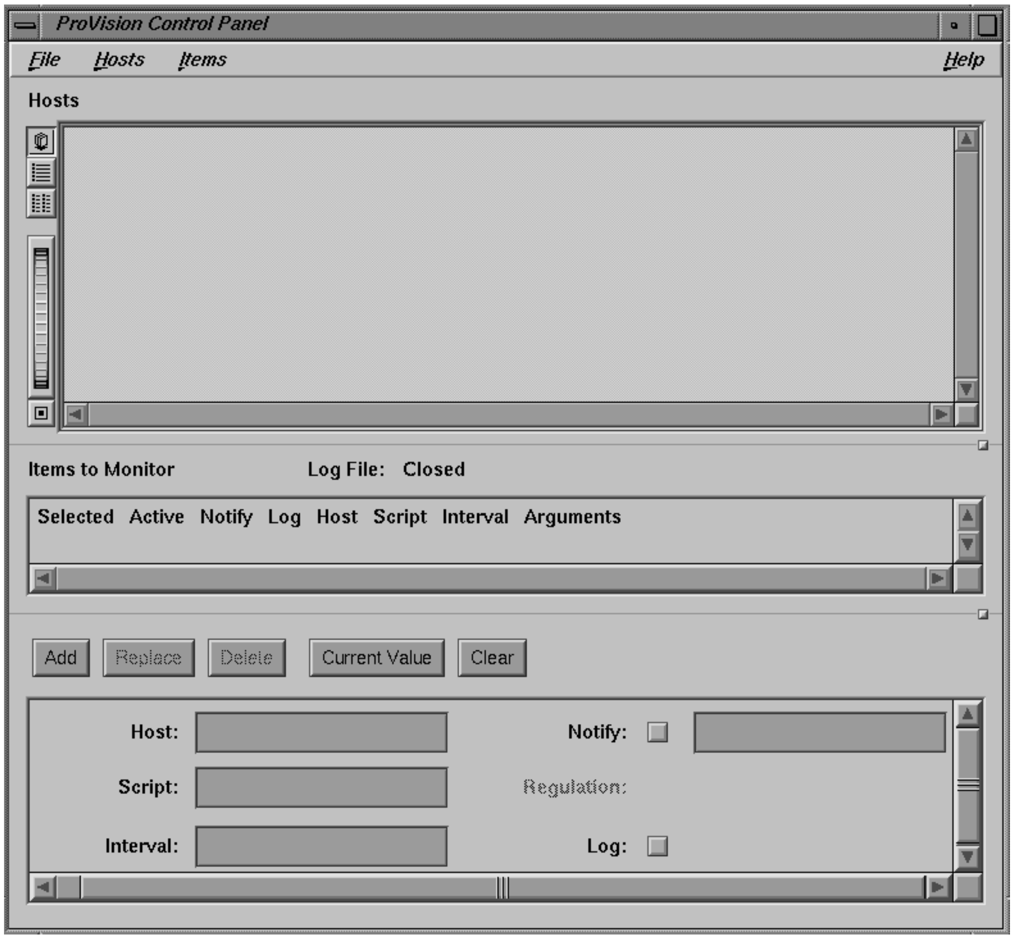


Figure 4-2 The pvcontrolpanel Window

There are four main sections of the *pvcontrolpanel* window. From the top of the window to the bottom, these sections are:

Menu Bar	The top bar, with the File, Hosts, Items, and Help menus. This is discussed in the section titled “Using the pvcontrolpanel Menu Bar.”
Hosts	All the hosts and collections currently monitored by this instance of *provision* and any other icons you may have added are shown in this area. This is discussed in the section titled “The pvcontrolpanel Hosts Area.”
Items to Monitor	All the items currently being monitored on any host are listed in this area. This is discussed in the section titled “The pvcontrolpanel Items to Monitor Area.”
Script Configuration	This area is where you enter information about the scripts and hosts to be monitored. This is discussed in the section titled “The pvcontrolpanel Script Configuration Area.”

Using the pvcontrolpanel Menu Bar

There are four menus available on the *pvcontrolpanel* menu bar. The menus and their choices are listed below.

The File Menu contains options dealing with the *pvcontrolpanel* configuration files and contains the options to restart and quit the session. The following choices are provided:

- Read Config

  This option reads a previously stored monitoring configuration from a file. You can also drag an icon representing a previously stored config file from the directory view onto the *pvcontrolpanel* icon to start *pvcontrolpanel* with that configuration. For more information on config files, see “The provision Configuration File.”

- Save Config

  This option saves the current monitoring configuration in a file.

- Save Config as...

This option saves the current configuration to a different filename.

- Quit

  This option ends the *pvcontrolpanel* session.

The Hosts Menu allows you to control the arrangement of the hosts in your *pvcontrolpanel* window. The following choices are available:

- View as Icons

  This option tells *pvcontrolpanel* to represent the hosts in your window with large icons, arranged alphabetically left to right.

- View as List

  This option tells *pvcontrolpanel* to represent the hosts in your window with smaller icons in a single column alphabetized list.

- View in Columns

  This option tells *pvcontrolpanel* to represent the hosts in your window with smaller icons, in an evenly columnized, vertical, alphabetized list.

- Add Icon

  This option adds an icon for a named host to your hosts area. You must first enter the hostname in the script configuration area.

- Remove Icon

  This option removes the selected icon from your hosts area.

Icons in the Hosts section represent each host that is currently communicating in some way with provision. A new icon is not added for each addition item monitored on a listed host unless it is specifically requested with the "Add Icon" menu choice. Also, host and collection icons can be dragged from the *propel* graphical tools (described in Chapter 2, "Software Distribution, User and Host Management With propel") or from the IndigoMagic desktop and tools, and can be dropped into the Hosts area of *pvcontrolpanel*.

The Items Menu contains options dealing with the operation of the *pvcontrolpanel* activity. An *Item* is any configured monitoring unit, for example, monitoring a script on a particular host at a particular interval. The following choices are provided:

- Start All Items

  This option starts all currently configured monitoring.

- Stop All Items

  This option stops all monitoring activity.

- Close Log File

  This option closes the current log file.

- Show Available Variables and Scripts

  This option brings up a window with a list of all available variables for monitoring, and all available scripts. To select a variable or script, place the mouse cursor over the desired list item and double-click with the left mouse button. This window is discussed further in “The Available Variable and Script Window.”

- MIB Browser

  This option invokes the MIB browser. For more information, see the section titled “The MIB Browser.”

- Add

  This option takes the information entered in the script configuration area and adds the entry to the Items area, and the specified host to the Hosts area.

- Delete

  This option deletes the selected item from the Items area.

- Delete All

  This option deletes all items and monitoring instructions.

- Replace

  This option changes the selected item by replacing it with a new item according to the current entries in the script configuration area.

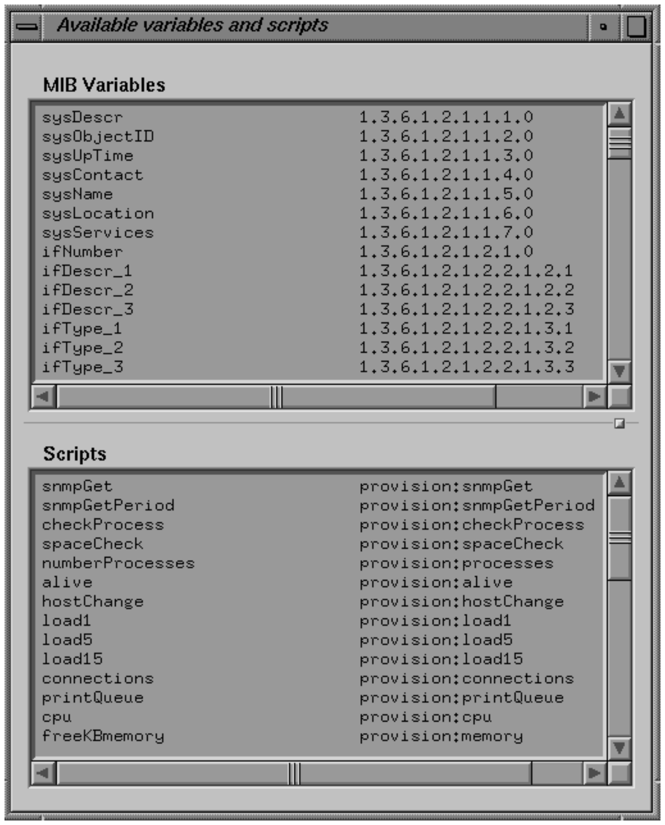
- Current Value

  This option runs the selected script once and returns the current value. The script will be run locally, although scripts can be written that execute other scripts on remote systems.

The Help Menu invokes the online help utility to provide help on all aspects of using *provision*.

The Available Variable and Script Window

When you select the “Show Available Variables and Scripts” choice from the Variables menu (or from the *Configure One Graph* window in *pvgraph*), you see the window shown in Figure 4-3.



MIB Variables

```
sysDescr                        1.3.6.1.2.1.1.1.0
sysObjectID                     1.3.6.1.2.1.1.2.0
sysUpTime                       1.3.6.1.2.1.1.3.0
sysContact                      1.3.6.1.2.1.1.4.0
sysName                         1.3.6.1.2.1.1.5.0
sysLocation                     1.3.6.1.2.1.1.6.0
sysServices                     1.3.6.1.2.1.1.7.0
ifNumber                        1.3.6.1.2.1.2.1.0
ifDescr_1                       1.3.6.1.2.1.2.2.1.2.1
ifDescr_2                       1.3.6.1.2.1.2.2.1.2.2
ifDescr_3                       1.3.6.1.2.1.2.2.1.2.3
ifType_1                        1.3.6.1.2.1.2.2.1.3.1
ifType_2                        1.3.6.1.2.1.2.2.1.3.2
ifType_3                        1.3.6.1.2.1.2.2.1.3.3
```

Scripts

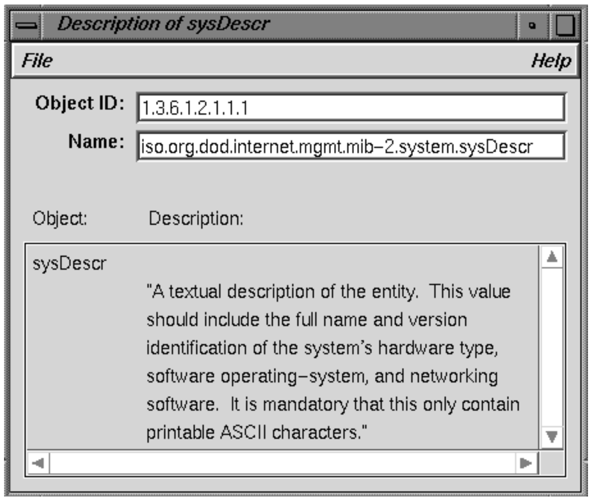
```
snmpGet                         provision:snmpGet
snmpGetPeriod                   provision:snmpGetPeriod
checkProcess                    provision:checkProcess
spaceCheck                      provision:spaceCheck
numberProcesses                 provision:processes
alive                           provision:alive
hostChange                      provision:hostChange
load1                           provision:load1
load5                           provision:load5
load15                          provision:load15
connections                     provision:connections
printQueue                      provision:printQueue
cpu                             provision:cpu
freeKBmemory                    provision:memory
```

Figure 4-3 The Available Variables and Scripts Window

This window lists available MIB variables that can be monitored in the upper half, and all available monitoring and notification scripts in the lower half. You can monitor MIB variables not listed in this window, but they must be specified by their full numeric Object ID (for example, the *sysServices* variable has an Object ID of 1.3.6.1.2.1.1.7.0). You can also monitor any script you have created that is not represented in this window, but it must be specified with its full name. For example, the *snmpGet* script's full name is *provision:snmpGet*.

The MIB variables are described in the MIB file. To see a variable's description, select the MIB browser from the *Add One Graph* window (in *pvgraph*) or select "MIB Browser" from the Variables menu in *pvcontrolpanel*.

Once the browser is up, press the *Variable...* button and enter the name of the variable you wish to have described in the *Name* field. Then, select the "Description" menu option from the Help menu of the *Variable* window. A new window appears, showing the description text. For example, Figure 4-4 shows the description window of the *sysDescr* variable.



Object ID: 1.3.6.1.2.1.1.1

Name: iso.org.dod.internet.mgmt.mib-2.system.sysDescr

Object:	Description:
sysDescr	"A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software. It is mandatory that this only contain printable ASCII characters."

Figure 4-4 Description Window for the sysDescr Variable

This procedure is also described in the section titled "Obtaining Descriptions of Variables" in this chapter.

The Default provision Scripts

The scripts shipped with *provision* are defined as follows:

alive	This script simply sends an ICMP ECHO (*ping*(1M)) request to the named remote system, and returns an error if it fails to get a response within a reasonable time. The arguments to this script are a test interval (in seconds) and a list of hosts to check. The script returns *true* or *false* for each system, and a status message if the script fails to fetch the data.
checkProcess	This script reads the process table from a remote system and checks for the existence of a particular process name. The arguments for this scripts are a test interval (in seconds), a list of hosts, and a process name. The script returns *true* or *false*, and a status message if the process does not exist, or if the script fails to get a response.
connections	This script returns the number of open network connections to a system, and an error if it is above a limit. The arguments are a test interval (in seconds), a list of hosts to check, and an upper bounds.
contextSwitch	This procedure returns the raw number of process switches that have occurred on a remote machine since the last boot, or an error if the script does not receive the information. The argument is a list of hosts.
contextSwitchPeriod	This procedure returns the number of context switches that have occurred on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts, an upper limit, and a lower limit.
cpu	This script returns the average percentage of CPU utilization on a system, or a status message if the number is out of bounds or if the script fails to retrieve the data. The arguments to this script are a test interval (in seconds), a list of hosts, a lower bound, and an upper bound.

hostChanged	This script watches for a change in the availability of a host. The argument to this script is a test interval (in seconds) and a list of hosts to watch. The script returns *true* or *false* for each host, and a status message if the status of a host changes.
ifCollisions	This procedure returns the raw network collisions that have occurred on a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
ifCollisionsPeriod	This procedure returns the raw number of network collisions that have occurred on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts, an upper limit and a lower limit.
ifInErrors	This procedure returns the raw number of input errors that have occurred on a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
ifInErrorsPeriod	This procedure returns the raw number of network read errors that have occurred on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts and an upper limit.
ifInPackets	This procedure returns the raw number of packets that have been received on a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
ifInPacketsPeriod	This procedure returns the raw number of network packets that have been read in on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts, an upper limit, and a lower limit.

ifOutErrors	This procedure returns the raw number of output errors that have occurred on a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
ifOutErrorsPeriod	This procedure returns the raw number of network write errors that have occurred on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts and an upper limit.
ifOutPackets	This procedure returns the raw number of packets that have been sent from a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
ifOutPacketsPeriod	This procedure returns the raw number of network packets that have been written out on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts, an upper limit, and a lower limit.
interrupts	This procedure returns the raw number of interrupts that have been received on a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
interruptsPeriod	This procedure returns the raw number of interrupts that have occurred on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts, an upper limit, and a lower limit.
load1	This procedure returns the current load average of a machine over the previous second, and a status if the load is out of bounds or the script does not receive the data. The arguments are a list of hosts to check, a low boundary, and a high boundary.

load5	This procedure returns the current load average of a machine over the previous 5 seconds, and a status if the load is out of bounds or the script does not receive the data. The arguments are a list of hosts to check, a low boundary, and a high boundary.
load15	This procedure returns the current load average of a machine over the previous 15 seconds, and a status if the load is out of bounds or the script does not receive the data. The arguments are a list of hosts to check, a low boundary, and a high boundary.
memory	This script returns the amount of free memory in kilobytes, and a status message if the number is out of bounds or the script fails to get the data. The arguments are a test interval (in seconds), a list of hosts, a lower bound, and an upper bound.
nfsChanged	This script performs essentially the same function as *nfsCheck*, but returns a status message only when the state of a remote server changes, that is, if a host that was formerly responding correctly ceases, or a host that was not responding begins to respond. The argument is a list of hosts.
nfsCheck	This script checks NFS server remote systems for correct response. The script returns a true or false value for each host. A true value indicates a correct response, and a false value indicates that the NFS server is not functioning correctly. A status message is also displayed if the server is not responding correctly or if the script fails to get the information. The argument is a list of hosts.
pageIn	This procedure returns the raw number of pages that have been paged in on a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
pageInPeriod	This procedure returns the raw number of pages that have been paged in on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts, an upper limit and a lower limit.

pageOut	This procedure returns the raw number of pages that have been paged out on a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
pageOutPeriod	This procedure returns the raw number of pages that have been paged out on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts, an upper limit, and a lower limit.
printQueue	This script checks the status of a remote printer queue. The arguments to this script are a test interval (in seconds), a list of hosts, and the remote printer name. The script returns *true* or *false* for the named printer, and a status message if the printer is down, or if the script fails to get the information.
processChanged	This script reads the process table from a system and checks for the existence of the specified name. The script returns a data field of true or false, and a status message if the process used to exist but has exited, or if the process did not exist before but has now started, or if the script fails to retrieve the information. The arguments for this script are a list of hosts, and a process name.
processes	This script checks the number of processes on a remote system and notifies you if the number is not in the specified bounds, and a status message if the number is out of script bounds, or if the script fails to get the data. The arguments for this script are a test interval (in seconds), a list of hosts, a low bound, and a high bound.
random	This script invokes a random number generator. The arguments are a test interval (in seconds), a list of hosts, a lower bound, and an upper bound. This script is used for testing purposes or demonstration.
snmpGet	This is a very simple script to query a system (or a collection of systems) for an *snmp* variable and return the value of the variable. The arguments for this script are a test interval (in seconds), a list of hosts, and a list of variables to be queried.

snmpGetPeriod	This is a very simple routine to query a system (or a collection of systems) for an *snmp* variable and return the change in it since the last query. The arguments for this script are a list of hosts and a list of variables.
spaceCheck	This script checks the available space on a given filesystem, and verifies that it is between the specified bounds. The arguments for this script are a test interval (in seconds), a list of hosts, a filesystem name, a low bound, and a high bound. The script returns a data field of the available space, and a status message if the check fails the bounds check or the script cannot get the data.
swap	This script returns the amount of free swap space on a host. The arguments are a test interval (in seconds), a list of hosts, a lower bound, and an upper bound. The script returns the amount of free space in kilobytes, and a status message if the number is out of bounds or the script fails to get the data.
swapIn	This procedure returns the raw number of processes that have been swapped in on a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
swapInPeriod	This procedure returns the raw number of pages that have been swapped in on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts, an upper limit, and a lower limit.
swapOut	This procedure returns the raw number of processes that have been swapped out on a remote machine, or an error if the script does not receive the information. The argument is a list of hosts.
swapOutPeriod	This procedure returns the raw number of pages that have been swapped out on a remote machine since the last check, or an error if the script does not receive the information. The arguments are a list of hosts, an upper limit, and a lower limit.

Adding Custom Scripts to provision

The *provision* application retrieves data from remote machines through commonly used protocols including *rstat* and SNMP. The *provisiond* daemon has an embedded sgitcl interpreter and uses sgitcl scripts to retrieve remote information.

When a request is made for a new sgitcl script from *pvgraph* or *pvcontrolpanel*, the *provisiond* daemon creates a new, private copy of the sgitcl interpreter. The daemon then calls a predefined script called *provision:wrapper* to gather the script information and return it. If the requested script is a valid SNMP Object ID, the routine *provision:snmpGet* is called to do the retrieval. If the script is a custom file you have created, *provision:wrapper* executes the file to retrieve the script data. Finally, the wrapper calls the script as an internal sgitcl routine which can be in any sgitcl tlib library.

You must create a file with a name ending in *.tlib* in the */usr/IRIXpro/lib* directory to hold your sgitcl script in order for *provision:wrapper* to locate the new script and call it as an sgitcl routine.

Your scripts are not required to be written in sgitcl. If your new script is not an sgitcl script, simply place the full pathname of the executable program or script in the */usr/IRIXpro/provision/scriptDefs* file.

A description of each script must be placed in the file */usr/IRIXpro/provision/scriptDefs*. This description is used to determine the type of any arguments, and the type of the return data from the script. If a script is customized or a new script is added, then this file must be updated. A description of each new MIB variable must be placed in the file */usr/IRIXpro/provision/varDefs*.

Custom Script Reply Format

All custom scripts must report their data back in a specific format. The format is that of an sgitcl list of lists. There is a list for each host containing three elements:

- the hostname
- the data
- a status string

The hostname and status string are optional. The status message is a script-generated error message that, if it exists, is sent to the selected *provision* notifier.

Custom Script Argument List Format

All scripts are called with a command line of the format:

```
host-list [argument]. . .
```

The host list is a space-separated list of hostnames or addresses. All arguments are defined in the individual script. Common arguments are low and high bounds on the data. When data comes in that is out of bounds, a status message is returned.

Custom Script sgitcl Routine Locations

All of the scripts provided with *provision* are found in the *tlib* file */usr/IRIXpro/lib/provision.tlib*. If a provided script does not meet your needs then the script file can be copied and edited to create a custom script. When you have edited the new script, restart the *provisiond* daemon. When the daemon restarts, all tlib files are searched for unknown procedures, so custom scripts should be kept in a custom script tlib file, which can include routines from any other tlib library files as well.

Custom Script sgitcl Extensions

The sgitcl programming language provides several extensions to fetch information. These include *rstat*, SNMP, and the Silicon Graphics object management system. There are sgitcl help pages for each call in these three extensions and a reference page on each library.

The pvcontrolpanel Hosts Area

This area of the *pvcontrolpanel* main window lists all hosts and collections currently being monitored by or otherwise known to your *provision* session. An icon appears with each host’s name. You can double-click a host icon and a dialogue window appears showing the items currently being monitored and any alarms received for the host or collection.

When an alarm comes in on a monitored host or collection, the object icon turns red to show you that an alarm has been received. When you double click the icon to view the alarm, the icon turns orange to show the alarm has been noted. If you then click the *Clear Alarms* button on the dialogue window, the icon returns to its default color.

The pvcontrolpanel Items to Monitor Area

All the items currently being monitored on any host are listed in this area. You can select which item is displayed in the Script Configuration Area and start and stop any item by clicking the provided buttons for each item being monitored.

The pvcontrolpanel Script Configuration Area

The script configuration section of the *pvcontrolpanel* window is functionally identical to the script configuration window used with *pvgraph*. This window is discussed in the section titled “The New Graph Window.” Some key differences are:

- A button is provided for you to specify logging for the script or variable.
- A button and command window are provided for you to specify notification and a notification command.
- A regulation time selector is provided if you choose notification. This controls the frequency with which you will be notified if the specified limit is reached. For example, if you have set a notification alarm if the free disk space is under 10000 blocks and you have specified a monitoring interval of 30 seconds, you can specify a regulation time of 10 minutes and you will only be notified at that time interval, rather than every 30 seconds.

Creating a Log File With pvcontrolpanel

To create a log file with *pvcontrolpanel* click the button labeled *log* when you configure a variable or script. The name of the log file used appears at the top of the Items to Monitor section.

When you wish to review the log you must select the Close Log File menu option from the Items menu or stop the actual logging . In order to stop all logging, close the log file, and not open a new log file, you must stop monitoring all items currently configured, delete all the items currently configured, and select the Close Log File menu option.

Alternately, you can select Close Log File from the Items menu, and the log file for the selected item will be closed and a new one opened. You can then review the log that was closed. This method is recommended.

Note: If you change the host or parameter being logged with a *modify* command, the log file will not be restarted, nor will it register this change in any way. Thus, when the log is viewed it will be presented as if the parameters had not changed, and any information collected after the change is attributed to the initial configuration.

Using pvcontrol

The *provision* package offers a text-based interface that replicates the functions of the *pvcontrolpanel* graphical logging and notification tool. The text-based interface to *provision* can be run on any shell window, X-terminal, or character-based terminal. As **root** or as a member of the user group **irixpro**, enter the command:

```
pvcontrol
```

When you enter the command, you see the following prompt:

```
pvcontrol>
```

To see a list of commands, type **pvhelp** (or simply **h**) and press **<Enter>** at the *pvcontrol* prompt. You see the following list:

```
Provision commands:

list [log | notify]     - list currently monitored items
listAlarms hostName     - list alarms reported for specified
                          host
clearAlarms hostName    - clear alarms for specified host
getCurrentValue hostName scriptName args
                        - get the value of the specified
                          script or variable
```

```
add hostName scriptName interval notifyCommand
regulationTime notify|nonotify log|nolog args
                        - add item to monitor
modify itemID hostName scriptName interval notifyCommand
regulationTime notify|nonotify log|nolog args
                        - modify an item that is being
                          monitored
delete all              - delete all items
delete itemID           - delete item with specified itemID

start all               - start monitoring all items
stop all                - stop monitoring all items
start itemID            - start monitoring specified item
stop itemID             - stop monitoring specified item

showAvail               - list all available variables and
                          scripts
browser                 - start the snmp browser
closeLog                - close the log file
logStatus               - check the status of the log file
readConfig fileName     - read specified configuration file
saveConfig              - save configuration file
saveConfigAs fileName   - save configuration file to new name
pvhelp                  - display this help
quit                    - quit

pvcontrol>
```

The commands have the following meanings:

`list [log | notify]`
: This command prints a list of items currently being logged or monitored for notification along with the itemID numbers. The itemID number is provided to allow more convenient manipulation of each specific item.

`listAlarms hostName`
: This command directs *pvcontrol* to list any alarms reported for the specified host.

`clearAlarms hostName`
: This command directs *pvcontrol* to clear all received alarms for the specified host.

```
getCurrentValue hostName scriptName args
```

This command directs *pvcontrol* to get the current value of the specified script or variable.

```
add host scriptName interval notifyCommand
regulationTime notify|nonotify log|nolog args
```

This command adds a new item to the list. You must supply an entry for each argument shown. When your new item is accepted, the itemID is displayed along with the parameters you used. For example, the command

`add myhost interrupts 1 “mail dhhill” 1 nonotify log`

produces this response:

```
4 off myhost interrupts 1 off on - mail dhhill 1
```

The itemID in the displayed response is 4.

In this command and in the *modify* command:

- The interval is the frequency with which the script is run or the variable is checked.
- The Notify Command is the shell command to run to notify you if the limit (set on a per-script basis in the arguments) is reached.
- The regulation time specifies how frequently you are notified if the script is chronically past the limits you have specified.
- The notify and log switches specify notification and logging.
- The arguments required vary based on the script or variable you select.

Note that the script is not actually being monitored or logged until you enter the command *start all* or *start itemID*.

```
modify itemID host scriptName interval notifyCommand
regulationTime notify|nonotify log|nolog args
```

This command modifies an item being monitored. You provide the itemID of the item, and the new values for the item. For example, to change the item used above, you might enter the command:

```
modify 4 myhost connections 5 “mail dhhill” 1
nonotify nolog 50
```

With this command you have changed the script to *connections*, the interval to 5, stopped logging the results, and changed the argument to 50. The new parameters of the item are displayed for you.

In this command and in the *add* command:

- The interval is the frequency with which the script is run or the variable is checked.
- The Notify Command is the shell command to run to notify you if the limit (set on a per-script basis in the arguments) is reached.
- The regulation time specifies how frequently you are notified if the script is chronically past the limits you have specified.
- The notify and log switches specify notification and logging.
- The arguments required vary based on the script or variable you select.

Using flags to the modify command, you can modify individual parameters of an item. The following flags are recognized:

-h [**hostname**]	– indicates new host name
-r [**regulation time**]	– specifies a regulation time
-s [**scriptname**]	– indicates new script
-i [**interval**]	– indicates new interval
-n [**on** | **off**]	– turns notification on or off
-c [**notify command**]	– specifies a notification command
-l [**on** | **off**]	– turns logging on or off

`-a` [**`arguments`**]	– indicates new arguments

Use the following command syntax with flags:

`modify itemID flag flag`

`delete all`	This command deletes all items currently configured.
`delete itemID`	This command deletes only the item with the specified itemID.
`start all`	This command starts monitoring all currently configured items.
`stop all`	This command stops monitoring all currently monitored items.
`start itemID`	This command starts monitoring the specified item.
`stop itemID`	This command stops monitoring the specified item.
`showAvail`	This command lists all available variables and scripts in text. The list of variables is quite long, and definitions of the variables can be obtained only through the SNMP Browser on a graphics system. Descriptions of the available scripts are in the section titled "The Available Variable and Script Window."
`browser`	This command starts the SNMP browser. The browser is a graphical-only tool, and so cannot display on a non-graphics system. The browser is described in the section titled "Obtaining Descriptions of Variables."
`closeLog`	This command directs *pvcontrol* to close the log file. A new log file is opened immediately.
`logStatus`	This command checks and reports the status of the log file.
`readConfig fileName`	This command directs *pvcontrol* to read the specified configuration file and use the monitoring and logging settings found in it.
`saveConfig`	This command saves the current configuration in the default configuration file.

`saveConfigAs fileName`
: This command saves the current configuration to a new configuration file.

`help`
: This command displays the list of available commands.

`quit`
: This command quits *pvcontrol*.

The commands available through *pvcontrol* are substantially similar to those available through the graphical *pvcontrolpanel*, and the description of that utility provides further helpful information.

Creating a Log File with pvcontrol

To create a log file with *pvcontrol* you must select logging as a command line option when you use the *add* or *modify* commands to select a variable or script. The name of the log file used is displayed in the following manner:

```
The log file is:/usr/IRIXpro/provision/Logs/0.950201-22:34:26
```

When you wish to review the log you can stop all monitoring action by deleting all items and entering the *closeLog* command at the *pvcontrol* prompt to stop all monitoring and logging, or you can simply enter the *closeLog* command and the current log file will be closed and a new one opened.

Note: If you change the host or parameter being logged with a *modify* command, the log file will not be restarted, nor will it register this change in any way. Thus, when the log is viewed it will be presented as if the parameters had not changed, and any information collected after the change is attributed to the initial configuration.

Using pvgraph

The second tool available directly from the *provision* window is *pvgraph*. This tool allows you to select command scripts and graph the values of certain variables and system statistics in a window. When you first bring up *pvgraph*, you see the following window (shown in Figure 4-5):

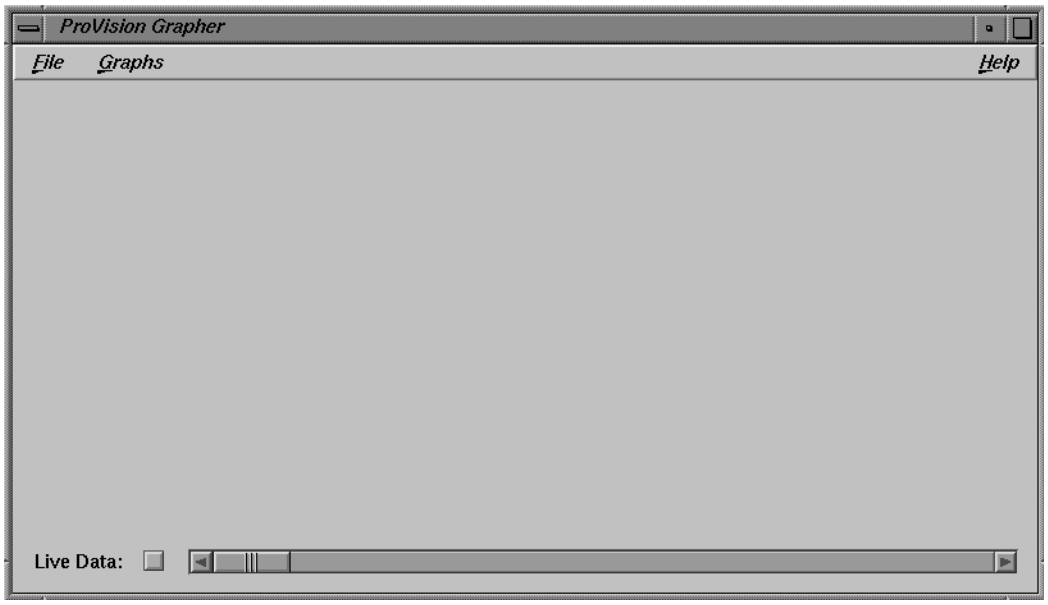


Figure 4-5 The pvgraph Window

The *pvgraph* window is blank when it comes up on your screen, and you create graphs by selecting options from the menu bar. The menu bar has three menus: File, Graphs, and Help. The options available in these menus are listed below.

The pvgraph File Menu

The File menu has the following choices:

Read Config...	This option reads a configuration file that specifies a set of graphs to run. You can also drag an icon representing a previously stored config file from the directory view onto the *pvgraph* icon to start *pvgraph* with that configuration.
Save Config	This option saves the current graphing configuration to a file.

Save Config As...
This option saves the current graphing configuration to a new filename.

Quit
Quits *pvgraph* and ends all graphing.

The pvgraph Graphs Menu

The Graphs menu has the following choices:

Add A Graph
Use this choice to add a new graph. This choice brings up the New Graph window, described in the section titled “The New Graph Window.”

Modify Selected Graph
Use this choice to change an existing graph. This choice brings up the Edit Selected Graph window with the parameters of the selected graph displayed in the fields for modification.

Delete Selected Graph
This choice deletes the selected graph.

Change Style of Selected Graph
This choice brings up the Graph Styles window. This window is described completely in the section titled “The Graph Style Window.”

Change Parameters of All Graphs
This choice brings up the Graph Parameters window. This window is described completely in the section titled “The Graph Parameters Window.”

Show Alarms
This choice shows all received *provision* alarms for the graphed items. See “Working With Graph Alarms” for more information.

Clear Alarms
This choice clears all received *provision* notification alarms. See “Working With Graph Alarms” for more information.

Start Selected Graph
This choice starts a previously stopped graph.

Stop Selected Graph
: This choice stops a selected graph.

Start All Graphs
: This choice starts all previously stopped graphs.

Stop All Graphs
: This choice stops all graphing.

The pvgraph Help Menu

The Help menu offers online help with *pvgraph*.

The New Graph Window

When you select the menu choice to add a graph to your *pvgraph* window, you see the new window shown in Figure 4-6.

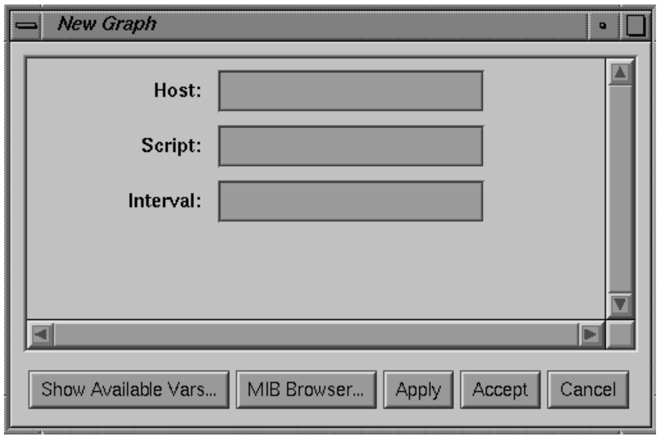


Figure 4-6 The New Graph Window

This window has several fields for you to fill in the parameters of the graph you wish to make. There are also other fields that may appear as you enter information. Certain scripts require more parameters than others, and if you enter the name of such a script in the Script field, additional fields appear below the basic fields. The fields require the following kinds of information:

Host Field	This field takes the name of a host. The host must be connected with the local system by the network, and the host must be running the *snmpd* daemon.
Script Field	This field takes a script or variable name. If you do not know the name of the script or variable you wish to use, press the *Show Available Vars* button at the bottom of the window and the Available Variable and Script window will appear. This window is described in the section titled "The Available Variable and Script Window." All distributed scripts are described in that section of this chapter.

Interval Field
: This field is where you specify the time interval (in seconds) at which the script will run and the results will be displayed. For example, if you enter **1**, the script will run and the graph will be updated every second.

Arguments Fields
: These fields are where any necessary arguments to the script are specified. When you enter a variable or script, appropriate fields appear for each needed argument. If the script or variable is not known to *provision*, a field titled *arguments* appears to receive any arguments required. To make a new script or variable known to *provision*, an entry must be placed in the */usr/IRIXpro/provision/scriptDefs* or */usr/IRIXpro/provision/varDefs* file.

At the bottom of the New Graph window, there are five buttons, labeled *Show Available Vars, MIB Browser, Apply, Accept,* and *Cancel*. The *Show Available Vars* button brings up the Available Variable and Script window, described in the section titled "The Available Variable and Script Window." The SNMP Browser button brings up the Browser, described in the section titled "The MIB Browser." Use the *Apply* button to add your graph and leave the New Graph window on the screen, or the *Accept* button to add the new graph and remove the New Graph window. The *Cancel* button removes the New Graph window without applying your changes.

If you add additional graphs, the window is subdivided for each graph. When you have more graphs than can fit on the window, you must enlarge the window to accomodate the new graphs.

Working With Graphs

When you have applied your graph to the *pvgraph* main window, the center of the window looks something like that shown in Figure 4-7.

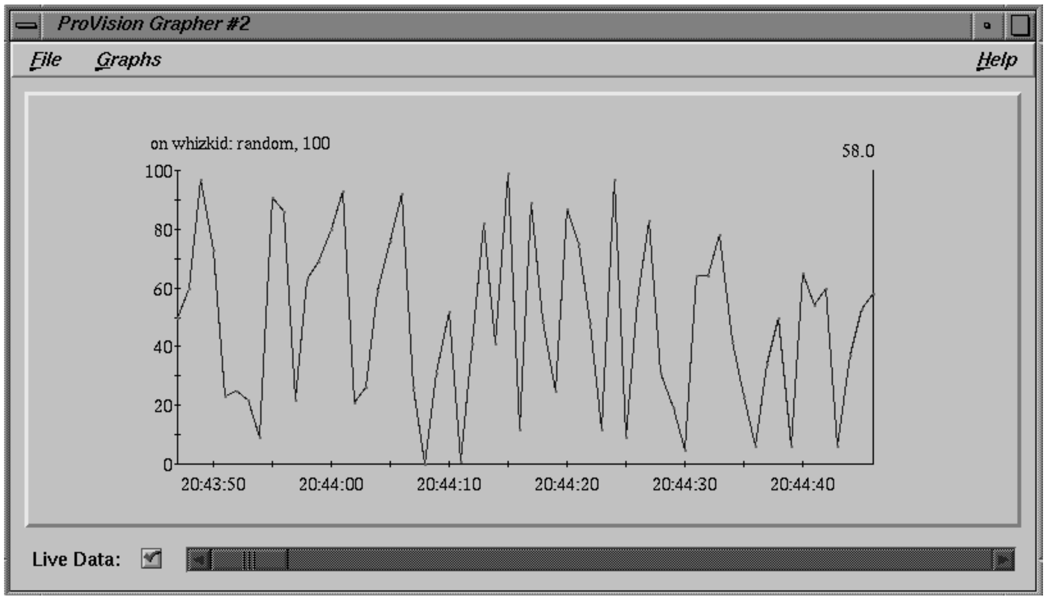


Figure 4-7 The pvgraph Window With One Graph

Note that there is a check box and a slider present at the bottom of the *pvgraph* window. When you begin your graph, the check box has a check mark, indicating that the graph being made is using data as it is collected in real time. The slider is grayed-out and inoperable. At any time you can click on this check box and the entire history of the graph is made available to you. The slider bar becomes active and you can use it to review your graph. When you wish to return to live graphing, simply click the check box again and the graph is updated. No data is lost during your review operation.

Working With Graph Alarms

When the script results or variable values being graphed exceed the low and high limits you specified when you added or modified the graph, an alarm is set off for you. This alarm is a visual cue to check the item being graphed. When the value of the script or variable has gone out of bounds, the graph turns red, as shown in Figure 4-8.

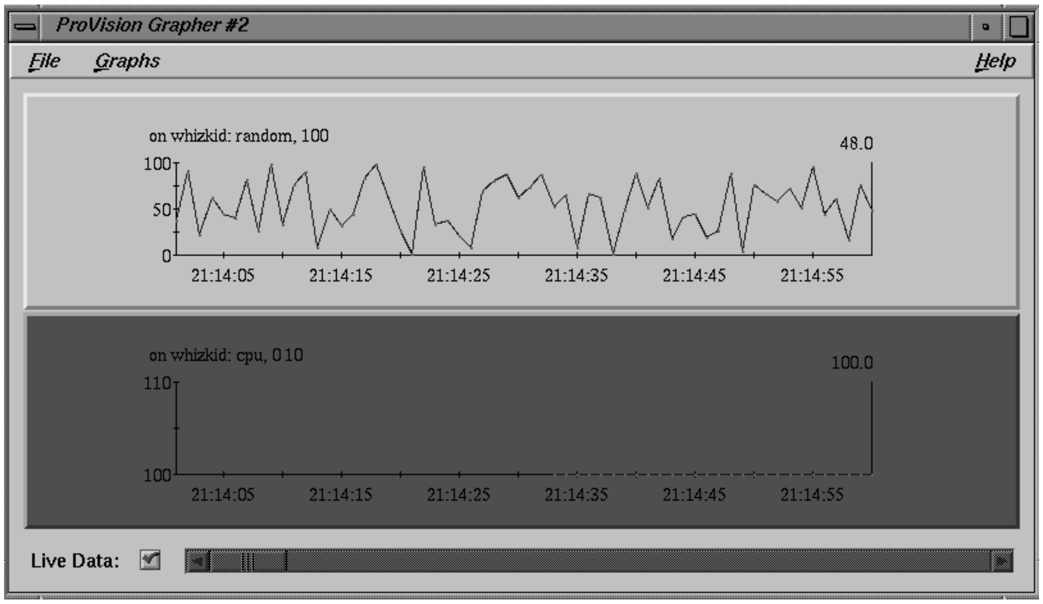


Figure 4-8 A Graph With an Alarm Showing

To clear alarms, select the Clear Alarms menu option from the Graphs menu. If you select the Show Alarms option, a window appears with a log of all alarms received since the last Clear Alarms command, or since the beginning of the *pvgraph* session.

Selecting a Graph

You can select a graph for further operations by placing the mouse cursor in the window section of the graph and clicking the left mouse button. The background of the selected graph turns yellow. Only one graph may be selected at a time. You may perform the following operations from the Graphs menu on selected graphs:

Modify Selected Graph
: This choice brings up the Modify Selected Graph window with the parameters of the selected graph displayed in the fields for modification. This window is identical to the New Graph Window except for the title.

Delete Selected Graph
: This choice deletes the selected graph.

Change Style of Selected Graph
: This choice brings up the Graph Style window. This window is described completely in the section titled "The Graph Style Window."

Stop Selected Graph
: This choice stops the selected graph.

Start Selected Graph
: This choice starts the selected graph.

The Graph Parameters Window

When you select the "Change Parameters of All Graphs" menu item from the *pvgraph* Graphs menu, you see the new window shown in Figure 4-9.

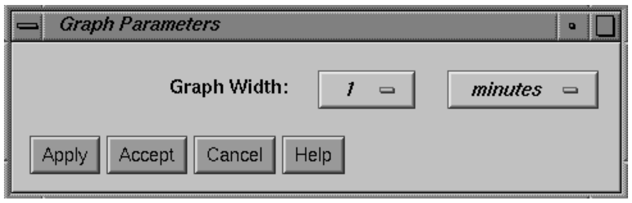


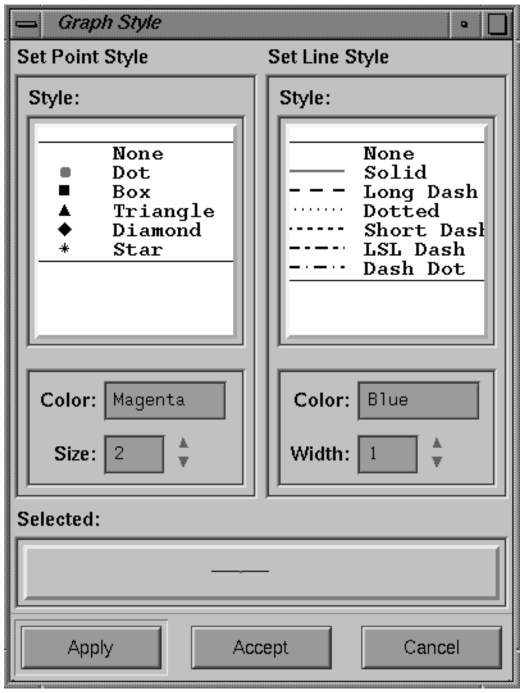
Figure 4-9 The Graph Parameters Window

What you are changing is the period of graph-time that is displayed in the window at any given moment. The parameters you can change are the graph width value and the time unit. The width value is simply the number of increments of the selected time unit. In the above example, the width value is 1 and the time unit is *minutes* for a width of 1 minute. You may select *1, 2, 5, 10, 20,* or *30* for the width value, and one of *seconds, minutes, hours, days,* or *weeks* for the time unit.

Once you have made your selections, you may press the *Apply* button to apply the change and leave the Graph Parameters window on the screen, or the *Accept* button to apply the changes and remove the Graph Parameters window. The *Cancel* button removes the window without applying your changes. The *Help* button invokes *provision*'s online help utility.

The Graph Style Window

If you select the "Change Style of Selected Graph" menu item from the Graphs menu in *pvgraph*, you see a new window on your screen, as shown in Figure 4-10.



Color: Magenta
Size: 2
Color: Blue
Width: 1

Figure 4-10 The Graph Style Window

This window allows you to select and modify the way the selected graph is presented in *pvgraph*. When the polling interval arrives on a graph, the new value of a variable (or the value of the output of the script being graphed) is placed on the graph as a point, and a line is drawn between the new point and the previous point. You may select the shape of the point marker, its size and color, and the style, width, and color of the connecting line. Click the style of marker and line you prefer. Any valid X color or value may be named in the Color field, and you can use the arrow buttons to increase or decrease the size of the line or marker.

When you have made your selection, press the *Apply* button to apply the new format to your graph, or the *Cancel* button to discard your unapplied changes.

The provision Configuration File

At any time during your *pvcontrolpanel* or *pvgraph* session, you can save the current graphing and / or monitoring selections in a configuration file. The options are in the File menu in both utilities:

```
Read Config...
Save Config
Save Config as...
```

When you first save your current state, use the “Save Config as...” option. When you select this option, you see a file selection window for your current working directory, such as that shown in Figure 4-11.

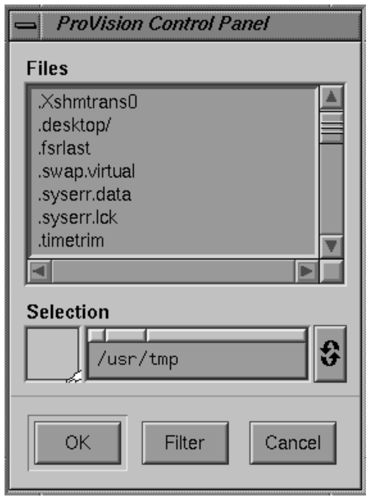


Figure 4-11 A File Selection Window

Select a new filename for your configuration file and click the *OK* button when you are satisfied with your selection. Your current state is now saved. If you wish to save your current selections again later, using the “Save Config” option from the Files menu of *pvcontrolpanel* or *pvgraph* will automatically bring up the file selection window with the most recently used config file specified. You can, however, change the name so as not to overwrite the existing config file. Using config files, you can create templates for commonly used monitoring and graphing scenarios. For example, you can have preset configuration files to monitor all systems’ network traffic or the swap rates on your servers.

Each *provision* configuration file is written in clear text and looks similar to the following example:

```
# provision config file written Mon Feb  6 14:29:52 PST 1995
by pvcontrolpanel
off on off  random provision:random  wookie 1 {}  -1 0 {200}
off on off  ifInOctets_1 1.3.6.1.2.1.2.2.1.10.1  wookie 1
{}  10 0 {2} {3}
off on off  ifOutOctets_1 1.3.6.1.2.1.2.2.1.16.1  wookie 1
{}  10 0 {2} {3}
off on off  random provision:random  irixpro 5 {}  600 0
{100}
```

Using pvgraph to View a Log

You can view log files created with *pvcontrol* or *pvcontrolpanel* using *pvgraph*. For more information on log files and how they are created, see “Creating a Log File With pvcontrolpanel” on page 101 or “Creating a Log File with pvcontrol” on page 107 in this chapter. A log file is simply a file containing a series of values for a script or variable accumulated over a period of time. The *provision* application stores the log files in the log directory */usr/IRIXpro/provision/Logs*. A log file is actually contained in two filenames in that directory. For example, a log might be placed in filenames similar to *0.950201-21:18:33.Desc*, and *0.950201-21:18:33.Data*. Filenames ending in *.Desc* contain information about what was logged, and filenames ending in *.Data* contain the actual log information.

To view a log as a graph, use the command syntax

`pvgraph` *`filename`*

to invoke *pvgraph* in log file mode. The *filename* argument can be any of the three filenames that refer to the desired log. For example, using the example filenames as shown above, you could invoke *pvgraph* in these ways:

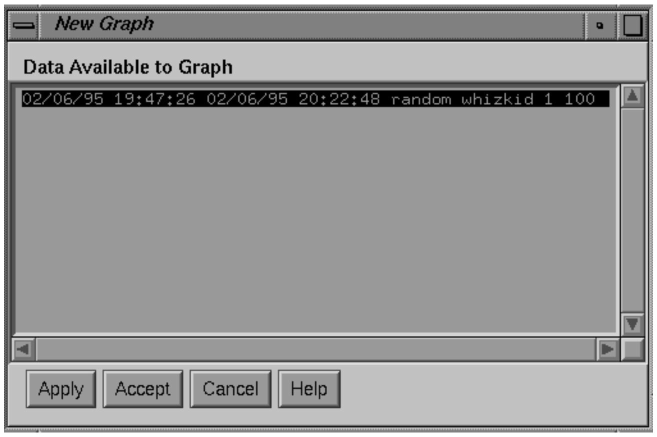
`pvgraph 0.950201-21:18:33`

`pvgraph 0.950201-21:18:33.Desc`

`pvgraph 0.950201-21:18:33.Data`

Each of the above commands results in the same action by *pvgraph*. By default, *pvgraph* looks for the given filename in */usr/IRIXpro/provision/Logs*, but you can specify any log file in any directory by issuing the pathname of the file on the command line.

When you invoke *pvgraph* with a log file name as a command line argument, *pvgraph* does not connect with the *provisiond* daemon as usual. Instead, the named log file is loaded. The log is not displayed as a graph, though, until you use the Add A Graph menu option from the Graphs menu. When you use this command, you see a different New Graph window, similar to Figure 4-12.



```
02/06/95 19:47:26 02/06/95 20:22:48 random whizkid 1 100
```

Figure 4-12 The New Graph Window with Log File Information

When you view a log file, it does not scroll by as usual; you must use the slider at the bottom of the window to move forward and back throughout the log.

If you are invoking *pvgraph* from the desktop or directory view rather than as a shell command, you can drag the file icon for the log you wish to view and drop it on the *pvgraph* icon and *pvgraph* will come up with the log loaded.

The MIB Browser

The MIB Browser is available through either *pvcontrolpanel* or *pvgraph*. The browser enables you to select a node on your network and view and change the contents of one or more Management Information Bases (MIBs) for that node. Browser communicates with a node that you select using Simple Network Management Protocol (SNMP). The node can be a workstation, router, bridge, hub, or gateway—any device that has an IP address and implements the SNMP protocol and agent.

Browser enables you to walk the tree of information represented by the MIBs and the SNMP Containment Tree, and to get the values of MIB variables. While you are using Browser, you can save the variable values that you receive to a file. You can set MIB variables if the SNMP and MIB implementations on the node you are browsing allow it and the community string you provide authorizes it.

Several MIB specifications are provided with Browser. The supplied MIB specifications are:

- `hp-ux_sgi`
- `mib-2`
- `rmon`

Browser is designed to be used by network managers experienced in managing various devices on the network. This section assumes that you are familiar with SNMP management terminology and technology, especially the MIBs for different devices. If you are not familiar with this terminology, the section titled “SNMP Management Glossary” on page 142 defines the basic terms.

This section explains how to

- start Browser
- use the Browser File menu
- use the Browser main window to specify the node you want to browse and begin navigating the SNMP Containment Tree
- navigate the SNMP Containment Tree to view subtrees, tables, and variables
- get descriptions of variables
- get and set the values of variables

In addition, an example of using Browser is provided. For complete information on Browser command line options, see the *browser*(1M) reference page.

Note: To enable Browser to get and set MIB variables on a Silicon Graphics workstation, that workstation must be running the SNMP daemon *snmpd*(1M), and your Display Station must be authorized in the file */usr/etc/snmpd.auth* on that workstation. See “Authorizing Browsing” for details.

Caution: With proper authorization, Browser lets you change some MIB variable values on devices you browse. Because MIB variable values can be critical to the operation of a device and your network, do not change values unless you understand the effects of your changes.

SNMP Agents

Browser uses Simple Network Management Protocol (SNMP) agents to obtain information. The SNMP agent for Silicon Graphics workstations is *snmpd*(1M). Vendor-specific SNMP agents are used to obtain information about other types of nodes (see Figure 4-13). For more information on SNMP see the section titled “SNMP Management Glossary.” For information on enabling SNMP agents, see “Enabling SNMP Agents.”

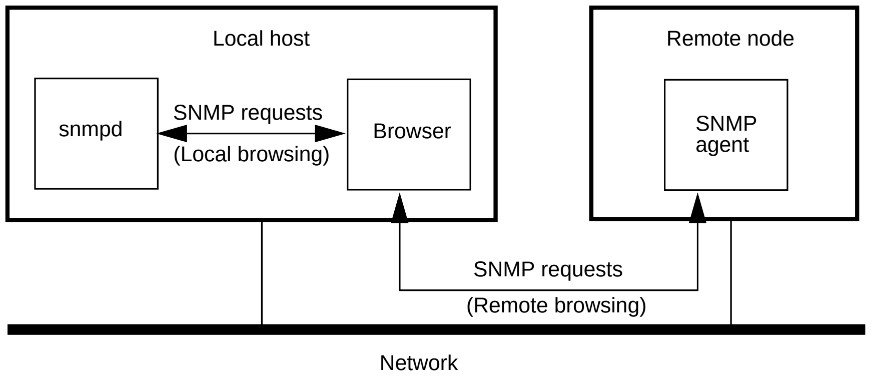


Figure 4-13 SNMP Agents and Browser

Enabling SNMP Agents

Browser must communicate with the SNMP agent on each node you wish to browse. SNMP agents and the procedures for enabling them are vendor-specific. The procedure for enabling the SNMP agent on Silicon Graphics workstations is described below. For other types of nodes, contact the system administrator for that node for help in enabling SNMP on that node.

The Silicon Graphics SNMP agent is *snmpd*(1M). The SNMP agent software is distributed with IRIX, and the hp-ux_sgi MIB and agent are distributed with IRIXpro. To configure a workstation so that *snmpd* is started automatically when the system is rebooted, copy the snmpd executable file to the workstation, and enter this command on the workstation as *root*:

```
chkconfig -f snmpd on
```

To see if the daemon is already running, enter this command:

```
ps -e | grep snmpd
```

If there is no output from this command, *snmpd* is not running. Enter this command as *root* to start *snmpd*:

snmpd

Authorizing Browsing

No special authorization other than a valid community string is required to browse on nodes other than Silicon Graphics workstations. (See “Browser Main Window” and “SNMP Management Glossary” for more information about community strings.)

If you want to get and set MIB variables on a Silicon Graphics workstation using Browser, you must perform several setup steps in addition to providing a valid community string while using Browser: confirm that the workstation you are browsing has SNMP agent software running; start it if necessary (see “Enabling SNMP Agents,” in this chapter); and authorize your Display Station to browse on that workstation.

To be authorized to browse a Silicon Graphics workstation, the Display Station’s host name must be specified in the file */usr/etc/snmpd.auth* on the workstation you are browsing. You must be superuser (*root*) to read or write */usr/etc/snmpd.auth*. For security reasons, the owner and permissions of this file should not be changed.

As an example, suppose that you want to browse a Silicon Graphics workstation named `tahoe`. Your workstation’s name is `sequoia`. First, confirm that *snmpd* is running on `tahoe`:

```
rsh guest@tahoe 'ps -e | grep snmpd'
```

Assuming that it is running, log onto `tahoe` as superuser and add this line to */usr/etc/snmpd.auth*:

```
accept sequoia:*
```

This line authorizes anyone using your workstation to browse the workstation `tahoe` when they give any community string. These users can perform both `get` and `set` operations.

By default, */usr/etc/snmpd.auth* contains this authorization line:

```
accept  *:public/get
```

This line authorizes any user from any host who provides the community `public` to get variable values for this workstation.

See the *snmpd*(1M) reference page and the file for more information about the syntax used in this file.

Starting Browser

To start Browser, click the *browser* button from a *provision* window.

The Browser main window appears. An example is shown in Figure 4-14.

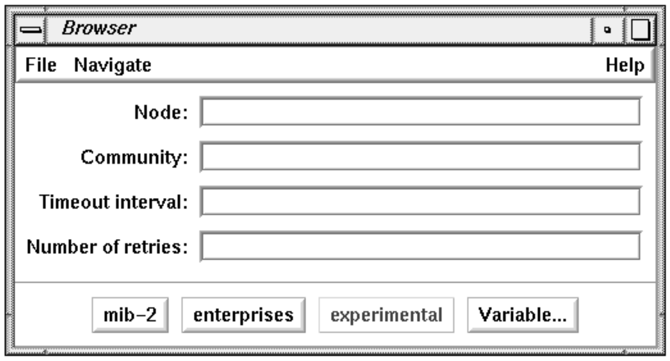
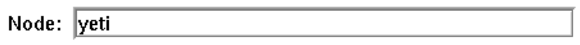


Figure 4-14 Browser Main Window

Browser Main Window

The entry fields in the Browser main window enable you to specify the node you wish to browse, a community string, a time-out value for accessing the SNMP agent on the node, and the number of retries to make when attempting to access a remote node.

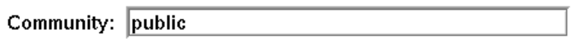
When you invoke Browser, the entry field shown in Figure 4-15 contains the name of your workstation. You can replace it with the name or address of the node you want to browse. A blank entry field is the same as specifying the name of your workstation.



Node: yeti

Figure 4-15 Node Entry Field

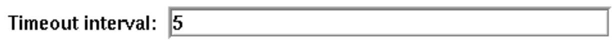
The Community entry field shown in Figure 4-16 contains the community string that is to be used in the SNMP packets sent to the node. The community string is an authorization password for the node you browse on. On Silicon Graphics workstations, valid community strings and other authorization information is specified in the file */usr/etc/snmpd.auth*. The default community string on Silicon Graphics workstations is "public". For other types of nodes, such as routers and bridges, the community string is specified for each device by a system administrator. A valid community string must be supplied in order to use Browser to view MIB information.



Community: public

Figure 4-16 Community Entry Field

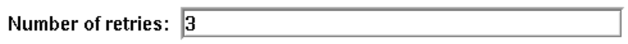
If Browser doesn't receive a reply from the SNMP agent on the specified node within the time-out value, it will try again. The default time-out value, shown in Figure 4-17, is 5 seconds.



Timeout interval: 5

Figure 4-17 Timeout Interval Entry Field

The Number of retries entry field, shown in Figure 4-18, specifies the number of retries when there has been no reply from the node. The default is 3.



Number of retries: 3

Figure 4-18 Number of Retries Entry Field

The *mib-2*, *enterprises*, and *experimental* buttons in the Browser main window, shown in Figure 4-19, provide quick ways to specify what you want to browse: the `mib-2` MIB, or the `enterprises` or `experimental` nodes in the

SNMP Containment Tree, respectively. When you click these buttons, a Subtree window appears. Subtree windows and Table windows are described in the next section. These buttons are grayed out if no MIB specifications in that portion of the SNMP Containment Tree are available to Browser. The term ‘‘grayed out’’ means that the button title is gray at that moment, rather than black. This indicates that the function or service represented by that button is unavailable.

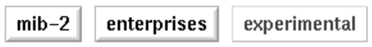


Figure 4-19 mib-2, enterprises, and experimental Buttons

When you click the *Variable...* button, shown in Figure 4-20, a Variable window appears. This window is used to get and set the values of specific MIB variables. It is explained in detail in “Obtaining and Setting Values Using the Variable Window” in this chapter.

Figure 4-20 Variable... Button

Browser Subtree and Table Windows

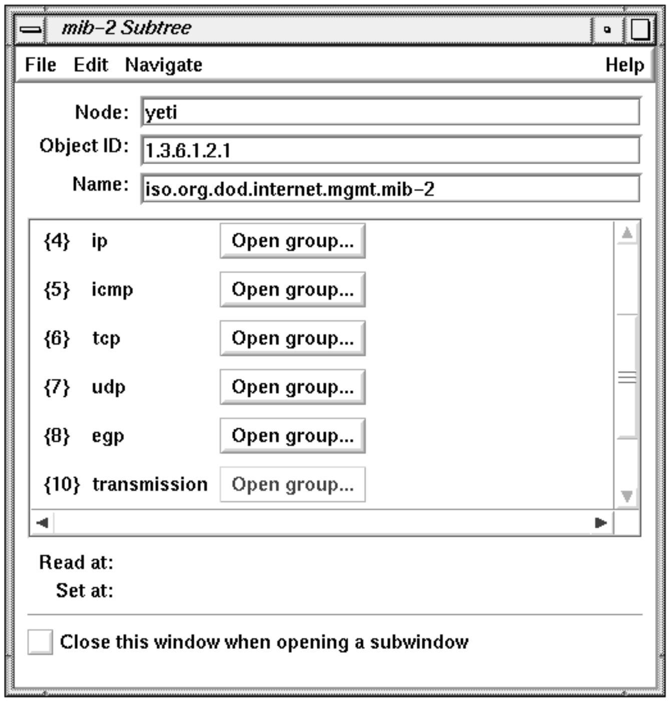
To display MIB information, Browser uses two types of windows: Subtree windows and Table windows. For every nonleaf node in the SNMP Containment Tree, Browser displays one of these types of windows:

- a Subtree window showing the subtrees of that node
- a Subtree window showing the variables and / or tables of that node
- a Table window showing the array of table variables in that table

The remainder of this section discusses examples of these windows.

Subtree Windows That Show Subtrees

Figure 4-21 shows the Subtree window for `mib-2`. It is an example of a Subtree window for a subtree that contains other subtrees.



Node: yeti
Object ID: 1.3.6.1.2.1
Name: iso.org.dod.internet.mgmt.mib-2

{4}	ip	Open group...
{5}	icmp	Open group...
{6}	tcp	Open group...
{7}	udp	Open group...
{8}	egp	Open group...
{10}	transmission	Open group...

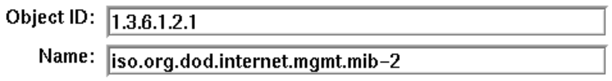
Figure 4-21 Subtree Window Showing Subtree Objects

The Node entry field, shown in Figure 4-22, contains the node name or address you specified in the Browser main window.

Node: yeti

Figure 4-22 Node Entry Field

The Object ID and Name entry fields, shown in Figure 4-23, contain two different representations of the name of the subtree displayed in the window. The Object ID entry field contains the numeric representation of the name (dot separated object numbers) and the Name entry field contains the text string representation (dot-separated object names).

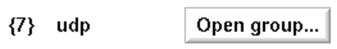


Object ID: 1.3.6.1.2.1

Name: iso.org.dod.internet.mgmt.mib-2

Figure 4-23 Object ID and Name Entry Fields

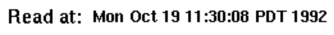
The scrolling display area in the center of the window contains one line for each object in the subtree, such as the udp line shown in Figure 4-24. The line begins with the object's number in curly braces followed by its object name. Clicking the *Open group...* button brings up a Subtree window for the object on this line. Its use is described more fully in "Navigation Using Buttons in the Subtree and Table Windows" in this chapter. A grayed-out button means there are no variables under this object in the MIB.



{7} udp

Figure 4-24 Object in a Display Area

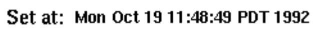
The Read At line provides status information during a "Get" operation (see "Obtaining, Setting, and Saving Variable Values" in this chapter), which is replaced by the current time after the operation is completed. An example is shown in Figure 4-25.



Read at: Mon Oct 19 11:30:08 PDT 1992

Figure 4-25 Read At Line

The current time is displayed on the Set At line after a "Set" operation (see "Obtaining, Setting, and Saving Variable Values" in this chapter). An example is shown in Figure 4-26.



Set at: Mon Oct 19 11:48:49 PDT 1992

Figure 4-26 Set At Line

Checking the "Close this window when opening a subwindow" check box, shown in Figure 4-27, specifies that you want this Subtree window to be closed when a new Subtree or Table window for a node in this subtree is opened. By default, each of the Subtree or Table windows you open for subtrees or tables within this subtree will have the same setting.

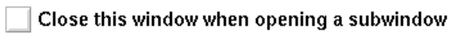
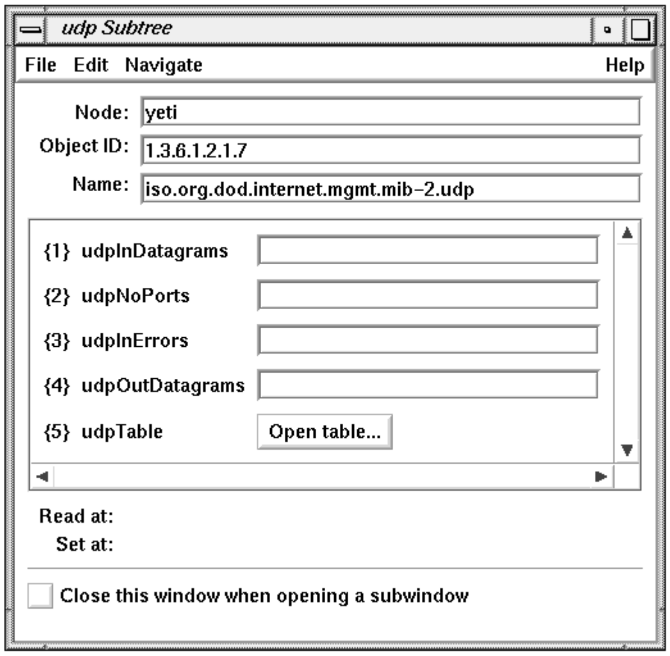


Figure 4-27 Close This Window When Opening a Subwindow Check Box

Subtree Windows That Show Variables and Tables

Figure 4-28 shows the Subtree window for `mib-2.udp`. It is an example of a Subtree window that shows the variables and / or tables of that subtree (in MIB terminology, this type of subtree is called a *group*).



Node: yeti
Object ID: 1.3.6.1.2.1.7
Name: iso.org.dod.internet.mgmt.mib-2.udp

{1} udpInDatagrams
{2} udpNoPorts
{3} udpInErrors
{4} udpOutDatagrams
{5} udpTable

Read at:
Set at:

Figure 4-28 Subtree Window Showing Variables and a Table

Most portions of this type of Subtree window are the same as the Subtree windows described in “Subtree Windows That Show Subtrees” in this chapter. However, the display area of this type of Subtree window contains entry fields and *Open table...* buttons rather than *Open group...* buttons.

Variables in the subtree are shown in the display area as an object number, an object name, and an entry field, as shown in Figure 4-29. When the object number (in braces) is appended to the object ID at the top of the window, it forms the complete object ID for the object. The entry field is gray for variables whose values are defined as read-only in the MIB and pink for variables that are defined as read-write or write-only in the MIB. If the entry field is pink, you can set the value of that variable (see “Obtaining and Setting Values Using the Edit Menu of a Subtree Window,” in this chapter).

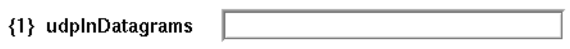


Figure 4-29 Variable Line in a Subtree Display Area

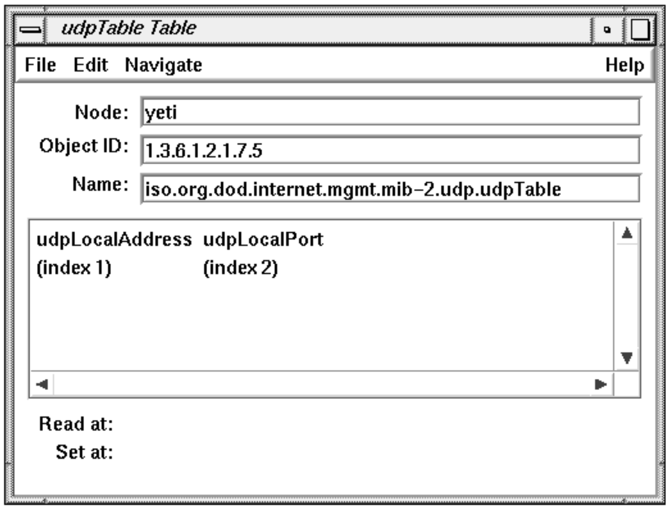
Tables in the subtree have lines that include their object number, their name, and the *Open table...* button, as shown in Figure 4-30. When you click an *Open table...* button, a Table window, described in the next section, appears.

{5} udpTable
Open table...

Figure 4-30 Table Line in a Subtree Display Area

Browser Table Windows

Figure 4-31 shows the default Table window for `mib-2.udp.udpTable`. When a Table window appears, the display area contains only the names of the table variables. Entry fields appear for the variables as you retrieve their values with “Get next row” in the Edit menu. (See “Obtaining and Setting Values Using the Edit Menu of a Table Window,” in this chapter.)



Node: yeti
Object ID: 1.3.6.1.2.1.7.5
Name: iso.org.dod.internet.mgmt.mib-2.udp.udpTable

```
udpLocalAddress  udpLocalPort
(index 1)        (index 2)
```

Read at:
Set at:

Figure 4-31 Browser Table Window

Navigating the SNMP Containment Tree

With Browser you can open a Subtree window for each subtree in an MIB that you want to browse and a Table window for each table in an MIB that you want to browse. Browser buttons and menus enable you to specify the subtree or table you want to view. When you use these buttons and menus, you are "navigating" the MIB Tree. The three navigation methods are described in the following sections.

Navigation Using the Buttons in the Main Window

The *mib-2*, *enterprises*, and *experimental* buttons in the Browser main window provide three starting points for browsing the SNMP Containment Tree. Clicking the *mib-2* button brings up a Subtree window for the MIB-II MIB. Clicking the *enterprises* and *experimental* buttons brings up Subtree windows for the Enterprises and Experimental subtrees, respectively.

Navigation Using the Navigate Menu

To use the Navigate menu from any window, follow these steps:

1. Press the left mouse button on Navigate in the menu bar.

   In the menu that appears, each choice except the last is the name of a subtree or table that is an object in the subtree in the window. Choices are highlighted as you move the cursor on them; if they have a rollover menu, it appears automatically. Figure 4-32 shows an example of the Navigate menu at the `mib-2` subtree with the cursor on udp.

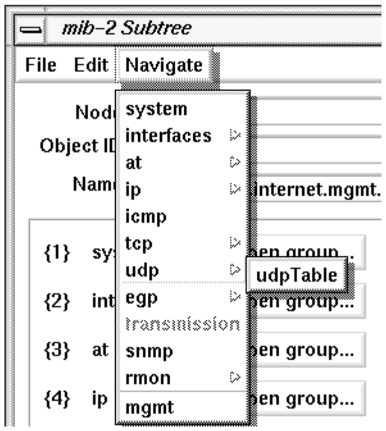


Figure 4-32 Navigate Menu

2. To view one of the subtrees of the subtree in the window, select one of the choices on the Navigate menu (not on a rollover menu).

   If you make the subtree selection shown in Figure 4-32, the window shown in Figure 4-28 appears.

3. To view subtrees or tables farther down in the hierarchy, move the cursor to a choice on a rollover menu and release the mouse button. In this way you can traverse the entire width and length of the subtree in the window.

4. To view the parent of the current subtree, select the last choice on the Navigate menu. It is the name of the parent of the subtree in the window.

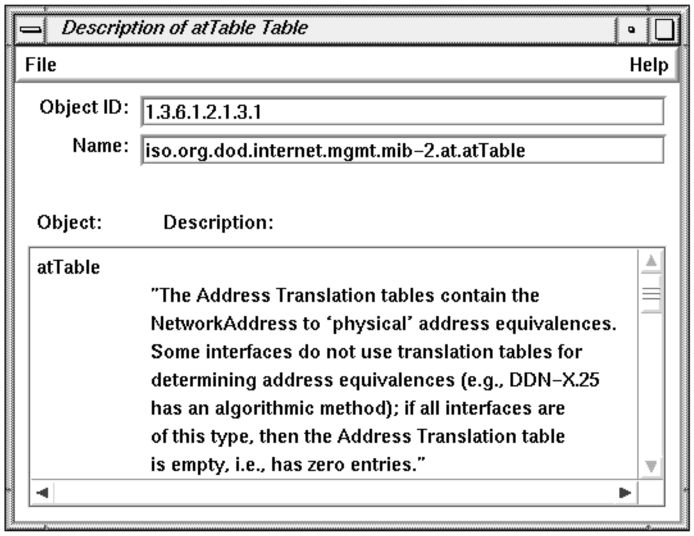
Navigation Using Buttons in the Subtree and Table Windows

Figure 4-21 shows an example of *Open group...* buttons in the display area of the `mib-2` Subtree window. When you click one of these buttons, a new Subtree window appears for this object. It is equivalent to choosing this subtree from the Navigate menu.

In Figure 4-28, `udpTable` is a table and has an *Open table...* button. Clicking an *Open table...* button is equivalent to choosing the table from the Navigate menu. Figure 4-31 shows the Table window for `udpTable` that appears when you click this button.

Obtaining Descriptions of Variables

To get a description of each of the objects in a subtree or each of the variables in a table, select "Description" from the Help menu of the Subtree or Table window. A Description window appears. Figure 4-33 shows an example.



Object ID: 1.3.6.1.2.1.3.1
Name: iso.org.dod.internet.mgmt.mib-2.at.atTable

Object: Description:

atTable
"The Address Translation tables contain the NetworkAddress to 'physical' address equivalences. Some interfaces do not use translation tables for determining address equivalences (e.g., DDN-X.25 has an algorithmic method); if all interfaces are of this type, then the Address Translation table is empty, i.e., has zero entries."

Figure 4-33 Description Window

Obtaining, Setting, and Saving Variable Values

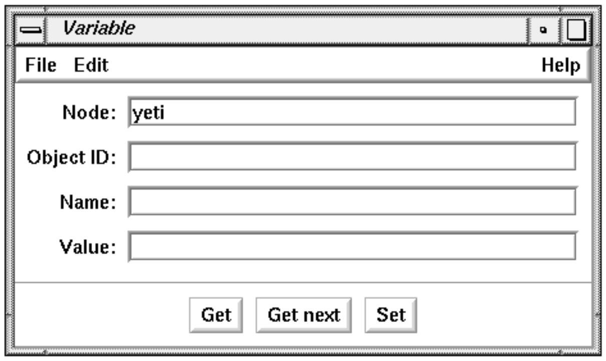
Browser enables you to obtain the values of MIB variables and set them if you have write access. (Write access is determined by the type of the variable and by your community. See “Authorizing Browsing” and “SNMP Management Glossary.”) Three types of Browser windows can be used to get and set variables:

- The Variable window enables you to get and set individual variables. The Variable window is described in “Obtaining and Setting Values Using the Variable Window,” in this chapter.
- Subtree windows enable you to get and set variables that aren’t part of tables. “Obtaining and Setting Values Using the Edit Menu of a Subtree Window” in this chapter describes how to do this.
- Table windows enable you to get and set variables that are part of tables. “Obtaining and Setting Values Using the Edit Menu of a Table Window,” in this chapter, describes how to do this.

Obtaining and Setting Values Using the Variable Window

Follow the steps below to use the Variable window to get and set variable values.

1. Click the *Variable...* button in the Browser main window. The window shown in Figure 4-34 appears.



Node: yeti
Object ID:
Name:
Value:

Figure 4-34 Variable Window

2. If you want to specify a variable by object identifier, fill in the Object ID entry field with the object identifier and append the following:

   ```
   .0
   ```

   The "dot zero" specifies that you want the value of the object; if you forget to use .0, Browser adds it automatically. For example, to specify mib-2.ip.ipForwarding (1.3.6.1.2.1.4.1), the Object ID entry field should look like the one shown in Figure 4-35.

Object ID: 1.3.6.1.2.1.4.1.0

Figure 4-35 Object ID Entry Field

3. To specify a variable in a table, enter its object identifier in the Object ID entry field. To construct its object identifier, you can use the object identifier of the table and append:

   ```
   .1.x.y
   ```

   The column number is represented by *x* (beginning with 1), and *y* is the value of `index` for the row you want. For example, the object identifier for the `ifDescr` variable (column 2) in the first row (index value of 1) of the `mib-2.interfaces.ifTable` (`1.3.6.1.2.1.2.2`) table is `1.3.6.1.2.1.2.2.1.2.1`. If the table you are using has more than one `index` column, create *y* by specifying each `index` value in order and separating them with periods. For example, if the value of `index1` is `127.1.9` and the value of `index2` is `7`, *y* is `127.1.9.7`.

4. If you want to specify the variable by name, fill in the Name entry field. You need not type in the complete hierarchical name, just the last component. Adding `.0` to the name is optional. If the Object ID and the name you fill in don't match, the Object ID is used.

5. To obtain the value of the variable, click the *Get* button. The value of the variable appears in the Value entry field. The Name entry field is automatically modified so that it contains the complete hierarchical name.

6. To set the value of a variable, enter the value in the Value entry field and click the *Set* button.

7. To obtain the value of the next variable, click the *Get next* button. Depth-first search is used to determine the next variable, so the right-most component of the object identifier varies fastest as the tree is traversed with *Get next*.
8. Continue obtaining and setting variables as necessary by modifying the Object ID, Name, and / or Value entry fields and using the *Get*, *Get next*, and *Set* buttons.

Obtaining and Setting Values Using the Edit Menu of a Subtree Window

You can obtain and set the values of variables from a Subtree window using the Edit menu:

1. Bring up the Subtree window that contains the variable whose value you want to obtain or set (see “Navigating the SNMP Containment Tree” in this chapter).
2. Select “Get” from the Edit menu to obtain the values of all of the variables. The current time is displayed on the Read At line.
3. Make changes in the entry fields for any variables whose values you want to change. Only variables whose entry fields are pink may be changed.
4. Select “Set” from the Edit menu to change variable values. The current time is displayed on the Set At line.

Obtaining and Setting Values Using the Edit Menu of a Table Window

You can obtain and set the values of variables in a table from its Table window using the Edit menu:

1. Bring up the Table window for the table you are interested in (see “Navigation Using the Navigate Menu” in this chapter).
2. To obtain the first row of variables in the table, select “Get next row” from the Edit menu. The current time is displayed on the Read At line.
3. To obtain other rows for the table, select “Get next row” from the Edit menu as many times as necessary.

4. Make changes in the entry fields for any variables whose values you want to change.
5. Select “Set” from the Edit menu to change variable values. The current time is displayed on the Set At line.

Browser File Menu

The File menu in each window gives you one or more of the window management and quitting choices listed below.

“Save MIB Values”	Save MIB values for this subtree for all nodes in the subtree that have open windows to a file. The values are appended to the file that you last specified with “Save MIB Values As...”.
“Save MIB Values As...”	Save MIB values for this subtree for all nodes in the subtree that have open windows to a file. When you select this choice, a file prompter appears. Use it to specify a filename The filenames are sorted by object identifier.
“Pop Main Window”	Display the Browser main window. This is useful if you have many windows open and want to locate the Browser main window quickly.
“Close Lower Level Windows”	Close the windows for all subtrees and tables below the subtree in this window. (You can close windows automatically as new windows are opened, using the check box “Close this window when opening a subwindow.” See “Subtree Windows That Show Subtrees,” in this chapter.)
“Close”	Close this window.
“Quit”	Quit Browser (available only from the File menu in the Browser main window).

Browser Example

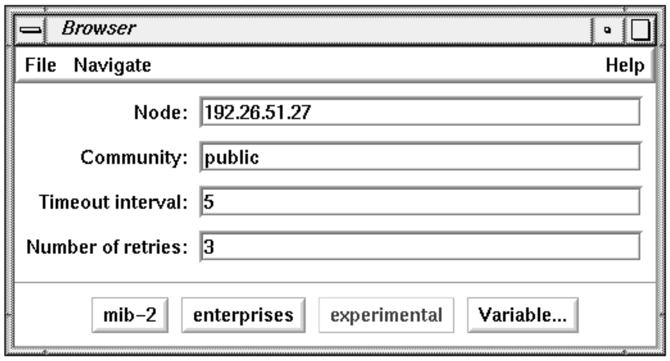
This section contains an example Browser session on a network that contains a Cisco router with IP address `192.26.51.27`.

To begin this session, invoke Browser through *provision*.

The Browser main window is placed on the screen. To browse the MIB for the Cisco router, first fill in the entry fields in the Browser main window:

Node	Enter the Cisco router's IP address, `192.26.51.27`.
Community	Enter the community string your workstation is authorized to use. In this example, the string `public` is used.
Time-out interval	Use the default value, 5 seconds, for the time-out interval. If the Browser doesn't receive a reply from the Cisco SNMP agent in 5 seconds, it will try again.
Number of retries	Use the default number of retries, which is 3. This entry field specifies the number of times that Browser tries again if no reply is received from the Cisco agent within the time-out interval.

Figure 4-36 shows the Browser main window after you've filled in the entry fields.



Node: 192.26.51.27
Community: public
Timeout interval: 5
Number of retries: 3

Figure 4-36 Example Browser Main Window

Suppose you want to get the values for the `lsystem` group in the Cisco MIB. To display the variables in this group, use the Navigate rollover menus to navigate through the MIB hierarchy to `lsystem`, as shown in Figure 4-37.

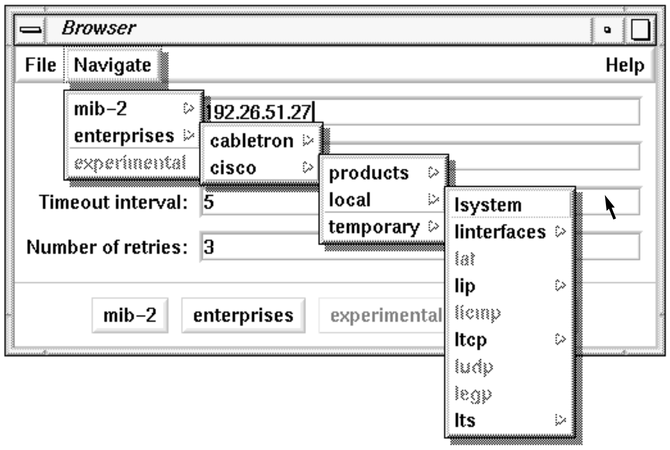
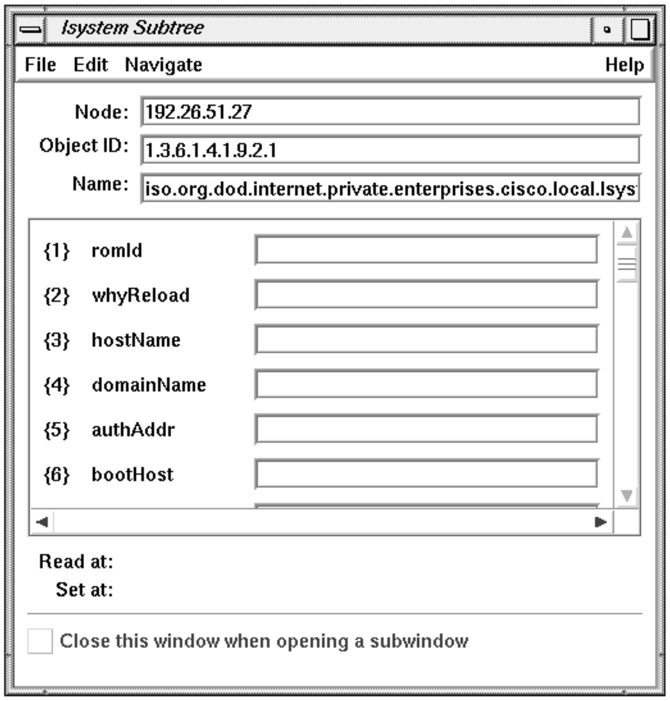


Figure 4-37 Navigate Rollover Menus for cisco.local.lsystem

When you release the mouse button, the cisco Subtree window shown in Figure 4-38 appears:



Node: 192.26.51.27
Object ID: 1.3.6.1.4.1.9.2.1
Name: iso.org.dod.internet.private.enterprises.cisco.local.lsys

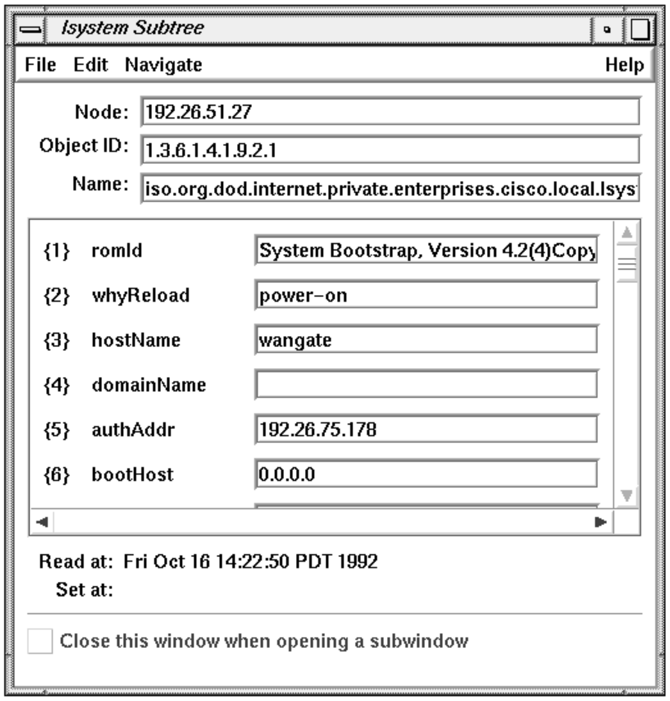
{1} romId
{2} whyReload
{3} hostName
{4} domainName
{5} authAddr
{6} bootHost

Read at:
Set at:

Close this window when opening a subwindow

Figure 4-38 Subtree Window for cisco.local.lsystem

To get the values for these variables, select “Get” from the Edit menu. The entry fields for the variables are filled in with their current values, and the current time is indicated at the bottom of the window. Figure 4-39 shows an example:



Node: 192.26.51.27
Object ID: 1.3.6.1.4.1.9.2.1
Name: iso.org.dod.internet.private.enterprises.cisco.local.lsys

{1}	romId	System Bootstrap, Version 4.2(4)Copy
{2}	whyReload	power–on
{3}	hostName	wangate
{4}	domainName	
{5}	authAddr	192.26.75.178
{6}	bootHost	0.0.0.0

Read at: Fri Oct 16 14:22:50 PDT 1992
Set at:

Figure 4-39 Subtree Window With Values for cisco.local.lsystem

Use the right scroll bar to adjust the display area so that you can examine the values of the variables that don’t fit in the default-size display area.

SNMP Management Glossary

This section describes some basic terms used in the SNMP management framework. It begins with basic concepts. Later definitions build on terms defined previously. Terms in *italics* are defined elsewhere in this section.

Agent

Software or firmware that gathers the information important to the device on which it resides. It also implements a protocol that exchanges that information with a *network management station*. *snmpd*(1M) is an example of an SNMP agent.

Network Management Protocol

The protocol used to convey management information between an *agent* and a *network management application*. The protocol used by Browser to query information from various agents is *SNMP*.

Simple Network Management Protocol (SNMP)

The *network management protocol* used by Browser to talk to *agents* on remote *managed nodes*. For Silicon Graphics workstations, the SNMP agent is *snmpd*(1M). Agents for other types of nodes may be implemented in software or firmware and are vendor-specific.

Management Information Base (MIB)

The specification for the virtual store of the information supported by an *agent*. Some MIBs have *RFC* status, which means they have been approved by the *IETF*.

MIBs are defined in *SMI* format. For instance, a router MIB is a collection of important information about a router defined in *SMI* format. An SNMP *agent* typically implements two MIBs, *MIB-II* and a device-specific MIB (an *Enterprise MIB*). However, the *agent* may not implement all of the objects defined in each MIB.

SNMP Containment Tree

A hierarchical tree of information gathered by agents about devices.

Subtree

A node with children in the *SNMP Containment Tree*.

MIB-II

The Internet-standard *MIB (RFC* 1213). This MIB has *managed objects* that are important for managing the TCP/IP suite of protocols.

Managed Object

A managed object is also known as a variable.

Variable

A leaf node in the *SNMP Containment Tree* is called a variable. In the *SMI* definition for each *MIB*, each variable is defined to be read-write, read-only, or write-only.

Object Identifier (Object ID)

An object identifier is a name in a dot notation that uniquely identifies an object (*subtree* or *managed object*) in the *MIB*. For example, the object ID for `sysContact` is `1.3.6.1.2.1.1.4`.

Enterprise MIBs

MIBs defined by different vendors for managing their devices. They can be specific to one device or vendor. For example, the CISCO *MIB* has objects for the Cisco router.

Community String

A password used by an *SNMP agent* for authentication purposes. The default string for Silicon Graphics workstations is “public”. Community strings are usually defined by system administrators.

Chapter 5

System Administration Request Management With problema

This chapter describes the *problema* software package. The following topics are covered:

- An overview of the *problema* system and its interfaces in "The problema Request Management System."
- Instructions on installing and configuring *problema* are provided in "Installing and Configuring problema."
- A description of the actions and processes that *problema* users and administrators are likely to use in "Using problema."
- A specific description of the user-level facilities of *problema* is provided in "General Operations."

The *problema* system administration request tracking service is based on the successful CASEVision / Tracker™ application, and brings the power and ease-of-use of that software to the system administrator. The configurability of Tracker has been maintained in *problema,* while custom-tailoring the application to the system administrator's needs. The Tracker product must be installed in order to use *problema* successfully.

Much of the in-depth documentation provided with Tracker is also accurate for *problema,* and should therefore be considered a valuable resource for understanding this product.

The *problema* request tracking software uses the same database software technology used in Tracker and in the rest of the IRIXpro suite of system administration tools. However, the *problema* application uses the *dml* language, as opposed to sgitcl in the rest of IRIXpro. Sample *dml* scripts are provided in the */usr/IRIXpro/sample/problema* directory that are designed to be run as **root** and **irixpro** *cron*(1) jobs. The script *problemaDBclean* removes the requests marked as DELETED from the database. The *problemaDBdue* script lists the requests that are not closed and past their expected due date.

The *problema* application has been designed to offer a convenient way for users to request system administration services and to offer administrators a convenient way to track those requests. A clear record is kept of each request, and the administrator is free to prioritize and assign each request in the most efficient way. With this system, the best interests of the user and the administrator are served by clarifying a process that is often handled informally.

The problema Request Management System

The *problema* request tracking system includes a database to hold system administration requests and graphical interfaces to that database. Online help is available at all times through the graphical interface windows, providing detailed instructions on the request filing, tracking, resolution and querying processes.

problema, miproblema, and suproblema

The *problema* application consists primarily of a database and three very similar interfaces to the information in the database. The difference between the interfaces is primarily in the level of detail and number of fields offered. The first interface is the one most users see regularly, called *miproblema*. This interface provides only the most basic fields necessary to submit a request to the administrators. The second interface is *problema*, which is also designed for users, but which provides a more complete interface to the database. The third interface is reserved for the system administrators or managers who make decisions and take action on the requests submitted by the users. This administrator's interface is called *suproblema*. The *suproblema* window contains some fields and options not available on the *problema* user-level version. All these interfaces are described and shown in the section titled "Using problema" on page 157.

In the course of this chapter, except where otherwise stated, all information is true for all *problema* interfaces. The *suproblema* interface is a superset of the *problema* interface, and the *problema* interface is a superset of the *miproblema* interface. The term *problema* is used in general to describe the database and its interfaces, rather than strictly the user's interface.

In addition to the functions available through the *miproblema* and *problema* interfaces, the *suproblema* interface to the *problema* database allows the administrator or manager to track a request that has been forwarded to a known vendor, and to add a technical description of the user's problem. It offers more action options than are available through the user's interfaces.

Typically, the *suproblema* interface is installed only on the host where the system administrators maintain the *problema* database and have regular access. Hosts that serve users generally have the *miproblema* and *problema* interfaces. However, you are free to install *suproblema* wherever you like and allow whomever you choose to use it, as best suits your needs.

The problema Tracking Process

This section describes the *problema* process in terms of the life cycle of a request in the system. The data flow diagram in Figure 5-1 illustrates the request tracking process.

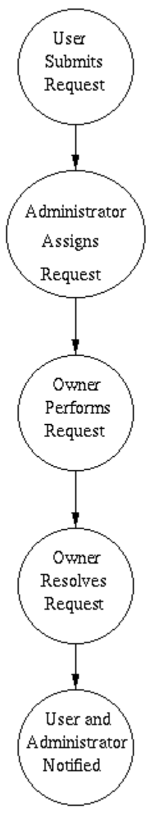


Figure 5-1 The problema Request Life Cycle

A request can go through many different states during its life cycle. The request tracking process has four major stages:

1. A user submits a request into the *problema* database. At this point the request is in the AWAITING_RESPONSE state.
2. The request is assigned to an owner to perform the work. If no action is required (for example, if the request is already resolved, must be deferred, or is a duplicate), the request is tagged appropriately and then returned to the database in the CLOSED or AWAITING_RESPONSE state for later action.
3. Assigned requests are then executed by their owners. The owner may also elect to defer, resolve, or tag the request as a duplicate, or the request may be forwarded to an outside vendor for attention.
4. Finally, the request is resolved when the work is done and the request is moved to the CLOSED state. If the work was not done correctly, the original requesting user can reopen the request, if need be, or submit a new request.

Installing and Configuring problema

The *problema* application is ready to run when you install IRIXpro on your system. All basic files necessary for standard usage are installed and configured for you. To use *problema* as it is distributed, you must have a CaseVision / Tracker user's license installed as well.

The *problema* application can be configured to match your needs and methods of administration. All necessary files for customization are installed with the application. To customize problema or create your own custom application, you must have a CaseVision / Tracker designer's license installed.

You can configure *problema* to accept the selected priorities, categories, and individuals who are authorized to execute *problema* operations. But first, you must install *problema* on each client system and the server, and run the application generation tools.

Installing problema on the Server

Follow these steps to install *problema* on your server:

1. Log in to your server as **root**.
2. Use the *inst*(1) program to install the *Tracker 2.0* product and the appropriate licenses on your server.
3. Execute the command */usr/IRIXpro/problema/problemainstall*. This command puts the *problema* commands in */usr/local/bin* and the *problema app-defaults* file in */usr/lib/X11/app-defaults*, and initializes your database. This process also adds *problema* online help to your *sgihelp* system. When this is done, *problema* is installed on your server.
4. Put */usr/local/bin* in your PATH environment variable, if it is not already there.
5. Execute the *suproblema* command.

Installing problema on Client Systems

Follow these steps to install *problema* on your client systems:

1. Log in to your client as **root**.
2. Use the *inst*(1) program to install the *Tracker 2.0* product and the appropriate licenses on your client.
3. Using either *inst*, NFS, *propel*, *rcp*, or media-based transfer, install the following parts of your *problema* software distribution on your client:
   - */usr/IRIXpro/problema/problema.pdl*
   - */usr/IRIXpro/problema/tools*
   - */usr/IRIXpro/problema/problemainstall*
4. Make sure your system has the */usr/local/bin* directory, or create it if needed.
5. Execute the command */usr/IRIXpro/problema/problemainstall*. This command puts the *problema* commands in */usr/local/bin*, the *problema app-defaults* file in */usr/lib/X11/app-defaults*, and determines the locations

of the server database. This process also adds *problema* online help to your *sgihelp* system. When this is done, *problema* is installed on your client.

6. Put */usr/local/bin* in your PATH environment variable, if it is not already there. If you do not wish to use the */usr/local/bin* location, the *problema* application can be started from the icon in the IRIXpro directory view, as can all IRIXpro applications.
7. Execute the *miproblema* or *problema* command.

The User irixpro

You must create a user called **irixpro** on your system if you wish to delete request records from your *problema* database when they have been closed, or if you wish to access Silicon Graphics Electronic Services.

Create the user **irixpro** according to the instructions in Chapter 5 of the *Personal System Administration Guide* or according to the instructions in Chapter 3 of your *IRIX Advanced Site and Server Administration Guide.* You may use any available UID for this account.

Configuration Files

The configurable parameters of *problema* are stored in the following files. Each file is described below:

- */usr/IRIXpro/problema/miproblema.pdl*
- */usr/IRIXpro/problema/problema.pdl*
- */usr/IRIXpro/problema/suproblema.pdl*
- */usr/IRIXpro/problema/problemagen*
- */usr/IRIXpro/problema/categories.h*
- */usr/IRIXpro/problema/vendor_categories.h*
- */usr/IRIXpro/problema/vendor_names.h*
- */usr/IRIXpro/problema/vendor_priorities.h*
- */usr/IRIXpro/problema/vendor_status.h*

- */usr/IRIXpro/problema/tools/lib/problema_notify*
- */usr/lib/X11/app-defaults/Miproblema*
- */usr/lib/X11/app-defaults/Problema*
- */usr/lib/X11/app-defaults/Suproblema*

miproblema.pdl

This is the configuration file for the *miproblema* view of the application. This is a plain text file, editable with any text editor. You must be logged in as **root** to edit this file. The file is extensively commented and you can alter it to change *miproblema* as you see fit. There is extensive documentation on modifying PDL files in Chapter 2 of the *CASEVision/Tracker Design Guide*, titled ‘’Using the Process Description Language.”

problema.pdl

This is the configuration file for the *problema* view of the application. This is a plain text file, editable with any text editor. You must be logged in as **root** to edit this file. The file is extensively commented and you can alter it to change *problema* as you see fit. There is extensive documentation on modifying PDL files in Chapter 2 of the *CASEVision/Tracker Design Guide*, titled ‘’Using the Process Description Language.”

suproblema.pdl

This is the primary configuration file for all of *problema*. This file specifically defines the *suproblema* application as it is distributed. This is a plain text file, editable with any text editor. You must be logged in as **root** to edit this file. The file is extensively commented to help you make your configuration choices. There is also extensive documentation on modifying PDL files in Chapter 2 of the *CASEVision/Tracker Design Guide*, titled ‘’Using the Process Description Language.” This file is where you set the allowable categories of requests, the allowable priorities, and the acceptable entries for each field.

problemagen

This script takes the changes made to the *suproblema.pdl* file and effects the changes in the *problema* database. If you use *problema* to create a custom application with its own database, use this file to specify the location of the database and the PDL file for the new application.

categories.h

The *categories.h* file controls the available categories for requests through *problema*. You may customize this file as you see fit, and you can select a default owner for each category of requests if you so choose. The format is:

```
Category         //      Owner
Misc,            //      jschmoe
```

Each category and owner pair are separated by two slash characters. On all but the final line, a comma (,) must follow the category name. Instructions are provided in the text of this file. This is a plain text file, editable with any text editor. You must be logged in as **root** to edit this file.

vendor_categories.h

This file controls the available vendor categories for request forwarding. Each category should correspond to an appropriate outside vendor.

vendor_names.h

This file names the available outside vendors for request forwarding. Each line in the file names a vendor.

vendor_priorities.h

This file names the available vendor priorities for request forwarding. Each line in the file names a possible priority.

vendor_status.h

This file controls the various options for the vendor's status on a request. The default values should cover most scenarios, but this file is configurable.

/usr/IRIXpro/problema/tools/lib/problema_notify

This file controls the e-mail process whereby users and other parties are notified of request transitions. This file is configurable.

/usr/lib/X11/app-defaults/Miproblema, Problema and Suproblema

These files control the X resources and general display characteristics for *miproblema*, *problema* and *suproblema*. It is not generally necessary to change these files. As distributed, they are all identical to each other.

Making Configuration Changes

The primary means of configuring the *problema* functionality is by editing the *suproblema.pdl* file, the *miproblema.pdl* file, and the *problema.pdl* file. These files are extensively commented, and written in the *Process Description Language*, which is described in detail in Chapter 2 of the *CASEVision/Tracker Design Guide*. Specific information regarding PDL syntax can be found in that guide.

The *problema* software is completely configurable for your site and organizational needs. The Process Description Language allows you to work within a basic framework of data structures and concepts to define:

- your request states
- the transitions between those states
- the fields that will make up each request record
- the allowable values for those fields

The Process Description Language also allows you to set up rules regarding the use of the data structures you have defined and to define new applications or new views of the *problema* database. Finally, you can use the PDL files to have *problema* execute IRIX shell commands of your choice as part of transitions, make field values available to your users as shell environment variables, and generally tailor the interaction between *problema* and other system utilities or applications.

The following list offers suggestions and ideas about custom uses for *problema* and the Process Description Language. The example used for customizing is a system administration group with special needs for quick response:

Request States

Given the definition of the *problema* application as a database product, you can define your own states for requests. For example, if your organization submits results back to the requesting party for final approval, you can create a state defined as AWAITING_APPROVAL.

Request Transitions

If you have defined custom states for your *problema* requests, you may also want to define the transition commands between those states. You can create action-based transitions, and those transitions can do more than simply change the state of a request. A transition may involve the notification of various parties, the execution of IRIX shell commands, or the generation of hardcopy reports. For example, if a system administration group must move quickly, the ASSIGN transition can be modified to use an e-mail pager to notify the request owner.

New Fields

Different organizations have different information they wish to track in the life cycle of a request. The *problema* application allows you to create and define request description fields as you see fit.

New Field Values

When you have defined a new field, you can also define the type of information and the possible values the field may have. There is a list (published in Table 2-1 of the *CASEVision/Tracker Design Guide*) of the available field types. You may set the parameters for acceptable values. For example, you may use a boolean type field to hold true-or-false information about the request.

New Rules

You can define rules about who in your organization may make transitions (each transition can have its own customized rules) and about which fields must be filled in for transitions to take place. For example, there can be transition commands restricted to certain users, or the vendor forwarding transitions can cause e-mail to be sent directly to the outside vendor.

New Applications

You can use the Process Description Language to create your own custom applications. Once you have defined a set of fields, transitions, states, and rules, you can set up a separate command name to execute the *problema* code and a separate database location, and your new application is complete. Once you have created the application, you must place the name of the application in the *problemagen* file and execute the file to generate the application.

New Views

You can create, in the same manner as with a new application, a custom view of the database. For example, you may wish to create a view of the database that allows only request submission. To do this, begin with a copy of the *problema.pdl* file and modify it to suit your needs. Information that is confidential or irrelevant may be screened from the view provided to different users.

IRIX Command Execution

As part of a transition command, you can direct *problema* to execute IRIX shell commands. A direct example of this is the list of people who are notified by e-mail when a transition takes place. However, any command you place in the PDL file can be executed if you create a rule to do so.

Environment Sharing

You can make shell environment variables available for users through the PDL files. For example, you may want to cause the ASSIGN transition command to make an environment variable available to the new owner of a request. The field names, their values, and the current transition as well as a list of modified fields at any transition can be made available as shell environment variables. You cannot, though, arbitrarily create environment variables or arbitrary data.

Application Interface

If you create your own custom applications, you can also choose to allow the various *problema* databases you have created to interact. For example, if you have created a separate *problema* application to handle your user service requests and one to handle requests for new hardware, you can link the two so that certain transitions (such as when hardware is installed) cause both applications to be updated at once.

The Process Description Language imposes very few rules on the application developer. The language works within the paradigm of the database, and actions are defined in terms of field values, request states, and transitions, and in terms of views of the database records beyond these limitations (which can more correctly be termed specializations), you are free to define the services *problema* renders to you and your users.

Using problema

The following sections describe the form and use of *suproblema*, *miproblema*, and *problema*. First, there is a description of each part of the interface window and information on using the features in the window. Then information is provided on using other features of *problema*. Finally, specific sections detailing operations on the server and client systems are provided.

The *suproblema* main window is shown in Figure 5-2.

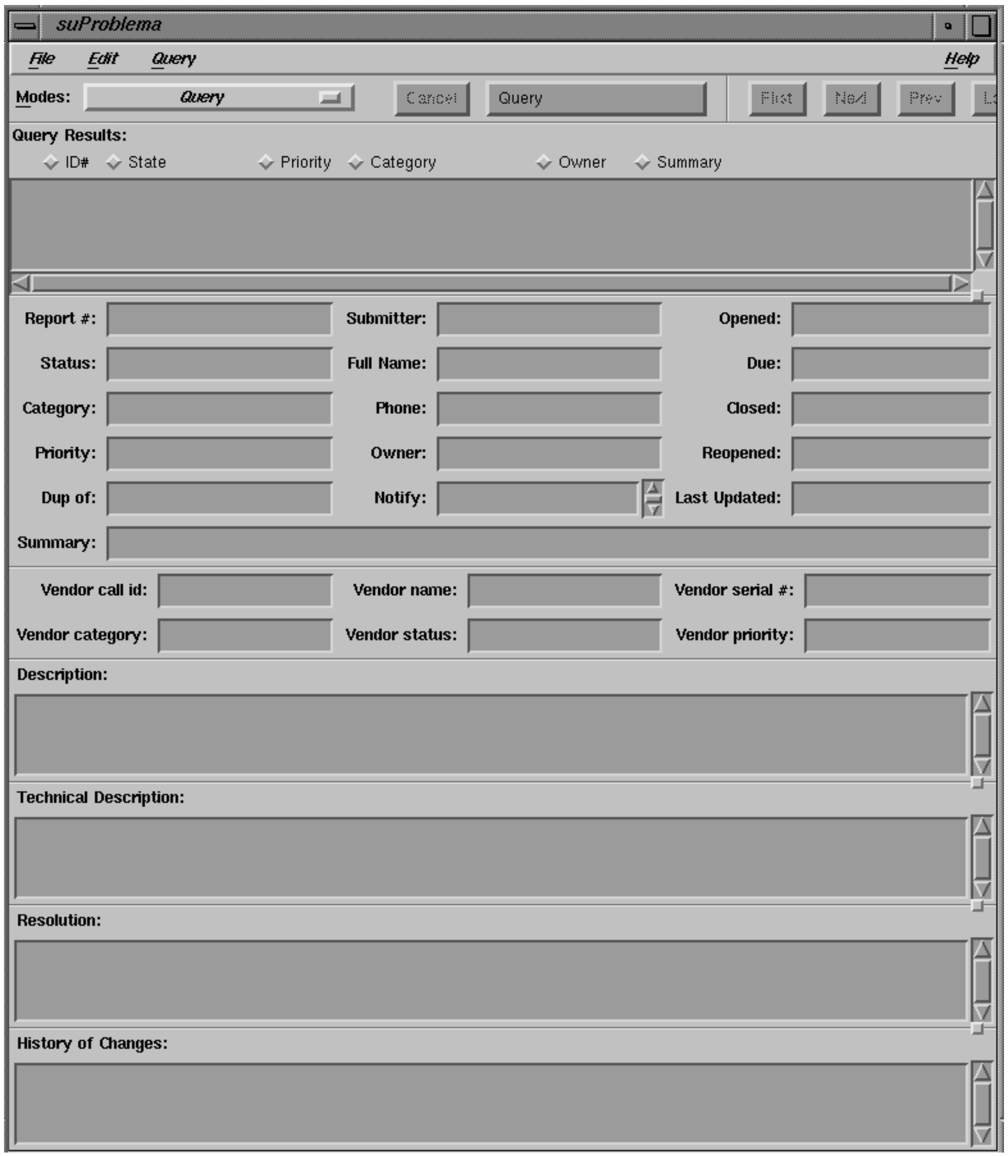


Figure 5-2 The suproblema Main Window

The *problema* main window is shown in Figure 5-3:

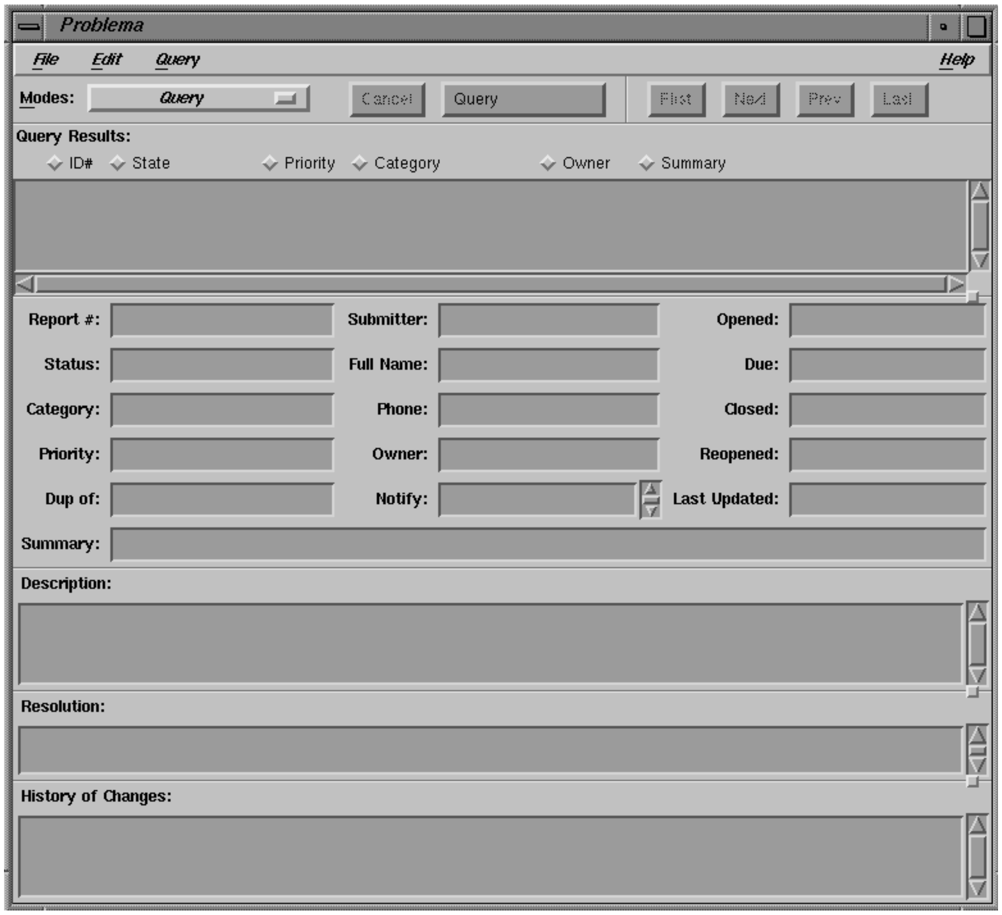


Figure 5-3 The problema Main Window

The *miproblema* window is shown in Figure 5-4:

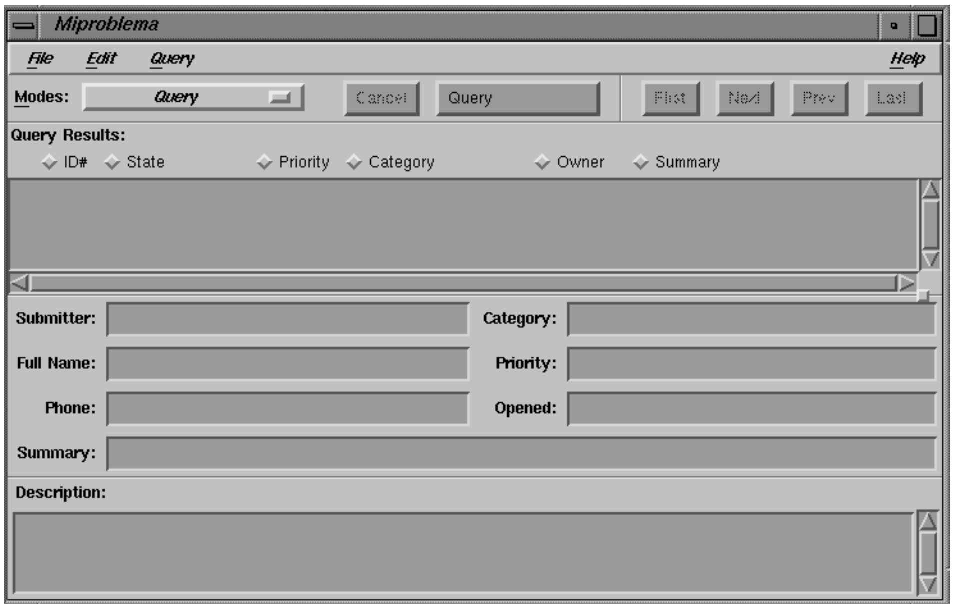


Figure 5-4 The miproblema Main Window

Main Window Anatomy

All the *problema* main windows have four parts:

menu bar	Lets you access the window menus.
control bar	Lets you display a list of requests and perform operations on individual requests.
query results area	Displays lists of requests that result from queries.
request form area	Lets you enter or display detailed information about a single request.

Each part of the *problema* main windows is described in detail in the sections below.

Menu Bar

The menu bar in each problema main window is shown in Figure 5-5.

:

Figure 5-5 The Menu Bar

The menu bar offers you four menus. The menus are:

File menu — Lets you print a file or exit the window.

Edit menu — Provides editing commands that you can apply to all fields as a group:

- “Reuse” takes the fields from a selected entry in the query results area and fills out the request form fields with those values.
- “Revert” restores the request form fields to their saved values after changes have been made.
- “Clear” clears all request form fields.

Query menu — Lets you reuse query selection criteria by making the following settings:

- “Save...” stores the current set of criteria.
- “Load...” enters a saved set of criteria into the appropriate fields.
- “Save as Default” saves your current criteria and reuses it as a default when you start up *problema.*
- “Case Sensitive Matches” toggles case sensitivity on or off when you are querying for a string match in a field.

Help menu — Lets you get product version information, summary information for the window, context information for components of the window, and an index of help topics for the system.

Control Bar

The control bar of each *problema* window is shown here in Figure 5-6, with the Query mode selected from the Modes menu

:

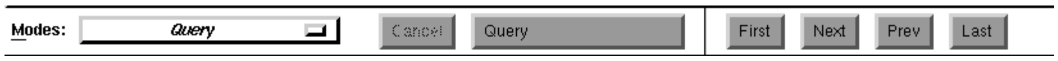


Figure 5-6 The Control Bar

The Modes menu on the left displays the actions that you can perform on requests in the database. At any given time, some actions may be disabled if they are not available for the request being acted upon. For example, if you are examining a closed request, the “resolve” action is not available.

The actions available from the *miproblema* Modes menu are:

- QUERY
- DISPLAY
- SUBMIT_REQUEST

The actions available from the *problema* Modes menu are:

- QUERY
- DISPLAY
- SUBMIT_REQUEST
- RESOLVE
- NOTIFYME
- EDIT
- REOPEN

The actions available from the *suproblema* Modes menu are:

- QUERY
- DISPLAY
- SUBMIT_REQUEST
- RESOLVE

- NOTIFYME
- EDIT
- REOPEN
- FORWARD_TO_VENDOR
- UPDATE_FROM_VENDOR
- ASSIGN
- DEFER
- DUPLICATE
- DELETE

When you select a command from the Modes menu, the information displayed in the rest of the window changes, as does the read / write permission on the fields below in the request form area. To the right of the Modes menu, the *apply command* button changes to reflect the command you selected from the Modes menu.

Commands have two categories: inspection commands that review requests and transition commands that can change the state of a request. The inspection commands, Query and Display, let you look up requests, either in a list of summaries or individually in detail. The transition commands change either the status of the request or the fields in the report. Any field required for a specific transition is highlighted, indicating that an entry is necessary. If no entry is made, then the command cannot be applied.

The list control buttons on the right of the control bar are:

- *First*
- *Next*
- *Previous*
- *Last*

These buttons let you choose a request from the query results area list for display in the request form area.

Query Results Area

The *problema* query results area is shown below in Figure 5-7. It is shown empty, as it is when the interface is first brought up.

:

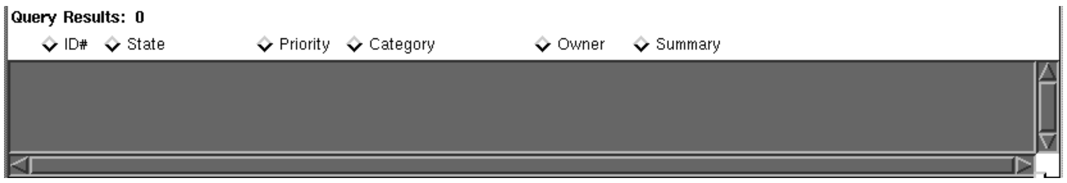


Figure 5-7 The Query Results Area

The number of requests in the current query is displayed at the top of the list area. When a query is made, the requests selected are listed in the grey area. If you use any of the list controls (buttons or menu items) to display a request in the request form area, the list in the grey area will scroll to the displayed request.

Two small icons that may appear to the left of a list entry in the grey area also help to track the status of a request:

- A small rectangle indicates that the request has been displayed in the form area at least once since its submission.
- A check mark indicates that the request has been edited at least once.

The query results area is divided into six columns, each with a title. The text in each column lines up under the title. Next to each title is a radio button. When you click one of these buttons, the button is highlighted and the list of requests is sorted accordingly. For example, if you click the ID# button, the list is sorted in ascending order according to the request identification number. You can sort only one characteristic at a time.

Request Form Area

The request form area contains fields for entering and displaying data. A blank *miproblema* request form area is shown here in Figure 5-8:

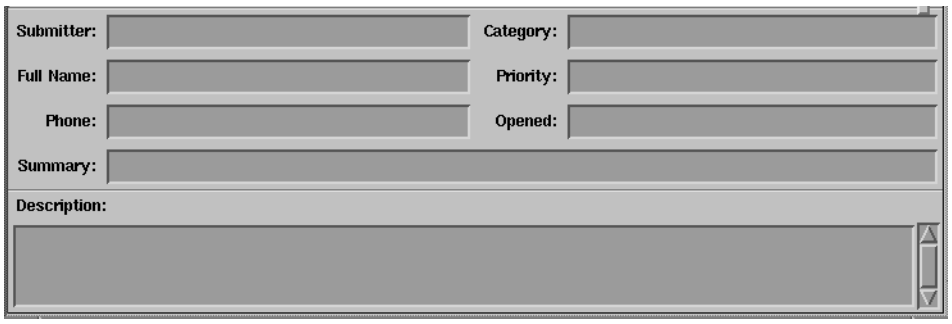


Figure 5-8 The miproblema Request Form Area

A blank *problema* request form area is shown in Figure 5-9:

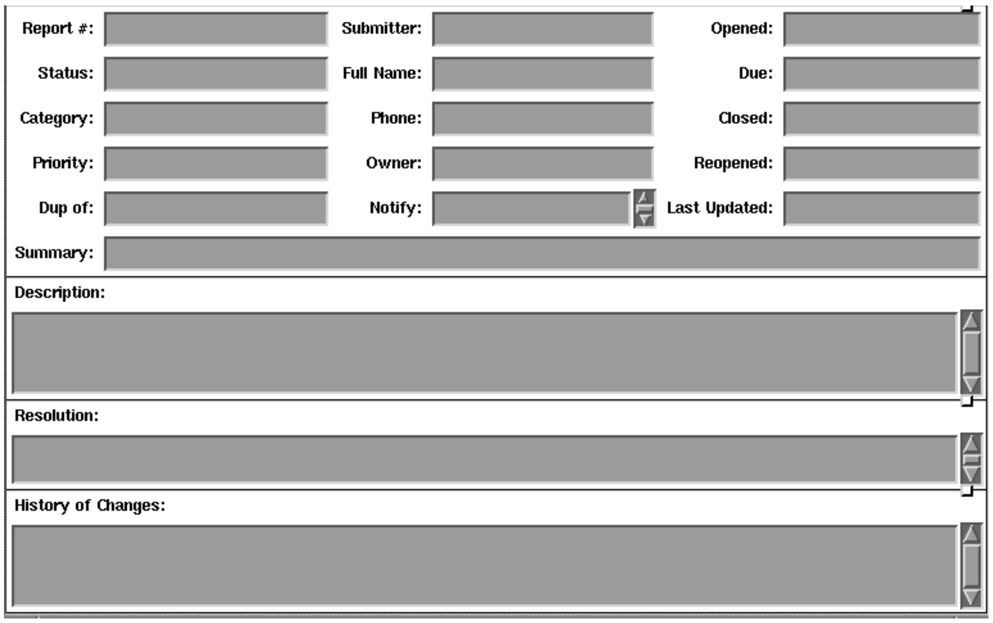


Figure 5-9 The problema Request Form Area

The request form area for *suproblema* has certain extra fields dealing with outside vendor forwarding and technical description, as shown here in Figure 5-10:

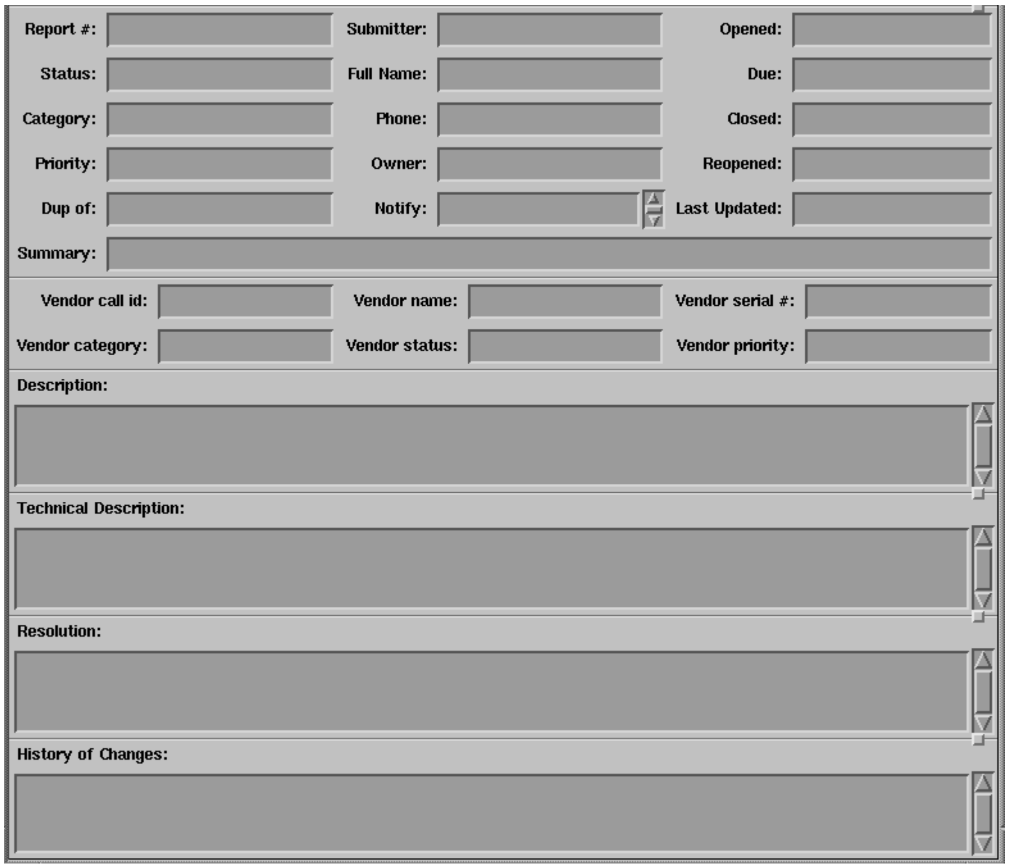


Figure 5-10 The suproblema Request Form Area

In all of the *problema* interfaces, fields change colors and take on highlighting according to your current operation. If a field has a red border, it is required to be completed before the operation can be applied. Fields may also have a red border if they are incorrectly entered. In either case, a highlighted field blocks the operation. If a field is the same color as the window background color, then it is writable, but optional. If the field turns black, it is not writable during this operation.

To display a popup menu containing options for any field, hold down the right mouse button while the cursor is over the field. At minimum, the menu contains the items Reuse (for using the last value displayed in this field), Revert (for returning to the saved value), and Clear (for blanking out the field). If predefined values are available for the field, they are displayed on the cascading menu attached to the Values item. The cascading menu features a tear-off perforation at the top. If you click the perforation, the menu is displayed in its own window.

The menu for fields that accept long text descriptions such as the Description and Resolution fields have an option called Edit... that lets you enter the text through your editor of choice. The editor default depends on your environment variable setting and defaults to *vi*(1) if no variables are set. See “Setting the Default Field Editor” on page 173 for more information.

The History of Changes field is automatically updated by *problema* each time an action is taken on a request.

Setting Dates in problema

The *problema* software provides a wide variety of date input formats. Dates can be supplied with as little information as the year or month, or specified to the nearest second.

Special formats allow dates to be supplied as a base time point plus or minus a time interval, and also as relative to the current year, month, day, hour or second. Date field values specify a point in time to the nearest second. You cannot, however, represent a time interval such as ‘’in six seconds’’ or ‘’in two days.’’

Date values are ordered from a starting point (the lowest value) and increasing toward later dates. Therefore, a date that occurs after a specified date will have a higher value. This enables you to use comparison operators such as < (before) and > (after).

When you use dates in transitions, you always enter a specific point in time. When you are using dates in queries, you can enter a range or condition as well as a specific time.

Setting a date and time in *problema* is handled in the same manner as CASEVision / Tracker, and Chapter 3 of the *CASEVision/Tracker User's Guide* offers a complete explanation of all date and time functions.

Query Operations

To query the *problema* database, perform the following steps:

1. Select Query from the Modes menu if it is not already selected. All fields in the window become writable.
2. Enter the desired selection criteria in the appropriate fields.
3. Click the *apply command* button to perform your query. All appropriate requests appear in summary form in the query results area.

   Leaving all fields blank brings all requests in the database into the list, with the first few displayed in the query results portion of the window. Entering one or more fields displays those requests with matching fields. Entering multiple fields has the effect of selecting only those requests that match all listed criteria. You can also use the operators in Table 5-1 to refine your queries. When you use these operators, you should enclose all values inside quotation marks ("").

Table 5-1 Logical Operators

Operator	Description
=range entry	equal to
<>	not equal to
<	less than
<=	less than or equal to
>	greater than

Table 5-1 Logical Operators

Operator	Description
>=	greater than or equal to
=match	regular expression match
contains	contains specified value (list fields only)
contains any	contains any of the specified values (list fields only)
contains only	contains only the specified values (list fields only)
= null	unset or non-existent test
<> null	set or existing test
[startrange:endrange]	range entry

The less-than (<) and greater-than (>) operators can be used with dates to mean before and after, respectively. For example, >June 25 means after June 25.

4. To display the detailed information for any request in the query results area, you can double-click the request directly, or you can click one of the list control buttons: First, Next, Prev, and Last.

   When you display a request, a small rectangular icon appears to the left of the request line in the query results area. If you modify the request, a check mark will appear next to the request as well. If you make a subsequent query, the icons will be cleared.

General Operations

All user operations can be performed from *miproblema* or *problema*, and all information in the database can be viewed from *problema*.

Procedure for Performing General Operations

For most *problema* operations, follow these steps:

1. Enter the command

   ```
   problema
   ```

   to invoke the application. The *problema* window is displayed.

2. Select the desired command from the Modes menu in the control bar. The *problema* window immediately enters the selected mode for performing the command. Note that the command is not executed until you click the *apply command* button, which is to the right of the *Cancel* button in the control bar. Depending on the command selected, the window makes these changes:

   - Fields become writable or read-only.
   - The command name now appears on the Modes menu button and on the *apply command* button.
   - All fields required for the command are highlighted in red.

3. Fill in the request form fields with the appropriate information. Remember that holding down the right mouse button displays a menu with some or all of these items:

Values	Lets you select any valid predefined value.
Clear	Clears the field.
Reuse	Inserts the value from the previous request (if any).
Revert	Restores the original value of the field for that request.

   As the required fields are entered, the highlight disappears. If a field is entered incorrectly, according to the rules for that field, it remains highlighted and must be corrected for the transition to take place. If you don't know the field's requirements, check online help.

4. Click the *apply command* button to perform the operation.

   Note that the *apply command* button is sensitive only when all required fields are filled in. Clicking the button performs the operation and changes the window and database (if affected) accordingly. Any default fields are filled in automatically. The command mode may change as well; for example, after a query, the window switches to display mode.

In addition, e-mail notifications are made to the owner, project manager, submitter, and any other parties indicated in the Notify field. The general format for a “SUBMIT_REQUEST” notification is:

```
Field czar set to: root.
Field category set to: Backup.
Field ENTITY_ID set to: 8.
Field STATE set to: AWAITING_RESPONSE.
Field summary set to: I’d like to schedule regular
backups of my /work file system..
Field bboard set to: root.
Field last_update set to: Thu Jul 7 16:24:53 PDT 1994.
Field due_date set to: Tue Jul 12 16:24:45 PDT 1994.
Field submit_date set to: Thu Jul 7 16:24:45 PDT 1994.
Field submitter set to: grant.
Field notify_list set to: (root, grant).

New value of history:
==========
Date:   Mon Jul 04 16:24:53 1994
User:   grant@vicksburg
==========

SUBMIT_REQUEST

New value of description:
I’d like to schedule regular backups of my /work file
system.

TRANSITION = SUBMIT_REQUEST
EXECUTING /bin/true
```

Specific Procedure for Submitting a Request

To submit a request:

1. From *problema*, select SUBMIT_REQUEST from the Modes menu on the control bar.

   The window makes these changes:

   - The Report # and Status fields turn read-only, since these are set automatically by the system.

- The Description field is highlighted in red, indicating that it is required.
- The *apply command* button changes to read SUBMIT_REQUEST, although the button is not yet enabled.

2. Fill in the Description field, which is required, and any other fields needed to provide as much request information as possible.

   After you make an entry in the Description field and move the cursor out of it, the highlighting is removed. At this point, the *apply command* and *Cancel* buttons are enabled, since you have fulfilled the Description requirement. Click the *apply command* button to submit your request.

Setting the Default Field Editor

Use one of the following methods to set your default editor for fields in any *problema* interface:

- The environment variable $WINEDITOR (for editors like *jot*(1) that bring up their own windows).
- The environment variable $EDITOR (for editors like *vi*(1) that do not supply their own windows).
- The *editorCommand* setting in your home directory's *.Xdefaults* file.

If none of these are set, the editor defaults to *vi*(1).

Assigning and Resolving Requests

Each new request should be reviewed using *suproblema* on the *problema* server, and resolved, deferred, or assigned to an owner. When the assigned requests have been fulfilled by their owners, they are closed (with the RESOLVE command) on the server.

The following options are available only from the *suproblema* Modes menu:

- FORWARD_TO_VENDOR
- ASSIGN
- DEFER
- DUPLICATE

- DELETE

Only authorized users can screen requests. The authorized users are specified in the "*suproblema.pdl*" file. You begin screening by querying and displaying requests, and you can then apply different transitions after the selected request displays.

Screening requests follows all the steps mentioned in the section "Procedure for Performing General Operations" on page 171. The difference is that you deal only with requests in the AWAITING_RESPONSE state.

No Action Commands

The screener has three transition commands for turning down requests:

DEFER	If work on the request is to be performed at a later date. This requires an entry in the Reopen Date field. When this command is used, *problema* automatically reopens the request on the specified date, switching its status from CLOSED to AWAITING_RESPONSE.
DUPLICATE	A request on the same topic has been previously submitted.
RESOLVE	The request has already been performed or is moot.

Assignment Commands

The ASSIGN command lets the screener specify the owner who will be executing the request. The ASSIGN command requires that the Owner field be changed.

The FORWARD_TO_VENDOR command indicates that the problem has been forwarded to an outside vendor of hardware or software for further action.

Forwarding a Request to Silicon Graphics

To automatically forward requests to Silicon Graphics, you must be an authorized Silicon Graphics Support Advantage customer, and you must have the electronic services package installed on your *problema* server. If these services are installed, *problema* uses the */usr/IRIXpro/lib/problematoes* script to contact Silicon Graphics when the FORWARD_TO_VENDOR action is taken and Silicon Graphics is the specified vendor.

When forwarding and updating requests, the user performing the actions is expected to be **irixpro**. See “The User irixpro” on page 151 for more information. This operation is synchronous, which means that suproblema waits on the request until it returns from Silicon Graphics or until the request action fails.

When forwarding requests, the key field is the Technical Description field. All pertinent information must be listed in this field.

Executing and Reopening Requests

After a request has been assigned, it is turned over to the owner for implementation. (If the owner performed the assignment, there is no turnover.) In addition to new requests, the owner may receive rejected resolutions (REOPEN requests). The owner then performs the requested work and when finished, closes the request by making a RESOLVE transition.

Resolving and Closing Requests

To resolve an existing request, you first have to bring it up with a query, as discussed in “Query Operations” on page 169.

When the request is displayed, perform these steps:

1. Select RESOLVE from the Modes menu. The Resolution field is highlighted.
2. Enter the explanation of the fix in the Resolution field.
3. Click the *apply command* button, which now reads RESOLVE. The status changes from AWAITING_RESPONSE to CLOSED.

Deleting Requests

If your database is growing too large for convenient use, you can use the DELETE command to mark closed requests for deletion. The deletion itself can be carried out any time at the administrator’s convenience through the use of the script: */usr/IRIXpro/sample/problema/problemaDBclean*. This script is designed to be run as a job in a non-peak use time through the *cron*(1) utility. It is suggested, however, that you make an archive copy of your database before deleting any closed requests. You can also use the delete operation to mark duplicate requests or mistaken requests that have been closed. The DELETE transition works only on requests with a CLOSED status.

You must be logged in as the user **irixpro** to perform DELETE transitions. See the section in this chapter titled "The User irixpro" on page 151 for complete information on creating the user **irixpro**. Note that you cannot simply use the command:

```
su irixpro
```

to log yourself in momentarily as **irixpro** to perform this transition. You must use one of the following commands:

```
su - irixpro
```

or

```
login irixpro
```

Overdue Requests

Each request submitted with any *problema* interface receives an automatic due date by which it is expected to be closed. The script */usr/IRIXpro/sample/problemaDBdue* generates a list of these overdue requests. This script is designed to be run as a job in a non-peak use time through the *cron*(1) utility. You must be logged in as the user **irixpro** to use this script. See the instructions in the section titled "Deleting Requests" on page 175 for more information on logging in as **irixpro** and the section titled "The User irixpro" on page 151 for more information on creating the **irixpro** user account.

The hp-ux_sgi MIB

The following is a copy, for reference, of the *hp-ux_sgi* MIB distributed with the *snmpd* package of IRIXpro in the file */usr/lib/netvis/mibs/hp-ux_sgi.mib*:

```
-- HP-UNIX DEFINITIONS ::= BEGIN

-- --
-- --  Hewlett-Packard definitons for unix agents
-- --  @(#) hp-unix $Date: 1994/11/02 01:03:49 $
-- --  $Revision: 1.3 $
-- -- ---------------------------------------------------
-- -- This MIB was editied by SGI to
-- -- specify which MIB objects
-- -- are supported in the SGI agent.
-- -- Only the object descriptions
-- -- were changed. No objects were removed or added.
-- -- Silicon Graphics, Inc., October 1994
-- -- ---------------------------------------------------
-- --

--IMPORTS
--enterprises, NetworkAddress, IpAddress, Counter, Gauge,
--                TimeTicks
--            FROM RFC1155-SMI
--        OBJECT-TYPE
--            FROM RFC-1212
--        DisplayString
--            FROM RFC1213-MIB;
--
hp               OBJECT IDENTIFIER ::= { enterprises 11 }
nm               OBJECT IDENTIFIER ::= { hp 2 }
system           OBJECT IDENTIFIER ::= { nm 3 }
interface        OBJECT IDENTIFIER ::= { nm 4 }
icmp             OBJECT IDENTIFIER ::= { nm 7 }
snmp             OBJECT IDENTIFIER ::= { nm 13 }
openView         OBJECT IDENTIFIER ::= { nm 17 }
```

```
general          OBJECT IDENTIFIER ::= { system 1 }
hpux             OBJECT IDENTIFIER ::= { system 2 }
hpsun            OBJECT IDENTIFIER ::= { system 10 }
sparc            OBJECT IDENTIFIER ::= { hpsun 1 }
computerSystem   OBJECT IDENTIFIER ::= { general 1 }
fileSystem       OBJECT IDENTIFIER ::= { general 2 }
processes        OBJECT IDENTIFIER ::= { general 4 }
cluster          OBJECT IDENTIFIER ::= { general 5 }
ieee8023Mac      OBJECT IDENTIFIER ::= { interface 1 }
trap             OBJECT IDENTIFIER ::= { snmp 1 }
snmpdConf        OBJECT IDENTIFIER ::= { snmp 2 }
authfail         OBJECT IDENTIFIER ::= { snmp 4 }
openViewTrapVars OBJECT IDENTIFIER ::= { openView 2 }

-- -- sysObjectId definitions.
-- -- These values are returned in the
-- -- .iso.org.dod.internet.mgmt.mib-2.system.sysObjectID.0
-- -- variable

hp386             OBJECT IDENTIFIER ::= { system 8 }
hp9000s300        OBJECT IDENTIFIER ::= { hpux 2 }
hp9000s800        OBJECT IDENTIFIER ::= { hpux 3 }
hp9000s700        OBJECT IDENTIFIER ::= { hpux 5 }
hpOpenView        OBJECT IDENTIFIER ::= { openView 1 }
sun4              OBJECT IDENTIFIER ::= { sparc 1 }
sun5              OBJECT IDENTIFIER ::= { sparc 2 }

-- -- the ComputerSystem Group

computerSystemUpTime OBJECT-TYPE
        SYNTAX        TimeTicks
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
               “Time since the last boot.
                Supported by SGI.”
        ::= { computerSystem 1 }
```

```
computerSystemUsers OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “Number of users logged on to system.
                Supported by SGI.”
        ::= { computerSystem 2 }

computerSystemAvgJobs1 OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “Average number of jobs in the last 1
                minute * 100.
                Supported by SGI.”
        ::= { computerSystem 3 }

computerSystemAvgJobs5 OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “Average number of jobs in the last 5
                minutes * 100.
                Supported by SGI.”
        ::= { computerSystem 4 }

computerSystemAvgJobs15 OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         andatory
        DESCRIPTION
                “Average number of jobs in the last 15
                minutes * 100.
                Supported by SGI.”
        ::= { computerSystem 5 }
```

```
computerSystemMaxProc OBJECT-TYPE
        SYNTAX         INTEGER
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “Maximum number of processes allowed in
                system.Implemented with Extensible SNMP
                Agent. Supported by SGI.”
        ::= { computerSystem 6 }

computerSystemFreeMemory OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “Free memory.  SunOS - not implemented.
                 Implemented with Extensible SNMP Agent.
                 Supported by SGI.”
        ::= { computerSystem 7 }

computerSystemPhysMemory OBJECT-TYPE
        SYNTAX         INTEGER
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “Physical memory.  SunOS - not implemented.
                 Implemented with Extensible SNMP Agent.
                 Supported by SGI.”
        ::= { computerSystem 8 }

computerSystemMaxUserMem OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “Maximum user memory.SunOS-not implemented.
                 Implemented with Extensible SNMP Agent.
                 Supported by SGI.”
        ::= { computerSystem 9 }

computerSystemSwapConfig OBJECT-TYPE
        SYNTAX         INTEGER
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
```

```
                “Swap space configured.SunOS-not implemented.
                 Implemented with Extensible SNMP Agent.
                 Supported by SGI.”
        ::= { computerSystem 10 }

computerSystemEnabledSwap OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “Enabled via swapon.SunOS - not implemented.
                 Implemented with Extensible SNMP Agent.
                 Not supported by SGI because not
                 applicable.”
        ::= { computerSystem 11 }

computerSystemFreeSwap OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “Currently free swap space.SunOS - not
                 implemented.
                 Implemented with Extensible SNMP Agent.
                 Supported by SGI.”
        ::= { computerSystem 12 }

computerSystemUserCPU OBJECT-TYPE
        SYNTAX         Counter
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “CPU used by users.  SunOS - not implemented.
                 Implemented with Extensible SNMP Agent.
                 Supported by SGI.”
        ::= { computerSystem 13 }
```

```
computerSystemSysCPU OBJECT-TYPE
        SYNTAX         Counter
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “CPU used by system.SunOS - not implemented.
                 Implemented with Extensible SNMP Agent.
                 Supported by SGI.”
        ::= { computerSystem 14 }

computerSystemIdleCPU OBJECT-TYPE
        SYNTAX         Counter
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “CPU idle.  SunOS - not implemented.
                 Implemented with Extensible SNMP Agent.
                 Supported by SGI.”
        ::= { computerSystem 15 }

computerSystemNiceCPU OBJECT-TYPE
        SYNTAX         Counter
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “CPU nice.  SunOS - not implemented.
                 Implemented with Extensible SNMP Agent.
                 Not supported by SGI.”
        ::= { computerSystem 16 }

-- -- The FileSystem Group

fileSystemMounted OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “The number of file systems mounted.
                 Supported by SGI.”
        ::= { fileSystem 1 }
```

```
-- -- The FileSystem Table

fileSystemTable OBJECT-TYPE
        SYNTAX         SEQUENCE OF FileSystemEntry
        ACCESS         not-accessible
        STATUS         mandatory
        DESCRIPTION
                “File system table.
                 Supported by SGI.”
        ::= { fileSystem 2 }

fileSystemEntry OBJECT-TYPE
        SYNTAX         FileSystemEntry
        ACCESS         not-accessible
        STATUS         mandatory
        DESCRIPTION
                “Each entry contains objects for a
                 particular file system.
                 Supported by SGI.”
        INDEX         { fileSystemID1, fileSystemID2 }
        ::= { fileSystemTable 1 }

FileSystemEntry ::= SEQUENCE {
    fileSystemID1
        INTEGER,
    fileSystemID2
        INTEGER,
    fileSystemName
        DisplayString,
    fileSystemBlock
        INTEGER,
    fileSystemBfree
        INTEGER,
    fileSystemBavail
        INTEGER,
    fileSystemBsize
        INTEGER,
    fileSystemFiles
        INTEGER,
    fileSystemFfree
        INTEGER,
    fileSystemDir
        DisplayString
}
```

```
fileSystemID1 OBJECT-TYPE
        SYNTAX        INTEGER (SIZE (0..4294967295))
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “First file system ID.
                 Supported by SGI.”
        ::= { fileSystemEntry 1 }

fileSystemID2 OBJECT-TYPE
        SYNTAX        INTEGER (SIZE (0..4294967295))
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Second file system ID.
                 Supported by SGI. Always set to 1.”
        ::= { fileSystemEntry 2 }

fileSystemName OBJECT-TYPE
        SYNTAX        DisplayString
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Name of mounted file system.
                 Supported by SGI.”
        ::= { fileSystemEntry 3 }

fileSystemBlock OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Total blocks in file system.
                 Supported by SGI.”
        ::= { fileSystemEntry 4 }

fileSystemBfree OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Free blocks in file system.
                 Supported by SGI.”
        ::= { fileSystemEntry 5 }
```

```
fileSystemBavail OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Free blocks avail to non-superuser.
                 Supported by SGI.”
        ::= { fileSystemEntry 6 }

fileSystemBsize OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Fundamental file system block size.
                 Supported by SGI.”
        ::= { fileSystemEntry 7 }

fileSystemFiles OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Total file nodes in file system.
                 Supported by SGI.”
        ::= { fileSystemEntry 8 }

fileSystemFfree OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Free file nodes in file system.
                 Supported by SGI.”
        ::= { fileSystemEntry 9 }

fileSystemDir OBJECT-TYPE
        SYNTAX        DisplayString
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “File system path prefix.
                 Supported by SGI.”
        ::= { fileSystemEntry 10 }
```

```
-- -- The Processes Group

processNum OBJECT-TYPE
        SYNTAX         Gauge
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                “The number of processes running.
                 Supported by SGI.”
        ::= { processes 1 }

-- -- The processes table

processTable OBJECT-TYPE
        SYNTAX        SEQUENCE OF ProcessEntry
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “Processes Table.
                 Implemented with Extensible SNMP Agent.
                 Supported by SGI.”
        ::= { processes 2 }

processEntry        OBJECT-TYPE
        SYNTAX        ProcessEntry
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “Each entry contains information about a
                 process running on the system.
                 Supported by SGI.”
        INDEX        { processPID }
        ::= { processTable 1 }

ProcessEntry ::= SEQUENCE {
    processPID
        INTEGER,
    processIdx
        INTEGER,
    processUID
        INTEGER,
    processPPID
        INTEGER,
    processDsize
        Gauge,
processTsize
    Gauge,
processSsize
    Gauge,
processNice
    Gauge,
processMajor
    INTEGER,
processMinor
    INTEGER,
processPgrp
    INTEGER,
processPrio
    INTEGER,
processAddr
    INTEGER,
processCPU
    Gauge,
processUtime
    TimeTicks,
processStime
    TimeTicks,
processStart
    TimeTicks,
processFlags
    INTEGER,
processStatus
    INTEGER,
processWchan
    INTEGER,
processProcNum
    INTEGER,
processCmd
    DisplayString,
processTime
    INTEGER,
processCPUticks
    Counter,
processCPUticksTotal
    Counter,
processFss
    INTEGER,
processPctCPU
    Gauge,
    processRssize
        Gauge,
    processSUID
        INTEGER,
    processUname
        DisplayString,
    processTTY
        DisplayString
}

processPID OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "The process ID (pid).
                 Supported by SGI."
        ::= { processEntry 1 }

processIdx OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Index for pstat() requests.SunOS -
                 not implemented.
                 Not supported by SGI."
        ::= { processEntry 2 }

processUID OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Process User ID.
                 Supported by SGI."
        ::= { processEntry 3 }

processPPID OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Parent process ID.
                 Supported by SGI."
        ::= { processEntry 4 }

processDsize OBJECT-TYPE
        SYNTAX        Gauge
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Process data size.
                 Not supported by SGI in this release."
        ::= { processEntry 5 }

processTsize OBJECT-TYPE
        SYNTAX        Gauge
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Process text size.
                 Not supported by SGI in this release."
        ::= { processEntry 6 }

processSsize OBJECT-TYPE
        SYNTAX        Gauge
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Process stack size.
                 Not supported by SGI in this release."
        ::= { processEntry 7 }

processNice OBJECT-TYPE
        SYNTAX        Gauge
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Process nice value.
                 Supported by SGI."
        ::= { processEntry 8 }

processMajor OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Process tty major number.
                 SunOS - not implemented.
```

```
                  Supported by SGI.”
        ::= { processEntry 9 }

processMinor OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Process tty minor number.
                 SunOS - not implemented.
                 Supported by SGI.”
        ::= { processEntry 10 }

processPgrp OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Process group of this process.
                 Supported by SGI.”
        ::= { processEntry 11 }

processPrio OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Process priority.
                 Supported by SGI.”
        ::= { processEntry 12 }

processAddr OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Address of process (in memory).
                 Supported by SGI.”
        ::= { processEntry 13 }
```

```
processCPU OBJECT-TYPE
        SYNTAX        Gauge
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Processor utilization for scheduling.
                 Supported by SGI.”
        ::= { processEntry 14 }

processUtime OBJECT-TYPE
        SYNTAX        TimeTicks
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “User time spent executing.
                 Supported by SGI.”
        ::= { processEntry 15 }

processStime OBJECT-TYPE
        SYNTAX        TimeTicks
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “System time spent executing.
                 Supported by SGI.”
        ::= { processEntry 16 }

processStart OBJECT-TYPE
        SYNTAX        TimeTicks
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Time Process started.
                 Supported by SGI.”
        ::= { processEntry 17 }
```

```
processFlags OBJECT-TYPE
        SYNTAX        INTEGER {
                incore(1),
                sys(2),
                locked(4),
                trace(8),
                trace2(16)
                }
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Flags associated with process.
                 SunOS - values found in
                /usr/include/sys/proc.h.
                 Supported by SGI.”
        ::= { processEntry 18 }

processStatus OBJECT-TYPE
        SYNTAX        INTEGER {
                sleep(1),
                run(2),
                stop(3),
                zombie(4),
                other(5),
                idle(6)
                }
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The process status.
                 SunOS - sleep(1), wait(2), run(3),
                 idle (4), zombie(5), stop(6)
                 Supported by SGI.”
        ::= { processEntry 19 }

processWchan OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “If processStatus is sleep,
                 value sleeping on.
                 Supported by SGI.”
        ::= { processEntry 20 }
```

```
processProcNum OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Processor this process last run on.
                 SunOS - not implemented.
                 Not supported by SGI in this release."
        ::= { processEntry 21 }

processCmd OBJECT-TYPE
        SYNTAX        DisplayString
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Command the process is running.
                 Supported by SGI."
        ::= { processEntry 22 }

processTime OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Resident time for scheduling.
                 Not supported by SGI."
        ::= { processEntry 23 }

processCPUticks OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Ticks of cpu time.
                 Supported by SGI."
        ::= { processEntry 24 }
```

```
processCPUticksTotal OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Total ticks for life of process.
                 SunOS - not implemented.
                 Supported by SGI.”
        ::= { processEntry 25 }

processFss OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Fair Share Schedular Group.
                 SunOS - not implemented.
                 Not supported by SGI.”
        ::= { processEntry 26 }

processPctCPU OBJECT-TYPE
        SYNTAX        Gauge
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Percent CPU * 100 for this process.
                 Supported by SGI.”
        ::= { processEntry 27 }

processRssize OBJECT-TYPE
        SYNTAX        Gauge
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Resident Set Size for process
                 (private pages).
                 Supported by SGI.”
        ::= { processEntry 28 }
```

```
processSUID OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “saved UID.
                 Supported by SGI.”
        ::= { processEntry 29 }

processUname OBJECT-TYPE
        SYNTAX        DisplayString
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “User name.
                 Supported by SGI.”
        ::= { processEntry 30 }

processTTY OBJECT-TYPE
        SYNTAX        DisplayString
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Process TTY.  SunOS - not implemented.
                 Not supported by SGI in this release.”
        ::= { processEntry 31 }

-- -- The Ieee8023Mac Group
-- --         This group is not implemented on Solaris.
-- --         This group is not implemented on SGI platforms.

-- -- The Ieee8023Mac Table

ieee8023MacTable OBJECT-TYPE
        SYNTAX        SEQUENCE OF Ieee8023MacEntry
        ACCESS         not-accessible
        STATUS        mandatory
        DESCRIPTION
                “A list of IEEE 802.3 Interface entries.
                 Not applicable to SGI.”
        ::= { ieee8023Mac 1 }

ieee8023MacEntry OBJECT-TYPE
        SYNTAX        Ieee8023MacEntry
        ACCESS         not-accessible
```

```
            STATUS        mandatory
            DESCRIPTION
                    “Each entry contains statistics for
                     ieee 802.3 interfaces.
                     Not applicable to SGI.”
            INDEX        { ieee8023MacIndex }
            ::= { ieee8023MacTable 1 }

    Ieee8023MacEntry ::= SEQUENCE {
        ieee8023MacIndex
            INTEGER,
        ieee8023MacTransmitted
            Counter,
        ieee8023MacNotTransmitted
            Counter,
        ieee8023MacDeferred
            Counter,
        ieee8023MacCollisions
            Counter,
        ieee8023MacSingleCollisions
            Counter,
        ieee8023MacMultipleCollisions
            Counter,
        ieee8023MacExcessCollisions
            Counter,
        ieee8023MacLateCollisions
            Counter,
        ieee8023MacCarrierLostErrors
            Counter,
        ieee8023MacNoHeartBeatErrors
            Counter,
        ieee8023MacFramesReceived
            Counter,
        ieee8023MacUndeliverableFramesReceived
            Counter,
        ieee8023MacCRCErrors
            Counter,
        ieee8023MacAlignmentErrors
            Counter,
        ieee8023MacResourceErrors
            Counter,
        ieee8023MacControlFieldErrors
            Counter,
        ieee8023MacUnknownProtocolErrors
            Counter,
    ieee8023MacMulticastsAccepted
        Counter
}

ieee8023MacIndex OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The index value that uniquely identifies the
                 interface to which this entry is
                 applicable. The interface identified by a
                 particular value of this index is the same
                 interface as identified by the
                 same value of ifIndex.
                 Not applicable to SGI.”
        ::= { ieee8023MacEntry 1 }

ieee8023MacTransmitted OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The number of frames successfully
                 transmitted.
                 Not applicable to SGI.”
        ::= { ieee8023MacEntry 2 }

ieee8023MacNotTransmitted OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The number of frames not transmitted.
                 Not applicable to SGI.”
        ::= { ieee8023MacEntry 3 }
```

```
ieee8023MacDeferred OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "The number of frames deferred because the
                 medium was busy.
                 Not applicable to SGI."
        ::= { ieee8023MacEntry 4 }

ieee8023MacCollisions OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Total number of transmit attempts that were
                 retransmitted due to collisions.  SunOS
                 with Intel 82586
                 Ethernet driver - total number of
                 collisions.
                 Not applicable to SGI."
        ::= { ieee8023MacEntry 5 }

ieee8023MacSingleCollisions OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Number of transmit attempts that
                 are involved in a
                 single collision and are
                 subsequently transmitted
                 successfully.  SunOS - this is always 0.
                 Not applicable to SGI."
        ::= { ieee8023MacEntry 6 }
```

```
ieee8023MacMultipleCollisions OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of transmit attempts that
                are involved in between 2 and 15
                collision attempts and are
                subsequently transmitted successfully.
                SunOS with Intel 82586 is always 0.
                SunOS with AMD 7990 LANCE driver -
                number of transmit attempts that are
                involved in between 1 and 15 collision
                attempts and are subsequently
                transmitted successfully.”
        ::= { ieee8023MacEntry 7 }

ieee8023MacExcessCollisions OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of transmit attempts that are
                 involved in more than 15 collision
                 attempts and are subsequently
                 transmitted successfully.  SunOS with
                 Intel 82586 Ethernet driver - this is
                 always 0.”
        ::= { ieee8023MacEntry 8 }

ieee8023MacLateCollisions OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of transmit attempts aborted
                 because a collision occurred after the
                 allotted channel time had elapsed.”
        ::= { ieee8023MacEntry 9 }
```

```
ieee8023MacCarrierLostErrors OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of times that carrier sense was lost
                when attempting to transmit.”
        ::= { ieee8023MacEntry 10 }

ieee8023MacNoHeartBeatErrors OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of times no heart beat was indicated
                after a transmission.”
        ::= { ieee8023MacEntry 11 }

ieee8023MacFramesReceived OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of frames successfully received.”
        ::= { ieee8023MacEntry 12 }

ieee8023MacUndeliverableFramesReceived OBJECT-TYPE
        SYNTAX        Counter
        ACCESS         read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of frames received that were not
                 delivered because the software buffer was
                 overrun when frames were sent faster than
                 they could be received.”
        ::= { ieee8023MacEntry 13 }

ieee8023MacCRCErrors OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of Cyclical Redundancy
                 Check (CRC) errors detected.”
        ::= { ieee8023MacEntry 14 }
```

```
ieee8023MacAlignmentErrors OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of frames received that were both
                 misaligned and had bad CRC.”
        ::= { ieee8023MacEntry 15 }

ieee8023MacResourceErrors OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of frames received that were lost
                 due to lack of resources.”
        ::= { ieee8023MacEntry 16 }

ieee8023MacControlFieldErrors OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of frames received with errors in
                the control field. SunOS - this is always 0.”
        ::= { ieee8023MacEntry 17 }

ieee8023MacUnknownProtocolErrors OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of frames dropped because
                 the type field or sap field referenced
                 an invalid protocol.  SunOS - this is
                 always 0.”
        ::= { ieee8023MacEntry 18 }
```

```
ieee8023MacMulticastsAccepted OBJECT-TYPE
        SYNTAX        Counter
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Number of accepted multicast addresses.”
        ::= { ieee8023MacEntry 19 }

-- -- The Icmp Group

icmpEchoReq OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The number of milliseconds it takes for an
                ICMP echo request to respond. IcmpEchoReq is
                    -1 if there is an internal error,
                    -2 if the echo request timed out,
                    -3 if the echo reply is not the correct
                       reply,
                    -4 if the packet size is too large, and
                    -5 if the timeout is invalid.

                To request the ICMP response time for IP
                address a1.a2.a3.a4 with a timeout of t
                and a packet size of s, send a request for
                icmpEchoReq.s.t.a1.a2.a3.a4.

                For example, suppose one wanted to find out
                the number of milliseconds it took to ping
                15.2.112.113, with time out of 8 seconds,
                and packet size of 75.  Accordingly,
                icmpEchoReq.75.8.15.2.112.113 would identify
                the number of milliseconds.
                Not supported by SGI in this release.”
        ::= { icmp 1 }
```

```
-- -- The Trap Group

trapDestinationNum OBJECT-TYPE
        SYNTAX        Gauge
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "The number of trap destinations.
                 Supported by SGI."
        ::= { trap 1 }

-- -- The Trap Destination Table

trapDestinationTable OBJECT-TYPE
        SYNTAX        SEQUENCE OF TrapDestinationEntry
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                "List of addresses to which
                 the agent sends traps.
                 Supported by SGI."
        ::= { trap 2 }

trapDestinationEntry OBJECT-TYPE
        SYNTAX        TrapDestinationEntry
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                "Each entry contains the address of
                 a management station.
                 Supported by SGI."
        INDEX        { trapDestination }
        ::= { trapDestinationTable 1 }

TrapDestinationEntry ::= SEQUENCE {
    trapDestination
        NetworkAddress
}
```

```
trapDestination OBJECT-TYPE
        SYNTAX        NetworkAddress
        ACCESS        read-write
        STATUS        mandatory
        DESCRIPTION
                “Address to which the agent sends traps.
                 Supported by SGI only as read-only since
                 SGI offers more granularity to specify
                 where traps should be sent. It gives the
                 entries found in /etc/snmpd.trap.conf”
            ::= { trapDestinationEntry 1 }

-- -- The SnmpdConf Group

snmpdConfRespond OBJECT-TYPE
        SYNTAX        INTEGER {
                    true(1),
                    false(2)
                }
        ACCESS        read-write
        STATUS        mandatory
        DESCRIPTION
                “The SNMP agent was configured to
                 respond to all objects if
                 snmpdConfRespond is true.  HP-Internal
                 use only.
                 Not supported by SGI.”
        ::= { snmpdConf 1 }

snmpdReConfigure OBJECT-TYPE
        SYNTAX        INTEGER {
                    reset(1)
                }
        ACCESS        read-write
        STATUS        mandatory
        DESCRIPTION
                “The agent will re-configure itself if
                 snmpdReConfigure is set to reset(1)
                 Not supported by SGI since the hpsnmpd
                 subagent has no configuration file per se,
                 except for the trapDestinationTable which
                 will be supported as read-write in the
                 next release.”
        ::= { snmpdConf 2 }
```

```
snmpdFlag OBJECT-TYPE
        SYNTAX        INTEGER {
                    removetrap(1),
                    netwareproxy(2)
                }
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "Indicates the capability of the agent.
                 Supported by SGI as returning 1 always."
        ::= { snmpdConf 3 }

snmpdLogMask        OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-write
        STATUS        mandatory
        DESCRIPTION
                "The agent's log mask.
                 Supported by SGI.
                 Corresponds to the -l command
                 line option passed to hpsnmpd."
        ::= { snmpdConf 4 }

snmpdVersion        OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                "The agent's version number.
                 Supported by SGI. Version 1.0.0 (100)
                 in this release."
        ::= { snmpdConf 5 }

snmpdStatus OBJECT-TYPE
        SYNTAX        INTEGER {
                    up(1),
                    down(2)
                }
        ACCESS        read-write
        STATUS        mandatory
        DESCRIPTION
                "Indicates the status of the agent.  Setting
                 the agent to down will kill it.
                 Supported by SGI."
        ::= { snmpdConf 6 }
```

```
snmpdSize OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The amount of memory the
                 agent has allocated.
                 Supported by SGI. Unit is in pages.
                 Each page is 4096 bytes.”
        ::= { snmpdConf 7 }

snmpdWhatString OBJECT-TYPE
        SYNTAX        DisplayString
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The what string of the agent.
                 Supported by SGI.”
        ::= { snmpdConf 9 }

-- -- The Cluster Group
-- --
-- --         This group is not implemented on SunOS/Solaris.
-- --         This group is not implemented on SGI platforms.

isClustered        OBJECT-TYPE
        SYNTAX        INTEGER {
                standalone(1),
                rootserver(2),
                cnode(3)
            }
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “Describes whether machine
                 is clustered or not.”
        ::= { cluster 1 }

clusterTable        OBJECT-TYPE
        SYNTAX        SEQUENCE OF ClusterEntry
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “List of nodes on the cluster.”
        ::= { cluster 2 }

clusterEntry OBJECT-TYPE
        SYNTAX         ClusterEntry
        ACCESS         not-accessible
        STATUS         mandatory
        DESCRIPTION
                "Each entry contains information
                 about the clustered node."
        INDEX {clusterID }
        ::= { clusterTable 1 }

ClusterEntry ::= SEQUENCE {
    clusterID
        INTEGER,
    clusterMachineID
        OCTET STRING,
    clusterType
        DisplayString,
    clusterCnodeName
        DisplayString,
    clusterSwapServingCnode
        INTEGER,
    clusterKcsp
        INTEGER,
    clusterCnodeAddress
        IpAddress
}

clusterID OBJECT-TYPE
        SYNTAX         INTEGER
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                "The cnode id."
            ::= { clusterEntry 1 }

clusterMachineID OBJECT-TYPE
        SYNTAX         OCTET STRING
        ACCESS         read-only
        STATUS         mandatory
        DESCRIPTION
                "The cnode machine id."
            ::= { clusterEntry 2 }
```

```
clusterType OBJECT-TYPE
        SYNTAX        DisplayString
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The cnode type (r or c).”
            ::= { clusterEntry 3 }

clusterCnodeName OBJECT-TYPE
        SYNTAX        DisplayString
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The cnode name.”
            ::= { clusterEntry 4 }

clusterSwapServingCnode OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The swap serving cnode.”
            ::= { clusterEntry 5 }

clusterKcsp OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “KCSP.”
            ::= { clusterEntry 6 }

clusterCnodeAddress OBJECT-TYPE
        SYNTAX        IpAddress
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The cnode IP Address.”
            ::= { clusterEntry 7 }
```

```
clusterCnodeID        OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The machine’s cnodes id”
            ::= { cluster 3 }

-- -- The AuthFail Group

authFailTable OBJECT-TYPE
        SYNTAX        SEQUENCE OF AuthFailEntry
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “List of managers that caused
                 an authentication failure.
                 This list has a maximum size.
                 Supported by SGI.”
        ::= { authfail 1 }

authFailEntry OBJECT-TYPE
        SYNTAX        AuthFailEntry
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “Each entry contains the ip
                 address of the management station.
                 Supported by SGI.”
        INDEX        { authIpAddress }
        ::= { authFailTable 1 }

AuthFailEntry ::= SEQUENCE {
    authIpAddress
        NetworkAddress,
    authTime
        TimeTicks,
    authCommunityName
        OCTET STRING
}

authIpAddress OBJECT-TYPE
        SYNTAX        NetworkAddress
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The ip address of the management
                 station that sent a
                 request to the agent with
                 an incorrect community name.
                 Supported by SGI.”
            ::= { authFailEntry 1 }

authTime OBJECT-TYPE
        SYNTAX        TimeTicks
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The time since the agent received
                 the un-authenticated request.
                 Supported by SGI.”
            ::= { authFailEntry 2 }

authCommunityName OBJECT-TYPE
        SYNTAX        OCTET STRING
        ACCESS        read-only
        STATUS        mandatory
        DESCRIPTION
                “The community name used in the request
                 Supported by SGI.”
            ::= { authFailEntry 3 }
```

```
-- -- the OpenView trap variables Group
-- -- These object ID’s cannot be retrieved
-- -- from the SNMP Agent, but
-- -- instead document the objects sent
-- -- with OpenView SNMP traps
-- --
-- --   This group is not implemented on SGI platforms.

openViewSourceId OBJECT-TYPE
        SYNTAX        INTEGER
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “The identifier of the software generating
                 the trap/event. This number is used by HP
                 OpenView software when it sends an event to
                 trapd.  It identifies which software
                 component sent the event. This object
                 cannot be retrieved from the SNMP agent.”
        ::= { openViewTrapVars 1 }

openViewSourceName OBJECT-TYPE
        SYNTAX        OCTET STRING
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “The source of the event (may not be the
                 machine upon which the event was
                 generated).  This string is used by HP
                 OpenView software when it sends an event.
                 It identifies for which source (node) the
                 event is generated. This object cannot be
                 retrieved from the SNMP agent.”
        ::= { openViewTrapVars 2 }
```

```
openViewObjectId OBJECT-TYPE
        SYNTAX        OCTET STRING
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “The OpenView object identifier associated
                 with the source of the trap/event. This
                 object cannot be retrieved from the SNMP
                 agent.”
        ::= { openViewTrapVars 3 }

openViewData OBJECT-TYPE
        SYNTAX        OCTET STRING
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “Any miscellaneous data sent with an
                 OpenView trap/event. This object cannot
                 be retrieved from the SNMP agent.”
        ::= { openViewTrapVars 4 }

openViewSeverity OBJECT-TYPE
        SYNTAX        OCTET STRING
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “The OpenView event severity associated with
                 the trap/event.This object cannot be
                 retrieved from the SNMP agent.”
        ::= { openViewTrapVars 5 }

openViewCategory OBJECT-TYPE
        SYNTAX        OCTET STRING
        ACCESS        not-accessible
        STATUS        mandatory
        DESCRIPTION
                “The OpenView event category associated with
                 the trap/event. This object cannot be
                 retrieved from the SNMP agent.”
        ::= { openViewTrapVars 6 }

END
```

Index

We'd Like to Hear From You

As a user of Silicon Graphics documentation, your comments are important to us. They help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics to comment on:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please include the title and part number of the document you are commenting on. The part number for this document is 007-2446-001.

Thank you!

Three Ways to Reach Us

The **postcard** opposite this page has space for your comments. Write your comments on the postage-paid card for your country, then detach and mail it. If your country is not listed, either use the international card and apply the necessary postage or use electronic mail or FAX for your reply.

If **electronic mail** is available to you, write your comments in an e-mail message and mail it to either of these addresses:

- If you are on the Internet, use this address: techpubs@sgi.com
- For UUCP mail, use this address through any backbone site: *[your_site]*!sgi!techpubs

You can forward your comments (or annotated copies of manual pages) to Technical Publications at this **FAX** number:

415 965-0964