Paragon ™
System

Commands Reference Manual

April 1996

Order Number: 312486-005

Paragon™ System Commands Reference Manual

Intel® Corporation

WARNING

Some of the circuitry inside this system operates at hazardous energy and electric shock voltage levels. To avoid the risk of personal injury due to contact with an energy hazard, or risk of electric shock, do not enter any portion of this system unless it is intended to be accessible without the use of a tool. The areas that are considered accessible are the outer enclosure and the area just inside the front door when all of the front panels are installed, and the front of the diagnostic station. There are no user serviceable areas inside the system. Refer any need for such access only to technical personnel that have been qualified by Intel Corporation.

CAUTION

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

LIMITED RIGHTS

Preface

This manual describes the Paragon™ system commands. The system commands let you run applications and manage partitions. You issue system commands at your shell prompt.

In this manual, "operating system" refers to the operating system that runs on the nodes of the Paragon™ supercomputer.

This manual assumes you are proficient with the use of the operating system.

Organization

This manual contains a "manual page" for each system command, organized alphabetically. Each manual page provides the following information:

- Command syntax including all switches and arguments.
- Descriptions of all switches and arguments.
- Description of what the command does (including hints on using the command).
- Examples of using the command.
- List of related commands.

Notational Conventions

This section describes the following notational conventions:

- Type style usage.
- Command syntax descriptions.

Type Style Usage

The text of this manual uses the following type style conventions:

Bold	Identifies command names and switches, system call names, reserved words, and other items that must be used exactly as shown.
Italic	Identifies variables, filenames, directories, partitions, and user names. Italic type style is also occasionally used to emphasize a word or phrase.
`Plain-Monospace`	Identifies computer output (prompts and messages), examples, and values of variables.
`Bold-Italic-Monospace`	Identifies user input (what you enter in response to some prompt).
`Bold-Monospace`	Identifies the names of keyboard keys (which are also enclosed in angle brackets). A dash indicates that the key preceding the dash is to be held down *while* the key following the dash is pressed. For example:

`<Break>` **`<s>`** **`<Ctrl-Alt-Del>`**

Command Syntax Descriptions

In this manual, the syntax of each system command is described in the "Syntax" section of the command's manual page. The following notational conventions apply to these syntax descriptions:

Bold	Identifies command names and switches (i.e., items that you must use exactly as shown).
Italic	Identifies arguments (that is, items whose values you must supply when you invoke the command).
[ ]	(Brackets) Surround optional items (that is, items that can be omitted).
|	(Vertical bar) Separates two or more items of which you may select *only* one.
{ }	(Braces) Surround two or more items (separated by vertical bars) of which you *must* select only one.
...	(ellipsis) Indicates that the previous item may be repeated.

For example, consider the syntax description of the **mkpart** command:

mkpart [-sz *size* **| -sz** *h***X***w* **| -nd** *nodespec*] [**-ss** | [[**-sps** | **-rq** *time*] [**-epl** *priority*]]] [**-mod** *mode*] *partition*

This syntax description shows the following:

- You may choose one of **-sz** *size*, **-sz** *h***X***w*, or **-nd** *nodespec.*
- You may choose **-ss**. If you do not choose **-ss**, you may choose one or both of **-rq** *time* and **-epl** *priority.*
- You may choose **-mod** *mode.*
- The *partition* argument is required. It is in italics because it is a variable name.

Applicable Documents

For more information about the Paragon system documentation, refer to the *Paragon™ System Technical Documentation Guide.*

Comments and Assistance

Intel Scalable Systems Division is eager to hear of your experiences with our products. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
Phone: 800-421-2823
Internet: support@ssd.intel.com

United Kingdom Intel Corporation (UK) Ltd.
Scalable Systems Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056
(44) 793 431062
(44) 793 480874
(44) 793 495108

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

Intel Japan K.K.
Scalable Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Scalable Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 677-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 677-9147

If you have comments about our manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com
(Internet)

Table of Contents

Appendix A Files Manual Pages

ALLOCATOR ALLOCATOR

Starts and stops the allocator daemon which allocates nodes and controls access to partitions on a Paragon system.

Syntax

allocator start | stop

Arguments

start	Starts the allocator daemon.
stop	Stops the allocator daemon.

Description

You can execute the **allocator** command to start or stop the allocator daemon. You must be *root* to use this command.

The allocator daemon handles requests to create partitions, remove partitions, or change partition characteristics. It allocates nodes for partitions in the *.compute* partition and controls how applications load and execute in the partitions. The allocator daemon starts when the system boots in multiuser mode and runs in the *.service* partition of a Paragon system.

The **allocator start** command does the following:

- Copies the allocator log file */etc/nx/allocator.log* to */etc/nx/allocator.log.last.*
- Reads the allocator configuration file */etc/nx/allocator.config*.
- Starts the allocator daemon.
- Logs allocator internal errors in the file */etc/nx/allocator.log*.
- Reads in the file */etc/nx/.badnodes* and removes the bad nodes from the set of nodes available in the root partition.

Configuration File

The **allocator start** command reads the configuration file */etc/nx/allocator.config* for configuration information. Configuration strings within this file allow you to enable or disable gang-scheduling, set the number of gang-scheduled partitions in the system, specify the maximum depth to which subpartitions or active entities can overlap in a gang-scheduled partition, specify the minimum permissible rollin quantum, specify whether the application switch **-plk** can be used in a gang-scheduled partition, and specify whether the allocator must validate user accounts with the Paragon Multi-User Accounting and Control System (MACS).

The configuration strings are as follows:

- *SPACE_SHARE=boolean*
- *NUM_GANG_PARTS=integer*
- *DEGREE_OF_OVERLAP=num*
- MIN_RQ_ALLOWED=*time*
- *REJECT_PLK=boolean*
- *USE_MACS=boolean*

For detailed information on the configuration strings in *allocator.config*, see the *allocator.config* manual page.

Space Sharing

When you specify space sharing, the allocator daemon denies requests to create a partition or load an application that would overlap with an existing partition. Overlap is defined as follows:

- Partitions overlap each other if the intersection of their set of nodes is not empty.
- Applications overlap each other if the intersection of their set of nodes is not empty.
- A partition overlaps an application if the intersection of their set of nodes is not empty and the partition contains at least one application.

ALLOCATOR *(cont.)* ALLOCATOR *(cont.)*

Space sharing prevents overlapping partitions or applications as follows:

- When you try to create a partition that overlaps another partition or an application, the allocator rejects the partition creation and the **mkpart** command returns an error.
- When you attempt to execute an application that overlaps another application or an active partition, the allocator rejects the request. The application returns an error message and does not execute.

Examples

Enter the following to stop the allocator daemon:

```
# /sbin/init.d/allocator stop
```

Enter the following to restart the allocator daemon:

```
# /sbin/init.d/allocator start
```

Files

/sbin/init.d/allocator	Command path.
/usr/sbin/allocator	Binary file for the allocator daemon.
/etc/nx/allocator.log	The allocator log file.
/etc/nx/allocator.log.last	Previous allocator log file.
/etc/nx/.badnodes	Lists nodes that failed to boot.
/etc/nx/allocator.config	Allocator configuration file.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

ALLOCATOR *(cont.)* ALLOCATOR *(cont.)*

See Also

commands: **chpart**, **lspart**, **mkpart**, **pspart**, **rmpart**, **showpart**

files: **allocator.config**

Paragon™ System User's Guide

application *application*

Executes a parallel application.

Syntax

application [**-sz** *size* | **-sz** *h***X***w* | **-nd** *h***X***w***:***n*] [**-rlx**]
[**-pri** *priority*] [**-pt** *ptype*] [**-on** *nodespec*] [**-pn** *partition*]
[**-nt** *nodetype*] [**-pkt** *packet_size*] [**-mbf** *memory_buffer*]
[**-mex** *memory_export*] [**-mea** *memory_each*]
[**-sth** *send_threshold*] [**-sct** *send_count*]
[**-gth** *give_threshold*] [**-noc** *correspondents*]
[**-plk**] [*application_args*] [**\;** *file* [**-pt** *ptype*]
[**-on** *nodespec*] [*application_args*]] ...

Arguments

application File name of the application. The application must be either linked using the **-nx** switch, or be linked using the **-lnx** switch and contain the **nx_initve()** system call.

-sz *size* Specifies the *size* of the node set on which to run the application. The argument *size* must be an integer that ranges from 1 to the number of nodes in the partition. By default, the application runs on a rectangular set of nodes. If no rectangle of *size* nodes is available, the application runs on any available node set.

-sz *h***X***w* Specifies a rectangular, contiguous node set on which to run the application. The argument *h* is an integer representing the rectangle's height. The argument *w* is an integer representing the rectangle's width. The character **X** is a separator that can be upper or lower case. If the application cannot be allocated as specified, the application fails immediately.

-nd *h***X***w:n* Specifies a rectangular, contiguous node set that is *h* nodes high by *w* nodes wide on which to run the application. The upper left corner of the set of nodes is node number *n*. The values for *h*, *n*, and *w* must be a positive integers. You can use an uppercase or lowercase letter **X** between the *h* and *w*.

Specifying neither the **-sz** switch nor the **-nd** switch causes the default application size to equal the value of the *NX_DFLT_SIZE* environment variable, or all the nodes in the partition if the *NX_DFLT_SIZE* environment variable is not defined or does not equal a positive integer.

NOTE

If your default number of nodes, as specified by the environment variable *NX_DFLT_SIZE*, is greater than the number of nodes available in the specified partition, you may get a "exceeds partition resources" or "request overlaps with nodes in use" error.

-rlx
Relaxes the requirement that the exact specified number of nodes be available for an application to run. With the **-rlx** switch, the application runs on as many nodes as possible, up to the number of requested nodes. Note that there must be at least one available node for the application to attempt to execute. If no available nodes exist, the application fails immediately.

NOTE

The **-rlx** switch can be used to relax the default size, the **-sz** *size* switch, or the **-nd** switch. It cannot be used to relax the switch **-sz** *h*X*w*.

-pri *priority*
Sets the application's priority level. The argument *priority* must be an integer ranging from 0 (low priority) to 10 (high priority). The default is a priority of 5. The priority level determines how much processor time an application gets when more than one application is allocated on a node.

-pt *ptype*
Sets the process type of each process in the application. The argument *ptype* must be an integer greater than or equal to 0. The default is a process type of 0.

-on *nodespec*
Specifies a subset of nodes on which to load the application. By default, the application is loaded on all the application's nodes. This switch allows you to specify certain nodes from the entire node set on which to run a program. Use this switch when you are running multiple programs as a single application.

NOTE

Do not use the **-on** switch if you just want to run a single program on a specific number of nodes. Use the **-sz** switch or the **-nd** switch instead.

application (*cont.*) *application* (*cont.*)

The *nodespec* must be one of the following node specifiers:

x	Node number.
x..*y*	Range of node numbers from *x* to *y*, inclusive.
n	The last node of the partition.
nspec[,*nspec*]...	List of nodes for the application. Each *nspec* argument is a node specifier. You can list nodes using any combination of the node specifiers *x*, *x*..*y*, or **n**. Do not put spaces in this list.

The numbers you use with the **-on** switch are node numbers within the application. The range of node numbers is from 0 to one less than the number of nodes allocated to the application.

NOTE

Do not specify the same node number in *nodespec* more than once. If you specify the same node twice, two processes are created on the specified node, but one of the processes is terminated shortly after creation with the error "setptype: Ptype already in use."

Care needs to be exercised when using the **-on**, **-sz**, and **-nd** switches. If you use the **-on** switch when you really should be using the **-sz** or **-nd** switch, the application will be allocated more nodes than it needs.

Additionally, you need to be careful when using the **-on** and **-rlx** switches together. Using these switches together can create situations where it is not known whether a program is running on all nodes. Recall that use of global synchronizing operations such as **gopen()** and **gdsum()** in situations where the program is not running on every node in the application causes the synchronizing operations to hang. They hang because these operations block until every node calls them.

-pn *partition* Specifies the partition in which the application runs. The *partition* argument must be a pathname of an existing partition. You must have execute permission on the partition. If you do not use this switch, the default partition used is the value of the *NX_DFLT_PART* environment variable, or the *.compute* partition if the *NX_DFLT_PART* variable is not set.

application (cont.)

-nt *nodetype*

Specifies nodes having the attributes defined by *nodetype* as the node set on which to run the application. The *nodetype* argument is one of the following:

attribute

Selects nodes having the specified attribute. For example, when *attribute* equals the string **mp**, only MP nodes are selected. The standard node attributes are shown in the "Node Attributes" section.

!*attribute*

Selects nodes *not* having the specified attribute. For example, when *attribute* equals the string **!io**, only nodes that are *not* I/O nodes are selected. Note that no white space may appear between the **!** and *attribute*.

[*relop*][*value*]*attribute*

Selects nodes having a specified value or range of values for the attribute. For example, the string **>=16mb** selects nodes with 16M bytes or more of RAM. The string **32mb** selects nodes with exactly 32M bytes of RAM. And, the string **>1proc** selects nodes with more than one processor.

The *relop* can be =, >, >=, <, <=, !=, or ! (!= and ! mean the same thing). If the *relop* is omitted, it defaults to =.

The *value* can be any nonnegative integer. If the *value* is omitted, it defaults to 1.

The *attribute* can be any attribute shown in the "Node Attributes" section, but is usually either **proc** or **mb**. (Other attributes have the value 1 if present or 0 if absent.)

No white space may appear between the *relop*, *value*, and *attribute*.

ntype[,*ntype*]...

Selects nodes having *all* the attributes specified by the list of *ntype*s, where each *ntype* is a node type specifier of the form *attribute*, **!***attribute*, or [*relop*][*value*]*attribute*. For example, the string **32mb, !io** selects non-io nodes with 32M bytes of RAM.

You can use white space (space, tab, or newline) on either side of each comma, but not within an *ntype*.

-pkt *packet_size*

Sets the number of bytes in each packet size (*packet_size*) used for sending messages. The *packet_size* argument is an integer value. If a message is larger than the *packet_size* value, the application sends messages in several packets that are each *packet_size* bytes long. The minimum, maximum, and default values for *packet_size* are as follows:

- minimum: **sizeof(xmsg_t)**
- maximum: 8192
- default:
  8192 bytes or (*memory_each* / 2) - **sizeof(xmsg_t)** (whichever is less)

NOTE

For the default, minimum, and maximum values of the message-passing configuration switches **-pkt**, **-mbf**, **-mex**, **-mea**, **-sth**, **-sct**, and **-gth**, the name *full_packet_size* represents the value *packet_size* + **sizeof(xmsg_t)**. The type **xmsg_t** is defined in the include file *<mcmsg/mcmsg_xmsg.h>*, and defines the message header sent with each packet. The size of this type is currently 64 bytes.

-noc *correspondents*

Sets the total number of other processes from which each process expects to receive messages. The argument *correspondents* is an integer value. The default value for *correspondents* is **numnodes()**.

-mbf *memory_buffer*

Sets the total memory allocated in bytes for message buffers in each process (*memory_buffer)*. The argument *memory_buffer* is an integer value. Its minimum, maximum, and default values are as follows:

- minimum:
  (8 * **sizeof(xmsg_t)**) * (*correspondents* + 2)) + 20 * **sizeof(xmsg_t)**)
- maximum: 32MB + (10 * *full_packet_size*)
- default: 1MB + 128K bytes

-mex *memory_export*

Sets the total memory in bytes allocated for buffering messages received from other nodes. The argument *memory_export* is an integer value. The minimum, maximum, and default values for *memory_export* are as follows:

- minimum: 2 * (*correspondents* + 2) * (2 * *full_packet_size*)
- maximum: *memory_buffer* - 128K bytes
- default: *memory_buffer* - 128K bytes

-mea *memory_each*

Sets the memory in bytes allocated to each node in the application for buffering messages received from other nodes (*memory_each*). The argument *memory_each* is an integer value. The application uses memory in the *memory_buffer* segment that is outside of the correspondents multiplied by the *memory_each* value for buffering messages from any sending node, when needed. The minimum, maximum, and default values for *memory_each* are as follows:

- minimum: 2 * *full_packet_size*
- maximum:
  (*memory_export* / 2) / (*correspondents* + 2) or 1MB - 31 (whichever is less)
- default:
  10 * (*full_packet_size*) or maximum *memory_each* (whichever is less)

-sth *send_threshold*

Sets the threshold in bytes for sending multiple packets (*send_threshold*). The argument *send_threshold* is an integer value. If a sending node has at least *send_threshold* bytes of memory free in its *memory_each* segment on the receiving node, it will send multiple packets of a message right away. Otherwise, it will send one packet and wait for an acknowledgment that a receive has been posted. The minimum, maximum, and default values for *send_threshold* are as follows:

- minimum: none
- maximum: *memory_each* - 1
- default: *memory_each* / 2

-sct *send_count* Sets the number of bytes to immediately send (*send_count*) when the memory available is above the *send_threshold* value. The argument *send_count* is an integer value. The minimum, maximum, and default values for *send_count* are as follows:

- minimum: *packet_size*
- maximum: *memory_each*
- default: *memory_each* / 2

-gth *give_threshold*

Sets the lower bounds in bytes for forward-flow control information (*give_threshold*). As messages on a receiving node are consumed by the user's program, the message memory becomes available to store new messages. The receiving node tells its sending nodes about the newly available free memory by "piggy-backing" the information on other messages going to the sender. This communication between sending and receiving nodes is known as forward-flow control. The sender only sends a message when it knows the receiving node has enough memory available in which to store the message. The receiver keeps track of how much memory it has told the sender is available. When there are no other messages to "piggy-back" information on and the amount of memory the receiver has told the sender is available goes below *give_threshold*, the receiver sends a special message to the sender telling it how much memory is actually available. The argument *give_threshold* is an integer value whose minimum, maximum, and default values are as follows:

- default: *packet_size*
- maximum: *memory_each* / 2
- minimum: *packet_size*

-plk Locks the data area of each process into memory. This switch functions like the **plock()** function. See the *OSF/1 Programmer's Reference* for information on the **plock()** function. This switch also conditions message-passing code to run more efficiently by assuming that all data buffers are locked into memory. The default behavior is to not lock.

The **-plk** switch locks the following parts of your application into physical memory:

- The entire *data segment*. This is the part of memory that contains global variables. This area is locked when the program is loaded.

- The area from the beginning of the *stack* or *heap* to the end of the buffer. One of these areas is locked the first time you use the buffer in a message-sending or message receiving call. The area locked depends on where you use the application buffer. If its used on the *stack*, the area from the beginning of the *stack* to the end of the buffer is locked. If you use an application buffer that is located on the *heap*, the area from the beginning of the *heap* to the end of the buffer is locked. The *stack* is the part of memory that contains local variables, while the *heap* is the part of memory that is allocated by the **malloc()** function (C) or the **ALLOCATE** statement (Fortran).

All areas of memory not mentioned in this list, including the node segment (the part of memory that contains executable instructions), are not locked and are still subject to paging. Note that locking is done a page at a time. To lock a single byte, the system must lock the entire 8K byte virtual memory page containing that byte.

application_args
Additional arguments specific to the application.

file
Loads the executable file specified by the *file* argument onto some or all of the same nodes as the application specified by the *application* argument. The file must be compiled and ready to execute. It can be linked with or without the **-nx** switch, but it must not call **nx_initve()**.

The **-pt** and **-on** switches following the *file* argument specify the process type and nodes for *file*. The *application_args* following the *file* argument specifies additional application-specific arguments for the executable file. The command-line switches you can use with the *file*s are as follows:

- Any application switches with the first file. The switches you use with the first file affect the entire application. The **-pt** and **-on** switches you use with the first file affect the first file only.

- Only the **-pt** and **-on** switches with the second and subsequent files. These switches affect the second and subsequent files only.

NOTE

The escaped semicolon (\;) before the *file* argument must be preceded and followed by a space or tab. Otherwise, it will be considered part of the preceding or following argument.

NOTE

If you forget the backslash before the semicolon, the first program is run as an application by itself and the second program runs after the first program finishes. This usually results in unexpected behavior from the programs.

Description

The switches described in this manual page are available for applications linked with the **-nx** switch or for applications linked with the **-lnx** switch that call **nx_initve()** or **nx_initve_rect()**.

An application linked with the **-lnx** switch that calls either the **nx_initve()** or the **nx_initve_rect()** function can override the command line switches.

When you specify the **-sz** *size* switch, the operating system attempts to allocate the application in a square group of nodes. If this is not possible, the operating system attempts to allocate the application in a rectangular group of nodes that is either twice as wide as it is high or twice as high as it is wide. If this is not possible, the operating system attempts to allocate the application in any available nodes. In this case, nodes allocated to the application may not be contiguous (that is, they may not all be physically next to each other). No matter what the shape of the application, node numbers within the application (as returned by **mynode()**) will always be sequential from 0.

To get better performance from an application that uses parallel loops you can control the number of CPUs per board used to process these loops. You can specify the number of CPUs by using the *DFLT_NCPUS* environment variable. For example, setting *DFLT_NCPUS* to 2 causes each node to use two CPUs to process parallel loops regardless of any setting determined during boot time. Use of this environment variable allows you to see the effects that multiple CPUs can have on looping code for debugging or performance purposes. Note that when *DFLT_NCPUS* is zero, an application compiled with **-Mconcur** uses the maximum number of CPUs available on each node. For more information on how to use the *DFLT_NCPUS* environment variable, see the *Paragon User's Guide*.

Parallel applications can be *gang-scheduled* to make more efficient use of system resources. In gang scheduling, an application is allowed to run for a time period, called the *rollin quantum*, and then is "rolled out". Once this application is "rolled out", another application is "rolled in" in to take its place. If the rollin quantum is long, much time may pass before you see any response to a `<Ctrl-c>` or `<Ctrl-z>`.

NOTE

Interrupting or suspending an application that is "rolled out" will not take effect until the application is "rolled in" again.

Node Attributes

The hardware characteristics of each node are described by a comma-separated series of strings called *attributes*. The following shows the most common node attributes. An attribute that is indented is a more specific version of the attribute from the previous level of indentation. For example, **net** and **scsi** nodes are specific types of **io** node; **enet** and **hippi** nodes are specific types of **net** node (and also specific types of **io** node).

Attribute	Meaning
bootnode	Boot node.
gp	GP (two-processor) node.
mp	MP (three-processor) node.
mcp	Node with a message coprocessor.
*n***proc**	Node with *n* application processors (not counting the message coprocessor).
*n***mb**	Node with *n*M bytes of physical RAM.
io	Any I/O nodes.
net	I/O node with any type of network interface.
enet	Network node with Ethernet interface.
hippi	Network node with HIPPI interface.
scsi	I/O node with a SCSI interface.
disk	SCSI node with any type of disk.
raid	Disk node with a RAID array.
tape	SCSI node with any type of tape drive.
3480	Tape node with a 3480 tape drive.
dat	Tape node with a DAT drive.
IDstring	SCSI node whose attached device returned the specified *IDstring*. For example, a disk node might have the *IDstring* **NCR ADP-92/01 0304**.

Node attributes are not case sensitive, therefore, **GP**, **gp**, and **Gp** are equivalent.

Using Node Attributes with an Application Size

If you use the **-nt** switch together with the **-sz** switch, the **-nd** switch, or the environment variable *NX_DFLT_SIZE*, the application runs on the specified nodes with the specified attributes, as follows:

- For **-sz** *size* or *NX_DFLT_SIZE*, at least the specified number of nodes with the specified attributes must be available in the partition.

- For **-sz** *h***X***w*, at least one rectangle of nodes of the specified size and shape, all of which have the specified attributes, must be available somewhere in the partition.

- For **-nd** *h***X***w***:***n*, the specified rectangle of nodes must be available and all the nodes must have the specified attributes.

If the specified nodes with the specified attributes are not available in the partition, the command fails with an error message and the application does not run. You can use the **-rlx** switch with the **-sz** or **-nd** switches to relax the requirement that a specified number of nodes must be available. In these cases, nodes that qualify (meet the specified attributes) in the partition are used to run the application. The number of nodes used can range from a single node up to one less than the full set of requested nodes.

Examples

The following examples assume that *myapp*, *mymgr*, and *myworker* are parallel applications that linked with the **-nx** switch.

1. To run *myapp* on all nodes in the default partition, enter the following:

   ```
   % myapp
   ```

   This application runs only if all the nodes in the default partition are available. To relax the requirement that all the nodes you request must be available, enter the following:

   ```
   % myapp -rlx
   ```

   The **-rlx** switch allows the application to run even when all the nodes requested for the application are not available.

2. To run *myapp* with a priority of 7 on 50 nodes in the default partition, enter the following:

   ```
   % myapp -pri 7 -sz 50
   ```

3. To run *myapp* on an 8x8 rectangular node set in the default partition, enter the following:

   ```
   % myapp -sz 8X8
   ```

4. To run *myapp* on an 8x8 rectangular node set anchored at node 0 (zero) in the default partition, enter the following:

   ```
   % myapp -nd 8X8:0
   ```

application (*cont.*) *application* (*cont.*)

5. To allocate all the nodes of the *mypart* partition to the application *mymgr*, load *mymgr* onto node 0 (zero) of the default partition with process type 1, and load *myworker* onto all nodes *but* node 0 with process type 0, enter the following:

```
% mymgr -on 0 -pt 1 -pn mypart \; myworker -on 1..n
```

6. To run *myapp* on all the MP nodes in the default partition (it fails if less than all the MP nodes in the default partition are available), enter the following:

```
% myapp -nt mp
```

   To relax the requirement that all the MP nodes you request must be available, enter the following:

```
% myapp -nt mp -rlx
```

7. To run *myapp* on all the MP nodes in the default partition that have greater than 16M bytes of memory, enter the following:

```
% myapp -nt "mp, >16mb"
```

   Remember, if any characters special to your shell (such as >, <, or white space) appear in the *nodetype* string of the **-nt** switch, you must enclose the *nodetype* string in quotes, or you must precede the special characters with a backslash character.

8. To run *myapp* on 5 MP nodes in the default partition (it fails if less than 5 MP nodes are available), enter the following:

```
% myapp -sz 5 -nt mp
```

9. To run *myapp* on 5 MP nodes in the default partition (it fails if less than 5 MP nodes are available), enter the following:

```
% setenv NX_DFLT_SIZE 5
% myapp -nt mp
```

10. To run *myapp* on a 2-by-4-node rectangle of MP nodes in the default partition (it fails if no such rectangle of MP nodes is available anywhere in the partition), enter the following:

```
% myapp -sz 2x4 -nt mp
```

11. To run *myapp* on a 3-by-3-node rectangle of MP nodes in the upper left corner of the default partition (it fails if the specified rectangle is not available or does not consist entirely of MP nodes), enter the following:

```
% myapp -nd 3x3:0 -nt mp
```

Errors

`Bad node specification`

You specified a node number that is greater than the largest node number in the partition with the **-sz** switch, or you used an improperly-formatted *nodespec* with the **-on** switch.

`Exceeds partition resources`

You specified an application size with **-sz** *size* that is greater than the partition size, or the *NX_DFLT_SIZE* environment variable specifies a size greater than the partition size. If you did not specify a partition with the **-pn** switch, check the size of the partition specified by *NX_DFLT_PART* environment variable.

`Give count invalid or out of range`

You specified a *give_threshold* argument with the **-gth** switch that is invalid or out of range.

`Invalid priority`

You specified a priority that is not between 0 (zero) and 10.

`Memory buffer invalid or out of range`

You specified a buffer size with the **-mbf** switch that is invalid or out of range.

`Memory each invalid or out of range`

You specified a buffer size with the **-mea** switch that is invalid or out of range.

`Memory export invalid or out of range`

You specified a buffer size with the **-mex** switch that is invalid or out of range.

application *(cont.)* ***application*** *(cont.)*

`Packet size invalid or out of range`

You specified a packet size with the **-pkt** switch that is invalid or out of range.

`Partition not found`

You specified a partition with the **-pn** switch that does not exist. If you did not use the **-pn** switch, check the value of the *NX_DFLT_PART* environment variable.

`Partition permission denied`

You specified a partition with the **-pn** switch that you do not have execute permission for. If you did not use the **-pn** switch, check the value of *NX_DFLT_PART* environment variable.

`Request overlaps with nodes in use`

You tried to load an application in a partition that may overlap an existing partition in the compute partition.

`Send threshold invalid or out of range`

You specified a send threshold with the **-sth** switch that is invalid or out of range.

`Send count invalid or out of range`

You specified a send count with the **-sct** switch that is invalid or out of range.

`The application and the OS are of incompatible revisions`

Your application is out of date. You need to recompile and relink your application.

`Use of -plk not allowed in gang-scheduled partition.`

You tried to use the **-plk** switch in a gang-scheduled partition or a partition that has a gang-scheduled ancestor. The use of the **-plk** switch is controlled in the allocator configuration file. See the **allocator** and **allocator.config** manual pages for more information about the allocator configuration file.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **lspart**, **mkpart**, **pspart**, **showpart**

calls: **mynode()**, **nx_initve()**, **nx_initve_rect()**, **nx_load()**

OSF/1 Programmer's Reference: **plock(2)**

ASYNC ASYNC

Diagnostic station: Connects a console device for asynchronous communications between the diagnostic station and the Paragon system.

Syntax

async [**-f**] [**-k**] [**-s**] [**-t**] *device*

Arguments

-f	Uses the **fscan** interface for asynchronous communications. Makes sure the kernel is downloaded before it exits.
-k	Sends a a carriage return to the boot node to start the node.
-s	Spins between the keyboard and serial line. This supports diagnostic systems that do not provide the **select**() call for non-stream devices.
-t	Returns 1 if the device is available (not locked), else returns 0. The return value can be checked using the Bourne shell variable *$?*.
device	Name of the device to be used for asynchronous communications. The device name can be either an absolute pathname, relative pathname, or a simple name for a TTY device. If you use a relative pathname or a simple name for a device, the pathname or name is assumed to be relative to the */dev* directory.

Description

The **async** command runs on the diagnostic station and is for use by the system administrator only.

The **async** command provides serial communications to the Paragon system console with the following communication parameters: 19.2K baud, 8 data bits, 2 stop bits, and no parity. These communications parameters cannot be changed.

When the **async** command connects to a device, the command creates a file named */tmp/LOCK.XXX*. The XXX variable is the device name minus the directory. For example, when connected to */dev/tty1a*, the **async** command creates a file called */tmp/LOCK.tty1a*. This file is a lock so one user only can use the **async** command with this device. However, other commands such as the **cu** command may use the device, causing unpredictable results.

ASYNC *(cont.)*   ASYNC *(cont.)*

The lock file contains the process ID (PID) of the process that owns the lock. The **async** command first checks to make sure the process still exists on the system. If the process does not exist, the lock is ignored.

When the **async** command is running, entering a tilde (~) begins a command sequence. The following command sequences have special meaning:

~!	Allows command execution on the diagnostic station. After the command completes, control is returned to the console. You can use the **sh** command to create a shell if you want to enter more than one command.
~**.** (dot)	Exits the program. Be careful using this when using **rlogin** to log in to the diagnostic station. Use two tildes and a dot (~~**.**) when logged in via the **rlogin** command.
~q	Exits the program. The **~q** sequence is identical to ~**.** (dot). Use when logged in via the **rlogin** command to avoid killing the **rlogin** process.

These command sequences work only after entering a carriage return on the command line.

Files

/usr/local/bin/async	Contains the executable for the **async** command.
/tmp/LOCK.XXX	Lock file used to lock the device so no other users can use the **async** command on the device.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

console, fscan, scanio

AUTODDB AUTODDB

Diagnostic station: Collects system debug information after a crash.

Syntax

autoddb [**-Aq**] [**-f** *file* | **-F** *file*] [**-n** *nodes*] [**-N** *nodes*] [**-t** *number*] [**-m** "*list*"] [**-s**] [**-S**]

Arguments

-A	Directs **autoddb** to collect debug information from the system nodes that are running an alternate operating system (for example, SUNMOS) as well as the nodes running the default operating system. The default is to check the nodes that are running the default operating system only. The alternate operating system on a node must be capable of responding to a **ping** from the **fscan** command.
-f *file*	Specifies a log file for reporting debug information. The default is a log file with the name ***autoddb**.date-time*. The *date-time* suffix has the format *DD-MMM-YY-HH.MM.SS*, where *DD-MMM-YY* is the date, month, and year and HH.MM.SS is the hour, minute, and second when the file was created. This allows for the existence of multiple log files reflecting different runs of **autoddb**. The log file(s) resides in the */u/autoddb* directory.
-F *file*	Same as the -f switch except the debug information is appended to the file specified by the *file* argument if the file exists. The default is that *file* is overwritten.
-n *nodes*	Specifies a single node or node list from which to collect debug information. The default is the boot node, any failed nodes, the nodes listed in the *DEBUG_NODES* variable in */usr/paragon/boot/autoddb*, and the nodes listed in *SERVICE_NODES* when there are no failed nodes. The **-n** switch overrides the default; debug information is only reported for the node specified by the *node* argument.
-N *nodes*	Adds node(s) to the default list of nodes that will be investigated.
-q	Suppresses output of debug information to the terminal. The default is that debug information is displayed on the terminal and reported to a log file.
-t *number*	Specifies the last task number that performs a *t/uT* on each node. By default, task information is returned for *$task0* and *$task2*. You can use this switch when the system hangs to get *t/uT* information beyond these tasks. For example, setting the *number* argument to five returns information for *$task0*, *$task2*, *$task3*, *$task4*, and *$task5*.

AUTODDB *(cont.)* AUTODDB *(cont.)*

-m "*list*"	Mail resulting log file to given user e-mail *list*.
-s	Simple run. Does not allow additional nodes to be investigated. Does not print stack traces for $task[2,0].0 (ddb command: **trace/uT**). The **-s** switch should only be used if you are planning to follow up with ddb commands.
-S	REAL Simple run. Operates the same as the **-s** switch, except does not do node follow-up. This switch should only be used if you are planning to follow up with ddb commands.

Description

The **autoddb** command runs on the diagnostic station and is intended for use by the system administrator only.

To execute the **autoddb** command you need to be in the */usr/paragon/boot* directory. When **autoddb** executes the following occurs:

- Opens a log file for the debug information.
- Uses ***fscan*** to poll each node to see if it is in the *running*, *dead*, or *debugger* state.
- Displays a message about any dead nodes that are found.
- Runs the diagnostic utility, **statusutl**.
- Collects debug information about the boot node (unless it is dead), nodes that are in *debugger* state, nodes listed in the *DEBUG_NODES* string, and nodes specified using the **-n** or **-n** and **-N** switches. Directs all output to the log file (in the */u/autoddb* directory) and the terminal (unless the **-q** switch is specified). Note that if no nodes are in *debugger* state, **autoddb** checks nodes listed in *SERVICE_NODES* as well. After debug information is collected from a node, the node resumes its operations.

Before running the **autoddb** command, you can use the *DEBUG_NODES* and *SERVICE_NODES* strings in the command script */usr/paragon/boot/autoddb* to specify from which nodes the command collects debug information. For information on these strings, see the "Specifying Nodes for Which Information is Returned" section of this manpage.

You should also check that the variables *CONFIGURATION* and *TOP_BACKPLANE* are set correctly in the *autoddb* command script. The values should be the same as the variables in the **reset** script. For example, the following lines show typical values for these variables:

```
$CONFIGURATION="full";
$TOP_BACKPLANE="D";
```

This specifies a full system configuration and the top backplane as backplane D.

The **autoddb** script tries to determine the settings for $CONFIGURATION and $TOP_BACKPLANE by looking for the RST_TOP_BACKPLANE and RST_CONFIGURATION settings in the *MAGIC.MASTER* file. If they are there, you do not need to customize **autoddb**.

Also before running **autoddb**, check that the bootmagic string *BOOT_CONSOLE* is set as follows in the magic file (*MAGIC.MASTER*):

```
BOOT_CONSOLE=f
```

This setting requests booting with the **fscan** console. If *BOOT_CONSOLE* is not set to **f**, edit the magic file to make this change.

To make sure that debug information gets collected when an autoreboot occurs, modify the specification for the **reboot** command in the file */usr/paragon/boot/fscan.cfg* as follows:

```
define reboot "sh /usr/paragon/boot/autoddb; \
ksh /usr/paragon/boot/reset skip ignorelock autoreboot"
```

This modification runs the **autoddb** command after the system watchdog detects a node error but before the system is booted.

As of R1.3, **autoddb** is implemented in a Perl script. The */bin/sh* version of **autoddb** is called *autoddb.sh* and is in the same location of **autoddb**. If the Perl version does not seem to be working correctly, you can use the *autoddb.sh* script.

Specifying Nodes

DEBUG_NODES=node_list

Specifies additional nodes on the Paragon system from which **autoddb** collects debug information. By default, debug information is collected for the boot node and nodes in the *debugger* state. The default value for *node_list* is a null string. For example, this setting checks the boot node and all nodes in the debugger state.

```
$DEBUG_NODES="";
```

AUTODDB *(cont.)* AUTODDB *(cont.)*

When you specify a list of nodes for the *node_list* argument, debug information is also collected for *node_list*. To specify a list of nodes supply one or more numbers with individual nodes separated by blanks. Usually, *node_list* consists of service nodes other than the boot node. You can also include I/O nodes and any suspected problem nodes. For example, this setting returns debug information for nodes 1 through 4; nodes 6, 8, and 9; the boot node; and any nodes in the *debugger* state. The boot node and any nodes in the *debugger* state always have information returned.

```
$DEBUG_NODES="1 2 3 4 6 8 9";
```

Note that the more nodes **autoddb** collects information about, the longer it takes to run. The **autoddb** command will add at least 60 seconds to its execution time for each additional node.

SERVICE_NODES=node_list

List of service nodes examined whenever no machine nodes are discovered in the *debugger* state. By default, **autoddb** returns information on the boot node, and any failed nodes, and the nodes listed in the *DEBUG_NODES* variable in */usr/paragon/boot/autoddb*. By setting this string you instruct **autoddb** to check the specific nodes in *node_list* whenever it fails to find machine nodes in the *debugger* state.

The default value for *node_list* is a null string. For example, this setting instructs **autoddb** to not check nodes beyond the default set.

```
$SERVICE_NODES="";
```

To specify a list of nodes, supply one or more numbers with individual nodes separated by blanks. For example, this setting returns debug information for nodes 6, 8, and 9 when **autoddb** can't find any machine nodes in the *debugger* state:

```
$SERVICE_NODES="6 8 9";
```

Examining Output

After you run **autoddb** you can review the output for some easy-to-spot problems: out of memory and NIC errors. Look at the summary section first for any problems **autoddb** has noticed.

Out of memory

Review the *!db_sys* output and look for the number of free pages. If this number is 29 (or very near 29), the node being examined is out of memory.

AUTODDB *(cont.)* AUTODDB *(cont.)*

NIC errors

Look at the *machine comm* output and find the line 'errors'. If anything follows the word 'errors' then the NIC has detected either a hardware or software error. Some of the various errors (and your options) are:

pr-par0 -- pr-par7
: These are processor-port parity errors. These errors are caused by parity inconsistencies between the NIC and the i860. To fix this problem, replace the node.

net-par0 or net-par1,
: These are network parity errors. These errors are occur when data is passed between the local MRC and the local NIC. To fix this problem, reseat the local node. If re-seating the node doesn't work, check for bent backplane connectors by running the PSD mesh test. If no backplane connectors are bent, replace the local node. If replacing the local node fails to fix the problem, replace the local MRC. Finally, if none of this works, replace the local backplane.

xmt-overrun, rcv-overrun, or rcv-underrun
: These three errors are the most common and most difficult to diagnose. They could originate from either hardware or software errors. To diagnose these errors, further examination beyond **autoddb**, is required.

Files

/usr/paragon/boot/autoddb	Specifies the command path.
/usr/paragon/boot/fscan.cfg	Specifies the **fscan** configuration file on the diagnostic station.
/usr/paragon/boot/MAGIC.MASTER	Specifies the default magic file on the diagnostic station.
/u/autoddb/autoddb.date-time	Log file(s) containing debug information.

See Also

commands: **fscan, fscan.cfg, reset**

files: **MAGIC.MASTER**

BOOTMESH BOOTMESH

Loads files for the operating system onto the system's nodes.

Syntax

bootmesh [-bdDEGHKMoRSvVwZ] [**-e** *file*] [**-k** *file*] [**-m** *file*]
[**-n** *node*] [**-s** *file*] [**-t** *millisecs*] [**-z** *seconds*]

Arguments

-b	Broadcasts the operating system image to all nodes. The default is to download each node individually.
-d, -D	Displays debug messages.
-e *file*	Specifies the emulator file to use for booting on the Paragon system. The default is the pathname specified by the bootmagic file (see the **bootmagic** manual page for more information).
-E	Does not download the emulator(s).
-G	Does not send the **goto** command.
-H	Displays help messages.
-k *file*	Specifies the kernel file to use for booting on the Paragon system. The default is the pathname specified by the bootmagic file (see the **bootmagic** manual page for more information)
-K	Does not download the kernel file.
-m *file*	Specifies the bootmagic file to use for booting. The default is the memory-resident bootmagic file. See the **bootpp** and **parsemagic** manual pages.
-M	Does not download the bootmagic file.
-n *node*	Specifies a single node to boot. The default is to boot all nodes specified in the bootmagic file (see the **bootmagic** manual page for more information).
-o	Boots the same kernel/server file on all nodes.
-R	Does not reset the NIC memory loader.

BOOTMESH *(cont.)* BOOTMESH *(cont.)*

-s *file*	Specifies the server file for booting. The default is to boot all nodes specified in the bootmagic file (see the **bootmagic** manual page for more information).
-S	Does not download the server file.
-t *millisecs*	Specifies timeout value for polling nodes.
-v, -V	Displays all messages (verbose mode).
-w	Does not display warning messages.
-z *msecs*	Specifies the number of microseconds to sleep between **goto** commands. The *msecs* parameter is an integer value that specifies the number of microseconds to sleep.
-Z	Sends goto commands to each node in turn. The default is to broadcast these commands to all nodes.

Description

The **bootmesh** command runs during the booting sequence on the boot node only. Note that the */sbin/bootmesh.sh* script executes the **bootmesh** command. You can edit this script to change the **bootmesh** command-line arguments. The **bootmesh** command does the following:

- Reads the information in the bootmagic file.
- Downloads the files for the operating system, including files for the microkernel, server, emulator, and alternate operating system (if the bootmagic string *BOOT_ALT_NODE_LIST* is specified in the bootmagic file). For information about setting the bootmagic string *BOOT_ALT_NODE_LIST*, see the **bootpp** manual page.
- Starts the kernel on each system node.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

bootpp, parsemagic

BOOTPP BOOTPP

Diagnostic station: Preprocesses information for the bootmagic file.

Syntax

bootpp [**-dDHuvVWZ?**] [**-a** *file*] [**-b** *file*] [**-c** *file*] [**-e** *file*] [**-k** *file*] [**-K** *file*] [**-l** *node_list*] [**-m** *file*] [**-M** *file*] [**-n** *node*] [**-p** *string*] [**-P** *num*] [**-s** *file*] [**-S** *file*] [**-x** *size*] [**-y** *size*] [**-z** *file*]

Arguments

-a *file*	Creates a new *DEVCONF.TXT* file. The default is */usr/paragon/boot/DEVCONF.TXT.*
-b *file*	Specifies the bootmagic file on the diagnostic station to use for booting. The bootmagic file is the output file created by **bootpp**. The default is */usr/paragon/boot/bootmagic.*
-c *file*	Specifies the hardware configuration file on the diagnostic station to use for booting a Paragon system. The default is */usr/paragon/boot/SYSCONFIG.TXT.*
-d	Specifies debug support.
-D	Provided for backward compatibility. Creates a *DEVTAB* bootmagic string.
-e *file*	Specifies the emulator file on the Paragon system to use for booting. The default is */mach_servers/emulator.*
-H	Displays help messages during execution.
-k *file*	Specifies the kernel file on the Paragon system to use for booting the I/O nodes. The default is */mach_servers/mach_kernela.*
-K *file*	Specifies the kernel file on the Paragon system to use for booting the compute nodes. The default is */mach_servers/mach_kernela.*
-l *node_list*	Specifies the list of nodes on a Paragon system to use for booting. The *node_list* is a string that is list of node numbers separated by commas.
-m *file*	Specifies the magic file on the diagnostic station to use for booting a Paragon system. This file is an input file for **bootpp** and overrides the defaults. The default magic file is */usr/paragon/boot/MAGIC.MASTER.*

BOOTPP *(cont.)* BOOTPP *(cont.)*

-M *file*
Specifies the file on the diagnostic to use for verifying bootmagic strings. The default is */usr/paragon/boot/magic.lis*. Use this switch with the **-V** switch.

-n *node*
Specifies the first node to boot on a Paragon system. The default is 0.

-p *string*
Specifies a value for a bootmagic string. The *string* has the following form:

name=value

The *name* is the name of the bootmagic string and the *value* is the value of the bootmagic string. For example, the following switch specification specifies a bootmagic string:

```
-p "BOOT_FIRST_NODE=7"
```

This specifies the boot node to be node 7.

You can specify the **-p** switch multiple times in a **bootpp** command line. No checking is done to validate the bootmagic string. See the **bootmagic** manual page for list of valid bootmagic strings.

-P *num*
Specifies whether to generate pager node information. The default is the value of the bootmagic string *PAGER_NODE* as specified in the *MAGIC.MASTER* file. The value for the *num* argument indicates the following:

0	Specifies that the boot node is the pager node. This is the default and it creates a two-level paging tree.
1	Requests the **bootpp** command to generate the pager node information. This creates a three-level paging tree.

See the *Paragon™ System Administrator's Guide* for more information on paging trees.

-s *file*
Specifies the server file on the diagnostic station for booting the I/O nodes on a Paragon system. The default is */mach_servers/startup*.

-S *file*
Specifies the server file on the diagnostic station for booting the compute nodes on a Paragon system. The default is */mach_servers/startup*.

-u
Specifies booting the boot node only (boots the system like a single-node system). This may be useful for system maintenance.

BOOTPP *(cont.)*

-v	Displays all help messages.
-V	Verifies the bootmagic strings. The *magic.lis* file must exist on the diagnostic station.
-W	This switch has no function, but is provided for compatibility with previous versions of the command.
-x *size*	Specifies the number of nodes in the system's X dimension.
-y *size*	Specifies the number of nodes in the system's Y dimension.
-z *file*	Specifies a kernel file on the diagnostic station that is used for a checksum with the kernel on the Paragon system's nodes. The default is */usr/paragon/boot/mach_kernel.*
-Z	Suppresses the kernel checksum operation, but read access to the kernel file is validated.

NOTE

The checksum operation compares a copy of the kernel file on the diagnostic node with the kernel file downloaded to the Paragon system's nodes. If the kernel file on the diagnostic station does not match the kernel file on the system's nodes, booting cannot complete.

-?	Same as the **-H** switch.

Description

The **reset** command executes the **bootpp** command on the diagnostic station immediately before booting a Paragon system. Users do not execute this command directly.

The **bootpp** command creates the bootmagic file */usr/paragon/boot/bootmagic*. This file contains hardware and software configuration information for the Paragon system being booted. The bootmagic file consists of a set of strings with the following form:

```
name=value
```

The string values are terminated with a new line and the bootmagic file is terminated with a null character. See the **bootmagic** manual page for a list of the bootmagic strings.

BOOTPP *(cont.)* BOOTPP *(cont.)*

Generating Bootmagic Strings

The **bootpp** command uses the following information to prepare the bootmagic file:

1. Default configuration parameters are used in the absence of any other inputs.

2. The hardware configuration file *SYSCONFIG.TXT* provides a description of the Paragon hardware. The default file is */usr/paragon/boot/SYSCONFIG.TXT*. You can specify an alternate hardware configuration file with the **-c** switch. Values in this file override the master magic file and default configuration values (such as, *mesh_x*, *mesh_y*, *bootnode*, *node_list*).

3. The file *MAGIC.MASTER* provides bootmagic strings that override the default configuration parameters. The default file is */usr/paragon/boot/MAGIC.MASTER*. You can specify an alternate magic file with the **-m** switch.

4. The file *BADNODES.TXT* contains the list of nodes on a Paragon system that have failed or are nonfunctional.This file contains node numbers for each node that caused three successive reboots.

5. Command-line switches such as **-e**, **-n**, **-s**, **-S**, **-x**, and **-y** override the configuration files.

6. Strings for new configuration parameters can be specified on the **bootpp** command line with the **-p** switch or inserted into the bootmagic file with the following form:

   ```
   name=value
   ```

The following cases are exceptions to the rules for bootmagic strings:

1. When using defaults to generate a bootmagic file, you still have to use the following **bootpp** command switches to specify the hardware configuration: the **-l**, **-n**, **-x**, and **-y** switches. There are no defaults for these switches.

2. Pathnames of the bootmagic file, master magic file, and the hardware configuration file can only be specified using the defaults or **bootpp** command switches.

3. The list of operational nodes in the root partition on a Paragon system is always computed from the available information, for example the *BADNODES.TXT* file.

4. The time-of-day value is always determined by a direct query (**time(3)**) to the diagnostic station.

BOOTPP *(cont.)* BOOTPP *(cont.)*

Booting an Alternate Operating System

The **bootmesh** command can boot an alternate operating system (for example, SUNMOS) on the subset of nodes specified by the bootmagic string *BOOT_ALT_NODE_LIST*. This list of nodes can be generated by the **bootpp** command in two ways:

1. In the master magic file, use the bootmagic string *BOOT_ALT_NODE_LIST* to specify the list of nodes on which the alternative operating system is booted.

2. Specify the keyword ALTOS in the node description lines in the file *SYSCONFIG.TXT*. The specified nodes are put in *BOOT_ALT_NODE_LIST*.

The nodes listed in *BOOT_ALT_NODE_LIST* are removed from the bootmagic string *BOOT_NODE_LIST*, which specifies the set of nodes on which the Paragon OSF/1 server and kernel are booted. The nodes on which the alternate operating system is booted will be in the root partition. The pathname of the alternate operating system's kernel file is specified by the bootmagic string *BOOT_ALT_KERNEL_NAME*. The default alternate kernel file is */mach_servers/sunmos*.

NOTE

The alternate kernel file, for example */mach_servers/sunmos*, is by default not available as part of the Paragon system software. The alternate kernel file must be installed on the system before you can boot with the alternate operating system.

Files

/usr/paragon/boot/bootmagic	Specifies the bootmagic file. This is the output of the **bootpp** command.
/usr/local/bin/bootpp	Specifies the command path.
/usr/paragon/boot/DEVCONF.TXT	Specifies the device configuration file.
/usr/paragon/boot/MAGIC.MASTER	Specifies the master magic configuration file.
/usr/paragon/boot/SYSCONFIG.TXT	Specifies the system configuration file.

BOOTPP *(cont.)*   BOOTPP *(cont.)*

Errors

```
Invalid syntax on line num for 'Cabinet' command!
```

The cabinet number is missing or is out of range in a **CABINET** command line in the *SYSCONFIG.TXT* file.

```
Invalid syntax on line num for 'Backplane' command!
```

The backplane name is missing or is not **A**, **B**, **C**, or **D** in a **BP** command line in the *SYSCONFIG.TXT* file.

```
Invalid syntax on line num for 'Slot' command!
```

The slot number is missing or is out of range in a **S** command line in the *SYSCONFIG.TXT* file.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes.*

See Also

commands: **bootmesh**, **cbs**, **parsemagic**, **reset**

files: **bootmagic**, **BADNODES.TXT**, **DEVCONF.TXT**, **MAGIC.MASTER**, **SYSCONFIG.TXT**

CBS CBS

Displays the cabinet, backplane, slot (CBS) node numbering for a Paragon system.

Syntax

cbs [**-b** *number*] [**-c** *number*] [**-h**] [**-m** *path*] [**-x** *width*]
[**-y** *height*] [**-w** *number*] [*node* ...] [*cbs_number* ...]

Arguments

-b *number*	Specifies that there are *number* backplanes in each cabinet.
-c *number*	Specifies that there are *number* cabinets in a Paragon system.
-h	Help
-m *path*	Pathname of bootmagic file. The *path* can be a relative or absolute pathname.
-w *number*	Specifies to use *number* of columns in the output.
-x *width*	Specifies the mesh width (cabinets*4). The *width* must be multiple of 4.
-y *height*	Specifies the mesh height (backplanes*4). The *height* must be multiple of 4.
node	Specifies the root-partition node number of a node on a Paragon system.
cbs_number	Specifies the CBS number of a node on a Paragon system.

Description

CBS is a node numbering system for the Paragon system. Cabinet numbering starts on the cabinet's front right with 00 and increments across to the cabinet's front left. Backplane numbering starts with the letter A at the bottom of the cabinet and increments to the letter D at the top of the cabinet. Slot numbering starts with the number 0 (zero) on the right side of each backplane and goes to the number 15. A node number combines the two-digit cabinet number, the one-character backplane name, and the two-digit slot number.

CBS *(cont.)* CBS *(cont.)*

If you execute the **cbs** command without arguments, the command prints the CBS information for all the available nodes in the Paragon system. By default there are four columns of output that shows both the root-partition node numbering and the CBS numbering.

For example, in the following diagram the CBS number indicated by the two stars (**) is specified as 01B10.

```
  +-----------+-----------+
D | 12 11 4 3 | 12 11 4 3 |
  | 13 10 5 2 | 13 10 5 2 |
  | 14  9 6 1 | 14  9 6 1 |
  | 15  8 7 0 | 15  8 7 0 |
  +-----------+-----------+
C | 12 11 4 3 | 12 11 4 3 |
  | 13 10 5 2 | 13 10 5 2 |
  | 14  9 6 1 | 14  9 6 1 |
  | 15  8 7 0 | 15  8 7 0 |
  +-----------+-----------+
B | 12 11 4 3 | 12 11 4 3 |
  | 13 ** 5 2 | 13 10 5 2 |
  | 14  9 6 1 | 14  9 6 1 |
  | 15  8 7 0 | 15  8 7 0 |
  +-----------+-----------+
A | 12 11 4 3 | 12 11 4 3 |
  | 13 10 5 2 | 13 10 5 2 |
  | 14  9 6 1 | 14  9 6 1 |
  | 15  8 7 0 | 15  8 7 0 |
  +-----------+-----------+
       01          00
```

Examples

The following example shows how to display node-number information about specific nodes on a Paragon system using the root partition node number:

```
% /sbin/cbs 23 43
ROOT =   23, CBS = 00D01
ROOT =   43, CBS = 01C02
```

This example shows that node 23 has the CBS number 00D01 and node 43 has the CBS number 01C02.

CBS *(cont.)* CBS *(cont.)*

The following example shows how to display node-number information about a specific node on a Paragon system using the CBS number for the node:

```
% /sbin/cbs 00D01
CBS = 00D01, ROOT =    23, DELTA =    49
```

This example shows that the node with the CBS number 00D01 has a root partition node number of 23 and a Touchstone DELTA System node number of 49.

The following example shows how to display CBS information for all the available nodes in the system:

```
% /sbin/cbs
Configuration = Cabinets  2, Backplanes 4, Mesh   8 X 16

Root ID = CBS ID

7 = 00D03     15 = 00D02     23 = 00D01     31 = 00D00
41 = 01C10     42 = 01C05     43 = 01C02     44 = 00C13
45 = 00C10     46 = 00C05     49 = 01C09     50 = 01C06
51 = 01C01     52 = 00C14     53 = 00C09     54 = 00C06
57 = 01C08     58 = 01C07     59 = 01C00     60 = 00C15
61 = 00C08     62 = 00C07     64 = 01B12     65 = 01B11
66 = 01B04     67 = 01B03     68 = 00B12     69 = 00B11
70 = 00B04     73 = 01B10     74 = 01B05     75 = 01B02
76 = 00B13     77 = 00B10     78 = 00B05     81 = 01B09
82 = 01B06     83 = 01B01     84 = 00B14     85 = 00B09
86 = 00B06     87 = 00B01     89 = 01B08     90 = 01B07
91 = 01B00     92 = 00B15     93 = 00B08     94 = 00B07
96 = 01A12     97 = 01A11     98 = 01A04     99 = 01A03
100 = 00A12    101 = 00A11    102 = 00A04    103 = 00A03
105 = 01A10    106 = 01A05    107 = 01A02    108 = 00A13
109 = 00A10    110 = 00A05    113 = 01A09    114 = 01A06
115 = 01A01    116 = 00A14    117 = 00A09    118 = 00A06
121 = 01A08    122 = 01A07    123 = 01A00    124 = 00A15
125 = 00A08    126 = 00A07
```

The following example shows how to display CBS information about specific node in a system configuration that has one cabinet and four backplanes by specifying the number of cabinets and backplanes in the system:

```
% /sbin/cbs -c 1 -b 4 8
ROOT =     8, CBS = 00D14
```

CBS *(cont.)*

If the node number is outside the configuration, then asterisks (*) will be printed for CBS number as follows:

```
% /sbin/cbs  -c 1 -b 4 128
ROOT =    8, CBS = *****
```

The following example shows how to display information for all the nodes in a non-standard system configuration by specifying number of cabinets and backplanes:

```
% /sbin/cbs  -c 1 -b 4

Configuration = Cabinets  1, Backplanes 4, Mesh   4 X 16

Root ID = CBS ID

0 = 00D12     1 = 00D11     2 = 00D04     3 = 00D03
4 = 00D13     5 = 00D10     6 = 00D05     7 = 00D02
8 = 00D14     9 = 00D09     10 = 00D06    11 = 00D01
12 = 00D15    13 = 00D08    14 = 00D07    15 = 00D00
16 = 00C12    17 = 00C11    18 = 00C04    19 = 00C03
20 = 00C13    21 = 00C10    22 = 00C05    23 = 00C02
24 = 00C14    25 = 00C09    26 = 00C06    27 = 00C01
28 = 00C15    29 = 00C08    30 = 00C07    31 = 00C00
32 = 00B12    33 = 00B11    34 = 00B04    35 = 00B03
36 = 00B13    37 = 00B10    38 = 00B05    39 = 00B02
40 = 00B14    41 = 00B09    42 = 00B06    43 = 00B01
44 = 00B15    45 = 00B08    46 = 00B07    47 = 00B00
48 = 00A12    49 = 00A11    50 = 00A04    51 = 00A03
52 = 00A13    53 = 00A10    54 = 00A05    55 = 00A02
56 = 00A14    57 = 00A09    58 = 00A06    59 = 00A01
60 = 00A15    61 = 00A08    62 = 00A07    63 = 00A00
```

CBS *(cont.)* CBS *(cont.)*

Files

/sbin/cbs	Specifies the command path.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

bootpp, **parsemagic**

CHPART CHPART

Changes a partition's characteristics.

Syntax

chpart [**-epl** *priority*] [**-g** *group*] [**-mod** *mode*] [**-nm** *name*] [**-o** *owner*[**.** *group*]] [**-rq** *time* | **-sps**] *partition*

Arguments

-epl *priority*
Changes the partition's effective priority limit to the value of the *priority* argument. The *priority* argument is an integer from 0 to 10 inclusive. The **-epl** switch can be used on a gang-scheduled or space-shared partition only. You must have write permission on the specified partition.

-g *group*
Changes the partition's group. The *group* argument can be either a group name or number. You must be the owner of the specified partition and a member of the specified new group, or the system administrator.

-mod *mode*
Changes the partition's protection modes. The *mode* value can be specified as a three-digit octal number with the form *nnn* (see the **chmod** command) or a nine-character string with the form **rwxrwxrwx**, where a letter (**r**, **w**, or **x**) represents a permission granted and a dash (-) represents a permission denied (see the **ls** command's **-l** switch). You must be the owner of the partition or *root* to use this switch. See the *OSF/1 Command Reference* for more information about the **chmod** and **ls** commands.

-nm *name*
Changes the partition's name. This switch changes the partition's name only. The *name* argument must be a simple name (without dots). You must have write permission on the parent partition of the specified partition.

You can only change the partition's name "in place;" there is no way to move a partition to a different parent partition.

-o *owner*[*.group*]
Changes the partition's owner to *owner*. If the *group* argument is specified, this also changes the partition's group to the value of the *group* argument. The *owner* and *group* values can be either user/group names or numeric user/group IDs. You must be *root* to use this switch.

CHPART *(cont.)* CHPART *(cont.)*

-rq *time*

Specifies gang scheduling for the partition and changes the partition's rollin quantum to the value of the *time* argument. If the **-rq** switch is used on a space-shared partition, this switch changes the partition's scheduling to gang scheduling. The value of the *time* argument is one of the following:

n	*n* milliseconds. If *n* is not a multiple of 100, it is rounded up to the next multiple of 100.
*n***s**	*n* seconds.
*n***m**	*n* minutes.
*n***h**	*n* hours.
0	Infinite time. When an application is rolled in, it runs until it exits.

The *time* value must be less than 24 hours; the minimum rollin quantum for your system is determined by your system administrator.

The **-rq** switch can be used only on a gang-scheduled or space-shared partition, and cannot be used together with the **-sps** switch. To use the **-rq** switch, you must have write permission on the specified partition.

-sps

Changes the partition to a space-shared partition.

The **-sps** switch can be used only on a space-shared or gang-scheduled partition, and cannot be used together with the **-rq** switch. If the partition is currently gang-scheduled, it must not contain any overlapping subpartitions or any applications. To use the **-sps** switch, you must have write permission on the specified partition.

partition

Absolute or relative pathname of a partition.

CHPART *(cont.)* CHPART *(cont.)*

Description

The **chpart** command lets you change the following partition characteristics:

- Rollin quantum.
- Effective priority limit.
- Protection modes.
- Name.
- Owner.
- Group.
- Scheduling type (gang scheduling or space sharing).

You can use the **chpart** command to change a partition's scheduling type from gang scheduling to space sharing, or change the scheduling from space sharing to gang scheduling. If a partition uses standard scheduling, you cannot change its scheduling characteristic.

You cannot change a partitions size or parent partitions. These characteristics are set when the partition is created.

Examples

The following changes the *mypart* partition to a gang-scheduled partition and sets the rollin quantum to 20 minutes:

```
% chpart -rq 20m mypart
```

The following changes the *mypart* partition to a space-shared partition:

```
% chpart -sps mypart
```

The following changes the effective priority limit for the *mypart* partition to 2:

```
% chpart -epl 2 mypart
```

CHPART *(cont.)* CHPART *(cont.)*

The following changes the protection modes of the *mypart* partition so that it is readable, writable, and executable by everyone:

```
chpart -mod 777 mypart
```

The following changes the owner of *mypart* to *smith*, but does not affect the group:

```
% chpart -o smith mypart
```

Errors

`Allocator internal error`

An internal error occurred in the node allocation server.

`Change to space shared partition not allowed.`

You tried to change a gang-scheduled partition to a space-shared partition, but the partition has an application running in it or the partition contains overlapping partitions.

`Exceeded allocator configuration parameters.`

You specified too many gang-scheduled partitions. See the **allocator** manual page for information about the maximum number of gang-scheduled partitions.

`Invalid group.`

You specified an invalid group name for the **-g** or **-o** switch.

`Invalid partition rename.`

You specified a partition name for the **-nm** switch that was not a simple name.

`Invalid priority.`

You specified an invalid priority limit for the **-epl** switch.

CHPART *(cont.)* CHPART *(cont.)*

```
Invalid user.
```

You specified an invalid user name for the **-o** switch.

```
Partition lock denied.
```

You specified a partition that is currently in use and being updated by someone else. You cannot change the characteristics of a partition that is currently being updated.

```
Partition not found.
```

You specified a partition that does not exist.

```
Partition permission denied.
```

You specified a partition for which you do not have the appropriate permissions or ownership for the operation you are trying to perform.

```
Scheduling parameters conflict with allocator configuration.
```

You specified a rollin quantum that is less than what is allowed. See the **allocator** manual page for information about the minimum rollin quantum.

Files

/usr/bin/chpart
Specifies the command path.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: ***application***, **lspart**, **mkpart**, **pspart**, **rmpart**, **showpart**

calls: **nx_chpart_epl()**, **nx_chpart_mod()**, **nx_chpart_name()**, **nx_chpart_owner()**, **nx_chpart_rq()**, **nx_chpart_sched()**

CONSOLE CONSOLE

Diagnostic station: Starts a console connection to the Paragon system.

Syntax

console

Description

The **console** command runs on the diagnostic station and is intended for use by the system administrator only.

The **console** command is a script that the **reset** command creates on the diagnostic station when the system is rebooted. The **console** command is created using the last console connection to the Paragon system. The **console** command supports the **async**, **fscan**, and the **scanio** console interfaces.

Example

The following shows an example console script:

```
#!/bin/sh
echo "mysys was last booted on Sun Oct 31 14:13:12 PDT 1993"
/usr/local/bin/fscan -bD -c full
```

Files

/usr/paragon/boot/console
Script file for the **console** command.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

async, **fscan**, **reset**, **scanio**

COREINFO COREINFO

Displays summary information of a core file or core directory.

Syntax

coreinfo [*corename*]

Arguments

corename	Pathname of a core file or core directory.

Description

The **coreinfo** command displays summary information about a core file or the core files located in a core directory. See the **core(4)** manual page for more information about core files and core directories. If *corename* is omitted, the command searches for a file or directory called *core* in the directory defined in the environment variable *CORE_PATH.* If *CORE_PATH* is undefined, the command searches the current working directory. An error is returned if the command fails to find a core file or core directory to report on.

The summary information includes the following:

- The number of nodes in the partition on which the failed application was loaded. This is omitted if a core file for a non-parallel application is specified.
- A table containing one line for each core file found in the core directory. If a single core file within a core directory is specified, a single line will be displayed.

The table contains the following information about the core files:

- The time that the process terminated (that is, faulted or killed). The table is ordered chronologically using the Date/Time column.
- The process ID (PID) of the faulting process,
- The node and process type of the process (for parallel applications only).
- The location in the program where the precess execution was halted. This address has an asterisk (*) next to it if the execution point being reported is for a thread other than the main user thread.

- The signal that terminated the process.
- The type of core file (FULL or TRACE).
- The name of the executable.

Examples

This example displays summary information about the default core directory. It indicates that the application *myapp* was running on four nodes when three of the processes encountered a segmentation violation (SIGSEGV). The fourth process (process zero on node two) was killed by the system. The three faulting processes dumped complete core files (FULL), while the non-faulting process dumped only stack trace data (TRACE). It is presumed from this output that the core action environment variables were defined as *CORE_ACTION_FIRST*=FULL, *CORE_ACTION_FAULT*=FULL, and *CORE_ACTION_OTHER*=TRACE when the application was executed. The following displays summary information about the default core directory.

```
# coreinfo
Summary information for directory: /usr/joe/core
Number of nodes: 4
Date/Time      Pid     Node   Ptype  Signal  Location    Type   Executable
---------      ---     ----   -----  ------  --------    ----   ----------
Oct 20 10:12   327684  3      0      SIGSEGV 0x000108fc  FULL   /home/joe/myapp
Oct 20 10:12   9       0      0      SIGSEGV 0x0001085c  FULL   /home/joe/myapp
Oct 20 10:12   65541   1      0      SIGSEGV 0x0001085c  FULL   /home/joe/myapp
Oct 20 10:12   262153  2      0      SIGKILL 0x000108e8  TRACE  /home/joe/myapp
```

This next example displays information about core file *core_save*. In this case, a non-parallel (or UNIX) application *testprog* encountered a bus error (SIGBUS) at 0x000102dc and dumped a complete core file. The absence of node and ptype information are the indicators of a non-parallel application.

```
# coreinfo core_save
Summary information for file: /usr/fred/core_save
Date/Time        Pid     Signal  Location    Type  Executable
---------        ---     ------  --------    ----  ----------
Oct 25 10:44     196973  SIGBUS  0x000102dc  FULL  /home/fred/testprog
```

COREINFO *(cont.)* COREINFO *(cont.)*

This final example displays information about a single process in a parallel core dump. The process of interest has pid 65541.

```
# coreinfo core/core.65541
Summary information for file: /usr/joe/core/core.65541
Number of nodes: 4
Date/Time      Pid    Node   Ptype  Signal  Location    Type  Executable
---------      ---    ----   -----  ------  --------    ----  ----------
Oct 20 10:12   65541  1      0      SIGSEGV 0x0001085c  FULL  /home/joe/myapp
```

See Also

core, pspart

CREATE_PF CREATE_PF

Creates a paging file.

Syntax

create_pf *size*[**M**] [*file*]

Arguments

size	Specifies the size of the paging file in blocks. The *size* argument must be an integer greater than 0 (zero).
M	Specifies that the block size is 1M byte. The default block size is 1K byte.
file	Specifies the file name of the paging file. The default is a the file *paging_file*, and it is created in the current directory. The file name may be an absolute or relative pathname.

Description

The **create_pf** command creates a paging file with a size of *size* blocks. A paging file is a file that provides additional paging space for the boot node. The additional disk space of the paging file allows the boot node to handle paging requests that are larger than the default paging space. If you append an **M** to the *size* argument, the **create_pf** command creates a paging file with a size of *size* megabyte blocks.

After creating the paging file, the **create_pf** command displays a message that the paging file is created.

Examples

The following example creates a paging file with the default name paging_file and a size of 10K bytes:

```
# /sbin/create_pf 10
```

The following message is printed when paging file is created:

```
Creating 'paging_file' of size (0 Meg) 10 blocks.
```

CREATE_PF *(cont.)* CREATE_PF *(cont.)*

The following example creates a paging file with the pathname */home/pf.new* and a size of 20M bytes:

```
# /sbin/create_pf 20M /home/pf.new
```

The following message is printed when the paging file is created:

```
Creating '/home/pf.new' of size (20 Meg) 20480 blocks.
```

Errors

```
create_pf: Invalid block count <n>
```

You specified an invalid size for the paging file.

Files

/sbin/create_pf	Specifies the command path

DEVSTAT

Displays the node numbers for the node to which a device is attached.

Syntax

devstat [-v] *file ...*

Arguments

-v	Specifies verbose mode. All messages are displayed.
file	Specifies the device file name.

Description

The **devstat** command outputs the root partition node number for the node that a device is attached to. Using the **-v** flag outputs device name and node number.

Examples

The following example displays the node number for the node the device */dev/io0/rz0a* is attached to:

```
# /sbin/devstat /dev/io0/rz0a
3
```

The command returns the root-partition node number 3.

Errors

```
<file>: No such file or directory
```

Specified device does not exist.

```
<file>: Invalid argument
```

Specified device is not a valid device, it may be a regular file.

DEVSTAT *(cont.)* DEVSTAT *(cont.)*

Files

/sbin/devstat	Specifies the command path

See Also

rmknod

OSF/1 Command Reference: **ls(1)**, **mknod(8)**,

DF DF

Display statistics on free disk space.

Syntax

df [**-ikn**] [**-t** *type*] [*file* | *file_system*]

Arguments

-i	Includes statistics on the number of free inodes.
-k	Causes the numbers to be reported in kilobytes. By default, all reported numbers are in 512-byte blocks.
-n	Prints out the previously obtained statistics from all mounted file systems. Use this flag if it is possible that one or more file systems are in a state such that they will not be able to provide statistics without a long delay (for example, a remote file system on a server that has crashed). When this flag is specified, **df** does not request new statistics from the file systems; for some remote file systems, the statistics displayed may be too old to be useful.
-t *type*	Displays statistics for the specified file system type only. If the **-t** flag is specified and a file or filesystem is specified, the **-t** flag is ignored. Available file system types include the following:

ufs	UNIX File System (Berkeley Fast File System) (default)
pfs	Parallel File System (PFS)
nfs	Network File System

If the **-t** flag is specified and a *file* or *file_system* argument is specified, the **-t** flag is ignored.

file	Pathname of a file.
file_system	Pathname of a mounted file system.

Description

Using the **df** command on a PFS file system gives information about the single disk partition on which the PFS file system is mounted. The **df** command does not give information about how much cumulative space is actually available for PFS file striping. Use the **showfs** command to get information about the cumulative amount of free space in a PFS file system.

If neither a file nor a file system is specified, statistics for all mounted file systems are displayed.

When file system disk usage exceeds 100% of the allowed space for users, the **df** command displays a negative number of free blocks. The allowed space for users is typically 90% of disk capacity, with 10% reserved for use by root only. However, system administrators may specify either less or more reserved space for use by root.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

Commands: **du(1)**, **mount(8)**, **showfs(1)**, **quot(8)**.

Functions: **fstatfs(2)**, **getmntinfo(3)**, **getpfsinfo(3)**, **mount(2)**, **statfs(2)**, **statpfs(3)**.

Files: **fstab(4)**, **pfstab(4)**.

EXPORTPAGING EXPORTPAGING

Exports a *Vnode* pager service from the non-boot nodes listed in the *EXPORT_PAGING* bootmagic string.

Syntax

exportpaging [**-vdn**]

Arguments

-v	Displays all messages.
-d	Displays debug information at runtime.
-n	Displays intended actions, but does not perform the **swapon**() system calls.

Description

The *EXPORT_PAGING* bootmagic string lists the MIO nodes (nodes with disks) that export a *Vnode* paging service. Paired with the *EXPORT_PAGING* bootmagic string is the *PAGER_NODE* bootmagic string. The *PAGER_NODE* bootmagic string lists the nodes that import their default paging service and the specific node that supplies this service. The boot node always exports a *Vnode* paging service and uses */mach_servers/pagingfile* as its default paging service. The boot node is the root of all paging trees, single or multi level. Using the correct definition of the *EXPORT_PAGING* and *PAGER_NODE* bootmagic strings, you can construct a multi-level paging tree. The purpose of a multi-level paging tree is to distribute the virtual memory (VM) page-out traffic over multiple disks, thus, reducing system paging bottlenecks.

The **exportpaging** command does the following:

- For each non-boot node listed in the bootmagic string *EXPORT_PAGING*, the **exportpaging** command finds the block-special paging device for that node and exports a *Vnode* paging service from that node.

- The default block-special device files */dev/io*/rz[0123456]b* are searched. The **exportpaging** command attempts to match the node numbers from the *EXPORT_PAGING* bootmagic string with the node numbers stored in the device special file. For more information, see the **rmknod(3)** manual page in the *OSF/1 Programmer's Reference.*

- When a match is detected the **swapon(2)** system call is called using the block-special device pathname. The **swapon(2)** system call instructs the *Vnode* pager at the specific node to register, with that node's UNIX server, a Mach port as an exported paging port. It is this exported paging port that will be utilized by other nodes as their default paging port.

EXPORTPAGING *(cont.)* EXPORTPAGING *(cont.)*

You can change the default block special paging partition/device *rz0b* by adding an entry to the file */usr/paragon/boot/DEVCONF.TXT* on the diagnostic station. In the *DEVCONF.TXT* file line which identifies the MIO node, add the *PAGE_TO* string plus the *device* argument using the following format:

```
MIO cbs rev PAGE_TO pagingdevice
```

The *device* argument specifies a block special disk dev ice (partition) at the MIO node. The following example specifies using the block special device (partition) *rz0d* instead of *rz0b*:

```
MIO 01D12 H04 PAGE_TO r20d
```

NOTE

The *PAGE_TO* string has no affect on the boot node. The boot node uses the default paging device *r20b*.

A specific MIO can be excluded from paging by the addition of the *NO_PAGER* string to the line in the *SYSCONFIG.TXT* file describing the MIO node.

```
MIO 01D12 H04 NO_PAGER
```

Limitations and Workarounds

Exit status is normally 0.

If a node that exports a *Vnode* paging service does not boot, all nodes that import their paging service from the dead node will hang.

For more information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **bootpp**

files: **DEVCONF.TXT**, **SYSCONFIG.TXT**

OSF/1 Programmer's Reference: **swapon(2)**, **rmknod(3)**

FIND

Finds files matching an expression.

Syntax

find *pathname ... expression*

Description

The **find** command recursively searches the directory tree for each specified pathname, seeking files that match a Boolean *expression* written using the terms given later. The output from **find** depends on the terms used in *expression.*

Expressions

In the following descriptions, the argument *number* is a decimal integer that can be specified as +*number* (more than *number*), -*number* (less than *number*), or *number* (exactly *number*).

-fstype *type* TRUE if the file system to which the file belongs is of the type *type* as follows:

nfs	Network File System
pfs	Parallel File System (PFS).
ufs	UNIX file system (Berkeley fast file system). This is the default.

-i *number* TRUE if file has inode *number.*

-inum *number* TRUE if file has inode *number.*

-name *pattern* TRUE if *pattern* matches a filename. You can use pattern-matching characters, provided they are quoted.

-perm [-]*octal_number*
TRUE if the file permission code of the file exactly matches *octal_number* (see the **chmod** command). If the optional - (dash) is present, this expression evaluates to TRUE if the file permission code of the file meets or exceeds *octal_number.*

The *octal_number* argument may be up to three octal digits.

-perm [-]*mode*	The *mode* argument is identical to the **chmod** command syntax. This expression evaluates to TRUE if the file has exactly these permissions. If the optional **-** (dash) is present, this expression evaluates to TRUE if the file meets or exceeds these permissions.
-prune	Always TRUE. Prunes the search tree at the file. That is, if the current pathname is a directory, **find** does not descend into that directory. In a networking environment, this flag keeps the **find** command from searching through remote file systems.
-type *type*	TRUE if the file *type* is of the specified type as follows:

b	Block special file
c	Character special file
d	Directory
f	Plain file
l	Symbolic link
p	FIFO (a named pipe)
s	Socket

-links *number*	TRUE if the file has *number* links. The argument *number* is a decimal integer that can be specified as +*number* (more than *number*), -*number* (less than *number*), or *number* (exactly *number*). See the **ln** command.
-user *user*	TRUE if the file belongs to *user*. If *user* is numeric and does not appear as a login name in the **/etc/passwd** file, it is interpreted as a user ID.
-nouser	TRUE if the file belongs to a user ID for which the **getpwuid()** function returns null.
-group *group*	TRUE if the file belongs to *group*. If *group* is numeric and does not appear as a group name in the **/etc/group** file, it is interpreted as a group ID.
-nogroup	TRUE if the file belongs to a group ID for which the **getgrgid()** function returns null.

-size *number*[**c** | **k**]
TRUE if the file is *number* blocks long (512 bytes per block). For this comparison, the file size is rounded up to the nearest block. If the **c** argument is present, the expression evaluates to TRUE if the file is *number* bytes long. If the **k** argument is present, the expression evaluates to TRUE if the file is *number* kilobytes long. For this comparison, the file size is rounded up to the nearest kilobyte.

The argument *number* is a decimal integer that can be specified as +*number* (more than *number*), -*number* (less than *number*), or *number* (exactly *number*).

-atime *number*
TRUE if the file was accessed in the past *number* days. The argument *number* is a decimal integer that can be specified as +*number* (more than *number*), -*number* (less than *number*), or *number* (exactly *number*).

-mtime *number*
TRUE if the file was modified in the past *number* days. The argument *number* is a decimal integer that can be specified as +*number* (more than *number*), -*number* (less than *number*), or *number* (exactly *number*).

-ctime *number*
TRUE if the file inode was changed in the past *number* days. The argument *number* is a decimal integer that can be specified as +*number* (more than *number*), -*number* (less than *number*), or *number* (exactly *number*).

-exec *command*
TRUE if the *command* runs and returns a 0 (zero) value as exit status. The end of *command* must be punctuated by a quoted or escaped **;** (semicolon). The command parameter { } is replaced by the current pathname. If shell quoting is used in *command*, each word in the command must be quoted separately. Also, the characters **;** (semicolon) and **{ }** (braces) must appear as separate words on a command line.

-ok *command*
This expression is equivalent to **-exec**, except that the **find** command first asks you whether it should start *command*. If your response begins with **y**, or the locale's equivalent of a **y**, *command* is started. The end of *command* must be punctuated by a quoted or escaped semicolon. If shell quoting is used in *command*, each word in the command must be quoted separately. Also, the characters **;** (semicolon) and **{ }** (braces) must appear as separate words on a command line.

-print
Always TRUE; causes the current pathname to be displayed. The **find** command assumes a **-print** expression, unless the **-exec**, **ls**, or **-ok** expressions are present.

-cpio *device*
Writes the current file to *device* in **cpio** format. See the **cpio** command.

-ncpio *size*
Sets the input/output size (5120 bytes by default) to *size*.

-newer *file*
TRUE if the current file was modified more recently than the file indicated by *file*.

FIND *(cont.)* FIND *(cont.)*

-depth	Always TRUE. This causes the descent of the directory hierarchy to be done so that all entries in a directory are affected before the directory itself. This can be useful when **find** is used with **cpio** to transfer files that are contained in directories without write permission.
e(*expression* **e)**	TRUE if *expression* is TRUE.
-ls	Always TRUE; causes *pathname* to be printed together with its associated statistics. These include, respectively, inode number, size in kilobytes (1024 bytes), protection mode, number of hard links, user, group, size in bytes, and modification time. If the file is a special file, the size field will instead contain the major and minor device numbers. If the file is a symbolic link, the pathname of the linked-to file is printed preceded by **->**. The format is identical to that of **ls -gilds** (note that formatting is done internally, without executing **ls**.)
-xdev	Always TRUE; causes **find** to not traverse down into a file system different from the one on which current pathname resides.

The primaries may be combined using the following operators (in descending order of precedence):

1. A parenthesized group of primaries and operators (parentheses are special to the shell and must be escaped).

2. The negation of a primary (! is the unary **not** operator).

3. Concatenation of primaries (the **and** operation is implied by the juxtaposition of two primaries or may be explicitly stated as **-a**).

4. Alternation of primaries (**-o** is the **or** operator).

To avoid unpredictable results when using a range expression to match a class of characters, use a character class expression rather than a standard range expression. For information about character class expressions, see the **grep** command.

Examples

1. To list all files in the file system with a given base filename, enter:

```
find / -name .profile
```

This searches the entire file system and writes the complete pathnames of all files named **.profile**. The / (backslash) tells **find** to search the root directory and all of its subdirectories. This may take a while, so it is best to limit the search by specifying the directories where you think the files might be.

FIND *(cont.)* FIND *(cont.)*

2. To list the files with a specific permission code in the current directory tree, enter:

   ```
   find . -perm 0600
   ```

   This lists the names of the files that have only owner-read and owner-write permission. The **.** (dot) tells **find** to search the current directory and its subdirectories. See the **chmod** command for details about permission codes. Alternatively, you could enter the following:

   ```
   find . -perm u+rw
   ```

3. To search several directories for files with certain permission codes, enter:

   ```
   find manual clients proposals -perm -0600
   ```

   This lists the names of the files that have owner-read and owner-write permission and possibly other permissions. The directories **manual**, **clients**, and **proposals**, and their subdirectories, are searched. Note that **-perm 0600** in the previous example selects only files with permission codes that match **0600** exactly. In this example, **-perm -0600** selects files with permission codes that allow at least the accesses indicated by **0600**. This also matches the permission codes **0622** and **2744**.

4. To search for regular files with multiple links, enter:

   ```
   find . -type f -links +1
   ```

   This lists the names of the ordinary files (**-type f**) that have more than one link (**-links +1**). Note that every directory has at least two links: the entry in its parent directory and its own **.** (dot) entry. See the **ln** command for details about multiple file links.

5. To remove all files named **a.out** or ***.o** that have not been accessed for a week and that are not mounted using **nfs**, enter:

   ```
   find / e( -name a.out -o -name (aa*.o(aa e) -atime +7 \
   -exec \rm {} e; -o -fstype nfs -prune
   ```

FIND *(cont.)* FIND *(cont.)*

6. To use the find command on PFS:

```
find /pfs -fstype pfs -print

/pfs
/pfs/f1
/pfs/f2
/pfs/f3
/pfs/d1
/pfs/d1/f1
/pfs/d1/f2
/pfs/d1/f3
/pfs/d1/f4
```

Files

/etc/group	Contains group information.
/etc/passwd	Contains user information.

Return Values

The **find** command returns a 0 (zero) if all of the paths were visited without error. **find** returns a nonzero value if it encountered an error.

See Also

Commands: **chmod(1)**, **cpio(1)**, **grep(1)/ egrep(1)/ fgrep(1)**, **ln(1)**, **sh(1)**, **test(1)**.

Functions: **stat(2)**.

Files: **fs(4)**.

FSCAN FSCAN

Diagnostic station: Provides fast-scan console communications between a diagnostic station and a Paragon system

Syntax

fscan [-Aqvwx] [**-b** *backplane*] [**-B** *file*] [**-c** *configuration*] [**-F** *file*] [**-s** *string*] [*node*]

Arguments

-A	Specifies that the **fscan** utility can communicate with the nodes specified in the bootmagic strings BOOT_NODE_LIST and BOOT_ALT_NODE_LIST. By default, the **fscan** utility communicates with the nodes specified in the bootmagic string BOOT_NODE_LIST only.
-b *backplane*	Specifies the top backplane in the system. The *backplane* argument must be either **A**, **B**, **C**, or **D**. The default value is **D**. Use this switch whenever the Paragon system does not have four backplanes. For example, if a system has three backplanes **A**, **B**, and **C**, then specify **-bC** for the top backplane **C**.
-B *file*	Specifies the name of the bootmagic file to read. The default is the file *bootmagic* in the current directory. The **fscan** utility reads the bootmagic file to determine boot parameters such as the boot node, node lists, and mesh size.
-c *configuration*	Specifies the system configuration. The *configuration* argument can be one of the following: **condo**, **full**, or **multi**. The default is **full**. The **condo** system is contained within one backplane consisting of between 1 and 16 nodes. (Some sites have multiple systems within the same cabinet. Although this is not a supported configuration, the **fscan** utility provides support for this configuration.) The **full** system has four backplanes. The **multi** system has two or three backplanes.
-q	Specifies that warning and informational messages be suppressed. Error messages are still reported.

FSCAN *(cont.)*   FSCAN *(cont.)*

-s *string* Sends the value of the *string* argument to the specified node, then resumes communications to the node. The default value is a space (**\b**). Sending the *string* argument to a node starts communications with a node and causes the node to send back a prompt. The *string* argument can be one of the following values:

\b	Space (0x20)
\n	Carriage return (0x13)
\r	Line feed (0x0A)
\t	Tab (0x09)
\	Single backslash (\)

-v Displays all messages. Use this switch for debugging your node connection.

-w Specifies running the **fscan** utility with the system watchdog without the console interface. Use this switch when the diagnostic station is not connected as a console, but the system watchdog is needed. Redirect standard output to a file and run the **fscan** utility in the background.

-x Connects to the boot node using a soft connect. A soft connect is a connection where initial handshaking is bypassed. Use this when the boot node is coming up and is vulnerable to interrupts.

node Node number of the node to connect the console to. The default is the boot node.

Description

The **fscan** utility runs on the diagnostic station and is used by the system administrator to establish fast-scan console communications with a Paragon system.

The **fscan** utility combines a fast console interface and the system watchdog. This is a diagnostic utility that evaluates the system's nodes and detects when nodes crash in the system. The **fscan** utility provides the following features:

- A console interface that lets you connect to any system node from your terminal.
- A system watchdog that starts when the system boots, detects when nodes crash, and automatically reboots the system.
- A command interface for accessing nodes from the console.

FSCAN *(cont.)* FSCAN *(cont.)*

The **fscan** utility can run with a console interface and the system watchdog, or run the system watchdog by itself. Running the system watchdog by itself improves performance in checking for node crashes. You run the system watchdog by itself using the **-w** switch with the **fscan** utility. The system watchdog can only be run by itself if the node connection is through the serial line. You can set up the serial connection using the **async** command. See the **async** manual page for more information.

NOTE

The **fscan** and **scanio** utilities cannot be used at the same time.

Console Commands

You can enter the following commands at your console prompt:

~* Invokes the **fscan** command interface and returns the `FSCAN>` prompt. At the prompt, you can use the **fscan** commands to change command settings, define new commands, and query the state of the Paragon system. The following example shows how to invoke the **fscan** prompt and execute an **fscan** command:

```
# ~*
FSCAN> ping
FSCAN>
#
```

At the `FSCAN>` prompt, the example uses the **ping** command to check the status of the node. Entering a carriage return without entering a command exits **fscan** and the console prompt returns.

~! Invokes a prompt that lets you execute a diagnostic station command on the node from the console. After the operating system command completes, control returns to the console. You can use the **sh** command to create a shell if you want to enter more than one command. After invoking this command from the console prompt, you should see the following prompt:

```
# ~!
UNIX>
```

After entering a command, console prompt returns.

~.
Exits the **fscan** utility and the console. Be careful using this when you are remotely logged in to the diagnostic station. Use **~~.** or **~q** when remotely logged in.

~q
Exits the **fscan** utility and the console. This command is identical to **~.**, but does not cause problems when you are remotely logged in to the diagnostic station.

~#
Invokes a NODE> prompt that lets you switch the console connection to another node. This is the same as the **fscan** utility's **switch** and **node** commands. After executing this command from the console prompt, you get a special prompt from which you can enter a node number. For a system with valid nodes 0 to 15, the following shows how to change the node the console is connected to:

```
# ~#
Valid Nodes - 0..15
Node: 2
Switching to node 2 (00A04) ...
```

After entering ~#, the **fscan** utility returns the valid node numbers available. Enter a valid node number at the prompt (in this example node 2). After a carriage return, the console is connected to the specified node. If you are connected to the boot node, a console prompt is return. Otherwise, no prompt is returned.

fscan Commands

You can enter the following commands at the FSCAN> prompt or in the **fscan** configuration file (default *fscan.cfg*):

adjust [*skew_factor*]
Adjusts the skew factor of the square-wave signal generated by the **calibrate** command. The *skew_factor* argument specifies the width of the diagnostic signal. If you do not specify a *skew_factor* argument, a skew factor based on the last **calibrate** command is used.

calibrate
Generates a square-wave signal between the connected node and the diagnostic station. The wave form, the wave frequency, the measured high and low pulse widths, and the calculated skew factor are displayed. This command helps to evaluate backplane skew when communicating with nodes in cabinets whose number is greater than 0 (zero).

define *cmd string*
Defines new commands you can execute from the **fscan** prompt. Enclose the *string* argument within double quotes if embedded spaces are used. The *string* argument must be a shell script or an executable.

FSCAN *(cont.)* FSCAN *(cont.)*

flush [*time*]

Flushes output from the scan lines to the console, until the node connected to the console goes silent. The *time* argument specifies the number of seconds to wait for output after the node goes silent. The default is one second.

global *string*

Sends the *string* argument to every node in the system. The *string* argument can be a sequence of the following characters:

^*X*	The caret (^) specifies sending the control character for the letter specified by the *X* variable. The *X* variable can be any character from A to Z.
\n	Carriage return (0x13)
\r	Line feed (0x0A)
\\^	Tab (0x09)
\\\\	Single backslash (\\)

For example, to send a control-P character to every node in the system, use the following command:

```
FSCAN> global ^P
```

local *string*

Sends the *string* argument to the node connected to the console. The *string* argument can be a sequence of the following characters:

^*X*	The caret (^) specifies sending the control character for the letter specified by the *X* variable. The *X* variable can be any character from A to Z.
\n	Carriage return (0x13)
\r	Line feed (0x0A)
\\^	Tab (0x09)
\\\\	Single backslash (\\)

For example, to send a carriage return to the node connected to the console, use the following command:

```
FSCAN> local \n
```

FSCAN *(cont.)* FSCAN *(cont.)*

more on | **off**	Turns scrolling on or off for the **rollcall** command. The default is **off**, which is no scrolling. When this is turned **on**, a **more** command prompt is displayed and you can scroll the output one screen at a time.
node [*node*]	Same as the **switch** command.
ping	Gets status of a node that is currently connected to the system.
rc [*count*]	Same as the **rollcall** command.
rollcall [*count*]	Checks the status of each node in the system and displays the node number, CBS number, and the state of the node. The *count* argument specifies how many times to check the status of a node if it does not respond. The default for the *count* argument is 1.

set *option* **on** | **off**

Turns options on or off. Any other value entered will generate an error message. The *option* argument can be one of the following:

autobucket	Setting this to **on** automatically disables the processor port on the MRC for any node that is not in a *running* state. A disabled processor port can be enabled again when the system is reset.
autoreboot	Setting this to **on** causes the **reboot** command to be executed if a node is in a *dead* or *debugger* state. This reboots the system even if a node drops into the debugger. This option has no effect if the polling option is **off**.
autoswitch	Setting this option to **on** causes **fscan** to automatically switch to any node that drops into the debugger. Once this switch happens, this option is turned off to prevent **fscan** from alternating between 2 or more nodes.
notify	Setting this option to **on** generates a message whenever a node is detected in the *dead* state or in the *debugger* state.
polling	Setting this to **on** activates polling in the system watchdog. The system watchdog begins checking the operations of the node the console is connected to. Setting this to **off** prevents polling. This is usually set in the **fscan** configuration file.

FSCAN *(cont.)* FSCAN *(cont.)*

switch [*node*] [*]	Switches the console connection to any node in the system. If *node* is not specified, the default is the boot node. If the specified *node* value is **?** (question mark), the list of valid nodes is displayed. If you specify an asterisk (*) as an argument, the `FSCAN>` prompt is returned. Otherwise, the console prompt is returned.
wait [*time*]	Waits until a node drops into the *debugger* state, then it connects the console to that node. The *time* argument specifies the amount of time to poll every node. Entering any key aborts the wait and returns you to the `FSCAN>` prompt.

Node States

The system watchdog queries and determines the state of each node on the system. Nodes in the system can have one of the following states:

running	The kernel is operating normally.
debugger	The kernel enters the debugger because of a panic in the kernel or the server. The node is effectively dead and the system is rebooted (if the system is configured for autoreboot). When the server panics and enters the debugger, the server must use the kernel debugger and set the node state to *debugger.* You can also use **`<Ctrl-P>`** to enter the debugger.
dead	The node does not respond to three attempts to get its state. This means the node is dead or the node is configured with the **scanio** console. Either way, the node is flagged as dead and the system is rebooted (if the system is configured for autoreboot).

In addition, there are two transient node states:

suspicious	The node did not respond to the first request for its state. Because the **fscan** utility is interrupt driven, the node should respond. However, declaring a node as dead is serious so the node is given two more chances.
comatose	The node did not respond to the second request for its state. The next time the node does not respond it is flagged as dead and the system is rebooted (if the system is configured for autoreboot).

These states are transient, because the next time the node status is taken the state changes.

Configuration File

The configuration file *fscan.cfg* contains **fscan** commands that specify how to start the **fscan** utility when the system is booted. The **reset** command looks for this file in your current directory (usually */usr/paragon/boot*). If the **reset** command does not find the file, it automatically creates one in your current directory. The default configuration file contains the following commands:

```
set polling on
set autoreboot off
set autoswitch off
set autobucket off
set notify off
define reboot \
   "ksh /usr/paragon/boot/reset skip ignorelock autoreboot"
```

The default configuration file sets polling and automatic rebooting to on; sets automatic node switching, automatic processor disabling, and notification of dead processors to off; and defines the **reboot** command.

You can edit the *fscan.cfg* file with any standard editor (for example, **vi**). The comment character # (pound sign) can be used in column one. All characters to the right of the comment character are ignored.

Starting a Console Interface and the System Watchdog

Do the following on the diagnostic station to boot up the system with a console interface and the system watchdog:

1. Edit the *MAGIC.MASTER* file in the directory */usr/paragon/boot* on the diagnostic station. Change the *BOOT_CONSOLE* string as follows:

   ```
   BOOT_CONSOLE=f
   ```

   This requests booting with the **fscan** console.

2. Edit the *fscan.cfg* file in the directory */usr/paragon/boot* on the diagnostic station. Change *polling* to **on** as follows:

   ```
   set polling on
   ```

   This requests to start the system watchdog with *polling* on.

3. Use the **reset** utility to reset the system.

   ```
   DS# reset
   ```

   The **reset** script calls the **fscan** utility during the reboot and establishes the console connection to the boot node of the system.

4. When the system completes booting, the console prompt (#) is returned, the console is connected to the boot node, and the system watchdog is started automatically. You now have console communications with the boot node. By default, the **fscan** utility always connects to the boot node first. You can change this in the **fscan** command line by specifying which node to connect to first.

As long as the system is up and running, the system watchdog continues to run. The only way to stop the watchdog is to stop the **fscan** utility. If you are using **fscan** as a console, use the **~q** command to stop **fscan** and kill the system watchdog. Otherwise, use the **kill** command with the process ID (PID) for the **fscan** job. You can find the PID for the **fscan** job in the file */tmp/FSCAN.LOCK*.

Setting Up a Serial Interface with the System Watchdog

Do the following on the diagnostic station to set up a serial interface running the system watchdog only:

1. Edit the *MAGIC.MASTER* file in the directory */usr/paragon/boot* on the diagnostic station. Change the *BOOT_CONSOLE* string as follows:

   ```
   BOOT_CONSOLE=cm
   ```

   In the argument **cm** the **c** requests a serial connection if an MIO board is present and the **m** requests an interface to the system mesh. If the MIO board is not present, the **c** is ignored and you are connected to the mesh. If necessary, change the options in the configuration file *fscan.cfg* to turn **autoreboot on** or **off**.

2. Invoke the **reset** command with the **watchdog** argument to start the system watchdog:

   ```
   DS# reset watchdog
   ```

   The **reset** script does the following:

   - Executes the **fscan** utility with the **-w** switch to start the system watchdog only.

   - Executes the **async** command to set up serial communications with the boot node.

Rebooting the System Automatically

You can reboot a Paragon system either automatically or using the **reboot** command. Do the following to set up an automatic reboot:

- Set the **autoreboot** option to **on** in the **fscan** configuration file.
- Define the **reboot** command in the fscan configuration file or on the **fscan** command line.
- Reset the system using the **reset** command with the **watchdog** switch.

When the system watchdog detects that a node is either in the *dead* or *debugger* state, it causes the system to reboot.

NOTE

When the Paragon system is booting, the **async** command is invoked for a short period of time to ensure the kernel was downloaded correctly and starts. The **async** command displays the following message when the kernel is downloaded correctly:

```
COFF header addr
```

The **async** command exits with a status of 2 and **fscan** is invoked.

Setting Up the reboot Command

For a Paragon system to automatically reboot, you must define the **reboot** command. You define the **reboot** command either in the *fscan.cfg* file or on the **fscan** command line. For example, the following line in an *fscan.cfg* file defines a **reboot** command:

```
define reboot "ksh /usr/paragon/boot/reset skip ignorelock \
autoreboot"
```

The **reboot** command must always be defined to use the **reset** command with the **autoreboot** argument.

FSCAN *(cont.)* FSCAN *(cont.)*

Defining the **reboot** command allows you to change how a system is rebooted. For example, the **reboot** command can be defined to execute **autoddb** before rebooting the system, for example:

```
define reboot "sh /usr/paragon/boot/autoddb; ksh
/usr/paragon/boot/reset skip ignorelock autoreboot"
```

If you have a problem with the return status when invoking scripts in the **reboot** command, use the Korn shell (**ksh**) to invoke commands in the script because other shells may cause problems.

Lock File

The **fscan** utility creates a lock file, */tmp/FSCAN.LOCK*, to make sure only one user at a time uses the utility. If the lock file exists when you execute the **fscan** utility, the **fscan** utility exits and displays the following message:

```
ERROR: FSCAN is locked by pid #XXXX
```

When **fscan** exits successfully, the lock file is removed. The lock file will remain if **fscan** is killed with the **kill** command or the system crashes. The lock file contains the PID for the **fscan** job. You can use this PID to kill the **fscan** job if **fscan** is not being used as a console.

Bad Nodes File

The bad nodes file identifies the nodes in the system that have failed. This is an ASCII file that has a single node entry per line. This file has the name *BADNODES.TXT* and is located in the */usr/paragon/boot* directory. This file is updated by the system watchdog as it finds bad nodes or potentially bad nodes. You can edit this file as needed with a standard system editor (for example, **vi**). Any node that has an entry in this file has it processor port disabled.

The *BADNODES.TXT* file has the following format:

```
cbs_number reason_for_removal
```

The following example *BADNODES.TXT* file contains two entries:

```
01A12     <watchdog> Node failed 3 times.
# The following bad node information added on 8/30/93.
01A11     removed to test some code.
```

FSCAN *(cont.)* FSCAN *(cont.)*

The example shows that node `01A12` caused the system watchdog to reboot the system three times. This node is listed as a bad node by the system watchdog. The information on node `01A11` was added by editing the file. The node was removed to perform tests. Comments are added by placing a # (pound sign) in the first column of a line in the file. Text in a comment line is ignored.

Files

The following files are on the diagnostic station:

/usr/local/bin/fscan	Contains the executable for the **fscan** utility.
/tmp/FSCAN.LOCK	Contains the lock that prevents multiple users from running the **fscan** utility.
/usr/paragon/boot/fscan.cfg	Contains **fscan** commands that specify the **fscan** configuration. This **fscan** utility looks for this file in the current directory or in the file specified by the **-F** switch.
/usr/paragon/boot/MAGIC.MASTER	Contains system booting information.
/usr/paragon/boot/BADNODES.TXT	Contains the list of bad nodes in the system.

The following file is on the Paragon system:

/sbin/watchdog	Contains the executable for the **watchdog** command.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **async, cbs, reset, scanio**

files: **BADNODES.TXT, MAGIC.MASTER**

FSCK FSCK

Provides file system consistency check and interactive repair

Syntax

fsck -p [**-m** *mode*] [**-P***s*[..*e*]] [**-R**]

fsck [**-b** *block#*] [**-c**] [**-y**] [**-n**] [**-o**] [**-m** *mode*]
[*file_system*] ...

Arguments

-b Use the block specified immediately after the flag as the super block for the file system. Block 4, 8, or 32 is usually the alternate super block.

-l Limit the number of parallel checks to the number specified in the following argument. By default, the limit is the number of disks, running one process per disk. If a smaller limit is given, the disks are checked round-robin, one file system at a time.

-m Use the mode specified in octal immediately after the flag as the permission bits to use when creating the lost+found directory rather than the default 1777. In particular, systems that do not want to have lost files accessible by all users on the system should use a more restrictive set of permissions such as 700.

-y Assume a yes response to all questions asked by **fsck**; this should be used with great caution as this is a free license to continue after essentially unlimited trouble has been encountered.

-n Assume a no response to all questions asked by **fsck** except for ''CONTINUE?'', which is assumed to be affirmative; do not open the file system for writing.

-o Check the file system, even if it has been marked clean.

-c If the file system is in the old (static table) format, convert it to the new (dynamic table) format. If the file system is in the new format, convert it to the old format provided the old format can support the file system configuration. In interactive mode, **fsck** will list the direction the conversion is to be made and ask whether the conversion should be done. If a negative answer is given, no further operations are done on the file system. In preen mode, the direction of the conversion is listed and done if possible without user interaction. Conversion in preen mode is best used when all the file systems are being converted at once. The format of a file system can be determined from the first line of output from **dumpfs**.

FSCK *(cont.)* FSCK *(cont.)*

-P*s*[..*e*]

Process the */etc/fstab* file starting with pass *s* and continuing through the optional pass *e*. If pass *e* is not specified, the end-of-file of the */etc/fstab* file ends the processing. This flag is not active when one or more *file_system* names are specified on the command line. The **fsck** command passes are organized in the */etc/fstab* file as follows:

Pass	**File system(s) type**
1	Root (/)
2	Required for single-user operation
3	Boot-node local file system
4 and above	Remote-node file system, NFS and PFS.

The **-P** flag is not valid if one or more *file_system* names are specified on the command line. This flag is only valid for the default list of file systems read from the */etc/fstab* file.

-R

Use the **fork()** call instead of the operating system's internal remote fork when file system is checking disks in parallel.

Description

The **fsck** command checks file systems in parallel if they are on different drives. For example, drives */dev/io0/rz0f* and */dev/io0/rz1f* can be checked in parallel.

The first form of **fsck** preens a standard set of file systems or the specified file systems. It is normally used in the run command script during automatic reboot. Here **fsck** reads the */etc/fstab* table to determine which file systems to check. Only partitions in **fstab** that are mounted ''rw'' or ''ro'' and that have non-zero pass number are checked. File systems with pass number 1 (normally just the root file system) are checked one at a time. When pass 1 completes, all remaining file systems are checked, running one process per disk drive. The disk drive containing each file system is inferred from the longest prefix of the device name that ends in a digit; the remaining characters are assumed to be the partition designator.

The system takes care that only a restricted class of innocuous inconsistencies can happen unless hardware or software failures intervene. These inconsistencies include unreferenced inodes, link counts in inodes that are too large, missing blocks in the free map, blocks in the free map that are also in files, and wrong counts in the super-block.

The preceding inconsistencies are the only ones that **fsck** with the **-P** argument corrects; if it encounters other inconsistencies, it exits with an abnormal return status and an automatic reboot will then fail. For each corrected inconsistency one or more lines are printed identifying the file system on which the correction takes place, and the nature of the correction. After successfully correcting a file system, **fsck** prints the number of files on that file system, the number of used and free blocks, and the percentage of fragmentation.

If sent a QUIT signal, **fsck** finishes the file system checks, then exits with an abnormal return status that causes an automatic reboot to fail. This is useful when you want to finish the file system checks during an automatic reboot but do not want the machine to come up multiuser after the checks complete.

Without the **-P** argument, **fsck** audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent, you are prompted for concurrence before each correction is attempted. Note that some of the corrective actions which are not correctable under the **-P** argument will result in some loss of data. The amount and severity of data lost can be determined from the diagnostic output. The default action for each consistency correction is to wait for you to respond **yes** or **no**. If you do not have write permission on the file system **fsck** defaults to a **no** action.

The **fsck** has more consistency checks than its predecessors **check, dcheck, fcheck,** and **icheck** combined.

If no file systems are given to **fsck**, then a default list of file systems is read from the file */etc/fstab*.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free map.
2. Blocks claimed by an inode outside the range of the file system.
3. Incorrect link counts.
4. Size checks: directory size not of proper format; partially truncated file.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks: file pointing to unallocated inode; inode number out of range; **.** (dot) or **..** (dot dot) not the first two entries of a directory or having the wrong inode number.
8. Super Block checks: more blocks for inodes than there are in the file system.
9. Bad free block map format.

FSCK *(cont.)* FSCK *(cont.)*

10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with your concurrence, reconnected by placing them in the *lost+found* directory. The name assigned is the inode number. If the *lost+found* directory does not exist, it is created. If there is insufficient space its size is increased.

Because of inconsistencies between the block device and the buffer cache, the raw device should always be used.

Examples

1. The following command line starts the **fsck** command processing the */etc/fstab* file starting with pass 2 and continuing until the end-of-file.

   ```
   # fsck -P2
   ```

2. The following command line starts the **fsck** command processing the */etc/fstab* file starting with pass 2 and ending with pass 2.

   ```
   # fsck -P2..2
   ```

3. The following command line starts the **fsck** command processing the */etc/fstab* file starting with pass 2 and ending with pass 4.

   ```
   # fsck -P2..4
   ```

Files

/usr/sbin/fsck	Specifies the command path
/etc/fstab	Contains default list of file systems to check

Limitations And Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

FSCK *(cont.)* FSCK *(cont.)*

See Also

Commands: **fsdb(8)**, **newfs(8)**, **reboot(8)**

Files: **fs(5)**, **fstab(5)**

FSPLIT FSPLIT

Splits one file containing several Fortran program units into several files containing one program unit each.

Syntax

fsplit [*filename*]

Arguments

filename	Identifies the file that you want to split into individual files. If you omit this argument, **fsplit** reads standard input.

Description

Use **fsplit** to split a single file containing several Fortran program units (program, subroutine, function, or block data) into several files, each of which contains only one program unit. The first non-comment line of each program unit (usually a **program**, **subroutine**, **function**, or **block data** statement) marks the beginning of each kind of program unit; the **end** statement marks the end.

The original file remains unchanged; the new files are named as follows:

- Each named program unit (one that specifies a name in its **program**, **subroutine**, **function**, or **block data** statement) is put in a file named *name.f*, where *name* is the name of the program unit.
- An unnamed block data subprogram is put in a file named *blockdataXXX.f*, where *XXX* is a number that corresponds to the order of the unnamed block data subprogram in the original file.
- A main program that does not contain a **program** statement is put in a file named *mainXXX.f*, where *XXX* is a number that corresponds to the order of the unnamed main program in the original file.
- A file with the same name as a program unit that already exists in the current directory is put in a file named *zzzXXX.f*, where *XXX* is a number that corresponds to the order of the duplicate program unit in the original file.

WARNING

If your file defines multiple program units with the same name, the resulting *name.f* file will contain only the first program unit defined.

FSPLIT *(cont.)*   FSPLIT *(cont.)*

Examples

Consider the following source file (named *file.f*):

```
      program a
c  This is the first main program named a.
      end

      program a
c  This is the second main program named a.
      end

      integer a-m
c  This is the first unnamed main program unit.
      end

c  This is the second unnamed main program unit.
      integer n-z
      end

      subroutine b
c  This is the first subroutine named b.
      end

      subroutine b
c  This is the second subroutine named b.
      end

c  This is the function named c.
      function c
      end

      block data d
c  This is the block data program unit named d.
      end

      block data
c  This is the first unnamed block data program unit.
      end

      block data
c  This is the second unnamed block data program unit.
      end
```

FSPLIT *(cont.)* FSPLIT *(cont.)*

Before using **fsplit**, the directory contains the following:

```
% ls
file.f
```

After using **fsplit**, the directory contains the following:

```
% fsplit file.f
a.f
a.f already exists, put in zzz000.f
main000.f
main001.f
b.f
b.f already exists, put in zzz001.f
c.f
d.f
blkdta000.f
blkdta001.f
zzz002.f
% ls
a.f blkdta000.f c.f file.f main001.f zzz001.f
b.f blkdta001.f d.f main000.f zzz000.f zzz002.f
```

Examining the various files reveals the following:

```
% cat blockdata000.f
      block data
c  This is the first unnamed block data program unit.
      end
% cat blockdata001.f
      block data
c  This is the second unnamed block data program unit.
      end
% cat main000.f
      integer n-m
      end
% cat main001.f
      integer n-z
      end
% cat a.f
      program a
c  This is the first main program named a.
      end
% cat b.f
      subroutine b
```

FSPLIT *(cont.)* FSPLIT *(cont.)*

```
c  This is the first subroutine named b.
      end
% cat c.f
      function c
      end
% cat d.f
      block data d
c  This is the block data program unit named d.
      end
% cat zzz000.f
      program a
c  This is the second main program named a.
      end
% cat zzz001.f
      subroutine b
c  This is the second subroutine named b.
      end
```

The files *a.f*, *b.f*, and *main000.f* contain the first program units defined in the original source file. The file *main000.f* contains the second unnamed program unit. The file *zzz000.f* contains the first duplicate unit and the file *zzz001.f* contains the second duplicate unit. This example highlights some of the problems in using the **fsplit** command.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

GETMAGIC GETMAGIC

Displays the bootmagic strings.

Syntax

getmagic [**-m** *file*] [**-w**] [*name* ...]

Arguments

-m file	Specifies the bootmagic master file to use for listing bootmagic strings. By default the command displays the bootmagic strings that are resident in memory.
-w	Expands the node numbers for bootmagic strings that contain lists of node numbers. Spaces are added to replace commas in the node list specification. Lists that use the dot notation (..) to list consecutive nodes are expanded to display the complete list nodes. Spaces separate the node numbers. See the "Examples" section for an example.
name	Specifies a bootmagic string. If the bootmagic string does not exist, the command displays a blank line. See the **bootmagic** manual page for a list of the bootmagic strings.

Description

Use the **getmagic** command to display the values of the bootmagic strings that are loaded on a Paragon system when the system is booted. This command does not list all the bootmagic variables a system supports. It only lists the bootmagic strings that are loaded on the system when the system is booted. These bootmagic strings are downloaded to the system when the system is reset.

Examples

The following example displays a specific bootmagic string.

```
# getmagic BOOT_NODE_LIST
0..5,8..9
```

The following example uses the **-w** switch to display the value for a specific bootmagic string. The values are expanded with white spaces.

```
# getmagic -w BOOT_NODE_LIST
0 1 2 3 4 5 8 9
```

GETMAGIC *(cont.)* GETMAGIC *(cont.)*

The following example lists of all the bootmagic strings loaded on a system:

```
# getmagic
BOOT_FIRST_NODE=3
BOOT_CONSOLE=f
BOOT_STARTUP_NAME=/mach_servers/startup
BOOT_EMULATOR_NAME=/mach_servers/emulator
BOOT_KERNEL_NAME=/mach_servers/mach_kernel.db
BOOT_ROOT_DEV=rz0a
BOOT_HOWTO=0x0
BOOT_COMPUTE_STARTUP_NAME=/mach_servers/startup.compute
BOOT_COMPUTE_KERNEL_NAME=/mach_servers/mach_kernel.compute.db
BOOT_ALT_KERNEL_NAME=/mach_servers/sunmos
BOOT_ARCH=paragon
BOOT_MY_NODE=3
ROOT_FS_NODE=3
ROOT_DEVICE_NODE=3
NETSERVER=3
ALLOCATOR_NODE=3
BOOT_NUM_NODES=8
BOOT_MESH_X=4
BOOT_MESH_Y=4
BOOT_IO_NODE_LIST=0,3
BOOT_COMPUTE_NODE_LIST=1..2,4..5,8..9
BOOT_NODE_LIST=0..5,8..9
EXPORT_PAGING=3
PAGER_NODE=<0..2,4..9>3
BOOT_TIME=758584809
BOOT_REMOTE_KERNEL=137.46.14.189:/usr/paragon/boot/mach_kernel.db
DEFER_REFRESH=startup
BOOT_GREEN_LED=Dci
BOOT_RED_LED=Dcgl
DISABLE_BOOTMESH=0
BOOT_LOAD_SYMBOLS=1
BOOT_DISK_NODE_LIST=0,3
BOOT_DAT_NODE_LIST=3
BOOT_ENET_NODE_LIST=0,3
```

GETMAGIC *(cont.)* GETMAGIC *(cont.)*

Files

/sbin/getmagic	Specifies the command path.

Errors

```
getmagic: Error: cannot get bootmagic from kernel.
```

The bootmagic file must be readable by the user and the **getmagic** command must be executable by all users.

```
bootmagic file: No such file or directory
getmagic: Error: cannot open bootmagic file
```

The bootmagic file does not exists.

```
bootmagic file: Permission denied
getmagic: Error: cannot open bootmagic file
```

You do not have read permission on the bootmagic file.

```
bootmagic file: Error 0
getmagic: Error: cannot read bootmagic file
```

The bootmagic file is empty or unreadable.

See Also

bootmagic, bootmesh, bootpp, parsemagic, reset

GPROF GPROF

Displays a call-graph execution profile for an application.

Syntax

gprof [**-absz**] [**-e** *routine* | **-f** *routine*]...
[**-E** *routine* | **-F** *routine*]...
[*object_file* [*profile_file*]...]

Arguments

-a	Do not print statically-declared functions. If the **-a** switch is given, all relevant information about the static function such as time samples, calls to other functions, and calls from other functions belongs to the function loaded immediately before the static function in the *a.out* file.
-b	Do not print a description of each field in the profile.
-e *routine*	Do not print the graph profile entry for *routine* and all its descendants (unless they have other ancestors that are not suppressed). More than one **-e** switch may be given. Only one *routine* may be given with each **-e** switch.
-E *routine*	Do not print the graph profile entry for *routine* and its descendants, (same as the **-e** switch) and also exclude the time spent in *routine* and its descendants from the total and percentage time computations. More than one **-E** switch may be given. Only one *routine* may be given with each **-E** switch.
-f *routine*	Print the graph profile entry only for *routine* and its descendants. More than one **-f** switch can be given. Only one *routine* may be given with each **-f** switch. The **-f** switch overrides the **-e** switch.
-F *routine*	Print the graph profile entry only for *routine* and its descendants (same as **-f**) and also use only the times of the printed routines in total time and percentage computations. More than one **-F** switch may be given. Only one *routine* may be given with each **-F** switch. The **-F** switch overrides the **-E** switch.
-s	Produce a profile file, *gmon.sum*,that represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of the **gprof** command (possibly also with the **-s** switch) to accumulate profile data across several runs of an application.
-z	Display routines that have zero usage, as indicated by call counts and accumulated time.

GPROF *(cont.)* GPROF *(cont.)*

object_file Specify the name of the executable file used by the **gprof** command to extract symbol table information. The object file should match the executable used to produce the profile file being analyzed. The default is *a.out.*

profile_file Specify the name of a directory containing profile files generated by a parallel application run or the name of a single profile file. The default is *gmon.out.* The **gprof** command checks whether *profile_file* specifies a directory or a file. If a directory is specified, the **gprof command** expects to find an *INFO* file in that directory. The *INFO* file contains information about the application that generated the profile files. The *INFO* file has the following format:

```
Controlling process: executable_name pid_value

pid          node        ptype       Executable
xxxxxxx      xxxxx       xxxx        full_path
xxxxxxx      xxxxx       xxxx        full_path
                 .
                 .
                 .
```

The **gprof** command expects the directory to contain one profile file for every process listed in the *INFO* file. The individual data files are named *executable_name.pid.node.ptype*, where *pid* is the process id, *ptype* is the process type and *node* is the node number as given in the *INFO* file. By default, the **gprof** command chooses the lowest *node:ptype* pair data file for the specified *object_file* as the *profile_file.* To view **gprof** output on other *node:ptype* pairs, the specific *executable_name.pid.node.ptype* data file must be specified as a *profile_file.*

More than one *profile_file* may be given, but only the first *profile_file* specified is assumed to be a directory containing multiple files. For example, to produce a summary profile of all the data files for the application binary *tst*, you would use the following command:

```
% gprof -s tst gmon.out/tst*
```

To view the summary, enter the following:

```
% gprof tst gmon.sum
```

GPROF *(cont.)* GPROF *(cont.)*

NOTE

Any *routine* names specified with the **-e**, **-f**, **-E**, and **-F** switches must be valid COFF symbols without the leading underscore generated by the compiler. For C symbols, the name of the function is valid. For Fortran symbols, it is necessary to add an underscore to the end of the routine name, since the Paragon Fortran compiler adds a trailing underscore for Fortran routines.

Description

Use the **gprof** command to produce a call-graph execution profile of applications processed with the Interactive Parallel Debugger (IPD) **instrument -gprof** command. A program creates profile files if it has been loaded under IPD and processed with the **instrument** command. Only programs that execute the *write-location* point of the **instrument** command will cause profile files to be written. For a complete description of the IPD **instrument** command, refer to the *Paragon™ System Interactive Parallel Debugger Reference Manual.*

The symbol table in the executable *object_file* is read and correlated with a *profile_file,* and the **gprof** command produces a flat profile, a call graph profile, and a cycle listing. The flat profile is similar to that provided by the **prof** command. The call graph profile provides total execution times and call counts for each function in the program, sorted by decreasing time.

The **gprof** command displays the call graph profile as follows:

- An application's call graph and any flow cycles within the call graph.
- Execution times on the edges of the call graph.
- Function names within the application. The names are sorted according to the amount of time they were used.
- A list of the children of each function with the propagation times for each function.
- Above each function entry, the amount of time for the function and when its descendants are propagated to the function's parents.

The cycle listing displays the cycles within the call graph and the members within the cycle. This includes the amount of time and call counts of each cycle.

GPROF *(cont.)* GPROF *(cont.)*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

There is a possibility of quantization errors with the **gprof** command. The granularity of the sampling is shown in the gprof output, but the granularity is statistical at best. The time for each execution of a function can be expressed by the total time for the function, divided by the number of times the function is called. This means that the time propagated along a call graph arc to parents of that function is directly proportional to the number of times that arc is traversed.

Parent functions that are not themselves profiled have the time of their profiled children propagated to them, but they do not have their own time propagated further. The call graph listing makes it appear as if they are spontaneously invoked.

Similarly, signal handlers, even though profiled, appear to be spontaneously invoked. Any profiled children of signal handlers have their times propagated accurately, unless the signal handler is invoked during the execution of a profiling routine. In this case, propagation cannot occur.

See Also

prof

Paragon™ System Interactive Parallel Debugger Reference Manual

GPS

Displays the current process status for all the nodes on a system.

Syntax

gps [-adejflm] [-oO *specifier*][=*header*],... **[-g** *glist*]
[-p *plist*] **[-s** *slist*] **[-t** *tlist*] **[-u** *ulist*]

BSD Compatible Syntax:

gps [aAeghkjlLmsSTuvwx] [oO *specifier*][=*header*],...
[t *tty*] [*process_number*]

The **gps** command displays the current process status. It is similar to the **ps** command except it can be used to display processes running on all the nodes.

Arguments

The following arguments can be used with **gps**:

-a	Prints information to standard output about all processes, except the process group leaders and processes not associated with a terminal.
-d	Prints information to standard output about all processes, except the process group leaders.
-e	Prints information to standard output about all processes, except kernel processes.
-f	Generates a full listing.
-g *glist*	Prints only information about processes that are in the process groups listed in *glist*. The *glist* is a list of process-group identifiers enclosed in "" (double quotes) and separated from one another by a comma or one or more spaces (or tabs), or both. Because of the way the shell treats spaces and tabs, you need to quote space-separated lists.
-j	Produces job control information, with fields specified for **user**, **pid**, **ppid**, **pgid**, **sess**, **jobc**, **state**, **tname**, **cputime** and **comm**.
-l	Generates a long listing.
-m	Prints all threads in a task, if the task has more than one.

GPS *(cont.)* GPS *(cont.)*

-o *specifier*[*=header*],...
: Specifies a list of format specifiers to describe the output format.

-O *specifier*[*=header*],...
: Same as **o**, except it displays the fields specified by **pid**, **state**, **tname**, **cputime**, and **comm** in addition to the specifiers supplied on the command line.

-p *plist*
: Displays only information about processes with the process numbers specified in *plist.* The *plist* argument is either a list of process ID numbers or a list of process ID numbers enclosed in "" (double quotes) and separated from one another by a comma or one or more spaces (or tabs), or both. Because of the way the shell treats spaces and tabs, you need to quote space-separated lists.

-s *slist*
: Displays information about processes belonging to the sessions specified in *slist.* The *slist* argument is either a list of session ID numbers or a list of session ID numbers enclosed in "" (double quotes) and separated from one another by a comma or one or more spaces (or tabs), or both. Because of the way the shell treats spaces and tabs, you need to quote space-separated lists.

-t *tlist*
: Displays only information about processes associated with the terminals listed in *tlist.* The *tlist* argument is either a list of terminal identifiers or a list of terminal identifiers enclosed in "" (double quotes) and separated from one another by a comma or one or more spaces, or both. Because of the way the shell treats spaces and tabs, you need to quote space-separated lists.

-u *ulist*
: Displays only information about processes with the user ID numbers or login names specified in *ulist.* The *ulist* argument is either a list of user IDs or a list of user IDs enclosed in "" (double quotes) and separated from one another by a comma or one or more spaces, or both. Because of the way the shell treats spaces and tabs, you need to quote space-separated lists.

: In the listing, **ps** displays the numerical user ID unless the **-f** flag is used; then it displays the login name.

The following BSD compatible flags can be used with **ps** (note that these flags are not prefixed with a - (dash) character):

a
: Asks for information regarding processes associated with terminals (ordinarily only one's own processes are displayed).

A
: Increases the argument space.

e
: Asks for the environment to be printed, as well as the arguments to the command.

GPS *(cont.)* GPS *(cont.)*

g Asks for all processes. Without this flag, **ps** only prints *interesting* processes. Processes are deemed to be uninteresting if they are process group leaders. This normally eliminates top-level command interpreters and processes waiting for users to log in on free terminals.

h Repeats the header after each screenful of information.

j Produces job control information, with fields specified by **user**, **ppid**, **pgid**, **sess**, and **jobc**.

l Asks for a detailed list, with fields specified by **ppid**, **cp**, **pri**, **nice**, **vsize**, **rssize** and **wchan**.

L Lists all available format specifiers.

m Prints all threads in a task, if the task has more than one.

o *specifier*[=*header*],...
Specifies a list of format specifiers to describe the output format.

O *specifier*[=*header*],...
Same as **o**, except it displays the fields specified by **pid**, **state**, **tname**, **cputime**, and **comm** in addition to the specifiers supplied on the command line.

s Gives signal states of the processes, with fields specified by **uid**, **cursig**, **sig**, **sigmask**, **sigignore**, and **sigcatch**.

S Prints usage summaries (total usage of a command, as opposed to current usage).

t*tty* Lists only processes for the specified tty.

T Lists all processes on your tty.

u Produces a user oriented output. This includes fields specified by **user**, **pcpu**, **pmem**, **vsize**, **rssize**, and **start**.

v Produces a version of the output containing virtual memory statistics. This includes fields specified by **cputime**, **sl**, **pagein**, **vsize**, **rssize**, **pcpu**, and **pmem**.

w Uses a wide output format (132 columns (bytes) rather than 80); if this flag is doubled (**ww**), uses an arbitrarily wide output. This information determines how much of long commands to print.

x Asks even about processes with no terminal.

GPS *(cont.)* GPS *(cont.)*

process_number	Restricts output to the specified process. This argument must be entered last on the command line.

Description

This command can only be used by super user.

The **ps** command shows only processes running in the service partition, but the **gps** command can be used to display processes running on all the nodes in the system.

Output formats for each process include the process ID (**pid**), control terminal of the process (**tname**), CPU time used by the process (**cputime**) (this includes both user and system time), the state of the process (**state**), and an indication of the command that is running (**comm**).

The state is given by a sequence of letters, for example, **RWN**. The first letter indicates the status of the process:

R	Runnable process.
U	Uninterruptible sleeping process.
S	Process sleeping for less than about 20 seconds.
I	Idle (sleeping longer than about 20 seconds) process.
T	Stopped process.
H	Halted process.

Additional characters after these, if any, indicate additional state information:

W	Process is swapped out (shows a blank space if the process is loaded (in-core)).
>	Process has specified a soft limit on memory requirements and is exceeding that limit; such a process is (necessarily) not swapped.

An additional letter may indicate whether a process is running with altered CPU scheduling priority (**nice**):

N	Process priority is reduced.
<	Process priority has been artificially raised.
+	Process is a process group leader with a controlling tty.

Format Specifiers

The following list contains all format specifiers that can be used with **ps**:

Specifier	**Header**	**Meaning**
comm	COMMAND	Command arguments (and environment with BSD e flag)
ucomm	COMMAND	Command name for accounting
logname	LOGNAME	User's login name
flag	F	Process flags
status	STATUS	Process status
uid	UID	Process user ID (effective UID)
ruid	RUID	Process user ID (real UID)
svuid	SVUID	Saved process group ID
rgid	RGID	Process group (real GID)
svgid	SVGID	Saved process group ID
pid	PID	Process ID
ppid	PPID	Parent process ID
cp	CP	Short-term CPU utilization factor (used in scheduling)
wchan	WCHAN	Address of event on which a process is waiting (an address in the system). A symbol is chosen that classifies the address, if available, from the system; otherwise, it is printed numerically.
nwchan	WCHAN	In this case, the initial part of the address is trimmed off and is printed hexadecimally, for example, 0x80004000 prints as 4000.
cursig	CURSIG	Current signal
sig	PENDING	Signals pending to this process
sigmask	BLOCKED	Current signal mask
sigignore	IGNORED	Signals being ignored

Specifier	**Header**	**Meaning**
sigcatch	CAUGHT	Signals being caught
user	USER	Username
ruser	RUSER	User ID
pgid	PGID	Process group ID
jobc	JOBC	Current count of processes qualifying PGID for job control
sess	SESS	Session ID
tdev	TDEV	Major/minor device for controlling tty
tname	TT	Controlling tty device name
longtname	TT	Long controlling tty device name
tpgid	TPGID	Foreground process group associated with tty
tsession	TSESS	Session associated with tty
state	STAT	Symbolic process status
pri	PRI	Process priority (nonpositive when in non-interruptible wait)
usrpri	UPR	Base scheduling priority
nice	NI	Process scheduling increment (see the setpriority() call).
vsize	VSZ	Process virtual address size
rssize	RSS	Real memory (resident set) size of the process (in 1024 byte units)
u_procp	UPROCP	Address of process in user area
umask	UMASK	Process umask
acflag	ACFLG	Process accounting flag
start	STARTED	Start time of process. If start time was more than 24 hours ago, gives the date.
lstart	STARTED	Start time and date of process
cputime	TIME	Current CPU time used
usertime	USER	Time spent in user space
systime	SYSTEM	Time spent in system
pcpu	%CPU	Percent CPU usage. This is a decaying average of up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young), it is possible for the sum of all %CPU fields to exceed 100%.
pmem	%MEM	Percent real memory usage
sl	SL	Sleep time
pagein	PAGEIN	Number of disk I/Os resulting from references by the process to pages not loaded in core.

GPS *(cont.)* GPS *(cont.)*

Specifier	Header	Meaning
minflt	MINFLT	Page reclaims
majflt	MAJFLT	Page faults
nswap	NSWAP	Swaps
inblock	INBLK	Block input operations
oublock	OUBLK	Block output operations
msgsnd	MSGSND	Messages sent
msgrcv	MSGRCV	Messages received
nsignals	NSIGS	Signals received
nvcsw	VCSW	Voluntary context switches
nivcsw	IVCSW	Involuntary context switches
scount	SCNT	Suspend count

Compound format specifiers are made up of groups of individual format specifiers, as follows:

Specifier	Meaning
RUSAGE	minflt, majflt, nswap, inblock, oublock, msgsnd, msgrcv, nsigs, nvcsw, nivcsw
THREAD	user, pcpu, pri, scnt, wchan, usertime, sys-time
DFMT (default printing format)	pid, tname, state, cputime, comm
LFMT (BSD l format)	uid, pid, ppid, cp, pri, nice, vsz, rss, wchan, state, tname, cputime, comm
JFMT (j format)	user, pid, ppid, pgid, sess, jobc, state, tname, cputime, comm
SFMT (BSD s format)	uid, pid, cursig, sig, sigmask, sigignore, sigcatch, stat, tname, comm
VFMT (BSD v format)	pid, tt, state, cpu-time, sl, pagein, vsz, rss, pcpu, pmem, comm

Specifier	**Meaning**
UFMT (BSD u format)	uname, pid, pcpu, pmem, vsz, rss, tt, state, start, cputime, comm
F5FMT (f format)	uname, pid, ppid, pcpu, start, tt, cpu-time, comm
L5FMT (l format)	flag, state, uid, pid, ppid, pcpu, pri, nice, rss, wchan, tt, cpu-time, ucomm
FL5FMT (lf format)	flag, state, uid, pid, ppid, pcpu, pri, nice, rss, wchan, start, cputime, comm
process_flags	Flags associated with process as in <sys/proc.h>

The flags associated with process as in <sys/proc.h> (see process_flags in the preceding table) are as follows:

Symbolic Constant	**Flag Value (Hex)**	**Meaning**
SLOAD	0x00000001	In core
SSYS	0x00000002	Swapper or pager process
STRC	0x00000010	Process is being traced
SWTED	0x00000020	Another tracing flag
SOMASK	0x00000200	Restore old mask after taking signal
SWEXIT	0x00000400	Working on exiting
SPHYSIO	0x00000800	Doing physical I/O
SPAGV	0x00008000	Init data space on demand, from vnode
SSEQL	0x00010000	User warned of sequential vm behavior
SUANOM	0x00020000	User warned of random vm behavior
STIMO	0x00040000	Timing out during sleep
SOUSIG	0x00100000	Using old signal mechanism
SOWEUPC	0x00200000	Owe process an addupc() call
SCTTY	0x00800000	Has a controlling terminal
SXONLY	0x02000000	Process image read-protected
SNOCLDSTOP	0x40000000	No SIGCHLD when children stop
SEXEC	0x80000000	Process called exec

<defunct>	A process that has exited but whose parent process has not waited for it is marked **<defunct>**.
<exiting>	A process that is blocked trying to exit is marked **<exiting>**.

The **ps** program examines memory to get the filename and arguments given when the process was created. The method is inherently somewhat unreliable because a process can destroy this information, so the names cannot be counted on too much.

Examples

This first example shows how to list all processes running in the system. Displaying this information may take some time if the system is large.

```
gps -deaf
```

This example displays processes running in a subpartition by specifying the group ID:

```
gps -g PGID
```

The PGID variable is the process group ID of the application which you can get using the **pspart** command output under the PGID column. Use the **pspart** command as follows to list the processes running in the compute partition and to find the PGID value:

```
pspart .compute
```

```
   PGID  USER       SIZE PRI START      TIME ACTIVE   TOTAL TIME COMMAND
 197281  root          4   5 14:37:36   0:24.59 100%     0:24.59 test -sz 2
```

Enter the following:

```
gps -g 197281
```

This may display the following:

```
   PID TT  STAT      TIME COMMAND
 65548 p0  RW +   1:59.41 test -sz 2
197281 p0  SW +   0:00.04 test -sz 2
327692 p0  RW +   1:59.38 test -sz 2
```

GPS *(cont.)*

This final example displays a long listing of processes running in the compute partition. By using the above **pspart** command output, you may enter:

```
gps -lg 197281
```

This may display the following:

```
F STAT   UID    PID   PPID %CPU PRI  NI  RSS WCHAN  TT       TIME COMMAN
  808001 RW +     0  65548 197281 99.0 -13   0 1.62 *      p0    2:22.37
testin
  808001 SW +     0 197281 197280  0.0 -13   0 496K *      p0    0:00.04
testin
  808001 RW +     0 327692 197281 99.0 -13   0 1.62 *      p0    2:22.35
testin
```

Files

/sbin/gps	Specifies the command path.

See Also

OSF/1 Command Reference: **ps(1)**

commands: **pspart**, **lspart**, **showpart**

calls: **nx_pspart()**

HOSTINFO HOSTINFO

Displays information about a Paragon system node that is running the Mach kernel.

Syntax

hostinfo [*node*]

Arguments

node	Root partition node number that is running the Mach kernel. The default is the node on which the **hostinfo** command is currently is running.

Description

The **hostinfo** command displays the following information about a node that is running the Mach kernel:

- The root partition node number of the node.
- The Mach kernel version number. This is not the version number of the Paragon operating system.
- The maximum number of processors the Mach kernel was configured for.
- The actual number of processors physically available in the hardware configuration.
- The amount of physical memory available on the node.
- The node's processor type.

HOSTINFO *(cont.)* HOSTINFO *(cont.)*

Examples

The following example shows a four node service partition.

```
% showpart .service
 USER GROUP ACCESS SIZE FREE RQ EPL
 root daemon 754      4    4  -  -
  +---------+
 0| . . * * |
 4| . . * * |
 8| . . . . |
12| . . . . |
  +---------+
```

You can use the **hostinfo** command to get kernel information as follows:

```
% /sbin/hostinfo
node 7, Mach kernel version 3.0.
Kernel configured for a single processor only.
1 processor is physically available.
Primary memory available: 16.00 megabytes.
CPU 0: i860XP ( Paragon XP/S ) UP
```

This shows kernel information about the node 7 which is the node that the **hostinfo** command is running on. This node has 16M bytes of memory available and is currently up and running.

You can use the **hostinfo** command to find out information about a specific host node, as follows:

```
% /sbin/hostinfo 3
node 3, Mach kernel version 3.0.
Kernel configured for a single processor only.
1 processor is physically available.
Primary memory available: 32.00 megabytes.
CPU 0: i860XP ( Paragon XP/S ) UP
```

This shows that node 3 has 32M bytes of memory available and is currently up and running.

Files

/sbin/hostinfo	Specifies the command path.

HOSTINFO *(cont.)* HOSTINFO *(cont.)*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

HSTAT HSTAT

Displays HiPPI device driver statistics.

Syntax

hstat [**-b** *node*] [**-a**] [**-v**]

Arguments

-b *node*	Number of the node board whose statistics you want to display.
-a	Show all statistics (rhippi, ifhip, hctlr).
-v	Verbose mode.

Description

The **hstat** command returns statistics for the HiPPI controller interface, raw driver interface, and network driver interface.

HiPPI Controller Interface Statistics

The **hstat** command returns the following SRC (source channel) statistics:

packets	packets sent
Kbytes	total Kbytes sent
intr	SRC channel interrupts
960busy	controller too busy for new SRC PD
qfull	PD queue is full
qnull	PD queue NULL head ptr
qhits	something to do in PD queue
qmiss	nothing to do in PD queue
chains	requests sent to i960 in chains
maxchain	largest chain sent to i960
ovrrun	SRC fifo full and no DST readys
align	not even number of HiPPI words
shtwrd	last word short, not HiPPI size
qflush	flushed queue in re-init cycle
errors	SRC channel errors
no_intc	interconnect deasserted
seqs	source sequence error
seqd	destination sequence error
timeout	timeout waiting for ready
parity	parity error

HSTAT *(cont.)* HSTAT *(cont.)*

The **hstat** command returns the following DST (destination channel) statistics:

packets	packets received
Kbytes	total Kbytes received
intr	DST channel interrupts
960busy	controller to busy for new DST PD
qfull	PD queue is full
qnull	PD queue NULL head ptr
qhits	something to do in PD queue
qmiss	nothing to do in PD queue
undrun	DST fifo empty and no bursts
qflush	flushed queue in re-init cycle
errors	DST channel errors
no_intc	interconnect deasserted
seqs	source sequence error
seqd	destination sequence error
towb	timeout waiting for burst
parity	parity error
llrc	LLRC error on burst of data
exbf	DST data exceeded buffer resource
arbt	aborted destination connection
algn	buffer addr and len was unaligned
cerr	CONT error, connection aborted
cabrt	CONT host aborted connection

HSTAT *(cont.)* HSTAT *(cont.)*

The **hstat** command returns the following OTHER statistics:

src_intr	SRC interrconnect asserted
dst_intr	DST interrconnect asserted
timeouts	watchdog, no controller response
init	controller init passes
resets	controller reset count
dmsg_intr	controller debug msg interrupts
ctlr_intr	controller type interrupts
filters	filters on DST filter list

HiPPI Raw Driver Statistics

The **hstat** command returns the following WRITES statistics:

packets	packets sent
bytes	total Kbytes sent
chains	requests sent to i960 in chains
errors	Write errors on SRC channel
qfull	PD queue is full
qhit	something to do in PD queue
qmiss	nothing to do in PD queue
reqfail	hctlr driver request failed

HSTAT *(cont.)* HSTAT *(cont.)*

The **hstat** command returns the following READS statistics:

packets	packets received
bytes	total Kbytes received
errors	READ errors on DST channel
Aborted	total READ requests aborted
timeouts	reason: timed out
deadport	reason: deadport (user app)
noFilters	reason: noFilter was set
unsetFilter	reason: Filter was being unset
ctlrAbort	reason: ctlrAborted connection

The **hstat** command returns the following OTHER statistics:

filter_qfull	hit max allowed filter setting
dn_errs	deadname notification from proxy port
no_rd_reqs	packet arrived but no request posted

HiPPI Network Driver Statistics

The **hstat** command returns the following OUTPUT statistics:

packets	packets sent
bytes	total Kbytes sent
intr	SRC channel (HiPPI-LE) interrupts
errors	Input errors on SRC channel
large pkts	total pkts, size greater than 60000 bytes
medium pkts	total pkts, size = 10000-60000 bytes
small pkts	total pkts, size = less than 10000 bytes
req_fail	hctlr driver request failed

The **hstat** command returns the following INPUT statistics:

packets	packets received
bytes	total Kbytes received
intr	DST channel (HiPPI-LE) interrupts
errors	Input errors on DST channel
large pkts	total pkts, size greater than 60000 bytes
medium pkts	total pkts, size = 10000-60000 bytes
small pkts	total pkts, size = less than 10000 bytes
no_smbufs	no small 1KB buffer for DST, pkt dropped
no_lgbufs	no large 64KB buffer for DST, pkt dropped

HSTAT *(cont.)* HSTAT *(cont.)*

Examples

```
# ./hstat -b5 -a

HiPPI Interface stats from Node (5):
-------------------------------------
SRC: packets= 3197   Kbytes = 295      errors = 2
     intr    = 3199  960busy= 0        chains = 0     qflush = 0
     qfull   = 0     qempty = 3199     qhits  = 1     qmiss  = 3198
     no_intc= 0      seqs   = 0        seqd   = 0     parity = 0
     timeout= 2      align  = 0        shtwrd = 0     ovrrun = 0
     maxchain=0

DST: packets= 6316   Kbytes = 1601036  errors = 0
     intr    = 9443  960busy= 5106     qflush = 0
     qfull   = 0     qempty = 0        qhits  = 6325  qmiss  = 1
     no_intc= 0      parity = 0        llrc   = 0     seqs   = 0
     seqd    = 0     undrun = 0        towb   = 0     exbf   = 0
     abrt    = 0     algn   = 0        cabrt  = 0     cerr   = 0

OTHER: timeout=0 dmsg=0 ctlr_intr=1 src_intr=1 dst_intr=1
       reset=1 init=1 filters=1

HiPPI RAW Driver (rhippi) stats from Node (5):
----------------------------------------------
WRITE: packets=0 bytes=0 KB chains=0 errors=0
       qfull=0 qhigh=0 qhit=0 qmiss=0 reqfail=0

READ:  packets=0 bytes=0 KB errors=0
       Aborted(0): timeouts=0  deadport=0
                   noFilters=0 unsetFilter=0 ctlrAbort=0

OTHER:  filter_qfull=0 qhigh=0 dn_errs=0 no_rd_reqs=0

HiPPI Network Driver (ifhip) stats from Node (5):
-------------------------------------------------
OUTPUT: packets=44 bytes=5 KB intr=0 errors=2
        large pkts=0 medium pkts=0 small pkts=44
        qhit=1 qmiss=45 req_fail=0

INPUT:  packets=36 bytes=4 KB intr=0 errors=0
        large pkts=0 medium pkts=0 small pkts=36
        no_smbufs=0 no_lgbufs=0
```

HSTAT *(cont.)* HSTAT *(cont.)*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

IFCONFIG IFCONFIG

Configures or displays network interface parameters.

Syntax

ifconfig *interface_id* [*address_family*] [*address* [*dest_address*]] [*parameters*]

Description

The **ifconfig** command assigns and displays an address to a network interface, and configures network interface parameters.

You use the **ifconfig** command at boot time to define the network address of each interface. You can also use the **ifconfig** command at other times to redefine the address of an interface or to set other operating parameters.

Any user can query the status of a network interface; only the superuser can modify the configuration network interfaces.

You specify an interface with the **ifconfig** *interface_id* syntax. The *interface_id* may be a string of the form *nameunit* (for example, **em0**) or *<node>nameunit* (for example, '**<11>em0**'). The *node* argument is the node number where the network controller is installed (using the root node numbering scheme).

If you do not specify an address or optional parameters, the **ifconfig** program displays the current configuration for the specified network interface only.

If a protocol family is specified by the *address_family* parameter, **ifconfig** reports only the configuration details specific to that protocol family.

When changing an interface configuration, an address family, which may alter the interpretation of succeeding parameters, must be specified. This family is required because an interface can receive transmissions in different protocols, each of which may require a separate naming scheme.

For the **inet** family, the *address_family* parameter is either a hostname or an Internet address in the standard dotted-decimal notation.

For the Xerox Network Systems family, addresses are *net:a.b.c.d.e.f*, where *net* is the assigned network number (in decimal), and each of the 6 bytes of the host number, *a* to *f*, are specified in hexadecimal. The host number may be omitted on 10-Mbps (Megabits per second) Ethernet interfaces, which use the hardware physical address, and on interfaces other than the first.

IFCONFIG *(cont.)* IFCONFIG *(cont.)*

The destination address (*dest_address*) argument specifies the address of the correspondent on the remote end of a point-to-point link.

Parameters

netmask *mask*	Specifies how much of the address to reserve for subdividing networks into sub-networks. This parameter can only be used with an address family of **inet**. The *mask* variable includes both the network part of the local address and the subnet part, which is taken from the host field of the address. The *mask* can be specified as a single hexadecimal number beginning with **0x**, in the standard Internet dotted-decimal notation, or beginning with a name. The *mask* contains 1s (ones) for the bit positions in the 32-bit address that are reserved for the network and subnet parts, and 0s (zeros) for the bit positions that specify the host. The *mask* should contain at least the standard network portion.
trailers	Requests the use of a trailer link-level encapsulation when sending messages. If a network interface supports **trailers**, the system will, when possible, encapsulate outgoing messages in a manner that minimizes the number of memory-memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol (see **arp**), this flag indicates that the system should request that other systems use **trailers** when sending to this host. Similarly, trailer encapsulations will be sent to other hosts that have made such requests. Currently used by Internet protocols only.
-trailers	Disables the use of a trailer link-level encapsulation. The use of **-trailers** may be disabled by default (check your vendor documentation).
up	Marks an interface as working (**up**). This parameter is used automatically when setting the first address for an interface, or can be used to enable an interface after an **ifconfig down** command. If the interface was reset when previously marked with the parameter **down** (see the following section for a description of this parameter), the hardware will be reinitialized.
down	Marks an interface as not working (**down**), which keeps the system from trying to transmit messages through that interface. If possible, the **ifconfig** command also resets the interface to disable reception of messages. Routes that use the interface, however, are not automatically disabled.
arp	Enables the use of the Address Resolution Protocol (ARP) in mapping between network-level addresses and link-level addresses. This parameter is on by default.

IFCONFIG *(cont.)* IFCONFIG *(cont.)*

-arp Disables the use of the ARP. Use of this parameter is not recommended, however, as your system will then only be able to communicate with other hosts that are configured with the parameter **-arp**.

-broadcast *address*
Specifies the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part consisting of all 1s (ones). Note that the computation of the host part is dependent on **netmask** (see the description of the **netmask** parameter).

debug Enables driver-dependent debug code. This might turn on extra console error logging. (See your hardware documentation for further information.)

-debug Disables driver-dependent debug code.

dest_address Specifies the correspondent on the other end of a point-to-point link.

ipdst Specifies an Internet host willing to receive IP packets encapsulating packets bound for a remote network. For an Network Systems (NS) case, an apparent point-to-point link is constructed, and the address specified will be taken as the NS address and network of the destinee.

alias Establishes an additional network address for this interface. This is sometimes useful when changing network numbers and one wishes to accept packets addressed to the old interface.

delete Removes the network address specified. This would be used if you incorrectly specified an alias, or if it was no longer needed. If you have incorrectly set an NS address having the side effect of specifying the host portion, removing all NS addresses will allow you to respecify the host portion.

metric *number* Sets the routing metric, or number of hops, for the interface to the value of *number*. The default value is 0 (zero) if *number* is not specified, indicating that both hosts are on the same network. The routing metric is used by the **routed** daemon, with higher metrics indicating that the route is less favorable.

IFCONFIG *(cont.)* IFCONFIG *(cont.)*

Examples

The following example configures a HIPPI network interface on node 10, setting the network address to 192.9.2.5, the netmask to 255.255.255.0, and disabling trailer link-level encapsulation.

```
# ifconfig '<10>ifhip0' 192.9.2.5 netmask 255.255.255.0 -trailers
```

To query the status of serial line interface **sl0** , enter:

```
$ ifconfig sl0

sl0: flags=51<UP,POINTOPOINT,RUNNING>
    inet 192.9.201.3 ---> 192.9.354.7 netmask 0xffffff00
```

To configure the local loopback interface, enter:

```
# ifconfig lo0 inet 127.0.0.1 up
```

Only a user with superuser authority can modify the configuration of a network interface.

Files

/usr/sbin/ifconfig	Specifies the command path

See Also

Commands: **netstat**(1)

INITPART

Initializes configuration files for the default root partition.

Syntax

initpart

Description

The **initpart** command is run automatically when the system goes into multiuser mode. This command creates the following files:

- The partition information file */etc/nx/.partinfo* for the default root partition. This file helps creates the root partition.
- The initial allocator configuration file */etc/nx/allocator.config*. The file */etc/nx/allocator.config* specifies the defaults that the allocator daemon uses for configuration information.

The file */etc/nx/.partinfo* is created using the information provided in bootmagic strings such as *BOOT_MESH_X*, *BOOT_MESH_Y*, and *BOOT_NODE_LIST*.

Examples

The following example creates new default files for the */etc/nx/.partinfo* and */etc/nx/allocator.config* files:

```
# /sbin/initpart
```

Errors

```
<boot-magic-name> not found in bootmagic
```

The specified bootmagic string is not in the *bootmagic* file.

Files

/sbin/initpart	Specifies the command path.

INITPART *(cont.)* INITPART *(cont.)*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

allocator, allocator.config, mkpart

INQUIRE

Displays SCSI device information.

Syntax

inquire [**-p**] *device*

Arguments

-p	Reports the device information for a RAID controller and its RAID drives. If the device is not a RAID, the inquire command prints the device information and indicates that the device is not a RAID.
device	Specifies the device special file of the SCSI peripheral to inquire about. Only SCSI pass-through devices may be used successfully, for example:

```
% inquire /dev/io*/scsi?
```

Description

Use the **inquire** command to display information from a SCSI peripheral regarding the device type, vendor and product identification, and revision level. The information displayed is in the following format:

```
device_type vendor_ID product_ID revision_level
```

The device type displayed may be one of the following: **disk**, **tape**, **printer**, **worm**, **cdrom**, or **medium changer**. All other fields are vendor specific and are displayed as returned from the device.

Examples

The following example invokes the **inquire** command on the SCSI device 0 (zero) attached to I/O node 0 (zero):

```
# inquire /dev/io0/scsi0
disk NCR ADP-92/01 0302
```

This command displays that the device at SCSI device 0 (zero) is a RAID controller.

INQUIRE *(cont.)* INQUIRE *(cont.)*

The following example uses the **-p** option to get device information from the RAID and its individual disks attached to SCSI device 1.

```
# inquire -p /dev/io1/scsi1
 disk NCR ADP-92/01 0306
 disk MAXTOR MXT-1240S I1.2
 disk MAXTOR MXT-1240S I1.2
 disk MAXTOR MXT-1240S I1.2
 disk MAXTOR MXT-1240S I1.2
 disk MAXTOR MXT-1240S I1.2
```

In this example, the **inquire** command displays that SCSI device1 is a RAID controller with five disk drives attached.

Files

/sbin/inquire Specifies the command path.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

ace, **acf**, **amd**, **apc**, **apr**, **arc**, **can**

IPD

Starts the Interactive Parallel Debugger.

Syntax

ipd

Description

The **ipd** command starts the Interactive Parallel Debugger (IPD) program. IPD is a complete symbolic, source-level debugger for parallel programs that run under the operating system operating system. Beyond the standard operations that allow the debugging of serial programs, IPD offers custom features that allow you to debug parallel programs. IPD lets you debug parallel programs written in C, C++, Fortran, and assembly language. IPD also allows you to examine core files for post-mortem debugging.

IPD consists of a set of debugging commands, for which help is available from within IPD. After running IPD, entering either **help** or **?** at the IPD prompt returns a summary of all IPD commands.

IPD Commands

The following commands are available when you start IPD. Refer to the *Paragon™ System Interactive Parallel Debugger Reference Manual* for complete command descriptions.

alias Display or set aliases.

alias [*alias_name* [*command_string*]]

assign Assign a value to a program variable, address, or register.

Assign a value to a program variable:
assign [*context*] [*file*{}] [*procedure*()] [#*line*] *variable* [,*count*] = *expression*

Assign a value to a program address:
assign [*context*] [-*size_switch*] *address*[:*address*|,*count*] = *expression*

Assign a value to a register:
assign [*context*] [-*size_switch*] *register_switch* = *expression*

break

Set a breakpoint or display current breakpoints.

Display breakpoint information:
break [*context*] [**-full**]

Set code breakpoint at procedure:
break [*context*] [*file*{}] *procedure*() [**-after** *count*]

Set code breakpoint at source line number:
break [*context*] [*file*{}] [*procedure*()] #*line* [**-after** *count*]

Set code breakpoint at instruction address:
break [*context*] *address* [**-after** *count*]

commshow

Display the handles (names) for communicators assigned by the debugger.

List all handles for communicators:
commshow

Display the handle for a communicator specified by a variable or expression in the current scope of context:
commshow [*context*] *expression*

Display the handle for a communicator specified by an expression containing global or static C variables:
commshow [*context*] *file*{} *expression*

Display the handle for a communicator specified by an expression containing local procedure variables:
commshow [*context*] [*file*{}] *procedure*() *expression*

Display the handle for a communicator specified by an expression containing variables local to a block in C or C++:
commshow [*context*] [*file*{}] #*line expression*

Display the handle for a communicator specified at a memory location:
commshow [*context*] *address*

context Set the debug context, defining the default set of processes and nodes to which debug commands apply.

Set the debug context to compute partition processes:
context ({**all** | **nodes** | *nodelist*} **:** {**all** | *ptypelist*})

Set the debug context to the host process (host-node model):
context (host)

Set the debug context to service partition processes (host-node model):
context (host : {**all** | *ptypelist* })

Set the debug context using rank values as process identifiers (MPI applications):
context (*communicator* : {**all** | *ranklist*})

Display the context in which the application was loaded:
context [-pid]

continue Continue execution of processes stopped by command or breakpoint in the current context.

continue [*context*] [**-nosignal**]

coreload Load core files for examination.

coreload [-all | -fault | -first | -nonfault | *context*] [*core_name*] [**-pn** *partition*] [**-sz** *size*]

disassemble Display machine code listing of process instructions.

Disassemble from current execution point:
disassemble [*context*] [,*count*]

Disassemble starting from an instruction address:
disassemble [*context*] *address* [:*address* | ,*count*]

Disassemble starting from procedure:
disassemble [*context*] [*file*{}] *procedure*() [,*count*]

Disassemble starting from a source line number:
disassemble [*context*] [*file*{}] [*procedure*()] *#line* [: *#line* |,*count*]

display Display the value of the specified variable, memory address, or processor registers.

Display the value of variable or expression in current scope of context:
display [*context*] [*-format_switch*] [*variable* | *expression*] [,*count*]

Display the value of an expression containing global or static C variables:
display [*context*] [*-format_switch*] *file*{} [*variable* | *expression*] [,*count*]

Display the value of an expression containing a local procedure variable:
display [*context*] [*-format_switch*] [*file*{}] *procedure*()
[*variable* | *expression*] [,*count*]

Display the value of an expression containing variables local to a block in C or C++:
display [*context*] [*-format_switch*] [*file*{}] #*line* [*variable* | *expression*]
[,*count*]

Display the value of a memory address:
display [*context*] *address* [:*address*|,*count*]

Display the contents of all processor registers:
display [*context*] **-register**

Display the contents of one processor register:
display [*context*] [*-format_switch*] *-register_name*

exec Read and execute IPD commands from the specified file.

exec [**-echo** | **-step**] *filename*

exit Terminate a debug session and exit IPD.

exit

flush Set performance monitoring instrumentation flush policy.

List current flush policy:
flush [*context*]

Change performance monitoring event trace buffer **flush** policy:
flush [*context*] [**-stop** | **-wrap** | **-continue**]

frame	Display the stack traceback(s) of the current or specified context. **frame** [*context*]
help	Display IPD commands and syntax. List all commands: { **help** | **?** } Obtain syntax help: { **help** | **?** } *command*
instrument	Add, remove, or display program instrumentation for performance monitoring. Instrument program: **instrument** [*context*] [[**-on**] *perf_name* [*start_location* [*,stop_location* [*,write_location*]]] [-**bufsize** *value*] [[**-force**] *path_name*]] Immediately write performance data and terminate monitoring: **instrument** [*context*] **-write** Remove performance monitoring instrumentation: **instrument** [*context*] **-off** [**-nowrite** | **-write**] [*perf_name*] List performance monitoring instrumentation information: **instrument** [*context*]
kill	Terminate and remove processes in the current or specified context. **kill** [*context*] [**-force**] [**-fault** | **-nonfault** | **-notfirst**]
list	Display source code lines. List from current execution point: **list** [*context*] [*,count*] List starting from procedure: **list** [*context*] [*file*{}] *procedure*() [*,count*] List starting from a source line number. **list** [*context*] [*file*{}] [*procedure*()] #*line*[**:** #*line* | *,count*]

IPD *(cont.)* IPD *(cont.)*

load Load an application under debugger control.

load *filename* [<*infile*] [*program_args*]

log Turn debug session logging on or off, or display the name of the current log file.

log [[**-on**] *filename* | **-off**]

more Control scrolling of IPD information on the display.

more [**-on** | **-off**]

msgqueue Display messages sent but not yet received.

msgqueue [*context*] [*type*]

msgstyle Set or display how process identifiers within contexts are displayed and interpreted.

msgstyle [**-nx** | **-mpi**]

process Display information about user processes controlled by IPD.

process [*context*] [**-change**] [**-loadfile**] [**- full**]

quit Terminate a debug session and exit IPD.

quit

recvqueue Display pending receives.

recvqueue [*context*] [*type*]

remove Remove breakpoints, watchpoints, and tracepoints.

remove [*context*] [*actionpoint_number* [*actionpoint_number*] ...] | **-all**

rerun Reload and restart the execution of the program, clearing previous command line arguments.

rerun [<*infile*] [*program_args*]

run	Reload and restart execution of a program, reusing previous command line arguments. **run** [<*infile*] [*program_args*]
set	Set or display IPD variables. List all set variables: **set** List variable definition: **set** *variable_name* Define new or redefine old variable: **set** *variable_name string*
signal	Set or display the set of signals that is reported. Display the current signal-reporting mask: **signal** [*context*] Enable signal reporting for specified signals: **signal** [*context*] **-on** [*signo* [*signo*]... | **-all**] Disable signal reporting for specified signals: **signal** [*context*] **-off** signo]... | **-all**]
source	Set or display the current source directory search paths. Display source directory search path: **source** [*filename*] Set new source directory search path: **source** *filename directory* [*directory*] ... Add directories to source directory search path: **source** *filename* **-add** *directory* [*directory*] ... Remove directories from source directory search path: **source** *filename* **-remove** *directory* [*directory*] ...
status	Display the debugger status and partition information. **status**

step	Single step through the processes in the current or specified debug context. Step through source line(s): **step** [*context*] [**-call**] [,*count*] Step one machine instruction: **step** [*context*] **-instruction** [**-call**] [,*count*]
stop	Stop program execution in the current context. **stop** [*context*]
system	Execute a shell command. **system** *shell_command* or ! *shell_command*
threads	Control number of threads displayed for each process with the **display**, **frame, process**, and **type** commands. **threads** [**-off** | **-on**]
trace	Set a tracepoint or display current tracepoints. Display tracepoint information: **trace** [*context*] [**-full**] Set tracepoint at procedure: **trace** [*context*] [*file*{}] *procedure*() [**-after** *count*] Set tracepoint at source line number: **trace** [*context*] [*file*{}] [*procedure*()] #*line* [**-after** *count*] Set tracepoint at instruction address: **trace** [*context*] *address* [**-after** *count*]

IPD *(cont.)* IPD *(cont.)*

type Display the type of variables in the current or specified context.

Display type of variable in current scope of context:
type [*context*] *variable*

Display type of global or static C variable:
type [*context*] *file*{} *variable*

Display type of a C++ class member variable:
type [*context*] *class*::[*class*::]...*variable*

Display type of local procedure variable:
type [*context*] [*file*{}] *procedure*() *variable*

Display type of a variable local to a block (in C or C++):
type [*context*] [*file*{}] [*#line*] *variable*

unalias Delete previously defined aliases.

unalias {*alias_name* [*alias_name* ...] | **-all**}

unset Delete previously defined command line variables.

unset {*variable_name* [*variable_name*] ...] | **-all**}

wait Wait until all processes within the context have stopped running.

wait [*context*]

watch Set a watchpoint (data breakpoint) or display current watchpoints.

Display Watchpoint information:
watch [*context*] [**-full**]

Set Watchpoint on variable:
watch [*context*] [**-access** | **-write**] [*file*{}] [*procedure*()] *variable* [*#line*] [**-after** *count*]

Set Watchpoint on a memory address:
watch [*context*] [**-access** | **-write**] *address* [**-after** *count*]

The IPD command parameters are defined in general as follows. Some commands may have exceptions or special argument requirements:

alias_name	A string (the first character must be a letter) that you choose to represent a command.
command_string	The IPD command string that an *alias_name* represents. All of the text following the *alias_name* to the end of the command line, including spaces, the pound sign (#), and semicolons, are part of the *command_string*.

context

Defines the nodes and process types that the command affects. If you do not specify a context, the default context applies. Specify the context as one of the following:

({ **all** | **host** | *nodelist* }**:**{ **all** | *ptypelist* })

(host)

context (*communicator* : {**all** | *ranklist*})

The *nodelist* is the list of nodes, and the *ptypelist* is the list of processes on those nodes to which the command will apply. These can be specified as a single value, a comma-separated list, a range, a combination, or the special value **all**, indicating all nodes and/or process types.

The **host** argument sets the context to the host process (host-node model).

The *communicator* is the name of an MPI communicator group to which the ranks in a *ranklist* apply. The *ranklist* is an identifier for a process within an MPI communicator's group of processes. A single value indicates a single process. You can specify a range of ranks with the syntax *rank1*..*rank2*, where *rank2* > *rank1*. Specify a list of ranks by separating rank numbers with commas, using the syntax *rank, rank, rank...* The *ranklist* may include both a range of tanks and a list of ranks.

variable

The symbolic name of a variable. For assembly language programs, you can use symbolic names if you have used the proper assembler directives to produce the symbolic debug information. For C, C++, or Fortran programs, IPD follows the scoping rules of the language. IPD looks for the variable in the following four places, in order: in the current code block, in the current procedure, in the static variables local to the current file, and finally, in the global program variables. For C++ programs, IPD searches for class members after looking in the current procedure. To specify variables not in the current scope, prefix the variable name with the *file{}*, *procedure*() and/or *#line* qualifiers.

file{}	The name of the source module in which a variable resides. To refer to a file other than the location of the current execution point, you must prefix the variable name with *file*{}. When you refer to a procedure, you can omit the *file*{} name unless there are duplicate procedure names, because IPD can find the source file from the symbol table information.
procedure()	The name of the procedure in which a variable resides. You need to specify the procedure when the execution point is not in that procedure.
size_switch	Use *size_switch* when you assign a value to an address. It specifies how many bytes (1, 2, 4, or 8) are to be assigned to the given address. You may only assign whole numbers an address; these may be hexadecimal, octal, or decimal. Floats, complex, characters, and strings are not allowed. The *size_switch* can be one of the following:

byte	1 byte
short	2 bytes
long	4 bytes
double	8 bytes

address	A valid memory address. You can specify a range of addresses either as *start_address:end_address* (for example **0x208:0x21b**) or as *address,count*, where *count* is the desired number of bytes in the address range (for example **0x208,20**).
count	A positive integer used to denote a range of an array variable or address. First, you designate the beginning array element or address followed by a comma and the *count*; for example, **x(10),10**, or **0x208,8**. The count may be negative for the **list** and **disassemble** commands.
register_switch	The *register_switch* can be used to assign a value to a register or a floating-point register pair. The value must be numeric. When assigning to a single-word register, the default size is **long**. You can override the default with the **-double** size specification to assign to a floating-point register pair. Similarly, the default size is **double** for double-word registers (**-KI**, **-KR**, **-T**), but can be overridden with the **-long** switch. Other size specifications (**-byte** and **-short**) are not allowed.

IPD *(cont.)*   IPD *(cont.)*

When assigning to a floating-point register pair, always specify the even-numbered register of the pair. The register switches recognized for the **assign** command are **-r0, -r1, -sp, -fp,** and **-r4** through **-r31** for the integer registers, and **-f0** through **-f31** for the floating-point registers. You may also assign to the dual-operation floating-point registers, **-KI**, **-KR** and **-T**, and to the control registers, **-psr, -epsr, -fir, -fsr,** and **-db.** When assigning to the control registers, IPD reports a warning if you try to change the supervisor bits of those registers.

#line A source line number. The line number must be preceded with a pound sign (#). You only need to specify a line number if the variable is hidden by a variable with the same name in the current scope.

format_switch The *format_switch* overrides the symbol table information that would normally determine how a symbol's value would be printed. For variable displays, the *format_switch* can be one of the following:

alphanumeric	**double**	**real** (equivalent to the C float type)
complex	**float**	**string**
dcomplex	**hexadecimal**	**logical**
decimal	**octal**	

For register displays, the *format_switch* can be one of the following:

decimal	**hexadecimal**
double	**real**
float	

register_name A specific processor register. The register names follow the processor naming conventions. The following is a list of register names.

-r0, -r1, -sp, -fp, -r4 .. **-r31**	Integer register file
-f0 .. **-f31**	Floating-point register file
-psr and **-epsr**	Processor status register and extended processor status register
-db	Data breakpoint register
-dirbase	Directory base register
-fsr	Floating-point status register
-fir	Fault instruction register
-KI, **-KR**, and **-T**	"Konstant" registers and "temporary register"
-merge	Merge register

IPD *(cont.)*

perf_name — *perf_name* is the name of the performance utility that will be used to analyze the resulting performance data. The three supported performance utilities are prof, gprof and paragraph and the corresponding switches are **-prof**, **-gprof** and **-paragraph**.

start_location — The *start_location* is the point in the code at which performance data collection begins. This can be an entry or exit point to a procedure, a line number, or an address. The syntax for the *start_location* specification is one of the following:

> [**-entry**|**-exit**] [*file*{}] *procedure*()
>
> [*file{}*] [*procedure*()] #*line*
>
> *address*

stop_location — The location at which performance data collection ends. The *stop_location* can be an entry or exit point of a procedure, a line number, or an address. The syntax for the *stop_location* specification can be one of the following:

> [**-entry**|**-exit**] [*file*{}] *procedure*()
>
> [*file{}*] [*procedure*()] #*line*
>
> *address*

write_location — The location at which all performance data is written and performance monitoring is terminated. The *write_location* can be an entry or exit point of a procedure, a line number, or an address. The syntax for the *write_location* specification can be one of the following:

> [**-entry**|**-exit**] [*file*{}] *procedure*()
>
> [*file{}*] [*procedure*()] #*line*
>
> *address*

path_name — *path_name* can either be a single event trace file for all nodes and processes or *path_name* can be a directory name where a data file exists for each process in the application.

filename — The file name of the program that you want to load. Specify the path name if the file is not in the current directory.

infile	A program's input file argument. All of the program's standard input (stdin) will be read from *infile*. The *infile* is read during a **wait** command.
program_args	Arguments to be passed to the program. Anything following *infile* is assumed to be an argument. This includes any semicolons.

If the program was compiled with the -nx option, *program_args* should include any operating system command line arguments necessary for loading the application (such as **-pn** *partition*, **-sz** *num_nodes*, **-pt** *process_type*, **-nd** *node_list*, and so on).

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

Paragon^{TM} System Interactive Parallel Debugger Reference Manual

Paragon^{TM} System Application Tools User's Guide

LOAD_LEVELD

load_leveld, loadlevel: Balances the system load across the service nodes of a Paragon system.

Syntax

load_leveld [**-s**] [**-p** *file*]

loadlevel start | **stop**

Arguments

-s	Specifies *static load-leveling*. By default, load-leveler daemons exchange information only. If this switch is not supplied, the load-leveler daemons do not attempt to distribute processes in an attempt to even out the processing load across the system. To enable static load-leveling, *remote process creation* must also be enabled. By default, remote process creation is enabled. You enable remote process creation by setting the bootmagic string *ENABLE_FORK_REMOTE* to **T** or **t**. If this string is set to **F** or **f**, static load-leveling is disabled.
-p *file*	Specifies an alternate *parameters* file for load-leveling. The default file is */etc/load_level/parameters*. The *parameters* file contains load-leveler parameters that specify the load leveler's behavior. See the **parameters** manual page for more information about the *parameters* file and the load-leveler parameters.
start	Instructs the **loadlevel** script to starts the load-leveler daemons.
stop	Instructs the **loadlevel** script to stop the load-leveler daemons.

Description

Both the **load_leveld** and **loadlevel** commands control the load-leveler daemons on service nodes in the Paragon system. The **loadlevel** command is a script that can start or stop the daemons. The **load_leveld** command is the command within the **loadlevel** script that enables static load leveling and can specify an alternate *parameters* file. Typically, the **load_leveld** command is executed indirectly by the **loadlevel** script. However, it is possible to execute **load_leveld** directly from the command line. Both these commands can be run on a Paragon system by the system administrator only.

LOAD_LEVELD *(cont)* LOAD_LEVELD *(cont)*

The **load_leveld** command starts the load-leveler daemon on each of the service nodes of a Paragon system or on a subset of nodes specified by the *nodes_to_use* parameter in the *parameters* file. The load leveler improves system performance by balancing the load across the service nodes. The load leveler behaves identically on each service node. Specifically, no master/slave relationship exists among the nodes, no central repository for load-leveler information exists, and the nodes using Mach IPC messages passes load-leveler information among themselves.

The **loadlevel** script starts or stops the load-leveler daemons. When daemons are started, the **-s** and **-p** switches are applied in accordance to the embedded **load_leveld** command. To start the load leveler, enter the following:

```
# /sbin/init.d/loadlevel start
```

This command starts the load-leveler daemons with static load-leveling. A daemon is started on each service node in the service partition. The **loadlevel start** command is executed automatically during multi-user start up. You should only use this command if the load leveler has been stopped.

To stop the load leveler, use the **loadlevel stop** command as follows:

```
# /sbin/init.d/loadlevel stop
```

This command finds the processes for the load-leveler daemons and kills them.

Static Load-Leveling

The Paragon system supports static load-leveling only. During static load-leveling the system computes the loads on each node and then starts processes accordingly. This behavior is unlike dynamic load-leveling where processes that are currently running can be migrated to nodes with lighter loads.

When you start the load leveler, a load-leveler daemon runs on each service node. Each load-leveler daemon locally maintains load information about the service node it is running on (the *local node*) and the other service nodes (*remote nodes*). The OSF server uses the load information to determine whether the **fork()** function will create new processes on the local node or on a remote node. Each load-leveler daemon determines the load information as follows:

- The load average of all the service nodes is calculated.
- The load of the local node is compared against the load average. If the load of the local node is lower than or equal to the load average, the local node is the lightest-loaded node. If the load of the local node is higher than the load average plus the *minimum_overload*, the local node is considered overloaded.
- The nodes that are underloaded are identified. All nodes with load values lower than the load average minus the *minimum_underload* are considered underloaded.

- Each underloaded node is assigned a probability value proportional to the difference between its load and the local node's load.

The load-leveler daemon periodically updates this load information to determine the lightest-loaded node, or "fastnode." The "fastnode" can be either the local node or a remote node. The load leveler hands the "fastnode" value to the local OSF server. Until the "fastnode" value changes, all new processes originating on the local node are created on the "fastnode."

The OSF server's "fastnode" changes if the server receives a new value from the load-leveler daemon. This is controlled by the *fast_node_timeout* parameter. The OSF server discards old "fastnode" values after *FORK_REMOTE_TIMEOUT* seconds, and the server creates all processes on the local node if there is no new "fastnode" value. The bootmagic string *FORK_REMOTE_TIMEOUT* specifies the time-out period when the "fastnode" changes to the local node if the "fastnode" value is not updated. The default value for *FORK_REMOTE_TIMEOUT* is 60 seconds.

Load Information Exchange

Information exchange about load levels is controlled through the information exchange algorithm as described in the paper "A Distributed Load-balancing Policy for a Multicomputer". For information on where to get this paper, see the "See Also" section of this manual page.

You can use the following parameters in the *parameters* file to tune the load information exchange between load-leveler daemons:

- *minimum_overload, minimum_underload*
- *number_vector_elements*
- *re_dispatch_timeout*
- *send_timeout*
- *static_min_load_delta*

See the **parameters** manual page for more information about these parameters.

LOAD_LEVELD *(cont)*   LOAD_LEVELD *(cont)*

When the **loadlevel** script is executed, a load-leveler daemon is started on each service node. The load for the current node is calculated using a combination of weighted averages of three separate load values obtained from the Mach kernel and paging information. A load value for a node is determined as follows:

cpu_utilization + (page-ins per second * *pagein_load*) + (page-outs per second * *pageout_load*)

The calculation of *cpu_utilization* is based on the kernel's 5-second, 30-second, and 1-minute load averages. The relative weight for each of these values is specified with the parameters *first_weight_factor*, *second_weight_factor*, and *third_weight_factor*. Page loads are determined by the parameters *pagein_load* and *pageout_load*. Paging statistics are sampled according to an interval determined by the parameters *pgstat_max_interval* and *pgstat_pref_interval*.

A load average is a positive floating point value, typically less than 10. The load average represents the average number of processes that simultaneously want to use the processor. A node number and its corresponding load value are called a node/load pair.

Each load-leveler daemon uses a load vector to store and exchange load information. The load vector contains a fixed number of node/load pairs that are ordered. The size of the load vector size is specified in the parameter *number_vector_elements*. After the current node's node/load pair is calculated, the node/pair information is placed into the first slot (slot 0) of the load vector. The first half of the vector is sent periodically to the load-leveler daemon on a randomly selected node.

Because message-passing overhead can be expensive, each load-leveler daemon only exchanges load information with the load-leveler daemon on one randomly chosen node. Every *send_timeout* seconds the load-leveler daemons send load information. Between sends, the load-leveler daemons receive half-size load vector messages from other nodes. The received node/load pairs are shuffled together with those in the first half of the existing load vector to produce a new load vector. Specifically, the *received* pairs are sequentially placed in the *odd*-numbered slots of the new vector, and the *existing* pairs are sequentially placed in the *even*-numbered slots of the new vector. Thus at each shuffle, all node/load pairs move to a higher-numbered slot. The exception is the node/load pair in slot 0, which always contains information for the current node.

As new node/load pairs are received and combined with existing pairs, newer information replaces older information. The lower the slot number in which a node/load pair appears, the more recent that load information is. The pair in slot 0 is always that of the current node, and is the most recent of all.

NOTE

When the load-leveler daemon is running, the inet daemon uses the load information to determine which service node a login process is started on.

LOAD_LEVELD *(cont)*

Files

/sbin/init.d/loadlevel
Specifies the command path of the **loadlevel** script.

/usr/sbin/load_leveld
Specifies the command path of the **load_leveld** command.

/etc/load_level/parameters
Specifies load leveler configuration parameters.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

files: **bootmagic**, **parameters**

Amnon Barak and Amnon Shiloh, "A Distributed Load-balancing Policy for a Multicomputer," in *Software—Practice and Experience,* Vol. 15 (September 1985).

LS LS

Lists and generates statistics for files.

Syntax

ls [-aAbcCdfFgilLmnopPqrRstux1] [*file* ... | *directory* ...]

Arguments

-a	Lists all entries in the directory, including the entries that begin with a **.** (dot). Entries that begin with a **.** (dot) are not displayed unless 1) they are explicitly referenced, or 2) the **-a** flag is specified.
-A	Lists all entries, except . (dot) and .. (dot dot).
-b	Displays nonprintable characters in octal notation.
-c	Uses the time of last modification (file created, mode changed, and so on) for sorting (when used with **-t**) or for displaying (when used with **-l**). This flag has no effect when not used with either **-t** or **-l** or both.
-C	Sorts output vertically in a multicolumn format. This is the default when output is to a terminal.
-d	Displays only the information for the directory that is named, rather than for its contents. This is useful with the **-l** flag to get the status of a directory.
-f	Lists the name in each slot for each named directory. This flag turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; this flag uses the order in which entries appear in the directory.
-F	Puts a **/** (slash) after each filename if the file is a directory, an * (asterisk) after each filename if the file can be executed, an = (equal sign) after each filename if the file is a socket, an @ (at sign) for a symbolic link, and a | (vertical bar) for a FIFO.
-g	Displays the same information as **-l**, except for the owner.
-i	Displays the i-number in the first column of the report for each file.

-l Displays the mode, number of links, owner, group, size, time of last modification for each file, and pathname. If the file is a special file, the size field contains the device's node number and the major and minor device numbers. If the file is a symbolic link, the pathname of the linked-to file is also printed preceded by **->**. The attributes of the symbolic link are displayed. The **-n** flag overrides the **-l** flag.

-L Lists the file or directory the link references rather than the link itself, if the argument is a symbolic link.

-m Uses stream output format (a comma-separated series).

-n Displays the same information as **-l**, except that it displays the user and the group IDs instead of the usernames and group names.

-o Displays the same information as with **-l**, except for the group. The **-n** flag overrides the **-o** flag.

-p Puts a slash after each filename if that file is a directory.

-P Displays the stripe attributes for a PFS file. The stripe attributes displayed consist of:

- The file's *stripe unit size*, in bytes. This is the unit of data interleaving used in the PFS file.
- The file's *stripe factor*. This is the size of the PFS file's stripe group. The stripe factor is equal to the number of stripe directories, and when multiplied by the stripe unit equals the size of one PFS file stripe.
- The file's *stripe group*. The stripe group is a list of stripe directories in the UFS or NFS file systems (typically UFS mount points) that are the storage locations for the PFS file.

The stripe attributes of an individual PFS file can also be retrieved and set programmatically with the **fcntl()** system call. See the **fcntl(2)** manual page for more information.

-q Displays nonprintable characters in filenames as a ? (question mark) character, if output is to a terminal (default).

-r Reverses the order of the sort, giving reverse collation or the oldest first, as appropriate.

LS *(cont.)* LS *(cont.)*

-R	Lists all subdirectories recursively.
-s	Gives space used in 512-byte units (including indirect blocks) for each entry.
-t	Sorts by time of last modification (latest first) instead of by name.
-u	Uses the time of the last access instead of time of the last modification for sorting (when used with **-t**) or for displaying (when used with **-l**). This flag has no effect when not used with either **-t** or **-l** or both.
-x	Sorts output horizontally in a multicolumn format.
-1	Forces one entry per line output format; this is the default when output is not directed to a terminal.

Description

The **ls** command writes to standard output the contents of each specified directory or the name of each specified file, along with any other information you ask for with flags. If you do not specify a file or a directory, **ls** displays the contents of the current directory.

By default, **ls** displays all information in collated order by filename. The collating sequence is determined by the **LC_COLLATE** environment variable (see the **ctab** command).

There are three main ways to format the output:

1. List entries in multiple columns by specifying either the **-C** or **-x** flags. **-C** is the default format, when output is to a terminal.

2. List one entry per line.

3. List entries in a comma-separated series by specifying the **-m** flag.

The **ls** command uses **ioctl()** to determine the number of byte positions in the output line. If **ls** cannot get this information, it uses a default value of 80. Note that columns may not be smaller than 20 bytes or larger than 400 bytes.

Modes

The mode displayed with the **-l** flag is interpreted by the first character, as follows:

b	Block special file
c	Character special file
d	Directory
l	Symbolic link
p	First-In-First-Out (FIFO) special file
s	Local socket
-	Ordinary file

Permissions

The next nine characters are divided into three sets of three characters each. The first three characters show the owner's permission. The next set of three characters show the permission of the other users in the group. The last set of three characters show the permission of everyone else. The three characters in each set show read, write and execute permission of the file. Execute permission of a directory lets you search a directory for a specified file.

Permissions are indicated as follows:

r	Read
w	Write
x	Execute or search (directories)
-	No access

The group-execute permission character is **s** if the file has set-group-ID mode. The user-execute permission character is **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **-**) is **t** if the 01000 (octal) bit of the mode is set; see the **chmod** command for the meaning of this mode. The indications of set-ID and the 01000 bit of the mode are capitalized (**S** and **T**, respectively) if the corresponding execute permission is not set.

When the sizes of the files in a directory are listed, the **ls** command displays a total count in 512-byte units, including indirect blocks.

The **LC_TIME** environment variable controls the format of the date and time.

Examples

1. To list all files in the current directory, enter:

   ```
   ls -a
   ```

   This lists all files, including . (dot), .. (dot dot), and other files with names beginning with a dot.

2. To display information about the special file *rz0a* in the */dev/io* directories, enter:

   ```
   ls -l /dev/io*/rz0a
   ```

   ```
   brw-r--r-- 1 root system  3: 3, 0 Jun 05 09:08 io0/rz0a
   brw------- 1 root system  7: 3, 0 Jun 08 07:29 io1/rz0a
   brw------- 1 root system 11: 3, 0 Jun 08 07:29 io2/rz0a
   brw------- 1 root system 15: 3, 0 Jun 08 07:29 io3/rz0a
   ```

3. To display file striping information about a PFS file named *ls_sa*, enter:

   ```
   ls -P /pfs/ls_sa
   ```

   ```
   sunit 524288 sfactor 4    sdirs /home/.sdirs/vol3 /pfs/ls_sa
                                 /home/.sdirs/vol0
                                 /home/.sdirs/vol1
                                 /home/.sdirs/vol2
   ```

   This displays the stripe unit size (sunit), the stripe factor (sfactor), and the stripe group (sdirs)..

4. To display detailed information, enter:

   ```
   ls -l chap1 .profile
   ```

   This displays a long listing with detailed information about the files **chap1** and **.profile**.

5. To display detailed information about a directory, enter:

   ```
   ls -d -l . manual manual/chap1
   ```

   This displays a long listing for the directories **.** and **manual**, and for the file **manual/chap1**. Without the **-d** flag, this command lists the files in **.** and **manual** instead of providing detailed information about the directories themselves.

6. To list the files in the current directory in order of modification time, enter:

   ```
   ls -l -t
   ```

   This displays a long listing of the files that were modified most recently, followed by the older files.

Files

/etc/passwd	Contains user information.
/etc/group	Contains group information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

OSF/1 Command Reference: **chmod(1)**, **find(1)**, **ln(1)**, **stty(1)**, **ioctl(2)**

OSF/1 Programmer's Reference: **ioctl(2)**

calls: **fcntl(2)**

"Using Internationalization Features" in the *OSF/1 User's Guide*

LSIZE LSIZE

Sets or increases the size of one or more files.

Syntax

lsize [**-a**] *size file ...*

Arguments

-a	Increases the file size by *size* bytes. If you do not specify the **-a** switch, **lsize** changes the file size to be exactly *size* bytes.
size	Number of bytes used to set or increase the file size. To specify units other than bytes, append the appropriate letter to the *size* argument:

k	Kilobytes (1024 bytes)
m	Megabytes (1024K bytes)
g	Gigabytes (1024M bytes)

file	Pathname of a file to be changed. If it exists, you must have write permission for this file. If it does not exist, you must have write permission on the file's directory.

Description

Use the **lsize** command to set or increase the disk space allocated for one or more files. There are two forms of this command. The following form sets the file size to *size* bytes:

lsize *size file ...*

The second form increases the file size by *size* bytes:

lsize -a *size file ...*

If a specified file does not exist, this command creates the file and allocates *size* bytes for the file.

LSIZE *(cont.)* LSIZE *(cont.)*

The **lsize** command cannot decrease the size of a file. If you specify a file size smaller than the file's current size, the command has no effect on the file.

The **lsize** command allocates file space for the file from the file system's disk space (the actual number of disk blocks may be more than the size requested). The file's new file space is undefined.

NOTE

Because NFS does not support disk block preallocation, the **lsize** command is not supported on files residing in remote file systems that have been NFS mounted. The **lsize** command is supported on files in UFS and PFS file systems only.

Examples

The following command sets the size of the file *mydata* to 5M bytes:

```
lsize 5m mydata
```

The following command increases the size of the file *mydata* by 200K bytes:

```
lsize -a 200k mydata
```

Errors

`No space left on device, `*`n`*` bytes allocated for `*`file`*`.`

You specified a size to allocate for the file that exceeds the space available on the device.

`Invalid size.`

You specified a size that has an invalid format.

`Size `*`size`*` is too large.`

You specified a size that exceeds the size of an integer.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

Calls: **esize()**, **lsize()**

LSPART LSPART

Lists the subpartitions of a partition.

Syntax

lspart [**-l** | **-p**] [**-r**] [*partition*]

Arguments

-l	Displays the attributes of the nodes in each subpartition. In the ATTR column, the command displays the attributes common to every node in the subpartition. Node numbers are relative to the specified partition.
-p	Displays the attributes of the nodes in each subpartition. In the ATTR column, the command displays the attributes common to every node in the subpartition. Node numbers are relative to the *root* partition.
-r	Recursively lists information about all the subpartitions of the specified partition.
partition	Absolute or relative partition pathname of the partition to be listed. If you do not specify a partition argument, the **lspart** command uses the value specified in the *NX_DFLT_PART* environment variable. If *NX_DFLT_PART* is not set, the command uses the *compute* partition. You must have read permission on the specified partition to list its subpartition information.

Description

The **lspart** command displays information about subpartitions within a partition. Without the **-l** or **-p** switches, the command shows in a column format general information that applies to each subpartition. With the **-l** or **-p** switch, the command follows this general information with attribute information for each node in the subpartition.

The columns of information are as follows:

`USER`	Name of the user who owns the subpartition.
`GROUP`	Group name of the group that owns the subpartition.
`ACCESS`	Access permissions for the subpartition.
`SIZE`	Number of nodes allocated to the subpartition. Numbers separated by a slash (/) indicate that one or more of the nodes for the partition are unusable.

LSPART *(cont.)* LSPART *(cont.)*

FREE	Number of unallocated, usable nodes in the subpartition. A node is *free* when no application is running on it.
RQ	The rollin quantum or scheduling type of the partition, as follows:

-	The partition uses standard scheduling.
SPS	The partition uses space sharing.
time	The partition uses gang scheduling with a rollin quantum of *time*. The *time* is expressed as a number followed by an optional letter: no letter for milliseconds, **s** for seconds, **m** for minutes, or **h** for hours.

EPL	Subpartition's effective priority limit.
PARTITION	Name of the subpartition.
ATTR	The node attributes common to all the nodes in the subpartition. This column appears only with the **-l** and **-p** switches.

Asterisks (*) in all the fields except the *partition* field means you do not have read permission on the partition or subpartition. A dash (-) in a field means the characteristic is not set.

Examples

The following example lists the subpartitions of the partition *mypart*. The parent partition for the *mypart* partition is the *compute* partition.

```
% lspart mypart

USER    GROUP   ACCESS  SIZE  FREE  RQ       EPL   PARTITION
chris   eng     777     16       4  15m      3     mandlebrot
pat     mrkt    755      4       0  10m      10    slalom
```

The display shows that the *mypart* partition has two subpartitions: *mandelbrot* and *slalom*. The *mandelbrot* partition is owned by the user *chris* in the group *eng*. It has read, write, and execute permissions for everyone. The subpartition has a size of 16 nodes, has 4 free nodes, uses a rollin quantum of 15 minutes, and has an effective priority limit of 3. The *slalom* subpartition is owned by user *pat* in group *mrkt*. It has read, write, and execute permissions for the owner and has read and execute permissions for the group and other users. This subpartition has a size of 4 nodes, contains no free nodes, has a rollin quantum of 10 minutes, and has an effective priority limit of 10.

The following command has the same effect, but uses an absolute partition pathname:

```
% lspart .compute.mypart
```

To recursively list all of a partition's subpartitions, use the **-r** switch. For example:

```
% lspart -r mypart

  USER    GROUP   ACCESS  SIZE  FREE  RQ      EPL    PARTITION
mypart:
  chris   eng     777     16       4  15m     3      mandelbrot
  pat     mrkt    755      4       0  10m     10     slalom
mypart.mandelbrot:
  chris   eng     777     16       4  15m     10     hi_pri
  chris   eng     777     16       4  15m     1      lo_pri
```

The **lspart -r** output shows that *mypart.mandelbrot* has two subpartitions (*hi_pri* and *lo_pri*). while *mypart.slalom* has no subpartitions. Additionally, neither *mypart.mandelbrot.hi_pri* nor *mypart.mandelbrot.lo_pri* has any sub-subpartitions.

The following example shows that one or more of the nodes for the indicated partition are unusable.

```
% lspart mypart

  USER    GROUP   ACCESS  SIZE   FREE  RQ      EPL    PARTITION
  chris     eng   777     14/16     5  15m       3    mandlebrot
```

In this example, the size entry 14 / 16 shows 16 nodes allocated to the *mandelbrot* subpartition, of which 14 are usable and 2 are unusable. You cannot run applications or allocate any partitions on unusable nodes. Use the **showpart** command to find which specific nodes in a subpartition are unusable

The following example shows the display for a user who does not have read permission for the partition *bozo*:

```
% chpart -mod 000 bozo
% lspart
  USER    GROUP   ACCESS  SIZE  FREE  RQ      EPL    PARTITION
     *        *        *     *     *   *        *    bozo
```

NOTE

Asterisks (*) indicate you do not have read permission in the partition.

LSPART *(cont.)* LSPART *(cont.)*

You can use the **-l** switch to list the attributes of the nodes in each subpartition. Using this switch adds the ATTR column to the general information and also produces a list of attributes for every node in the partition. The list follows the general information that appears in the columns. For example:

```
% lspart mypart -l
 USER      GROUP     ACCESS  SIZE      FREE  RQ   EPL  PARTITION    ATTR
 chris     eng          777    16         4  15m    3  mandelbrot   1proc,GP,16mb

0..15 1proc,GP,16mb

 pat       mrkt         755     4         0 SPS    10  slalom       2proc,MP

0 2proc,MP,32mb
1..3 2proc,MP,64mb

 *         *              *     *         *   *     *  private      *
```

In the above example:

- The subpartition *mandelbrot* consists of 16, one-processor, GP nodes each having 16M bytes of memory. The information in the ATTR column shows attributes common to all nodes. The information following the general information shows the attributes for each node. In this case, all nodes have the same attributes.
- The subpartition *slalom* consists of 4,two-processor, MP nodes. The ATTR column shows that each node is a two-processor, MP node. The specific node information shows that node 0 has 32M bytes of memory; and nodes 1, 2, and 3 have 64M bytes of memory.
- The subpartition *private* has access permissions that do not grant the user read permission. Consequently, all characteristics appear as asterisks (`*`), and no node attributes appear.

Errors

`Partition not found`

You specified a partition that does not exist.

`Partition permission denied`

You do not have read permission for the specified partition.

LSPART *(cont.)* LSPART *(cont.)*

Files

/usr/bin/lspart	Specifies the command path.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

chpart, **mkpart**, **pspart**, **rmpart**, **showpart**

MACHID MACHID

Starts the *machid* name server that establishes a name space for Mach IDs.

Syntax

machid

Description

This command is only executed during a system reset when the system goes to multiuser mode.

The **machid** command starts the *machid* name server that the kernel uses to handle Mach IDs. For example, given the Mach ID of a task the *machid* server returns the Mach IDs of the task's threads. Every call to the *machid* server takes an authentication port.

Sensitive calls, like killing a task, require appropriate privileges (host privilege port).

The *machid* server registers with the *netname* service, under the name *MachID*.

Mach IDs are drawn from a space of 32-bit unsigned numbers. Zero is always an invalid Mach ID. Mach IDs are never reused.

Files

/mach_servers/machid
: Specifies the command path.

/usr/include/servers/machid.h
: Specifies the path of the include file *machid.h.*

/usr/include/servers/machid_debug.h
: Specifies the path of the include file *machid_debug.h.*

/usr/include/servers/machid_types.h
: Specifies the path of the include file *machid_types.h.*

MACHID *(cont.)*

Limitations and Workarounds

The *machid* server holds onto every send. Tasks expecting no-senders do not receive notifications.

The *machid* server does not check for the exhaustion of its name space.

The authentication is very primitive. General information calls are always approved. Calls to read memory, fetch register state, and modify kernel state are not distinguished, and require a valid authentication port. A privileged host port authorizes operations on all objects on that host. A task port authorizes operations on threads in that task. An object port authorizes operations on that object.

For additional information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

ms, snames

MAKEDEV MAKEDEV

Creates SCSI device entries.

Syntax

MAKEDEV [*node*]

Arguments

node	Node number for an I/O node with devices connected to it.

Description

Execute the **MAKEDEV** command in the */dev* directory.

If the *node* parameter is specified, the **MAKEDEV** command makes SCSI device entries for the specified node only. If the specified node does not have devices attached to it, the **MAKEDEV** command exits with an error message indicating that no device entries were created for the specified node.

If the *node* parameter is not specified, the **MAKEDEV** command does the following:

- Finds the list of all the available IO nodes in the system and builds appropriate device entries for each IO node under the */dev* directory.
- Checks all the SCSI IDs for each of the IO nodes in the system.
- Determines what type of devices (disk or tape) are attached to each node.
- Checks whether a device entry exists for each SCSI device and creates device entries in the */dev* directory for the ones that do not exist.
- Finds unnecessary device entries in the */dev* directory and prompts for removing them. If you choose to remove them, the command cleans up all of the unnecessary device entries.
- Checks for RAID level 5 devices and prompts to convert them to RAID level 3. The command uses the **ace** utilities to do the low level formatting, which takes about 10 minutes to convert each RAID level 5 device to RAID level 3. This destroys any data on the RAID level 5 devices.

MAKEDEV *(cont.)* MAKEDEV *(cont.)*

Examples

1. To update device entries on a specific I/O node, do the following:

```
# cd /dev
# ./MAKEDEV 7
```

   This updates the device entries on the I/O node 7.

2. To update device entries for all the I/O node:

```
# cd /dev
# ./MAKEDEV
```

Files

/dev/MAKEDEV	Specifies the **MAKEDEV** command path.
/dev/makedevnode	Contains information used by the **MAKEDEV** command to build one set of entries.
/dev/checkraidlevel	Contains information used by the **MAKEDEV** command to check on the RAID level.
/etc/devtab	Updated by the **MAKEDEV** command for each additional I/O node.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

rmknod

MD MD

Creates dependencies in a makefile from dependency files (*.d* files) created by the **cc -MD** command.

Syntax

md [-dDfvx] [-m *file*] [**-u** *file*] *file ...*

Arguments

-d	Deletes each dependency file (*.d* files) after each file is processed.
-D	Specifies debugging (expert mode).
-f	Forces an update of the dependencies even though the makefile is more recent than each of the dependency files. This implies that **md** was already run. This is the default.
-m *file*	Specifies an alternate makefile to write the dependency information to. The file must exist. If **-m** is not specified a file named either *makefile* or *Makefile* must exist in the current directory.
-u *file*	Specifies a makefile to write the dependency information to. If the file exists, the dependency information is updated in the file. If the file does not exist, the file is created and the dependency information is written to the file.
-v	Specifies verbose mode so additional messages are displayed.
-x	Removes old dependency information from the makefile.
file	Specifies a dependency file created by the **cc -MD** command. The *file* argument must have the form *source_file.d*, where the *source_file* is the root name of the application's source file.

MD *(cont.)* **MD** *(cont.)*

Description

When you compile C source file with the **cc -MD** command, the compiler creates a dependency file that lists the pathnames of the include files specified in the source file. The compiler creates the dependency file with the name *source_file.**d*** where *source_file* is the root name of the source file.

The **md** command processes the dependency file, creates a list of dependencies, deletes repeated entries from the list, sorts the list, and fills each line to a 78 character line. For example, repeated entries that appear in the list of dependencies as *../dir1/../dir2* is reduced to *../dir2*. The **md** command updates a makefile with the dependency information (a makefile or an alternate makefile), so the dependency file can be deleted (**-d** switch). This is done to save space.

The **md** command assumes that dependency information is sorted by object file name (*.o* file), and merges in (adds or replaces) the new dependency lines. The **md** command assumes that any specified dependency files were created before the makefile and are processed already.

Examples

The following example creates a dependency file and uses the **md** command to create the dependencies in a makefile. This example assumes a makefile named *makefile* exists in the current directory.

```
% cc -MD -o big big.c
% md big.d
```

The following example shows the output of the **md** command:

```
% more makefile
# Dependencies for File: big.o:
big.o: /usr/include/allocsys.h /usr/include/mach/machine/vm_types.h
big.o: /usr/include/machine/machtime.h /usr/include/nx.h \
/usr/include/pfs/pfs.h
big.o: /usr/include/standards.h /usr/include/sys/estat.h
big.o: /usr/include/sys/mode.h /usr/include/sys/time.h \
/usr/include/sys/types.h
big.o: /usr/include/time.h big.c
```

MD *(cont.)* MD *(cont.)*

Files

/usr/mach/bin/md	Specifies the command file.
Makefile	Specifies a **make** description file.
makefile	Specifies a **make** description file.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

cc(1), make(1)

MKPART

Makes a new partition.

Syntax

mkpart [**-sz** *size* | **-sz** *h***X***w* | **-nd** *nodespec*] [**-nt** *nodetype*] [**-rlx**] [**-ss** | [[**-sps** | **-rq** *time*] [**-epl** *priority*]]] [**-mod** *mode*] *partition*

Arguments

-sz *size*	Creates a partition whose size (number of nodes) is *size*. The *size* argument must be an integer ranging from 1 up to and including the number of nodes in the parent partition. If you specify the **-sz** switch, the **mkpart** command attempts to create a square partition. If this is not possible, it attempts to create a rectangular partition that is either twice as wide as it is high or twice as high as it is wide. If this is not possible, **mkpart** uses any available nodes. In this case, the nodes allocated to the partition may not be contiguous.
-sz *h***X***w*	Creates a contiguous rectangular partition that is *h* nodes high and *w* nodes wide. You can use an uppercase or lowercase letter **X** between the integers *h* and *w*.
-nd *nodespec*	Creates a partition that consists of the nodes specified by the *nodespec* argument. The *nodespec* argument must be one of the following:

x	Node with the node number *x*.
x..*y*	Range of nodes from numbers *x* to *y*.
*h***X***w*:*n*	Rectangular group of nodes that is *h* nodes high and *w* nodes wide and whose upper left corner is node number *n*. (You can use an uppercase or lowercase letter **X** between the integers *h* and *w*.)
nspec[,*nspec*]...	Specified list of nodes, where each *nspec* is a node specifier of the form *x*, *x*..*y*, or *h***X***w*:*n*. Do not put any spaces in this list.

The numbers you use with **-nd** are node numbers within the parent partition, which always range from 0 to one less than the size of the partition.

NOTE

Do not specify the same node number more than once with the **-nd** switch.

-nt *nodetype* Creates a partition that consists of specific types of nodes only. The *nodetype* argument is one of the following:

attribute Used to select nodes having the specified attribute. For example, when *node_attribute* equals the string **mp**, only MP nodes are selected. The standard node attributes are shown in the "Node Attributes" section.

!*attribute* Used to select nodes *not* having the specified attribute. For example, when *node_attribute* equals the string **!io**, only nodes that are *not* I/O nodes are selected. Note that no white space may appear between the **!** and *node_attribute*.

[*relop*][*value*]*attribute*

Used to select nodes having a specified value or range of values for the attribute. For example, the string **>=16mb** selects nodes with 16M bytes or more of RAM. The string **32mb** selects nodes with exactly 32M bytes of RAM. And, the string **>proc** selects nodes with more than one processor.

The *relop* can be =, >, >=, <, <=, !=, or **!** (!= and **!** mean the same thing). If the *relop* is omitted, it defaults to =.

The *value* can be any nonnegative integer. If the *value* is omitted, it defaults to 1.

The *node_attribute* can be any attribute shown in the "Node Attributes" section, but is usually either **proc** or **mb**. (Other attributes have the value 1 if present or 0 if absent.)

No white space may appear between the *relop*, *value*, and *attribute*.

ntype[,*ntype*]...

Used to select nodes having *all* the attributes specified by the list of *ntype*s, where each *ntype* is a node type specifier of the form *node_attribute*, **!***node_attribute*, or [*relop*][*value*]*node_attribute*. For example, the string **32mb, !io** selects non-io nodes with 32M bytes of RAM.

You can use white space (space, tab, or newline) on either side of each comma, but not within an *ntype* argument.

-rlx

Relaxes the requirement that the exact specified number of nodes must be available for the partition to be created. The partition is created on the nodes that meet the requirements only (up to the number of requested nodes). There must be at least one available node that meets the requirements for the partition to be created. Otherwise, the partition is not created.

NOTE

The **-rlx** switch can be used to relax the default size, the **-sz** *size* switch, or the **-nd** switch. It cannot be used with the switch **-sz** *h***X***w*.

-ss

Creates a partition that uses standard scheduling. You cannot use the **-ss** switch with the **-sps**, **-rq**, or **-epl** switches.

If you don't use the **-ss**, **-sps**, **-rq**, or **-epl** switch, the new partition uses the same scheduling, rollin quantum, and effective priority limit as its parent partition.

-sps

Creates a partition that uses space sharing. Requests are denied that attempt to create overlapping partitions or run applications that overlap within the partition.

You cannot use the **-sps** switch with the **-ss** or **-rq** switches. You can use the **-sps** switch with the **-epl** switch. If you use **-sps** switch without the **-epl** switch, the partition is created as a space-shared partition with the same effective priority limit as the parent partition.

MKPART *(cont.)* MKPART *(cont.)*

-rq *time* Creates a gang-scheduled partition with a rollin quantum of *time*, where *time* is one of the following:

n	*n* milliseconds. If *n* is not a multiple of 100, it is silently rounded up to the next multiple of 100.
*n***s**	*n* seconds.
*n***m**	*n* minutes.
*n***h**	*n* hours.
0	"Infinite" time. Once rolled in, an application runs until it exits.

The maximum rollin quantum is 24 hours. The minimum rollin quantum is set by the system administrator.

You cannot use the **-rq** switch with the **-ss** or **-sps** switches. You can use the **-rq** switch with or without the **-epl** switch. If you use the **-rq** switch without the **-epl** switch, the new partition is a gang-scheduled partition with the same effective priority limit as its parent partition.

-epl *priority* Creates a partition that uses gang scheduling with an effective priority limit of *priority*. The *priority* argument is an integer from 0 (zero) to 10 inclusive (0 is low priority, while 10 is high priority).

You cannot use the **-epl** argument with the **-ss** switch. If you use the **-epl** switch without either the **-rq** or **-sps** switches, the results depend on the scheduling type of the parent partition as follows:

- If the parent partition is a space-shared partition, the new partition is a space-shared partition with the specified effective priority limit.
- If the parent partition is a gang-scheduled partition, the new partition is a gang-scheduled partition with the specified effective priority limit and the same rollin quantum as the parent partition. If this would exceed the maximum number of gang-scheduled partitions, the new partition is created as a space-shared instead.

MKPART *(cont.)* MKPART *(cont.)*

-mod *mode*	Specifies the partition's protection modes. You can specify the *mode* value either as a three-digit octal number with the form *nnn*, or as a nine-character string with the form `rwxrwxrwx`. See the *OSF/1 Command Reference* for more information about the **chmod** and **ls** commands.
partition	The absolute or relative partition pathname of the partition to be created.

Description

Use the **mkpart** command to create a partition. The specified partition cannot already exist, and the parent partition must exist and give you write permission.

The **mkpart** command lets you specify most of the partitions's characteristics. You use the **mkpart** command's switches to set specific characteristics for a partition. When you create a partition, you become the new partition's owner and the new partition's group is set to your current group. By default, a new partition gets the same characteristics as the parent partition.

If you use the **-sz**, **-nd**, or **-nt** switch to specify a partition's size and any nodes you specify are not available, the **mkpart** command fails with an error message and the partition is not created. A node's availability is determined by the parent partition's scheduling type and whether the node is currently in use. For example, if the partition does not permit overlapping subpartitions, any node that is already allocated to a subpartition is not available.

You can use the switch **-rlx** to relax the requirement that the exact specified number of nodes must be available. When you use **-rlx**, the new partition may consist of *fewer* nodes than you requested. In other words, the new partition consists of *as many nodes as possible*, *up to* the requested number of nodes. However, there must be at least one node available or the **mkpart** command still fails.

If you do not use the **-sz**, **-nd**, or **-nt** switch, **mkpart** attempts to allocate all the nodes of the parent partition to the new partition. If the parent partition contains unusable nodes, only the usable nodes are allocated to the new partition.

You can use at most one **-sz** or **-nd** switch in a single **mkpart** command. You can use **-nt** alone, or with **-sz** or **-nd**. If you use **-nt** without **-sz** or **-nd**, the new partition consists of all the nodes of the specified type in the parent partition. If you use **-nt** together with **-sz** or **-nd**, the new partition consists of the specified nodes of the specified node type; if the specified nodes are not all of the specified node type and you did not use the **-rlx** switch, the command fails (see the "Examples" section for more information).

Nodes are always numbered from 0 to one less than the partition's size. In most cases, they are numbered from left to right and then top to bottom as they are located in the partition. If you use the **-nd** switch, the nodes in the new partition are numbered in the order you specify them in the **-nd** switch.

No matter what the shape of the application, node numbers within the application (as returned by **mynode**()) will always be sequential from 0.

The **-sps** switch specifies that a partition uses space-sharing. Space-sharing prevents active partitions or active applications from overlapping each other. When you run an application in a space-sharing partition, the partition checks if another application is running on any of the requested nodes for your application. If any of the nodes are in use, your application fails immediately. However, if all of the requested nodes are available, your application starts immediately and runs to completion without interruption. When you attempt to create a subpartition using nodes that already belong to a subpartition of the space-shared partition, the attempt to create the subpartition fails immediately.

If a space-shared partition overlaps with another partition, the entire space-shared partition can be interrupted by applications running in the other partition. This can only occur if the space-shared partition's parent is a gang-scheduled partition.

NOTE

The boot node must be in the service partition. It is an error if the boot node is not in the *.service* partition.

NOTE

Creating a new *.service* partition will not take effect until after a reboot.

Node Attributes

The hardware characteristics of each node are described by a comma-separated series of strings called *attributes*. The following shows the most common node attributes. An attribute that is indented is a more specific version of the attribute from the previous level of indentation. For example, **net** and **scsi** nodes are specific types of **io** nodes; **enet** and **hippi** nodes are specific types of **net** nodes (and also specific types of **io** nodes).

MKPART *(cont.)* MKPART *(cont.)*

Attribute	Meaning
bootnode	Boot node.
gp	GP (two-processor) node.
mp	MP (three-processor) node.
mcp	Node with a message coprocessor.
*n***proc**	Node with *n* application processors (not counting the message coprocessor).
*n***mb**	Node with *n*M bytes of physical RAM.
io	Any I/O nodes.
net	I/O node with any type of network interface.
enet	Network node with Ethernet interface.
hippi	Network node with HIPPI interface.
scsi	I/O node with a SCSI interface.
disk	SCSI node with any type of disk.
raid	Disk node with a RAID array.
tape	SCSI node with any type of tape drive.
3480	Tape node with a 3480 tape drive.
dat	Tape node with a DAT drive.
IDstring	SCSI node whose attached device returned the specified *IDstring*. For example, a disk node might have the *IDstring* **NCR ADP-92/01 0304**.

Node attributes are not case sensitive, therefore, **GP**, **gp**, and **Gp** are equivalent.

NOTE

All nodes are **mcp** nodes, GP nodes are always **1proc**, and MP nodes are always **2proc**. MP nodes acting as GP nodes are **1proc**.

Examples

1. To create a partition called *mypart* that consists of every node in the *.compute* partition, enter the following:

```
% mkpart mypart
```

MKPART *(cont.)* MKPART *(cont.)*

If every node in the *.compute* partition is not available, this command will fail. You can relax this requirement using the **-rlx** switch as follows:

```
% mkpart -rlx mypart
```

If some or all nodes in the *.compute* partition are not available, this command will create a partition using as many nodes that are available in the *.compute* partition.

2. To create a partition called *mypart* whose parent partition is the *.compute* partition and has a size of 50 nodes, enter the following:

```
% mkpart -sz 50 mypart
```

The following command is functionally identical but uses an absolute partition pathname:

```
% mkpart -sz 50 .compute.mypart
```

3. To create a partition called *mypart* that is an 8 by 8 rectangle and uses space sharing, enter the following:

```
% mkpart -sz 8X8 -sps mypart
```

4. To create a partition called *mypart* that is a 10-node high by 5-node wide rectangle and is located in the upper left-hand corner of the parent partition (assuming node 0 of the parent partition is in fact situated in the upper left-hand corner), enter the following:

```
% mkpart -nd 10X5:0 mypart
```

This creates a partition that is a contiguous rectangle.

5. To create a 25-node partition of MP nodes called *mypart*, enter the following:

```
% mkpart -sz 25 -nt mp -rlx mypart
```

The **-rlx** switch makes sure the partition is created even though there may not be 25 MP nodes available. If there are fewer than 25 MP nodes available, the partition is created with the available MP nodes.

6. To create a partition called *mypart* that consists of all 32M-byte MP nodes, enter the following:

```
% mkpart -nt mp,32mb mypart
```

If there are no 32M-byte MP nodes available in the system, the command fails.

MKPART *(cont.)* MKPART *(cont.)*

7. To create a 50-node partition called *mypart* that consists of two-processor nodes, enter the following:

```
% mkpart -sz 50 -nt 2proc mypart
```

If there are not at least 50 two-processor nodes in the system, the command fails.

Errors

```
mkpart: Bad node specification
```

You specified a node number with the **-nd** switch that is greater than the largest node number in the partition. You specified an improperly-formatted *nodespec* with the **-nd** switch. You specified the same node number twice with the **-nd** switch.

```
mkpart: Exceeded allocator configuration parameters.
```

You specified too many gang-scheduled partitions. See the **allocator** manual page for information about the maximum number of gang-scheduled partitions.

```
mkpart: Exceeds partition resources
```

You specified a partition size with **-sz** *size* that is greater than the number of nodes in the parent partition. You specified a rectangle with **-sz** *h***X***w* that does not fit in the largest contiguous rectangle of nodes within the parent partition. You specified a partition with **-nd** or **-nt** where the nodes with the requested attributes are not available.

```
mkpart: Invalid priority
```

You specified a partition whose priority is not between 0 (zero) and 10.

```
mkpart: Partition permission denied
```

You specified a partition in a parent partition that does not give you write permission.

```
mkpart: Request overlaps with nodes in use.
```

You tried to create a partition that may overlap an existing partition in the *.compute* partition.

MKPART *(cont.)* MKPART *(cont.)*

```
mkpart: Scheduling parameters conflict with allocator configuration.
```

You specified a rollin quantum that is less than currently allowed. See the **allocator** manual page for information about the minimum rollin quantum.

Files

/usr/bin/mkpart	Specifies the command path.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: ***application***, **chpart**, **lspart**, **pspart**, **rmpart**, **showpart**

calls: **nx_mkpart()**, **nx_mkpart_rect()**, **nx_mkpart_map()**

OSF/1 Programmer's Reference: **chmod(1)**, **ls(1)**

MOUNT MOUNT

mount, **umount**: Mounts and unmounts file systems.

Syntax

mount [**-adfrwv**] [**-o** *options*] [**-u** [*options*]]
mount [**-dfrwv**] [**-o** *options*] [**-u** [*options*]]
device | *node* | *device node*
mount **-a**[**frwv**] [**-t nfs** | **ufs** | **pfs**] [**-o** *options*]
[**-u** [*options*]]
mount [**-frwv**] [**-t nfs** | **ufs** | **pfs**] [**-o** *options*]
[**-u** [*options*]] *device* | *node* | *device node*

umount **-a** | **-A** [**fFv**] [**-t nfs** | **ufs** | **pfs**]
umount [**-fFv**] [**-h** *host*] [**-t nfs** | **ufs** | **pfs**]
device | *node* | *device node*

Arguments: mount

-a Mounts all file systems listed in */etc/fstab*. When other flags are specified, mounts all file systems that meet the other criteria.

-d Mounts a **ufs** file system even if it has not been unmounted cleanly or checked by **fsck** for consistency.

-f Performs the actions of the mount without actually mounting any file systems. Use with the **-v** flag to determine what **mount** is trying to do.

-o *option* Mounts the file system with the specified options. Options entered on the command line override options listed in */etc/fstab.*

-r Mounts the file system as read-only.

-t *type* Mounts the file system type indicated by the *type* argument. The possible values for the *type* argument are the following:

nfs	Network File System (NFS)
pfs	Parallel File System (PFS)
ufs	UNIX File System (UFS) (Berkeley fast file system) (default)

MOUNT *(cont.)* MOUNT *(cont.)*

-u [*options*]	Changes the options for a mounted file system. The argument *options* is a comma-separated list of options. If you do not supply *options*, they default to **rw**, and **noquota**. See the "Options" section in this manual page.
-v	Displays a verbose description of **mount**'s actions.
-w	Mounts the file system as read-write.

Arguments: umount

-a Unmounts all file systems listed in */etc/fstab*.

-A Unmounts all the file system listed in the kernel's list of mounted file systems.

-f Forcibly unmounts a file system, even if it is busy. Any active special files continue to work; the special files are dissociated from the file system itself, so access times are no longer updated. Attempts to access all other files return errors. This flag does not support **nfs** file systems; you cannot forcibly unmount an **nfs** file system.

-F Performs the actions of the unmount without actually unmounting any file systems. Use with the **-v** flag to determine what **umount** is trying to do.

-h *host* Unmounts all **nfs** file systems that are listed in */etc/fstab* and have been mounted from *host*. Use with **-A** to unmount all file systems in the kernel's internal mount table.

-t *type* Unmounts file systems of the specified type. Use with **-a** to unmount file systems listed in */etc/fstab*. Use with **-A** to unmount file systems listed in the kernel's internal mount table. The possible values for the *type* argument are the following:

nfs	Network File System (NFS)
pfs	Parallel File System (PFS)
ufs	UNIX File System (UFS) (Berkeley fast file system) (default)

-v Displays a verbose description of **umount**'s actions.

MOUNT *(cont.)* MOUNT *(cont.)*

Description

The **mount** command mounts the specified file systems, attaching *device* to the directory hierarchy at *node*. With no arguments, **mount** displays the kernel's table of mounted file systems.

The **mount** command announces to the system that a removable file system is present on *device*. For **ufs** file system types, *device* must be a block special file. For **nfs** file system types, *device* is a remote file system and can be specified in two forms: *rhost:path* and *path@rhost*. For **pfs** file system types, *device* must be a block special file.

The file system on *device* is attached to the local mount point *node*. The *node* must exist; for **ufs** and **nfs** file system types, *node* must be a directory. Its path name becomes the path name to the root of the newly mounted file system. For the **pfs** file system type, *node* must be a directory.

If either *device* or *node* is not provided, the appropriate information is taken from the *fstab* file.

The system maintains a list of currently mounted file systems. If you invoke the **mount** command without options, it prints the list.

The **umount** command unmounts file systems. It announces to the system that the removable file system *node* or whatever removable file system was previously mounted on *device* should be removed.

Options

Options are specified as a comma-separated string of option names. Some options take a value; the syntax for these is *option=value*. When a file system is mounted, the options are determined by first reading any options listed in */etc/fstab*, then applying *options* specified with **-o** on the command line, then applying the **-r** or **-w** command-line flag. If the **-u** switch is supplied without any *options* arguments, the default options **rw**, and **noquota** are applied.

Some options apply to any file system type; some options apply only to one.

General Options

By default, file systems are mounted so that binaries can be executed, set-UID and set-GID bits are honored, special files are interpreted, and I/O is asynchronous. The options that apply to all file systems can be set with **-o** to override the defaults. They can be modified with **-u** to change the options for a mounted file system without unmounting it. For example, when you boot to single-user mode the root file system is mounted with read-only access. If you want to modify a file, you must change the options on the root file system to read/write.

MOUNT *(cont.)* MOUNT *(cont.)*

noexec	Does not allow execution of any binaries on the mounted file system. This option is useful for a server that has file systems containing binaries for architectures other than its own.
nosuid	Does not allow set-UID or set-GID bits to take effect.
nodev	Does not interpret character or block special files on the file system. This option is useful for a server that has file systems containing special files for architectures other than its own.
synchronous	All I/O to the file system should be done synchronously.

The **mount** command also recognizes the **exec**, **suid**, **dev**, and **asynchronous** *options*. These *options* can be used with **-o** on the command line to override an option in */etc/fstab*. They cannot be used with **-u** to change the state of a mounted file system.

NFS Options

The following *options* arguments apply to the NFS file system type:

hard	I/O system calls will retry until the server responds (default).
soft	I/O system calls will fail and return an error after **retrans** request retransmissions.
spongy	Uses soft semantics for the **stat**, **lookup**, **fsstat**, **readlink**, and **readdir** file system operations and hard semantics for the others. This option is meant to be similar to hard, except that processes will not be hung forever when they trip over mount points to dead servers.
bg	If the first mount request times out, retries in background. By default, **mount** runs in the foreground; the **bg** option allows **mount** to continue trying to mount remote file systems without hanging.
nointr	I/O system calls cannot be interrupted.
noconn	Does not connect the socket. This option is generally used with the UDP transport protocol when the server sends replies from sockets other than the **nfs** server socket.
rsize=#	Sets read size to # bytes. The default read size is 8192 bytes. The minimum read size is 512 bytes; the maximum is 8192 bytes. The read size can be set to values between the minimum and maximum.

MOUNT *(cont.)*

wsize=*#*	Sets write size to *#* bytes. The default write size is 8192 bytes. The minimum write size is 512 bytes; the maximum is 8192 bytes. The write size can be set to values between the minimum and maximum.
retry=*#*	Sets retry count to *#*. The default retry count is 10000
retrans=*#*	Sets retransmission count for nfs rpc's to *#*. The default retransmission count is 10.
timeo=*#*	Sets initial nfs timeout to *#* in 0.1-second intervals. The default initial timeout is 1 second.
attrtimeo=*#*	Sets the attribute cache timeout value to # seconds. The default is 5 seconds.

transport=*type*[^:^*net_address*]

Specifies the underlying transport protocol. The default is UDP. The valid *types* are **tcp**, **udp**, **udg**, **pip**, **spp**, **ssp**, and **idp**. For most transport protocols, the address of the server does not need to be specified here. The format of the address depends on the protocol.

tcp	A shorter form of **transport=tcp**.
udp	A shorter form of **transport=udp**.
share=*#*	Determines how many connections are made to a single server. The appropriate value depends on the underlying transport, the number of file systems being mounted from a server, and the NFS implementation on the server. The acceptable values are:

0	Each mounted file system has its own connection.
1	The default. All mounts from a given server share a connection.
>1	**mount** opens the specified number of connections for one file system. This option can be helpful is the server is single-threaded.

The **hard**, **soft**, and **spongy** options are mutually exclusive. The **mount** command also recognizes the **fg** (foreground), **conn**, and **intr** options. These options specify the defaults.

PFS Options

The following *options* arguments apply to the PFS file system type. These options are available when using the **-o** or **-u** flags with the **mount** command to mount a PFS file system:

stripeunit=*n* The stripe unit size for this PFS file system, in bytes; that is, the size of the unit of data interleaving for regular stripe files. To specify units other than bytes, append the appropriate (upper- or lower-case) letter to the *n* argument:

k	Kilobytes (1024 bytes)
m	Megabytes (1024K bytes)
g	Gigabytes (1024M bytes)

The default stripe unit size is 64K bytes.

stripedirs=*path1:path2:path3 ...*
A colon-separated list of pathnames that specifies the set of stripe directories that define the stripe group for this PFS file system. Each stripe directory is used for stripe file storage. The default value of the **stripedirs** option is the set of all available stripe directories as defined in the */etc/pfstab* file. (See the **pfstab(5)** manual page for a description of the */etc/pfstab* file.) All stripe directories must be specified as full pathnames; relative pathnames are not accepted. The **stripegroup** option can be used in place of this option.

stripegroup=*groupname*
A keyword from the */etc/pfstab* file that identifies the stripe group (see the **pfstab** manual page). The default stripe group is *all*, indicating that the set of all available stripe directories will be used for striping in the PFS file system. The **stripedirs** option can be used in place of this option.

svr_buffering Enable PFS server buffering. The fileservers cache stripe-file data in their memory-resident, disk-block caches. These fileservers use a read-ahead and write-behind caching algorithm. PFS buffering is recommended only when the IO request size is less than 64K bytes; otherwise, the fieservers's cache may thrash..

The *device* special file specified in the command line must refer to a disk partition on which a standard UFS file system has been built (see the OSF/1 **newfs** command manual page). This partition is used by PFS to store meta-information for regular stripe files, and to store files of other types such as directories and symbolic links. Data associated with regular stripe files is stored in the **stripedirs**.

With no arguments, the **mount** command displays a list of all currently mounted file systems. If a file system is of type **pfs**, the stripe unit size and stripe directories are also displayed.

Examples: mount

1. The following example mounts the device **/dev/rx2a** on the node **/usr/local/bin**:

   ```
   mount /dev/rx2a /usr/local/bin
   ```

2. The following example mounts all NFS file systems listed in **fstab** and takes *device*, *node* and options from the file:

   ```
   mount -at nfs
   ```

3. The following example shows how to test a **mount** command without actually mounting the file system:

   ```
   mount -fv users@server2 /users
   ```

   The example also shows one way of specifying a remote file system for *device*:

4. The following example changes the options on the root file system to read/write. This change is necessary if you want to modify files on systems booted to single-user mode. The root file system for systems booted in single-user mode are mounted with read-only access. Note that by not supplying an *options* argument with the **-u** switch the default options **rw** and **noquota** are applied.

   ```
   mount -u /
   ```

5. The following example unsets the **nosuid** option for the file system that is already mounted on the node **/arch1/bin**:

   ```
   mount -u nosuid /srch1/bin
   ```

6. The following example mounts an NFS file system with some options:

   ```
   mount -o soft,bg,retry=15 server2:/usr/local/man /usr/man
   ```

   You do not need to specify **-t nfs**. If *device* contains a colon or an ampersand, **mount** assumes an NFS file system type.

7. The following example mounts a PFS file system (residing on the device */dev/io1/rz0a*) on the */usr/pfs* directory:

   ```
   mount -t pfs /dev/io1/rz0a /usr/pfs
   ```

   This example uses the default stripe unit size and stripe group.

MOUNT *(cont.)* MOUNT *(cont.)*

8. The following example mounts a PFS file system, but specifies that the stripe unit size be 32 kilobytes and the stripes be interleaved into stripe files within the given set of directories:

```
mount -t pfs -o \
stripeunit=32k,stripedirs=/strp/vol0:/strp/vol1:/strp/vol2 \
/dev/io1/rz0a /usr/pfs
```

9. The following example mounts a PFS file system, but specifies the stripe group via the **stripegroup** option rather than the **stripedirs** option. This example assumes the */etc/pfstab* file is set up appropriately (see the **pfstab()** manual page).

```
mount -t pfs -o stripeunit=32k,stripegroup=left \
/dev/io1/rz0a /usr/pfs
```

Examples: umount

1. The following example unmounts all NFS file systems that are listed in */etc/fstab*:

```
umount -a -t nfs
```

2. The following example unmounts all file systems in the kernel's internal table of mounted file systems:

```
umount -A
```

3. The following example unmount all NFS file systems that have been mounted from the host **server3**:

```
umount -A -h server3
```

4. The following example unmounts the PFS file system mounted in the previous examples:

```
umount /dev/io1/rz0a
```

MOUNT *(cont.)* MOUNT *(cont.)*

Notes

File systems on physically write-protected disks must be mounted read-only or errors will occur when access times are updated. This holds whether or not an explicit write is attempted.

The root file system cannot be unmounted.

UFS file systems can be mounted with quotas enabled, but the appropriate options must be specified in */etc/fstab*. They cannot be entered on the **mount** command line.

The **umount** accepts the **-f** flag and passes it to the appropriate file system. File systems that do not support forcible unmounts return EINVAL.

Mounting corrupted file systems will crash the system.

Files

/etc/fstab	Table of file systems mounted at boot
/etc/pfstab	Table of PFS stripe groups

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

Calls: **fcntl(2)**, **getmntinfo(3)**, **getpfsinfo(3)**, **mount(2)**, **fattach(3)**, **fdetach(3)**

commands: **ls(1)**, **mount(8)**, **newfs(8)**, **showfs(1)**, **df(1)**

Files: **fstab(4)**, **pfstab(4)**

MS MS

Displays Mach kernel status.

Syntax

ms [**-v**] [**-t**|**-th**|**-h**|**-ps**|**-p**|**-tt**|**-ipc**] [**-n** *node_list*] [*mach_id* ...]

Arguments

-v	Causes more information to be displayed in some modes.
-t	Displays task status.
-th	Displays thread status.
-h	Displays host status, including the host's default processor set.
-ps	Displays processor set status.
-p	Displays processor status.
-tt	Displays the most useful task information, including some information about each thread in the task. This is the default.
-ipc	Displays IPC task status. If **-v** is also specified, the *machid* server displays a summary of IPC data structure usage on a host. In this mode, the *machid* server takes at most one privileged host *machid* argument on the command line.
-n *node_list*	Specifies the nodes for which information is returned. *node_list* is a list of one or more node numbers separated with white space.
mach_id	Specifies the ID for a Mach object. A Mach object can be one of the following: thread, task, host, processor, and processor sets.

MS *(cont.)* **MS** *(cont.)*

Description

The **ms** command displays general status information for the Mach kernel.

The **ms** command can display the status of threads, tasks, hosts, processors, and processor sets. If no Mach IDs are specified, the **ms** command displays all objects of the correct type. Otherwise, the **ms** command displays the specified objects, coercing the types if necessary.

For example, when displaying task status, a thread is converted to the task containing the thread. A processor set is converted to the tasks contained in the processor set. A host is converted to the tasks running on the host. When displaying host status, a task is converted to the host on which it is running.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

Files

/usr/mach/bin/ms
Specifies the command path.

See Also

machid, hostinfo

NETSTAT NETSTAT

Displays network statistics.

Syntax

netstat [**-Aan**] [**-f** *address_family*] [**-p** *protocol*] [*system*] [*core*]

netstat [**-imnrsutdH**] [**-f** *address_family*] [**-p** *protocol*] [*system*] [*core*]

netstat [**-ntd**] [**-I** *interface*] [*interval*] [*system*] [*core*]

Arguments

-a Displays the state of sockets related to the Internet protocol. Includes sockets for processes such as servers that are currently listening at a socket but are otherwise inactive.

-A Displays the address of any protocol control blocks associated with sockets. Typically, this flag is used for debugging.

-d Displays the number of dropped packets; for use with the **-I***interface* or **-i** flags. You can also specify an *interval* argument (in seconds).

-f *address_family*
Limits reports to the specified address family. The address families that can be specified might include the following:

inet Specifies reports of the AF_INET family, if present in the kernel.

unix Specifies reports of the AF_UNIX family, if present in the kernel.

ns Specifies reports of the AF_NS family, if present in the kernel.

all Lists information about all address families in the system.

-H Displays the current ARP table (behaves like **arp -a**).

-i Displays the state of configured interfaces. (Interfaces that are statically configured into the system, but not located at system start, are not shown.)

NETSTAT *(cont.)* NETSTAT *(cont.)*

-I *interface*	Displays information about the specified interface. The *interface* argument may be a string of the form *nameunit* (for example, **em0**) or <*node*>*nameunit* (for example, '**<11>em0**'). The *node* argument is the root partition node number where the network controller is installed. An interval argument must also be entered when you use the **-I** flag.
-m	Displays information about memory allocated to data structures associated with network operations.
-n	Displays network address in numerical format. When this flag is not specified, the address is displayed as hostname and port number. This flag can be used with any of the display formats.
-p*protocol*	Displays statistics for *protocol*, which you can specify as a well known name or an alias. Supported protocol names and their aliases are listed in **/etc/protocols.** A null listing (0) means that there is no data to report. If routines to report statistics for a specified protocol are not implemented on this system, **netstat** reports that the protocol is `unknown`.
-r	Displays the host's routing tables. When used with the **-s** flag, shows the host's routing statistics instead of routing tables.
-s	Displays statistics for each protocol.
-t	Displays timer information; for use with the **-I***interface* or **-i** flags.
-u	Displays information about domain sockets (UNIX domain).

Description

The **netstat** command displays network-related data in various formats.

When an *interval* argument is specified, **netstat** displays a table of cumulative statistics regarding packets transferred, errors, and collisions for an interface. The first line of data, and every 24th line thereafter, contains cumulative statistics from the time the system was last rebooted.

The *interval* argument overrides other flags, except for the **-n**, **-t**, and **-d** modifiers. The *interval* argument can be used with the *system* and *core* arguments, but it must precede them.

The *system* and *core* arguments cause **netstat** to derive statistics from the file *core* based on kernel file *system.* The default values for *system* and *core* are **/vmunix** and **/dev/kmem**, respectively.

Default Display

When used without flags, the **netstat** command displays a list of active sockets for each protocol. The default display shows the following items:

- Local and remote addresses
- Send and receive queue sizes (in bytes)
- Protocol
- State

Address formats are of the form *host.port* or *network.port* if a socket's address specifies a network but no specific host address. The host and network address are displayed symbolically unless **-n** is specified.

Interface Display

The network interface display format provides a table of cumulative statistics for the following:

- Interface name
- Maximum Transmission Unit (MTU)
- Network address
- Packets received (**Ipkts**)
- Packets received in error (**Ierrs**)
- Packets transferred (**Opkts**)
- Outgoing packets in error (**Oerrs**)
- Collisions

  Note that the collisions item has different meanings for different network interfaces.
- Drops (optional with **-d**)
- Timers (optional with **-t**)

NETSTAT *(cont.)* NETSTAT *(cont.)*

Routing Table Display

A route consists of a destination host or network and a gateway to use when forwarding packets. Direct routes are created automatically for each interface attached to the local host when you issue the **ifconfig** command. Routes can be modified automatically in response to the prevailing condition of the network.

The routing-table display format indicates available routes and the status of each in the following fields:

Flags	Displays the state of the route as one or more of the following:
U	Up, or available.
G	This route is to a gateway.
H	This route is to a host
D	This route was dynamically created by a redirect.
M	This route was modified by a redirect.
Refs	Gives the current number of active uses for the route. Connection-oriented protocols hold on to a single route for the duration of a connection; connectionless protocols obtain routes in the process of sending to a destination.
Use	Provides a count of the number of packets sent using the route.
Interface	Indicates the network interface used for the route.

Examples

1. To show the state of the configured interfaces, enter:

```
$ netstat -i
```

2. To show the routing tables, enter:

```
   $ netstat -r
```

The resulting display looks like the following:

```
Routing Tables
Destination Gateway     Flags  Refs Use         Interface

Netmasks:
Inet        255.255.255.0

Route Tree for Protocol Family 2:
default     555.555.5.5 UG     13   38618       se0
localhost   555.555.5.4        UH   2           29     lo0
ethernet    555.555.5.3        U    98          66760  se0
```

(Output may be formatted differently on your system.)

3. To produce the default display for network connections, enter:

```
   $ netstat
```

The resulting display might include the following headings:

```
Active Internet connections
Proto Recv-Q Send-Q Local Address       Foreign Address (state)
```

Related Information

Commands: **vmstat**(1).

NEWFS NEWFS

Constructs a new file system.

Syntax

newfs [**-N**] [*newfs-options*] *special disk-type*

Arguments

-N Causes the file system parameters to be printed out without really creating the file system.

The following *newfs-options* arguments define the general layout policies.

-b *block-size* The block size of the file system in bytes.

-f *frag-size* The fragment size of the file system in bytes.

-m *%free_space*
The percentage of space reserved from normal users; the minimum free space threshold (*minfree*). The default value is 10%. See **tunefs**(8) for more details on how to set this option.

-o *opt_preference*
The file system can either be instructed to try to minimize the *time* spent allocating blocks, or to try to minimize the *space* fragmentation on the disk. If the value of *minfree* is less than 10%, the default is to optimize for space; if the value of *minfree* is greater than or equal to 10%, the default is to optimize for time. See **tunefs**(8) for more details on how to set this option.

-a *maxcontig* This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (refer to the **-d** option). The default value is 1. See **tunefs**(8) for more details on how to set this option.

-d *rotdelay* This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. The default is 0 (zero) milliseconds. See **tunefs**(8) for more details on how to set this option.

-e *maxbpg* This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. The default is about one-quarter of the total blocks in a cylinder group. See **tunefs**(8) for more details on how to set this option.

NEWFS *(cont.)* NEWFS *(cont.)*

-i *#bytes/inode* This specifies the density of inodes in the file system. The default is to create an inode for every four fragments. For example, in a file system with a block size of 65536 bytes (64K bytes) and a fragment size of 8192 bytes (8K bytes), one inode is created for every 32768 bytes (32K bytes) of data space. If, instead you want one inode for every two fragments, you must specify **-i** 16384.

-c *#cylinders/group*
The number of cylinders per cylinder group in a file system. The default value is 16.

-s *size* The size of the file system in sectors.

The following options override the standard sizes for the disk geometry. Their default values are taken from the disk label. Changing these defaults is useful only when using **newfs** to build a file system whose raw image will eventually be used on a different type of disk than the one on which it is initially created (on a write-once disk, for example). Note that changing any of these values from their defaults makes it impossible for **fsck** to find the alternate superblocks if the standard superblock is lost.

-r *revolutions/minute*
The speed of the disk in revolutions per minute.

-S *sector_size* The size of a sector in bytes (.e.g. RAID3 = 2048, Maxtor =2048, and IPI-3 = 65536).

-u *sectors/track*
The number of sectors per track available for data allocation by the file system. This does not include sectors reserved at the end of each track for bad block replacement (see **-p**).

-t *#tracks/cylinder*
The number of tracks per cylinder available for data allocation by the file system.

-p *spare_sectors/track*
Spare sectors (bad sector replacements) are physical sectors that occupy space at the end of each track. They are not counted as part of the sectors per track (**-u**) since they are not available to the file system for data allocation.

-x *spare_sectors/cylinder*
Spare sectors (bad sector replacements) are physical sectors that occupy space at the end of the last track in the cylinder. They are deducted from the sectors per track (**-u**) of the last track of each cylinder since they are not available to the file system for data allocation.

NEWFS *(cont.)* NEWFS *(cont.)*

-j *number_of_fragments*

Sets the number of fragments allowed in an indirect block. The default for *number_of_fragments* in a file system block is four. The ability to set this number allows for indirect blocks that contain fragments to be smaller than file system blocks. This implies a smaller buffer cache and thus more available memory.

-l *hardware_sector_interleave*

Used to describe perturbations in the media format to compensate for a slow controller. Interleave is physical sector interleave on each track, specified as the denominator of the ratio: sectors read/sectors passed over.

Thus, an interleave of 1/1 implies contiguous layout, while 1/2 implies logical sector 0 (zero) is separated by one sector from logical sector 1.

-k *sector0-skew/track*

Used to describe perturbations in the media format to compensate for a slow controller. Track skew is the offset of sector 0 (zero) on track N relative to sector 0 (zero) on track N-1 on the same cylinder.

Description

The **newfs** command creates a new file system on the specified *special-device* of type *disk-type*. If the disk has been labeled using **disklabel, newfs** builds a file system on the specified device, basing its defaults on the information in the disk label. If the disk has not been labeled using **disklabel**, **newfs** looks up the specified disk type in the disk description file */etc/disktab* to get default information on the specified special device. Typically the defaults are reasonable, however **newfs** has numerous options to allow the defaults to be selectively overridden.

You must be root to use this command.

The specified special device should either be previously labeled using **disklabel(8)** or its disk-type must be in */etc/disktab*. If the block-size and/or the size of the file system is specified, the specified values override what is in the disklabel or in */etc/disktab*.

If no prototype file is specified, **newfs** builds a file system with a single empty directory on it. The boot program block (block zero) is left uninitialized.

NEWFS *(cont.)* NEWFS *(cont.)*

If a prototype file is specified, **newfs** takes its directions from that file. The prototype file contains tokens separated by spaces or new-lines. A sample prototype specification follows; line numbers have been added to aid in the explanation:

```
1 proto_file
2 512 4872 110
3 d--777 3 1
4 usr d--777 3 1
5 sh ---755 3 1 /sbin/sh
6 jmp d--755 6 1
7 $
8 b0 b--644 3 1 0 0
9 c0 c--644 3 1 0 0
10 $
11 $
```

Historically line 1, as shown in the preceding example, is the name of a file to be copied onto block zero as the bootstrap program. Since operating system does not support using the System V file system as a root file system, this file name is ignored by **newfs** and no data is copied onto block zero of the specified special device.

Line 2 specifies the block size of the file system in bytes, the size of the file system in number of blocks of the just specified block size and the number of inodes in the file system.

Lines 3 to 9 tell **newfs** about files and directories to be included in this file system. Line 3 specifies the root directory. Lines 4 to 6 and 8 to 9 specify other directories and files. The $ on line 7 tells **newfs** to end the branch of the file system it is on, and continue from the next higher directory.

The **$** on lines 10 and 11 end the process, since no additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is **-bcd** to specify regular, block special, character special and directory files, respectively. The second character of the mode is either **u** or **-** to specify set-user-id mode or not. The third is **g** or **-** for the set-group-id mode. The rest of the mode is a 3-digit octal number giving the owner, group, and other read, write, execute permissions. Refer to **chmod(1)** for additional information.

NEWFS *(cont.)*   NEWFS *(cont.)*

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file. If the file is a regular file, the next token of the specification may be a path name from which the contents and size are copied. If the file is a block or character special file, two decimal numbers follow which give the major and minor device numbers. If the file is a directory, **newfs** makes the entries **.** (dot) and **..** (dot dot) and then reads a list of names and file specifications recursively for the entries in the directory. As noted previously, the scan is terminated with the token **$**.

FILES

/usr/sbin/newfs	Specifies the command path
/etc/disktab	Provides disk geometry and file system partition information

See Also

Commands: **chmod(1)**, **disklabel(8)**, **fsck(8)**, **tunefs(8)**

Files: **disktab(4)**, **fstab(4)**

NQS NQS

Supports batched job requests for the Intel Supercomputer network.

Description

The Network Queueing System (NQS) is an industry-standard queueing system that provides services that let you can submit jobs from a local or remote workstation for later execution on the Intel supercomputer system.

Manual pages are provided online and in the *Paragon™ System Network Queueing System Manual* for the following NQS commands:

nmapmgr	Invokes the map manager program. Allows the configuration and management of the net map used by NQS.
qcmplx	Displays the status of NQS complexes.
qdel	Deletes requests from the specified queue. You can also send an OSF/1 signal to any request that has already started to run.
qdev	Displays the status of NQS devices.
qlimit	Displays the supported resource limits and the shell strategy for a specified workstation.
qmgr	Invokes the NQS manager program. This is an interactive program that allows you to define, configure, and manage queues.
qpr	Queues user files for printing.
qstat	Displays queue status.
qsub	Submits requests to queues.

Refer to the *Paragon™ System Network Queueing System Manual* for detailed user, installation, and reference information on NQS.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

NQS *(cont.)* NQS *(cont.)*

See Also

Paragon™ System Network Queueing System Manual

NTPD NTPD

Network Time Protocol time synchronization daemon.

Synopsis

/usr/sbin/ntpd [**-a** *threshold*] [**-c** *file*] [**-d**] [**-D** *level*] [**-l**] [**-s**] [**-t**]

Options

-a *threshold*	Sets the threshold that limits how far **ntpd** will change the system clock. Its used as a sort of ultimate sanity check to prevent your system time from being changed a great deal. By default, the threshold is 1000 seconds. threshold is specified in units of seconds, or the string any to defeat the sanity check.
-c *config-file*	Specifies the location of the ntpd configuration file. By default, */etc/ntp.conf* is used.
-d	Increments the debug level by one. May be specified more than once to increment debug level by one each time.
-D *level*	Sets the debug level to the value specified.
-l	Causes **ntpd** to log a message each time the logical clock is changed. Normally, you would not specify this option unless you wanted to gather statistical information to analyze the logical clock behavior. If the **-l** option is specified, a message will be logged approximately every 2 minutes.
-s	Causes **ntpd** to never adjust the local clock.
-t	Causes **ntpd** to modify the value of **tickadj** in your kernel.

Discussion

ntpd is the network time synchronization daemon and is invoked at boot time. It implements a new revision of the Network Time Protocol first described in RFC-958. It maintains the host's time synchronized with a set of distributed time servers, each with varying accuracy and reliability. Multiple time server masters may exist, but there is no requirement for election of a single master.

NTPD *(cont.)* NTPD *(cont.)*

ntpd uses the **adjtime (2)** system call to slew the clock of the host by small amount in order to keep the clock synchronized. If the local clock exceeds the "correct" time by some threshold, then settimeofday (2) is used to make a step adjustment of the local clock.

When **ntpd (8)** is started on the machine, it reads configuration information from the */etc/ntp.conf* file, which contains information about other **ntp** time servers and host specific information. Configuration information is listed one entry per line, with fields separated by whitespace. Lines which begin with a "#" character are treated as comments. Here is a sample configuration file:

```
# Local clock parameters
#
# Precision of the local clock to the nearest power of 2
# ex.
#60-HZ   = 2**-6
#100-HZ  = 2**-7
#1000-HZ = 2**-10
precision -7
#
tickadj 5
#
peer        foo.umd.edu
peer        192.5.39.94
peer        bar.arpa
server      bogon.umd.edu
passive     bozo.umd.edu
#
# Configure a reference clock.
#           device      refid  stratum  precision  type
#          -------     -----   -------  ---------  ----
peer       /dev/tty03   WWV      1         -5      psti
#peer       /dev/null    LOCL     1         -5       local
```

There are two major types of information specified in the configuration file: local host information, and remote timer server specification. The local host information is used to describe the intrinsic properties of the local host's timekeeping machinery. The commands in this group are **precision** and **tickadj**.

The **precision** command takes a number which describes the resolution of the local clock, as a power of two. For example, a Paragon system typically has a 100 HZ clock and thus a **precision** of -7. If the symbol **hz** is defined in the namelist of */vmunix*, this value is automatically set based on the value of **hz**.

The **tickadj** command is used to specify the granularity of clock adjustment done by the **adjtime(2)** system call. If the **-t** option is specified when **ntpd** is invoked, the kernel variable **_tickadj** is modified via */dev/kmem*.

NTPD *(cont.)* NTPD *(cont.)*

The preferred method of setting **tickadj** is by changing the value in the kernel file *conf.c* instead of having **ntpd** set in this rude fashion.

The **driftfile** command can be used to specify the name of the file that the drift compensation register will be loaded from at initialization time and that updated values will be written into. The drift compensation value describes the intrinsic drift of your host's clock. By default, the file */etc/ntp.drift* will be used.

Three time server specifications are supported. They are **peer, server** and **passive**.

Each command takes either a dotted-quad internet address or a host name. Each host specified in any one of the three commands is eligible to be synchronized to, while random hosts which set up a peer relationship are not. The **peer** and **server** commands create an active polling situation; in the case of **peer**, the NTP packets are sourced in Symmetric-Active mode, while using **server** causes the packets to be in Client mode. When reachability is lost with a configured host in either of these two cases, the daemon will continue to poll to reacquire that host. A host specified using the **passive** command will not continue to be polled. If that host begins to poll us, it will be eligible as to be synchronized but will not be polled if reachability is lost.

It is recommended that the bulk of the peers configured should be specified with the **client** keyword; this will minimize resource usage on the remote NTP server. If your host will be serving as a redistribution point for a cluster of hosts, you should set up peer relationships with higher quality clocks (lower stratums) and other equal stratum clocks. In other words, if you are not redistributing time to others, you shouldn't need to configure any peers in your NTP configuration; client specifications are more appropriate.

To configure a reference clock, you should use something like the previous example. The first field after the peer keyword is the name of the file that the clock is connected to. This must be a complete path name with a leading slash (/) character. The next field is the reference-id that will be inserted into the packets generated from this NTP daemon. For a PSTI clock, this should be WWV. The next field is the clock. Actually, it is really the stratum that will be placed in the packet if this clock is selected by the local NTP daemon as the reference clock. Following that is the precision that will be inserted into the packet when this clock is selected. The final field is the type of the clock. Two types are supported: **psti** for the Precision Standard Time, Inc WWV clock (RIP) and **local** for the local time of the system. The local type of clock can be used to declare one host in an isolated network as having the "correct" time and then the other hosts on that network can synchronized to it.

NTPD *(cont.)* NTPD *(cont.)*

Files

/etc/ntp.conf	NTP daemon configuration file

See Also

adjtime(2), settimeofday(2)

RFC-958, *Network Time Protocol (Version 2) Specification and Implementation, Revised 15 April 1988*, David L. Mills, University of Delaware

PARAGRAPH PARAGRAPH

Invoke ParaGraph, a performance visualization tool for Paragon applications.

Syntax

paragraph [**-m** | **-s** | **-p**] [**-f** *filename*] [**-e** *environment_file*]
[*X Toolkit parameters*]

Arguments

-m
Forces monochrome display mode. This is useful for making black-and-white hardcopies from a color screen.

-s
Forces ParaGraph to allocate read-only colorcells from the default colormap. By default, ParaGraph attempts to allocate read/write colorcells. This allows you to change the colors used within the displays interactively. However, read/write colorcells can not be shared by different applications and are thus a limited resource. If you use the **-s** option, ParaGraph's colors can not be edited and the Colors entry in the options menu is disabled.

-p
Forces ParaGraph to allocate read/write colorcells from a private colormap. Use this option if not enough colorcells can be allocated from the default colormap because they have been used up by other applications.

-f *filename*
Specifies the name of a trace file that contains previously-saved performance data in the Paragon SDDF trace format.

-e *environment_file*
Specifies the name of a file containing a layout environment produced through the Save Layout command.

X Toolkit parameters
The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*).

Description

ParaGraph is a graphical display system for visualizing the behavior and performance of parallel programs on the Paragon multiprocessor. It takes as input execution profile data produced by the Paragon performance monitoring subsystem. The performance monitoring subsystem produces an execution trace during an actual run of a parallel program on a message- passing machine, and the resulting trace data can then be replayed pictorially with ParaGraph to provide a dynamic depiction of the behavior of the parallel program.

PARAGRAPH *(cont.)* PARAGRAPH *(cont.)*

ParaGraph provides several distinct visual perspectives from which to view the same performance data, in an attempt to gain insights that might be missed by any single view. ParaGraph is based on the X Window System and the Motif toolkit. Although ParaGraph is most effective in color, it also works on monochrome and gray-scale monitors. The user interface for ParaGraph is menu-oriented, with most user input provided by mouse clicks, although for some features keyboard input is also supported. ParaGraph provides a dynamic depiction of the parallel program while also providing responsive interaction with the user. Menu selections determine the execution behavior of ParaGraph both statically (e.g., initial selection of parameter values) and dynamically (e.g., pause/resume, single-step mode). As a further aid to the user, ParaGraph determines some relevant parameters automatically (e.g., time scale, number of processors) before the graphical simulation begins.

ParaGraph currently provides about 25 different displays or views, all based on the same underlying trace data, but each giving a distinct perspective. Some of these displays change dynamically in place, with execution time in the original run represented by simulation time in the replay. Other displays depict time evolution by representing execution time in the original run by one space dimension on the screen. The latter displays scroll as necessary (by a user-controllable amount) as simulation time progresses. The user can view as many of the displays simultaneously as will fit on the screen, and all visible windows are updated appropriately as the tracefile is read. The displays can be resized within reasonable bounds.

The ParaGraph main window provides the following features:

Title Bar	Lists the title of the tool and the name of the current tracefile.
Menu Bar	Provides a set of pulldown menus to access all the features of ParaGraph.
Button Panel	Contains three push buttons that control the features of the visualization.

When ParaGraph is invoked without the -f option, the push buttons in the button panel are insensitive (can not be used). To begin a ParaGraph session select the Open... menu item from the File menu. This will pop up a Motif File Selection box. You should now select the tracefile you wish to visualize. This will cause ParaGraph to open the file and display the file name in the title bar. The push buttons will now be sensitive.

ParaGraph displays can be selected from any of the middle menus (Utilization, Communication, Task and Other). Pressing a button in any of these menus will bring the corresponding display up. Displays can be removed from the screen by pressing the Close button (provided by the window manager) or by pressing the corresponding menu button again.

Once you have selected a tracefile, you can start the simulation by pressing the Start button. As a result, the button label will change to Pause. The visualization can be suspended by pressing this button again, which changes the button label to Resume. The Reset button is used to go back to the beginning of the trace file. The Step button can be used to step through the trace events one (or a few) at a time.

PARAGRAPH *(cont.)* PARAGRAPH *(cont.)*

Further information on how to use ParaGraph is available through ParaGraph's on-line help facility. To learn how to use the help facility, select On Help from the Help menu. ParaGraph provides context-sensitive help, so users can get help related to a particular ParaGraph activity by pointing and clicking at that part of the screen after selecting On Context from the Help menu.

Resources

You can configure ParaGraph by using an X resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *Paragraph* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *Paragraph* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following ParaGraph application resources are provided to configure ParaGraph:

Paragraph*ScrollValue

This resource determines whether ParaGraph displays scroll smoothly or jump scroll by a user-specified amount. The resource parameter is Smooth, Jump1/8, Jump1/4, Jump1/2 or Jump1.

Pargraph*ScaleWidth

This resource determines the number of pixels used to depict one time unit. The resource parameter is 1, 2, 4, 8 or 16.

Paragraph*BackingStore

This resource determines whether backing store is used in the ParaGraph displays. The resource parameter is on or off.

Paragraph*StepIncrement

This resource determines how many consecutive records from the tracefile are processed each time the step button is pressed. The resource parameter is a positive integer value.

Paragraph*helpfile

This resource determines the name of the help file used by ParaGraph's help utility.

Paragraph*font1

The font used to label the inside of the displays. This should be a fixed size font.

Paragraph*font2

The font used to label the Animation, Kiviat, Clock and Task Status displays. It should be a fixed size font that is bigger than font1.

Paragraph*font3

The font used to label the Animation and Topology displays for large node numbers. It should be a fixed size font that is smaller than font1.

PARAGRAPH *(cont.)* PARAGRAPH *(cont.)*

Paragraph*MPsupport
The resource used to turn ParaGraph's multi-process support on and off.

Paragraph*fgColor
The foreground color of the ParaGraph displays.

Paragraph*bgColor
The background color of the ParaGraph displays.

Paragraph*busyColor
The color used to depict the busy state in the Utilization displays and the Animation and Topology displays.

Paragraph*ovhdColor
The color used to depict the overhead state in the Utilization displays and the Animation and Topology displays.

Paragraph*idleColor
The color used to depict the idle state in the Utilization displays and the Animation and Topology displays.

Paragraph*ioColor
The color used to depict the I/O state in the Utilization displays and the Animation and Topology displays.

Paragraph*flushColor
The color used to depict the flush state in the Utilization displays and the Animation and Topology displays.

Paragraph*sendColor
The color used to depict the sending state in the Animation and Topology displays.

Paragraph*recvColor
The color used to depict the receiving state in the Animation and Topology displays.

Paragraph*trafColor
The color used to draw the queue values in the Communication Traffic display.

Paragraph*lgnd1Color through **Paragraph*lgnd5Color**
The colors used to color code message lengths in the color code legend display.

Paragraph*msgqColor
The color used to draw message queue values in the Communication Queues display.

PARAGRAPH *(cont.)* PARAGRAPH *(cont.)*

Paragraph*msghColor
The color used to draw message queue values high water mark in the Communication Queues display.

Paragraph*nwk1Color through **Paragraph*nwk7Color**
The colors used to color code the number of messages that travel across links in the Communication Network display.

Paragraph*clockColor
The color used to draw the time bar in the clock display.

Paragraph*bkbgColor
The color used to draw started tasks in the Task Status display.

Paragraph*bkedColor
The color used to draw finished tasks in the Task Status display.

Paragraph*kivtColor
The color used to draw utilization values in the Utilization Kiviat display.

Paragraph*kivhColor
The color used to draw the utilization high water mark in the Utilization Kiviat display.

Paragraph*task1Color through **Paragraph*task63Color**
The colors used to color code task numbers, message types and message distances.

Files

ParaGraph uses the following files:

Paragraph	Application defaults file for ParaGraph. This file defines resources that control the appearance and configuration parameters of ParaGraph.
Paragraph.hlp	Online help text for ParaGraph.
.pgrc	Defines the initial state of ParaGraph upon invocation, including open displays, screen position, and display colors.

PARAGRAPH *(cont.)* PARAGRAPH *(cont.)*

See Also

paraide, **xprof**, **xgprof**, **xipd**, **spv**

traceblockbegin(), **traceblockend()**

Paragon™ System Application Tools User's Guide

## PARAIDE	PARAIDE

Invoke **ParAide**, the graphical front end for tools in the Paragon application toolset.

Syntax

paraide [-**[no]menus**] [-**[no]icons**] [-**[no]shell**]
[**-rows** *minrows*] [**-cols** *mincols*] [*X Toolkit parameters*]

Arguments

-[no]**menus**	[Do not] display the menu bar. **-menus** is the default.
-[no]**icons**	[Do not] display the icon strip. **-icons** is the default.
-[no]**shell**	[Do not] display the shell panel. **-shell** is the default.
-rows *minrows*	Sets the minimum number of rows for the shell panel scrolling text area. The number of rows might be greater than *minrows* when the main window is created.
-cols *mincols*	Sets the minimum number of columns for the shell panel scrolling text area. The number of columns might be greater than *mincols* when the main window is created due to the text area being forced to stretch across the window.
X Toolkit parameters	The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*).

Description

ParAide provides a front end to the graphical tools in the Paragon application toolset and allows you to load a program into the mesh and open editor windows. The **ParAide** main window provides the following features:

Title Bar	Lists the title of the tool.
Menu Bar	Provides a set of pulldown menus to access all the features of ParAide.
Icon Strip	Contains icons for the graphical application tools in the Paragon toolset. You can launch any of these tools from **ParAide**. The tools include **XProf**, **XGprof**, **XIPD**, **ParaGraph**, and **SPV**.
Shell Panel	Contains a user shell to execute programs loaded into the mesh. Commands and output text appear in a scrolling text region.

Resources

You can configure **ParAide** by using a resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *Paraide* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *Paraide* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following application resources are provided to configure **ParAide**:

Paraide.showMenus
A boolean (True / False) value that controls creation of the menu bar for the **ParAide** main window

Paraide.showIconStrip
A boolean (True / False) value that controls creation of the icon strip for the **ParAide** main window.

Paraide.showShell
A boolean (True / False) value that controls creation of the shell panel for the **ParAide** main window

Paraide.shellRows
An integer that defines the minimum number of rows the shell panel scrolling text region should have.

Paraide.shellColumns
An integer that defines the minimum number of columns the shell panel scrolling text region should have.

Paraide.nodeSize
An integer that defines the size (in pixels) of the nodes drawn on the mesh.

Paraide.nonPartitionNodeColor
A color name that defines the color for mesh nodes that don't belong to the currently selected partition. Also used to color nodes that haven't been loaded.

Paraide.partitionNodeColor
A color name that defines the color for mesh nodes that belong to the currently selected partition. Also used to color nodes that have been selected for loading.

Paraide.partitionSelectedNodeColor
A color name that defines the color for mesh nodes that have been selected to be loaded into. Also used to color nodes loaded with a program.

Paraide.nonPartitionNodeStipple
A stipple pattern name that defines the stipple to be used with nodes that don't belong to the currently selected partition. Also used to render nodes that haven't been loaded.

Paraide.partitionNodeStipple
A stipple pattern name that defines the stipple to be used with mesh nodes that belong to the currently selected partition. Also used to render nodes that have been selected for loading

Paraide.partitionSelectedNodeStipple
A stipple pattern name that defines the stipple to be used with mesh nodes that have been selected to be loaded into. Also used to render nodes loaded with a program.

Environment Variables

EDITOR Name of the editor program to execute when a file is to be edited (from *New*, *Open...*, or from the open history dialog). If not defined, **ParAide** executes the *vi* program.

TGI_EDITOR Name of the editor program that overrides the contents of *EDITOR*. This is useful for defining an editor to be used specifically with **ParAide**.

NX_DFLT_PART
Name of the default partition ParAide selects within the partition selection dialog. If not defined, **ParAide** defaults to selecting the *.compute* partition.

SHELL The shell program to execute within the shell panel.

Files

ParAide uses the following files:

Paraide Application defaults file for ParAide. This file defines resources that control the appearance and configuration parameters of **ParAide**.

paraide.hlp Online help text for **ParAide**.

See Also

xprof, **xgprof**, **xipd**, **paragraph**, **spv**

Paragon™ System Application Tools User's Guide

PARSEMAGIC PARSEMAGIC

Diagnostic station: Returns values from the bootmagic file on the diagnostic station.

Syntax

parsemagic [**-dlLnstv**] [**-b** *letter*] [**-B** *file*][**-c** *config*][**-m** [*mode*]]

Arguments

-B *file*	Specifies the pathname for the bootmagic file. The default is the bootmagic file in the current directory.
-b [*letter*]	Specifies the backplane to use for a condo configuration. The letter argument can be **A**, **B**, **C**, or **D**, and can be specified in upper or lower case. The default is **D**.
-c [*config*]	Specifies the system configuration. The values for the *config* argument are **condo**, **full**, and **multi**. The default is **full**.
-d	Displays a description of the system.
-l	Returns a list of nodes with ranges expanded. This means node specifications in the format 3...7 are displayed as 3 4 5 6 7.
-L	Returns a list of mesh routing chips (MRCs) with ranges expanded. This means MRC specifications in the format 3...7 are displayed as 3 4 5 6 7.
-m [*mode*]	Used by the **reset** script to get information about the MRCs in a backplane. The *mode* parameter can be one of the following:

fast	Fast streaming mode.
slow	Slow streaming mode.
interlocked	Interlocked mode.

Only the first character of the *mode* parameter is examined.

-n	Displays the number of cabinets in the system.
-s	Displays the slot number of the boot node.

PARSEMAGIC *(cont.)* PARSEMAGIC *(cont.)*

-t	Tests the *BOOT_CONSOLE* string to see if it supports the scan interface. Returns 1 if it supports the scan interface, otherwise it returns 0 (zero).
-v	Verifies the existence of the bootmagic file. Returns 1 if the file exists and is readable. Returns 0 if the file is missing or is unreadable.

Description

The **parsemagic** command runs on the diagnostic station and is intended for use by the system administrator only.

The **parsemagic** command reads values from the bootmagic file and returns the values to a shell program. This command fails if there is no bootmagic file in the current directory or if you do not specify a bootmagic file pathname.

Files

/usr/local/bin/parsemagic	Contains executable for the **parsemagic** command.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

cbs, reset

PFSCK PFSCK

Identifies and repairs PFS file system inconsistencies.

Syntax

pfsck [**-DKhqv**] *pfs_filesystem_name* [*application_arguments*]

Arguments

-D	Enable verbose debug operation mode
-K	Remove orphaned stripe files
-h	Show help text
-q	Enable quiet operation mode
-v	Enable verbose operation mode
pfs_filesystem_name	Fully-specified pathname (from the mount point) of the mounted PFS file system to be checked. Note that only a mounted file system can be checked. Note also that *pfs_filesystem_name* cannot be the name of a symbolic link.
application_arguments	Any arguments described in the NX *application* manual page.

PFSCK *(cont.)* PFSCK *(cont.)*

Description

Use **pfsck** (PFS File System Checker) to identify and optionally repair the following PFS file system inconsistencies:

- *Orphaned PFS stripe files.* These are stripe data files that are not referenced by any PFS file. **pfsck** lists orphaned files in the file */usr/tmp/orphans**pfs_filesystem_name***, replacing any slash characters in *pfs_filesystem_name* with period characters. For example, if *pfs_filesystem_name* were `/pfs`, the orphaned files list would be in the file `/usr/tmp/orphans.pfs`.

- *Bad PFS files.* These are PFS files that reference one or more stripe data files that do not exist. **pfsck** lists bad files in the file */usr/tmp/pfs-files.bad**pfs_filesystem_name***, replacing any slash characters in *pfs_filesystem_name* with period characters. For example, if *pfs_filesystem_name* were `/pfs`, the orphaned files list would be in the file `/usr/tmp/pfs-files.bad.pfs`.

Note that repair (i.e., the removal of any orphaned stripe files) is performed only if explicitly requested via the **-K** argument. Note also, the **pfsck** only *lists* the bad PFS files. Typically, you would either explicitly remove bad files (using **rm**) or try to restore the missing PFS stripe files (from a previous backup or other archive).

NOTE

On a large PFS (many files, many stripes), the **pfsck** command may take an hour or more to complete. Consequently, **pfsck** should be done as part of your system PM process, *not* as part of the boot process.

In addition, the PFS file system must be mounted, but it must *not* be in use at the time it is being checked. A PFS file system that is in use may return erroneous inconsistencies.

Finally, because **pfsck** is an NX parallel application, **pfsck** will not run unless the system is "ready and able" to run parallel applications.

## PFSCK *(cont.)*		PFSCK *(cont.)*

Examples

1. Run **pfsck** using 121 nodes (1 controlling node and 120 worker nodes) to check the PFS file system named "/pfs." Orphaned stripe files are listed in the file */usr/tmp/orphans.pfs* and bad PFS files are listed in */usr/tmp/pfs-files.bad.pfs*. No stripe files are deleted.

```
# /usr/sbin/pfsck /pfs -pn .compute -sz 121
```

2. Delete any orphan files found in the "/cfs" PFS file system:

```
# /usr/sbin/pfsck -K /cfs -pn .compute -sz 17
```

3. Same as above, but do it quietly:

```
# /usr/sbin/pfsck -K -q /cfs -pn .compute -sz 17
```

Files

/usr/sbin/pfsck	The **pfsck** command.
*/usr/tmp/orphans**pfs_filesystem_name***	The list of orphaned stripe files.
*/usr/tmp/pfs-files.bad**pfs_filesystem_name***	The list of bad PFS files.

Limitations And Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

application

PGS

Show system-wide Virtual Memory (VM) paging file statistics.

Syntax

pgs [**-dvkph**] [**-n** *node*] [*interval*]

Arguments

-n *node*	Show statistics only for the specified node.
-d	Show only the boot-node default pager statistics.
-v	Show only vnode pagers statistics.
-k	Paging file units in K-bytes (default).
-p	Paging file units in VM pages.
-h	Usage information.
interval	Specifies the interval in seconds to show the statistics. If this argument is specified, the statistics are updated and shown every interval seconds.

Description

The **pgs** command shows the following paging file information:

Node Paging file	*Paging file name*: If the paging file is the boot-node default paging file then */mach_servers/paging_file* is shown, otherwise the vnode paging device name is listed.
Total	*Allocated file size*: Expressed in K-bytes (default) or VM pages (**-p**).
Free	*Free space*: Expressed in current output units (K-bytes/Pages)
Used	*Used space*: Current output units.
(%)	*Percentage utilization*: Percent of total space in use.
pgin	*Page-in requests*: Count of pagein requests.
pgout	*Page-out requests*: Count of pageout requests.

PGS *(cont.)* PGS *(cont.)*

Examples

- Show system wide paging file statistics:

```
# /usr/mach/bin/pgs

Node Paging file   (in Kbytes)   Total    Free    Used  (%)    pgin  pgout
------------------------------------------------------------------------
   7 /dev/io0/rz0b            1048576 1048568       8  0.00       0      0
   7 /mach_servers/paging_file  524288  475976   48312  9.21  68939  58287
 124 /dev/io5/rz0m             386096  384800    1296  0.34       2    161
 122 /dev/io4/rz0m             386096  384784    1312  0.34       6    163
   5 /dev/io3/rz0m             386096  386088       8  0.00       0      0
   3 /dev/io2/rz0m             386096  383496    2600  0.67       7    324
   1 /dev/io1/rz0m             386096  369928   16168  4.19    2756   6064
```

- Show system wide paging file statistics every 10 seconds:

```
/usr/mach/bin/pgs 10
```

- Show only the boot-node default paging file */mach_servers/paging_file* statistics:

```
/usr/mach/bin/pgs -d
```

Files

/usr/mach/bin/pgs	Executable file

PMAKE

Parallel make utility that maintains up-to-date versions of target files and performs shell commands.

NOTE

Although functionally similar, **pmake** is *not* the same as the System V command of the same name.

Syntax

pmake [-bcdeFikmnNpqrsStuUvw] [**-C** *dir*] [**-f** *file*]
[**-I** *dir*] [**-j** [*jobs*]] [**-l** [*load*]] [**-o** *file*]
[**-P** [*partition*]] [**-W** *file*] [*macro_definition* ...]
[*target* ...]

Arguments

-b	Has no effect; exists so older-version **make** dependency files continue to work.
-c	Does not try to find a corresponding Revision Control System (RCS) or Source Code Control System (SCCS) file and check it out if the file does not exist.
-C *dir*	Changes to directory *dir* before reading the description files or doing anything else. If multiple **-C** switches are specified, each is interpreted relative to the previous one: **-C/ -C**`etc` is equivalent to **-C /etc**. This is typically used with recursive invocations of **pmake**.
-d	Prints debugging information in addition to normal processing. The debugging information includes information about the files considered for processing, the comparison of file-times, the files that need processing, and the implicit rules being considered and actually applied.
-e	Does not reassign environment variables within the description file.
-f *file*	Reads *file* for a description of how to build the target file. If you do not specify the **-f** switch, **pmake** looks in the current directory for a description file named *makefile* or *Makefile*. If a – (dash) follows the **-f** switch, **pmake** reads standard input. You can specify more than one description file by entering the **-f** switch with the *file* argument more than once.
-F	Causes a fatal error if a description file is not present.

-i	Ignores error codes returned by commands and continues to execute until finished. This is similar to the pseudotarget command **.IGNORE:**, which can be specified in the description file. The **pmake** command normally stops if a command returns a nonzero exit code.
-I *dir*	Specifies a directory *dir* to search for description files to be included. You can specify the **-I** switch multiple times in a command line to specify multiple directories to search. The directories are searched in the order specified.
-j [*jobs*]	Specifies the maximum number of jobs that can run simultaneously. The default is the partition size, or 1 if the **pmake** command is running in the service partition. If there is more than one **-j** switch, the last one is effective. If the **-j** switch is given without an argument, the **pmake** command does not limit the number of jobs that can run simultaneously.
-k	Stops processing the current target if an error occurs, but continues with other branches that do not depend on the target that failed.
-l [*load*]	Specifies that no new jobs should be started if there are other jobs running and the load average is at least the value of *load* (a floating-point number). Specifying the switch with no argument removes a previous load limit.
-m	Searches for machine-specific subdirectories automatically. On a Paragon system, if a *PARAGON* subdirectory exists in the current directory, the **-m** switch adds the *PARAGON* subdirectory to the directory list specified by the *VPATH* special variable. See the "Special Variables" section for more information.
-n	Echoes commands that would be executed, but does not execute them.
-N	Disables all configuration file (*Makeconf*) processing.
-o *file*	Does not process *file* even if it is older than its dependencies, and does not process anything because of changes in *file*. Essentially, the file is treated as very old and its rules are ignored.
-p	Echoes all the environment variables, macro definitions, and target descriptions before executing any commands. This also prints the version information given by the **-v** switch. To print the database without trying to remake any files, use the following:

```
% pmake -p -f /dev/null
```

PMAKE *(cont.)* PMAKE *(cont.)*

-P [*partition*] Runs the **pmake** command as a parallel application in the partition specified by the *partition* argument. The name you specify for the *partition* argument must be *.compute* or the name of a subpartition in the *.compute* partition. If you specify the **-P** switch without an argument, the default partition is the value of the *NX_DFLT_PART* environment variable or the *.compute* partition if the *NX_DFLT_PART* variable is not set. If you do not specify the **-P** switch, the *.service* partition is the default partition.

-q Does not execute the commands in the description file. Returns a status code of zero if the object files are up-to-date; otherwise, returns a nonzero value.

-r Eliminates the built-in implicit rules and clears out the default list of suffixes for suffix rules.

-s Does not echo the commands being executed. This is similar to the pseudotarget command **.SILENT:**, which would be specified in the description file.

-S Stops processing the current target if an error occurs and does not continue to any other branch. This is the default. This cancels the effect of the **-k** switch. This is not necessary except in a recursive **pmake** where **-k** might be inherited from the top-level **pmake** via *MAKEFLAGS* or if you set **-k** in *MAKEFLAGS* in your environment.

-t Touches the targets; marks the files up-to-date without running commands to update them, or creates the target if it does not exist.

-u Does not unlink files that were automatically checked out from SCCS or RCS.

-U Has no effect; exists so older-version **make** dependency files continue to work.

-v Prints the version of the **pmake** command, a copyright, a list of authors, and a notice that there is no warranty. After this information is printed, processing continues normally. To get this information without doing anything else, use the following:

```
% pmake -v -f /dev/null
```

-w Prints a message containing the working directory before and after other processing in the directory. This may be useful for tracking down errors from complicated nests of recursive **pmake** commands.

PMAKE *(cont.)* PMAKE *(cont.)*

-W *file*
: Pretends that the target *file* has just been modified. When used with the **-n** switch, this shows you what would happen if you were to modify that file. Without **-n,** it is almost the same as running a *touch* command on the given file before running the **pmake** command, except that the modification time is changed only in the imagination of the **pmake** command.

macro_definition
: Specifies a macro to use with the definition file. Use the same macro syntax as required for the definition file. Enclose strings in quotes. Spaces and tabs are ignored. See the "Macros" section for more information on using macros.

target
: Name of the target to build or update. Target names are typically executable files, but this is not always the case. If *target* is not specified, **pmake** uses the first target in the definition file.

Description

The **pmake** command updates a target based on whether the target's dependencies have been modified relative to the last modification of the target. Targets are typically executable files, but this is not always the case. Typically you use **pmake** to recompile large programs, but you can use it for any applications where a target must be updated whenever files the target depends on change. The **pmake** command executes commands in a description file, also called a *makefile*, to build or update the target.

The **pmake** command is an extension of GNU **make**, which was written by Richard Stallman and Roland McGrath. It has been renamed **pmake** to emphasize its ability to execute several commands in parallel, and to distinguish it from the standard sequential **make** utility. This manual page provides a summary of the base GNU **make** features as well as the **pmake** extensions. For more detailed information about GNU **make** features, refer to *The GNU Make Manual.* This manual is available through the Intel Scalable Systems Division Customer Service Response Center (*support@ssd.intel.com*).

For switches that have arguments, spaces between switches and argument are optional. For example, when using the **-I** switch, the directory name may come directly after the switch (**-I***dir*) or a space may be inserted (**-I** *dir*).

Description Files

When a description file exists for a program, invoking **pmake** executes all the commands needed to build the program. The **pmake** command uses a definition file and the last-modification time of the target and the files that a target depends on to decide which targets need updating.

Description files contain a sequence of entries that define target names and dependencies and describe the rules for updating the targets. A typical entry includes a dependency line and a series of commands, and takes the following form:

target1 [*target2* ...] :[:][*dependency1* ...] [; *command*]
    [*command*] [; *command* ...]

The dependency line begins with one or more target names separated by spaces. A single or double colon separates the target(s) from a list of zero or more dependencies for the target(s). If no dependencies are given, the target files are always updated if they do not exist. Otherwise, a target is updated only if a dependency has changed since the target was last updated. A single command may also appear on the dependency line, separated from the dependencies by a semicolon. Alternatively, commands may appear on subsequent lines provided that each command line begins with a tab character. When a target requires updating, the specified commands are executed.

Dependency Lines

Dependency lines may take several forms:

target... : [*dependency*] ...

Single-colon rules. Words following the colon are added to the dependency list for the target(s). If a target is named in more than one single-colon rule, the dependencies for all of its entries are concatenated to form that target's complete dependency list. In that case, only one of the single colon rules may include commands for remaking the target.

target... :: [*dependency*] ...

Double-colon rules. When used in place of a single colon (:), the double colon (::) allows a target to be checked and updated with respect to alternate dependency lists. Each double colon rule is considered independently when deciding whether and how to update a particular target.

target... : *target-pattern* : *dep-pattern...*

Static-pattern rules. The *target-pattern* and *dep-pattern* values specify how to compute the dependencies for each target. Each target is matched against the *target-pattern* to extract a part of the target name, called the stem. The stem is then substituted into each of the *dep-pattern* values to make the dependency names, for example, the following dependency line specifies that *foo.c* and *foo.h* are dependencies of *foo.o*:

```
$(OBJECTS): %.o : %.c %.h
```

.s1.s2 :

Double-suffix rules. The rule tells how to make a file *foo.s2* from the file *foo.s1* where *foo* is an arbitrary stem and *s1* and *s2* are suffixes.

.s1 :

Single-suffix rules. The rule tells how to make a file *foo* from the file *foo.s1* where *foo* is an arbitrary stem and *s1* is a suffix.

target-pattern : *dependency-pattern...*

Pattern rules. The target and dependency patterns each contain the wild card character, % (percent). The % in the *target-pattern* is matched against a stem of a target name. That stem is substituted for the % in the *dependency-pattern*. This creates the dependency for the target, for example, %.o : %.c with the stem *foo* creates the target *foo.o* from the dependency *foo.c*.

Parallel Execution

The **pmake** command updates multiple target files in parallel. Parallel execution may occur either in the service partition or the compute partition. In the service partition, **pmake** relies on process migration and load balancing to ensure parallel execution. In the compute partition, **pmake** places commands on the available nodes within a partition and executes as a parallel application.

You can request parallel execution using either the **-j** or **-P** switch, or using both switches together. The **-P** switch specifies the partition in which **pmake** runs jobs. If you specify the **-P** switch without an argument, the default partition is the value of the *NX_DFLT_PART* environment variable or the *.compute* partition if the *NX_DFLT_PART* variable is not set. If you do not specify the **-P** switch, the *.service* partition is the default partition.

The **-j** switch specifies the maximum number of jobs that can run in parallel. If the **-j** switch is not used, the maximum number of jobs defaults to the number of nodes in the partition, or one job if the **pmake** command is running in the service partition. If you use the **-j** switch followed by the optional *jobs* argument, the **pmake** command runs up to the number of jobs you specify in parallel. The number of jobs the **pmake** command can run in parallel is not limited to the number of nodes in the partition, because multiple jobs can run on a node. If you specify the **-j** switch without the *jobs* argument, the maximum number of jobs the **pmake** command can run in parallel is unlimited.

The **-l** switch also affects parallel execution of the **pmake** command. This switch specifies that no new jobs should be started if there are other jobs running and the load average is at least the value of the *load* argument.This switch can limit parallel execution when the system load level is above the specified load level. The **-l** switch does not itself cause parallel execution. It affects parallel execution when you use it with the **-j** or the **-P** switches, only.

The **pmake** command relies on the dependencies defined in the description file to determine which files can be updated in parallel. For example, if the file *foo* is a dependency of the file *bar*, the **pmake** command ensures that the *foo* and *bar* files are not simultaneously updated. For this reason, it is important that the description file dependencies be complete when using the **-j** and **-P** switches.

When using the **-j** or **-P** switch, **pmake** becomes a parallel application. If you use the **-j** or **-P** switch after this on a recursive invocation of **pmake**, it is ignored since **pmake** is already executing as a parallel application. Therefore, when invoking **pmake**, choose the best recursion level at which to use the **-j** or **-P** switch.

PMAKE *(cont.)* PMAKE *(cont.)*

For example, if **pmake** is invoked with "**-j**2 **-P**.compute" switches on the following description file, the two targets *big* and *little* will be updated simultaneously.

```
all: big little

big:
        cd bigdir; $(MAKE)
little:
        cd littledir; $(MAKE)
```

If, in the above example, the *little* target is quickly updated, while the *big* target involves lots of compiles and takes several hours to build, the benefits of parallelism will be lost.

A better approach in this case is to invoke the top-level **pmake** without the **-j** or-**P** switches and use the switches at the second level instead. The better approach is shown below.

```
all: big little

big:
        cd bigdir; $(MAKE) -j8 -P.compute

little:
        cd littledir; $(MAKE) -j2 -P.compute
```

Commands

The commands to remake a target may be prefaced by one or more of the following special characters. The special characters are not passed to the shell but have the following effect within **pmake**:

-	**pmake** ignores any non-zero error code returned by the command.
+	**pmake** executes the command even if the **-n**, **-q** or **-t** switches are specified.
@	**pmake** does not print the command before executing it.

Included Description Files

Description files may be included within other description files by using the **include** directive as shown below. When **pmake** encounters an **include** directive within a description file, it temporarily stops processing the first description file, processes the included description file, and then resumes processing the original description file.

include *filename*

A file named *filename* is included and processed. An error occurs if the file is not found.

-include *filename*

A file named *filename* is included and processed. No error occurs if the file is not found.

Relative pathnames for the *filename* argument are interpreted as relative to the location of the file that includes them.

Configuration File Support

When you invoke **pmake**, it searches for the configuration file *Makeconf*. It searches the build tree starting from the current directory and continuing up to the root directory. It includes the first *Makeconf* file it finds and processes this file prior to processing any description file. Not having a *Makeconf* file does not produce an error. This file contains rules that override the default rules **pmake** uses.

The **pmake** command allows a software project to be organized into separate source and object directory trees. The source tree contains files that are read but not modified by **pmake**. The object tree contains the target files that **pmake** creates or updates. The trees are assumed to have the same structure so that each source directory has a counterpart with the same name within the object tree. If the *Makeconf* file contains a definition for the variable *OBJECTDIR*, it is interpreted as the root of the object directory tree. *OBJECTDIR* may be defined either as an absolute path or as a path relative to the location of the *Makeconf* file.

Before running any commands, **pmake** computes the relative path from the location of the *Makeconf* file to the current directory and appends that path to the root of the object directory tree to determine the current object directory. The **pmake** command then changes directories to the object directory, creating subdirectories as needed, and modifies the *123* to search for source files in the original directory. For example, assume a *Makeconf* file in */usr/foo* defines *OBJECTDIR* as */usr/obj*. If you invoke **pmake** from */usr/foo/bar*, the object directory is */usr/obj/bar* and */usr/foo/bar* will be added to the *VPATH*.

PMAKE *(cont.)* PMAKE *(cont.)*

The *Makeconf* file may include a definition for the variable *SOURCEDIR*. The *SOURCEDIR* variable is interpreted as a colon-separated list of alternate source directory trees. As with *OBJECTDIR*, *SOURCEDIR* may include both absolute pathnames and pathnames relative to the location of the *Makeconf* file.

If *SOURCEDIR* is defined, **pmake** computes the additional source directories to be searched in much the same manner as it computes the current object directory. The relative path from the location of the *Makeconf* file to the current directory is computed and then appended to each of the colon-separated path names in the *SOURCEDIR* variable. The resulting pathnames are then added to the *VPATH* and searched after the current directory.

Macros

Macros may be used to simplify and improve the portability of description files. Macros may be defined either on the command line or from within the description file. Macro definitions can have the following general forms:

MACRO = value
: Recursively expanding macro definition. The macro value is installed verbatim; if it contains references to other macros, those references are expanded when the macro is evaluated.

MACRO := value
: Simply expanding macro definition. The value is evaluated once, when the macro is installed; imbedded macro references are evaluated at that time.

override *MACRO = value*
: Override directive. Causes macro definition within a description file to override definition from the command line.

Macro references take the form $(*macro-name*) or ${*macro-name*} and may appear anywhere within the description file. Again, several special forms are supported.

$(...$(*MACRO*)...)
: Nested macro references.

$(*MACRO:suffix1=suffix2*)
: Suffix replacement references. The value of *suffix1* is replaced by *suffix2* in the expansion of *MACRO*. The *suffix1* must occur at the end of a word.

$(*MACRO:pattern1=pattern2*)
: Pattern replacement references.The *pattern1* and *pattern2* values each contain the wild card character, %. Occurrences of *pattern1* in the expansion of *MACRO* are replaced by *pattern2* with the % character matching any stem.

PMAKE *(cont.)* PMAKE *(cont.)*

$(*MACRO/reg-expression/replacement*)
: Pattern replacement references. The replacement string is substituted for the reg-expression within the macro expansion. The valid forms of regular expressions are described in **regexp(3)**. Semicolons may be used in place of the slashes that separate the *MACRO*, *reg-expression*, and *replacement* strings.

$(*MACRO*:*X*)
: C-shell style modifiers. *X* may be **t** (tail), **h** (head), **r** (root) or **e** (extension).

$(*MACRO?value1*:*value2*)
: Conditional expressions. Evaluates to *value1* if *MACRO* is defined and *value2* otherwise.

Special Macros

The following internal macros are automatically set as each target is processed:

$@	The name of the current target.
$*	The base name of the current target.
$<	The name of the current dependency file.
$?	The list of dependencies that are newer than the target.
$%	The name of the library member being processed.
$^	The list of all dependencies.
$$@	The current target (valid only on the dependency line).

Special Variables

Some variables have special meaning for the **pmake** command. Some of these variables are set automatically by the **pmake** command or they can be set as environment variables. The following special variables are supported:

cputype
: The CPU type of the target system in lower-case (for example, `i860`). This variable is set automatically by the **pmake** command.

CPUTYPE
: The CPU type of the target system in upper-case (for example, `I860`). This variable is set automatically by the **pmake** command.

PMAKE *(cont.)* PMAKE *(cont.)*

MAKE	The command line with which **pmake** was invoked, excluding switches. This variable is set automatically by the **pmake** command.
MAKEFILES	A list of description files to be read before any others. This variable may be set in the environment.
MAKEFLAGS	A list of the switches specified on the command line. This variable is set automatically by the **pmake** command.
MAKELEVEL	The current level of make recursion. This variable is set automatically by the **pmake** command.
OBJECTDIR	The root of the object tree where **pmake** will build its targets.
SHELL	The shell to use for command execution. The default is */bin/sh.* This variable may be set in the environment or in a description file.
SOURCEDIR	The roots of the source tree where **pmake** will search for sources.
SUFFIXES	The list of default, known suffixes from built-in suffix rules. This variable is set automatically by the **pmake** command.
target_machine	The machine architecture of the target system in lower-case (for example, **paragon**). This variable is set automatically by the **pmake** command.
TARGET_MACHINE	The machine architecture of the target system in upper-case (for example, **PARAGON**). This variable is set automatically by the **pmake** command.
VPATH	A colon-separated list of directories to search for dependency files. This variable may be set in the environment or in a description file.

Pseudotarget Names

The **pmake** command assigns special meanings to the following pseudotargets:

.DEFAULT	If it appears in the description file, the rule for this target is used to process a target when there is no other entry for it.
.EXIT	If defined in the description file, **pmake** processes this target and its dependencies after all other targets are built.

PMAKE *(cont.)* **PMAKE** *(cont.)*

.EXPORT	Variables listed as dependencies of this target are expanded and exported to the environment in which **pmake** runs it's commands. Unlike GNU **make**, **pmake** does not normally export variables defined within a definition file.
.IGNORE	Ignore errors. When this target appears in the description file, **pmake** ignores non-zero error codes returned from commands.
.INIT	If defined in the description file, this target and its dependencies are built before any other targets are processed.
.PHONY	The dependencies of this target are considered to be "phony" targets. When it is time to consider such a target, **pmake** will run its commands unconditionally regardless of whether a file with that name exists or what its last modification time is.
.PRECIOUS	List of files not to delete. **pmake** does not remove any of the files listed as dependencies for this target when interrupted. **pmake** normally removes the current target when it receives an interrupt.
.SILENT	Run silently. When this target appears in the description file, **pmake** does not echo commands before executing them.
.SUFFIXES	The dependencies of this target are the suffixes that **pmake** will search for when applying suffix rules.

Conditional Execution

The **pmake** utility provides conditional execution directives that control what parts of the description file **pmake** sees. The general format of a conditional directive is the following:

conditional-directive
text-if-true
endif

Another format is the following:

conditional-directive
text-if-true
else
text-if-false
endif

PMAKE *(cont.)* PMAKE *(cont.)*

There are four different conditional directives. They are:

ifeq (*arg1*, *arg2*)
The conditional evaluates to true if *arg1* is equal to *arg2*.

ifneq (*arg1*, *arg2*)
The conditional evaluates to true if *arg1* is not equal to *arg2*.

ifdef *variable-name*
The conditional evaluates to true if *variable-name* is defined.

ifndef *variable-name*
The conditional evaluates to true if *variable-name* is not defined.

Archive Support

Archive library members may be referenced within a description file using the following form:

lib(*member*)

Functions

Functions provide support for text processing within a description file. The **pmake** utility provides approximately 20 functions including functions to do pattern replacement, filtering, sorting and filename component selection. The general form of a function call is: $(*function arguments*) or ${*function arguments*}.

For a description of the available functions and their arguments, refer to *The GNU Make Manual.*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes.*

See Also

The GNU Make Manual, available through the Intel Scalable Systems Division Customer Service Response Center (*support@ssd.intel.com*).

PMAKE *(cont.)* PMAKE *(cont.)*

Copyright

Copyright ® 1988-1994 Intel Corporation

The **pmake** utility is an extension of GNU **make** and is distributed under the terms of the GNU General Public License. Intel will provide a complete copy of the **pmake** source code upon request. For more information, contact Intel's SSD Customer Service Response Center.

PROF PROF

Displays profile data.

Syntax

prof [**-ghsz**] [**-a** | **-c** | **-n** | **-t**] [**-o** | **-x**] [**-m** *mdata*] [*prog*]

Arguments

-a	Sort output lines by increasing symbol address.
-c	Sort output lines by decreasing number of calls.
-g	Include non-global symbols (static functions).
-h	Suppress the heading normally displayed on the report. (This is useful if the report is to be processed further.)

NOTE

To make the **-m** switch work properly when the node is not 0, two things must happen:
1. The name of the executable (*prog*) must be the last argument.
2. The *mdata* file must be specified completely, even if the file is in the default *mon.out* directory (e.g., *mon.out/hello.3145768.1.0*).

-m *mdata* Use *mdata* file as the input profile file. By default, the file with the lowest *node:ptype* pair from the directory *mon.out* is used. The data files in *mon.out* are named with the following form:

executable_name.pid.node.ptype

In this name *pid* is the process ID, the *node* is the number of the node on which the process is running, and *ptype* is the last process type the process had before the performance data was written.

-n	Sort output lines by symbol name.
-o	Display each symbol address in octal. The default is decimal.
-s	Display a summary of several of the monitoring parameters and statistics on the standard error output.

PROF *(cont.)* PROF *(cont.)*

-t	Sort output lines by decreasing percentage of total time (default).
-x	Display each symbol address in hexadecimal. The default is decimal.
-z	Include all symbols in the profile range, even if associated with zero number of calls and zero time.
prog	Correlate the symbol table in the executable file *prog* with the profile file specified by the **-m** argument. If you omit this argument, **prof** uses the symbol table in the executable file *a.out.*

Description

Use **prof** to interpret a profile file produced with the IPD **instrument** command. A program creates a profile file if it has been loaded under IPD and processed with the **instrument** command. For a complete description of the IPD **instrument** command, refer to the *Paragon™ System Interactive Parallel Debugger Reference Manual*

The symbol table in the executable *prog* is read and correlated with the profile file *mdata*. For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is displayed, together with the number of times that function was called and the average number of milliseconds per call.

The times reported in successive identical runs may vary by 5% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may beat with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes.*

See Also

gprof

Paragon™ System Interactive Parallel Debugger Reference Manual

PSPART PSPART

Shows status of the applications in a partition.

Syntax

pspart [**-r**] [*partition*]

Arguments

-r	Recursively displays status information for the specified partition and all its subpartitions.
partition	Absolute or relative pathname of a partition. If the *partition* argument is not specified, the value of the *NX_DFLT_PART* environment variable is used. If *NX_DFLT_PART* is not defined, the *compute* partition is used. You must have read permission on the specified partition to use this command.

Description

The **pspart** command shows the following status information about all of the applications running in a partition and subpartitions:

`PGID`	Process group ID of an application running in the specified partition. The process group ID of an application is always the same as the process ID of the application's controlling process.
`USER`	Name of the user who invoked the application.
`SIZE`	Number of nodes allocated to the application from the partition.
`PRI`	Application's priority.
`STARTED`	Starting time of the application's process. If the application was started more than 24 hours before the current time, the date it was started is displayed instead.

`TIME ACTIVE`

Amount of time the application has been active (rolled in) in the current rollin quantum. The time active is shown both in the following format:

```
[[hours:]minutes:]seconds.milliseconds percent
```

The percent is the percentage of the partition's rollin quantum. If the application is not active in the current rollin quantum, a dash (-) is shown for the time and percentage. If the partition uses space sharing, the time shown is the total amount of time the application has been running and the percentage is always 100%.

PSPART *(cont.)* PSPART *(cont.)*

`TOTAL TIME`	Total time the application has been rolled in since it was started. The time format is the following: `[[`*`hours`*`:]`*`minutes`*`:]`*`seconds`*`.`*`milliseconds`*`.` If the partition uses space sharing, the `TOTAL TIME` is always the same as `TIME ACTIVE`.
`COMMAND`	Command line with options used to start the application. See the "Examples" section of this manual page for information about the `COMMAND` column concerning faulting parallel applications.

The following information is displayed for active partitions:

`OWNER`	Owner of the active partition.
`GROUP`	Group the active partition belongs to.
`SIZE`	Size of the subpartition. All the nodes in the subpartition containing an active application are considered active, even if not all the nodes in the subpartition are actually in use by applications.
`PRI`	The current priority of the subpartition. This is the highest priority of all applications in the subpartition or the subpartition's effective priority limit, whichever is lowest.
`STARTED`	Time or date when the oldest application in the subpartition was started.
`TIME ACTIVE`	Amount of time the subpartition has been active (rolled in) in the current rollin quantum.
`TOTAL TIME`	Total time the subpartition has been rolled in since it was started.
`NAME`	Subpartition name.

PSPART *(cont.)* PSPART *(cont.)*

Examples

The following example shows the status of the applications in the partition *mypart*. The parent partition is the *compute* partition:

```
% pspart mypart
 PGID    USER    SIZE   PRI   START      TIME ACTIVE    TOTAL TIME COMMAND
  12345  pat      256   5      11:42:20  45.00 75%       0:04:41    mag -sz 256
  23456  chris   67     4      Jan 21    - -             0:12.30    boggle
  34567  smith   192    10     02:21:51  0:01:00 100%   2:12:03    myfft
Active Partitions
 OWNER   GROUP     SIZE PRI   START       TIME ACTIVE   TOTAL TIME NAME
  smith  eng        64  6      09:16:30    - -           1:18.10    subpart
```

The **pspart** command shows that the partition *mypart* has two active applications (*mag* and *myfft*), one inactive application (*boggle*), and one active subpartition (*subpart*).

This following example uses the **-r** switch to display status information about the applications, active partitions, and the subpartitions in the partition *mypart*.

```
% pspart -r mypart
mypart:
 PGID    USER    SIZE   PRI   START      TIME ACTIVE    TOTAL TIME COMMAND
  12345  pat      256   5      11:42:20  45.00 75%       0:04:41    mag -sz 256
  23456  chris   67     4      Jan 21    - -             0:12.30    boggle
  34567  smith   192    10     02:21:51  0:01:00 100%   2:12:03    myfft
Active Partitions
 OWNER   GROUP     SIZE PRI   START       TIME ACTIVE   TOTAL TIME NAME
  smith  eng        64  6      09:16:30    - -           1:18.10    subpart
mypart.subpart:
 PGID    USER    SIZE   PRI   START      TIME ACTIVE    TOTAL TIME COMMAND
  45678  smith   56     7      09:16:30  - -             1:18.10    span
```

The **pspart** command shows that the partition *mypart* has two active applications (*mag* and *myfft*), one inactive application (*boggle*), and one active subpartition (*subpart*). The subpartition *subpart* has the application *span* in it. The application is not active.

PSPART *(cont.)* PSPART *(cont.)*

This example shows how **pspart** returns information in the COMMAND column for a parallel application of which one or more processes fault and dump core.

```
% pspart otherpart
 PGID USER SIZE PRI START    TIME ACTIVE   TOTAL TIME COMMAND
  42   pat   12   5  12:51:22 0:06.30 63%   0:02:42    app (core dump)
```

In this example, the application *app* is running on some (not necessarily all) nodes in the partition *otherpart* defined by *NX_DFLT_PART*. The string "(core dump)" appended to the application name indicates that *app* is currently dumping core. Display of the application name *app*, in this example, indicates that the controlling process for *app* was found and is still running.

This next example illustrates a similar scenario except that the server cannot find the controlling process of the application. This condition could mean that the controlling process has died or is in the process of dying. When the server cannot find the controlling process of a parallel application in which processes have faulted, **pspart** substitutes the question mark character (?) for the application name in the COMMAND column. Note that "(core dump)" is still appended because a core dump is occurring or has occurred.

```
% pspart otherpart
 PGID USER SIZE PRI START    TIME ACTIVE   TOTAL TIME COMMAND
  42   pat  12   5   12:51:22 0:06.30 63%   0:02:42    ? (core dump)
```

Errors

`pspart: Partition not found`

You specified a partition that does not exist.

`pspart: Partition permission denied`

You do not have read permission for that partition.

PSPART *(cont.)* PSPART *(cont.)*

Files

/usr/bin/pspart Specifies the command path.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

calls: **nx_pspart()**

commands: **chpart**, **lspart**, **mkpart**, **rmpart**, **showpart, core, coreinfo**

OSF/1 Command Reference: **ps(1)**

RESET RESET

Diagnostic station: Resets the Paragon system from the diagnostic station.

Syntax

reset [autocfg] [autoreboot] [boot | noboot] [copy] [debug | ramdisk] [flash] [ignorelock] [skip] [watchdog] -f *file*

Arguments

autocfg	Creates the *SYSCONFIG.TXT* file, but does not reboot the system. The *DEVCONF.TXT* file must exist in the */usr/paragon/boot* directory.
autoreboot	Resets and automatically reboots the system when the system watchdog detects that a node crashed. This switch is used by the system watchdog only.
boot	Resets the system. This is the default. Because this is the default, only use this switch following the **autocfg** switch to force the system to reboot.
copy	Specifies using a remote kernel when resetting the system. See the "Description" section for more information.
debug	Resets the system and boots with the debug version of the kernel. You cannot use this switch with the **ramdisk** switch.
flash	Resets the system and uses the **async** command to attach the console to the serial line. The system does not boot. Use this argument to maintain flash programs.
ignorelock	Resets the system and ignores the lock on the scan device. A lock on the scan device is set with the */tmp/LOCK.SCAN* file. Use this switch when a lock remains active on the scan device after the **reset** script has been killed.
noboot	Does not reset the system.
ramdisk	Resets the system and boots the system ramdisk. Use this argument for installing the operating system or for system maintenance. You cannot use this switch with the **debug** switch.
skip	Resets the system, but does not connect to the console. Use this argument with applications that have an existing console interface.
watchdog	Resets the system and starts the system watchdog. The system watchdog will not start if the console is configured to use the scan lines. See the **fscan** manual page for more information.

RESET *(cont.)* RESET *(cont.)*

-f *filename*	Specifies a magic file to use instead of the *MAGIC.MASTER* file. The *filename* argument must be a simple filename. It cannot include any part of the pathname. The format of *filename* must follow the same format used in *MAGIC.MASTER* .

Description

The **reset** command runs on the diagnostic station and is intended for use by the system administrator to reset a Paragon system.

The first time you run the **reset** command, use the **autocfg** switch to create a new *SYSCONFIG.TXT* file. To create a new *SYSCONFIG.TXT* file and boot the system, use the **autocfg** switch followed by the **boot** switch as follows:

```
DS# reset autocfg boot
```

This creates a new *SYSCONFIG.TXT* file and boots the system in that order, because the **reset** command arguments are parsed and invoked in the order you enter them.

The **copy** switch specifies using a remote kernel when the system is reset. This sets the following bootmagic string in the *bootmagic* file:

```
BOOT_COPY_REMOTE_KERNEL=1
```

When this bootmagic string is set, the file specified in the bootmagic string *BOOT_REMOTE_KERNEL_NAME* is copied to the file specified in the bootmagic string *BOOT_KERNEL_NAME* before the system is booted. The **copy** switch assumes the files */sbin/bootmesh.h* and */sbin/rkernel* exist.

Environment Variables

You can control the behavior of the **reset** script with the following environment variables:

BOOT_DIR	Specifies the pathname of the boot directory on the diagnostic station. The default is */usr/paragon/boot.*
MAGIC_MASTER	Specifies the magic file to use. The value supplied must be a simple filename. It cannot include any part of the pathname. The default is *MAGIC.MASTER.*
SYSCONFIG	Specifies the pathname of the *SYSCONFIG.TXT* file. The default is the file *./SYSCONFIG.TXT.*

MACH_KERNEL	Specifies the pathname of the boot node kernel on the diagnostic station. The default is the file */usr/paragon/boot/mach_kernela* or */usr/paragon/boot/mach_kernelb* depending on the nodes in the system. If all the nodes in the system have B-step NICS *kernelb* is used. If some or all of the nodes in the system have A-step NICS then *kernela* is used.

For example, to change the *MAGIC.MASTER* file from *./MAGIC.MASTER* to *./MAGIC.new*, you could set the *MAGIC_MASTER* variable as follows:

```
DS# MAGIC_MASTER=MAGIC.new
DS# export MAGIC_MASTER
```

Reset Strings

You can control the behavior of the **reset** script with the following strings in the *MAGIC.MASTER* file. Putting these strings in *MAGIC.MASTER* is equivalent to setting the corresponding environment variable. Note that these strings are not bootmagic strings. They cause **reset** to set equivalent environment variables.

RST_CONFIGURATION	Specifies the system configuration: **condo**, **multi**, or **full**. The default is **full**. A **condo** system has single backplane, a **multi** system has two or three backplanes, and a **full** system has four or more backplanes.
RST_TOP_BACKPLANE	Specifies the top backplane in the system: **A**, **B**, **C**, or **D**, with **A** being closest to the floor. The default is **D**. This variable should be set to a value other than **D** only when *CONFIGURATION* is set to **condo** or **multi**.
RST_BOOT_DIR	Specifies the directory where boot configuration files, kernels, and other files used in booting are found. The default is the file */usr/paragon/boot.*
RST_SERIAL_DEVICE	Specifies the absolute pathname of the device where the serial line from the Paragon system is attached to the diagnostic station. The default is the file */dev/tty1a.*

RESET *(cont.)* RESET *(cont.)*

RST_MAGIC_MASTER	Specifies the file name of the *MAGIC.MASTER* file. The default is the value of the environment variable *MAGIC_MASTER*, or the file *./MAGIC.MASTER* if the environment variable is not set. This bootmagic string must be set in the default *MAGIC.MASTER* file, and does not take effect until after the default *MAGIC.MASTER* file has been parsed for strings that begin with *RST_*. The value of the bootmagic string must be a simple filename. It cannot include any part of the pathname. The **reset** script prepends the boot directory to the specified name. The boot directory is */usr/paragon/boot* by default, or whatever *RST_BOOT_DIR* is defined as.
RST_SYSCONFIG_TXT	Specifies the pathname of the *SYSCONFIG.TXT* file. The default is the value of the environment variable *SYSCONFIG*, or the file *./SYSCONFIG.TXT* if the environment variable is not set.
RST_DEVCONF_FILE	Specifies the pathname of the *DEVCONF.TXT* file. The default is the file *./DEVCONF.TXT*.
RST_MACH_KERNEL	Specifies the absolute pathname of the boot node kernel on the diagnostic station. The default is the value of the environment variable *MACH_KERNEL*, or is the file */usr/paragon/boot/mach_kernela* or */usr/paragon/boot/mach_kernelb* (depending on the types of nodes in the system) if the environment variable is not set.

For example, to change the boot directory from */usr/paragon/boot* to */usr/paragon/newboot*, you could add the following line to the *MAGIC.MASTER* file:

```
RST_BOOT_DIR=/usr/paragon/newboot
```

This specifies that the **reset** command use the directory */usr/paragon/newboot* to find the files for booting the Paragon system.

Specifying a Kernel File

The **reset** command determines which kernel file to download in the following order:

1. The **reset** command uses the environment variable *MACH_KERNEL*. If this variable is not set, **reset** uses *RST_MACH_KERNEL* (if defined) to set *MACH_KERNEL*.

2. If *$MACH_KERNEL* is not defined, then **reset** uses the bootmagic string *BOOT_REMOTE_KERNEL*.

RESET *(cont.)* RESET *(cont.)*

3. If the bootmagic string *BOOT_REMOTE_KERNEL* is not defined, then **reset** uses the environment variable *$BOOT_DIR*.

4. If *$BOOT_DIR* is not set, **reset** uses the boot directory defined by the bootmagic string *RST_BOOT_DIR* and either *mach_kernela* or *mach_kernelb* (depending on the types of nodes in the system). Thus, the complete pathname of the Mach kernel is either *$BOOT_DIR/mach_kernela* or *$BOOT_DIR/mach_kernelb*.

Using the *MACH_KERNEL* Environment Variable

The **reset** command uses the environment variable *MACH_KERNEL* to determine which kernel to load. Set the *MACH_KERNEL* environment variable as follows:

```
setenv MACH_KERNEL string
```

The *string* has the following form:

```
[system:]kernel_file
```

Where:

system	Name of the system where the kernel file resides. The system name cannot be an IP address. If *system* is not specified, the diagnostic station name is used as the system name.
kernel_file	Absolute or simple pathname of the kernel file. If the *path* is not specified, the value of *$BOOT_DIR* is used, or the default */usr/paragon/boot*.

The following example specifies a kernel name using the system name, full directory pathname, and kernel file name:

```
setenv MACH_KERNEL namu:/usr/myhome/kernel/mach_kernel
```

The following example specifies a kernel name using the node name and kernel file name:

```
setenv MACH_KERNEL namu:mach_kernel.gp
```

The **reset** command completes the kernel pathname using the environment variable *$BOOT_DIR* (if defined) to complete the kernel pathname. If *$BOOT_DIR* is not defined, then **reset** uses the default name */usr/paragon/boot*.

## RESET *(cont.)*		RESET *(cont.)*

The following example specifies the kernel name using kernel file name only:

```
setenv MACH_KERNEL mach_kernel.gp
```

The **reset** command completes the kernel pathname using the diagnostic station name and *$BOOT_DIR* (if defined). Again, */usr/paragon/boot* is used when *$BOOT_DIR* is not defined.

The following example specifies the kernel name using full pathname of the kernel file specification:

```
setenv MACH_KERNEL /usr/myhome/kernel/mach_kernel
```

The **reset** command completes the kernel pathname using the diagnostic station name.

Examples

You execute the **reset** script on the diagnostic station to reset a Paragon system. The command resets the nodes, programs the MRCs, and determines the console interface. The script determines which console interface to use (either the **async** program, the **scanio** program, or the **fscan** program) based on the value assigned to *BOOT_CONSOLE* in the *MAGIC.MASTER* file as follows:

```
BOOT_CONSOLE=s        Use scanio
BOOT_CONSOLE=f        Use fast scan
BOOT_CONSOLE=cm       Use the serial line and the mesh.
```

By default, the serial line is used with the **async** utility. Use the key sequence `~.` or `~q` to exit this program.

The following example resets and boots the system:

```
DS# reset
```

The following example creates a new *SYSCONFIG.TXT* file, but does not boot the system:

```
DS# reset autocfg
```

The following example creates a new *SYSCONFIG.TXT* file and boots the system. The order of the arguments must be as follows:

```
DS# reset autocfg boot
```

The following example boots the system with the debug version of the kernel:

```
DS# reset debug
```

This appends a *.db* to the value of the *MACH_KERNEL* variable and uses the resulting MACH kernel name to boot the system.

RESET *(cont.)* RESET *(cont.)*

Reset the system in single-user mode, except that the final connection to the boot node is not made, as follows:

```
DS# reset skip
```

Reset the system using the RAM disk as follows:

```
DS# reset ramdisk
```

When using the **ramdisk** switch, the **reset** script provides extensive error checking and prints an appropriate error message if a problem is detected.

Reset the system, but do not download any bootmagic strings, as follows:

```
DS# reset flash
```

After resetting the nodes and programming the MRCs, the console prompt is returned where you must press the **<Enter>** key to get the monitor prompt. Use this when you need to access the flash memory.

Start the system watchdog after resetting the system as follows:

```
DS# reset watchdog
```

This overrides the setting of the *START_WATCHDOG* variable, but the setting of the *BOOT_CONSOLE* still disables the system watchdog if it is not set to **Cm** or **cm**.

Request the system to automatically reset when the system watchdog detects a node crash as follows:

```
DS# reset autoreboot
```

Files

/usr/paragon/boot/bootmagic
Specifies information for booting a Paragon system.

/usr/paragon/boot/DEVCONF.TXT
Specifies the device configuration for a Paragon system.

/usr/paragon/boot/MAGIC.MASTER
Specifies the master version of the bootmagic strings for booting a Paragon system.

RESET *(cont.)* RESET *(cont.)*

/usr/paragon/boot/SYSCONFIG.TXT
Specifies the hardware configuration for a Paragon system. Created with the **autocfg** switch.

/usr/paragon/boot/reset
Specifies the command path name.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **async**, **bootpp**, **console**, **fscan**, **parsemagic**

files: **bootmagic**, **DEVCONF.TXT**, **MAGIC.MASTER**, **SYSCONFIG.TXT**

RMKNOD RMKNOD

Builds a special file on a remote I/O node.

Syntax

rmknod *special_file* **b** | **c** *major_device# minor_device# node*

Arguments

special_file	Name of a *special device file.*
b	Indicates that the special file corresponds to a block-oriented device (disk or tape).
major_device#	Major device number the operating system uses to find the device driver code.
minor_device#	Minor device number, in decimal or octal numbers, that is the unit drive or line number.
c	Indicates that the special file corresponds to a character-oriented device.
node	Node number of a remote I/O node that can be the boot node or any other I/O node.

Description

The **rmknod** command has the same syntax and functionality as the **mknod** command, except for the following:

- The additional argument *node* for the **rmknod** command. Use the *node* argument to specify the number for the remote I/O node on which the actual device is attached.
- The **rmknod** command does not support the **mknod** command's **p** switch.

See the **mknod(8)** manual page in the *OSF/1 System and Network Administrator's Reference* for the complete description of the **mknod** command.

RMKNOD *(cont.)* RMKNOD *(cont.)*

Examples

To create a special file for a tape drive, */dev/io1/rmt6*, with a major device number of 8 and a minor device number of 96 on node 95, enter:

```
# rmknod /dev/io1/rmt6 c 8 96 95
```

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

rmknod()

OSF/1 Programmer's Reference: **mknod(2)**

OSF/1 System and Network Administrator's Reference: **makedev(8), mknod(8)**

RMPART RMPART

Removes the named partition.

Syntax

rmpart [**-f**] [**-r**] *partition*

Arguments

-f	Removes a partition and any applications running in the partition. When you use the **-f** switch, the **rmpart** command terminates all the applications running in the specified partition and then removes the named partition. By default, the **rmpart** command does not remove partitions in which applications are running.
-r	Removes a partition and all its subpartitions if no applications are running in the partition or its subpartitions. This is a recursive operation; all subpartitions and their subpartitions in the specified partition are removed. When used with the **-f** switch, the **rmpart** command terminates all the applications running in the partition and its subpartitions, and removes the partition and all its subpartitions.
partition	The absolute or relative partition pathname of the partition to be removed.

Description

This command removes the named partition. Use the **-f** switch if the partition contains running applications. Use the **-r** switch to remove a partition and all its subpartitions.

To remove a partition, you must have write permission on its parent partition.

NOTE

Removing a partition from the service partition may not take effect until after a reboot.

Examples

If there are applications running in *mypart*, use the following command to terminate the applications and remove the partition:

```
rmpart -f mypart
```

RMPART *(cont.)* RMPART *(cont.)*

Use the following command to remove the *mypart partition* and all its partitions:

```
rmpart -r mypart
```

Errors

```
rmpart: Partition lock denied
```

You specified a partition that is currently in use and being updated by someone else. You cannot remove a partition that is currently being updated.

```
rmpart: Partition not empty
```

You specified a partition that contains subpartitions or has active processes.

```
rmpart: Partition not found
```

You specified a partition that does not exist.

```
rmpart: Partition permission denied
```

You specified a partition whose parent partition does not grant you write permission.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

calls: **nx_pspart()**

commands: **chpart**, **lspart**, **mkpart**, **pspart**, **showpart**

SAT SAT

Runs the Paragon system acceptance test (SAT).

Syntax

sat [**-bchxV**] [**-d** *dir*] [**-l** *log*] [**-m** *mins*] [**-o** *output*]
[**-p** *partition*] [**-r** *reps*] [*test* ...]

Arguments

-b	Causes the **sat** command to build tests from sources prior to running them. You must have write permissions in the directories in which the tests are being built. The tests are built in the same directories that hold the source code. Use the **-d** command line switch to specify an alternate directory. The default directory is */usr/lib/sat.*
-c	Runs tests concurrently. The default is to run the tests serially.
-h	Displays a help message and exits.
-x	Causes the acceptance test to terminate if a test fails. By default, testing continues after an error is reported in the transcript.
-V	Causes the **sat** command to write version information to standard output and then exit.
-d *dir*	Specifies a user-defined directory from which to run acceptance tests. Otherwise, the **sat** command executes tests from the directory */usr/lib/sat.*
-l *log*	Causes the **sat** command to write a transcript of the test session to the specified log file.
-m *mins*	Specifies the total length of time, in minutes, to run the tests. By default, the tests execute once without regard for time.
-o *output*	Causes the **sat** command to write a final report to a user-specified output file.
-p *partition*	Specifies the partition in which the **sat** parallel tests are run. If you do not specify a partition, the parallel tests run in the partition specified by the *NX_DFLT_PART* environment variable. If the *NX_DFLT_PART* environment variable is not defined, the tests run in the *.compute* partition.

SAT *(cont.)* SAT *(cont.)*

-r *reps*	Sets the number of times to repeat the list of tests. By default, list of tests executes once.
test	Specifies which SAT test(s) to run. The *test* argument may be either the name of an individual test or a subdirectory.

Description

The Paragon system acceptance test (SAT) provides a set of test suites designed to rigorously test Paragon system functionality and performance. The standard tests, which are run on the system before it leaves the factory, can also be used to test the system after it is installed at your site, after installing additional hardware or software, and periodically to perform system performance tests. The SAT is invoked with the **sat** command, a test driver, located in the */usr/bin/sat* directory. The **sat** command executes tests in the directory */usr/lib/sat* or in a directory you specify with the **-d** switch.

The SAT test hierarchy consists of "organizing" directories and test directories. Organizing directories are used to organize the test directories. Test directories contain run scripts, executable files, data files, sources, and makefiles for compiling and running tests. The **sat** command identifies a test directory by the presence of a *run* file and a *README* file.

The **sat** command creates a variety of temporary files while running. These files (and the temporary files created by the tests in */usr/lib/sat*) reside in */usr/tmp*. When **sat** exits, it removes any temporary files it created.

The rest of this manual page discusses **sat** command arguments.

Exit Values

An exit value of

0	Indicates successful completion of all tests.
1	Indicates test failures.
2	Indicates an error occurred while processing command arguments or scanning the directory hierarchy.

Running Specific Tests (*test*)

If you don't specify a test, **sat** runs all tests. If you specify one test, **sat** runs only that test. For example, to run only the Level 2 BLAS test, enter:

```
# sat blas2
```

To run several tests, put a space between the test names. For example:

```
# sat blas2 paranoia
```

You can also run all the tests in a SAT subdirectory. For example, to run all the tests under the standard SAT parallel subdirectory */usr/lib/sat/parallel* directory, enter:

```
# sat parallel
```

Displaying Help Information (-h)

The **-h** switch displays a help message on standard output and then exits. If you select **-h** and specify a test, the message briefly describes the test. If you do not specify a test, the message describes the **sat** command and usage information; it also lists the available tests.

Building Tests from Sources (-b)

The **-b** switch causes the **sat** command to build tests from sources prior to running them. You must have write permissions in the directories in which the tests are being built. Root permissions are required to build tests in the default directory */usr/lib/sat*. (Use the **-d** command line switch to specify an alternate directory.) Only the tests that include a *makefile* (or Makefile) along with the sources are built.

If you specify **-r** (repetitive test runs) with **-b**, the build occurs once, before running the tests. If you specify **-r** 0, the **sat** program builds the tests but doesn't run any of them. The build time does not affect the number of times the tests are run.

SAT *(cont.)* SAT *(cont.)*

If you specify **-m** (length of time, in minutes, to run the tests) with **-b**, the build occurs once, before running the tests.

The **-b** switch causes tests to be built from scratch. In other words, objects and executable files are removed prior to each build. If any errors occur during a build, the **sat** program exits without running any tests. If you do not select **-b**, the **sat** program runs the tests from the executable files that are present.

NOTE

As shipped from the factory, the SAT includes the executables for each test. Although the SAT is designed to remove executables prior to each build, the makefiles installed at the factory ensure that the required compilers and sources are present before removing any binaries. This prevents the SAT from accidentally deleting the executables on systems where they cannot be rebuilt.

Running Tests Concurrently (-c)

By default, the **sat** program tests in sequential mode. The **-c** switch specifies concurrent operation of tests, which means all the tests start simultaneously. Concurrent operation implies that the **sat** tests will compete for system resources. Thus, if you run the tests concurrently (or if the system is running other applications when you run the tests) system performance is likely to be affected. The SAT still reports accuracy and completeness results.

If you specify the **-r** switch with the **-c** switch, the **sat** program waits for each iteration of the tests to complete before beginning the next iteration. This prevents shorter tests from beginning their next iteration before longer tests complete their current iteration.

SAT *(cont.)* SAT *(cont.)*

Repeating Tests (-m, -r)

The **-m** switch specifies the length of time, in minutes, to repeat the list of tests. The specified time must be at least 1 minute. When the specified time limit is reached, the **sat** program terminates the tests even if you specified multiple tests or concurrent operation.

The **-r** switch sets the number of times to repeat the test(s). The number of repetitions must be 0 or more. Specifying **-r 0** in conjunction with **-b** (build tests), builds tests without actually running the **sat** program. The list of tests repeats the specified number of times whether they pass or fail. If you specify **-r** with concurrent operation (**-c**), the tests will run in concurrent mode until all tests have reached the specified count.

If you specify both **-r** and **-m**, execution continues until either the repetition count or the time condition is met.

If you specify **-b** (build tests) with **-m** or **-r**, the time to build the tests does not impact the time or repetition count.

If you specify **-x** (exit on error) with **-m** or **-r** and an error is encountered, the entire run exits regardless of the time or repetition count.

Specifying a Partition for Parallel Tests (-p)

Parallel SAT tests are run in a partition. The **-p** switch identifies the test partition. By default, parallel tests run in the partition specified by the *NX_DFLT_PART* environment variable. If that variable is not defined, the tests run in the *.compute* partition. You must have permission to execute programs in the specified partition. Refer to the *Paragon™ System User's Guide* for more information about the *.compute* and *.service* partitions.

Running Tests from a User-Specified Directory (-d)

By default, the **sat** command runs tests from the */usr/lib/sat* directory. Use the **-d** switch to specify a different directory. If you specify a non-existent directory, the **sat** program generates an error message and exits.

SAT *(cont.)* SAT *(cont.)*

Creating Log Files and Final Reports (-l, -o)

You can use the **-l** and **-o** switches to write a transcript of a test session to a file. The file's pathname can be either relative or absolute.

The **-l** switch produces a log file that contains a simple transcript of the test session. For example, to run the Livermore LOOPS test and save the transcript in a file called *logfile*, enter:

```
# sat -l logfile lloops
```

The **-o** switch produces an output file that contains expanded information about the test session, including complete test results, operating parameters, and configuration information. This output file is called a *final report*. For example, to run the Livermore LOOPS test and save expanded information in a file called *yourfile,* enter:

```
# sat -o yourfile lloops
```

Exiting on Error (-x)

If you specify **-x**, the **sat** program terminates and exits as soon as an error is found and reported. If you do not specify **-x**, testing continues after an error is reported.

Displaying Version Information (-V)

If you specify **-V**, the **sat** program displays its version number and exits.

Files

/usr/lib/sat/README	Contains a description of the Paragon SAT
/usr/lib/sat/help	Contains help information

Directories

/usr/lib/sat	Root of default test directory hierarchy
/usr/tmp	Directory for temporary files created by the **sat** command

SAT *(cont.)* SAT *(cont.)*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

Paragon™ System Acceptance Test User's Guide

SCANIO SCANIO

Diagnostic station: Establishes a scan-based console interface between the diagnostic station and a node on an Paragon system.

Syntax

scanio [**-eiqv**] [**-b** *file*] [**-c** *letter*] [**-f** *file*] [**-s** *string*] [**-t** *state*] [*node*]

Arguments

-b *file*	Specifies the *bootmagic* file pathname. The default is the file *bootmagic* in the current directory.
-c *letter*	Specifies the backplane to use for a condo configuration. The letter argument can be **A**, **B**, **C**, or **D**, and can be specified in upper or lower case. Backplane A is closest to the floor. By default, the **scanio** command assumes the node numbers begin in the upper left corner of the last cabinet (the one furthest away from the diagnostic station). (Some sites have multiple Paragon systems within the same cabinet. Although this is not a supported configuration, the **scanio** command provides minimal support for this configuration.)
-e	Specifies that the **scanio** command should exit after the node has been silent for a few seconds. Keyboard input is not accepted, except a keyboard interrupt which kills **scanio**.
-f *file*	Specifies reading characters from the file specified by the *file* argument rather than from the keyboard. It's normally used to define boot variables during system booting.
-i	Ignores checking of the bootmagic string *BOOT_CONSOLE* for the scan interface specification. The **reset** command uses this feature.
-q	Suppresses messages. This is the opposite of the **-v** switch.
-s *string*	Specifies using the *string* argument to set bootmagic strings. This switch is similar to the **-f** switch except that the data contained in the *string* argument is used in place of keyboard input.
-t *state*	Tests the state of the INT_FROM_NODE line. The *state* argument can be either 1 or 0 (zero). When the *state* argument is set to 0, the **scanio** command returns 0 if the INT_FROM_NODE is high and 1 if it is low. When the *state* argument is set to `1`, the **scanio** command returns 1 if the INT_FROM_NODE is high, and 0 if it is low.

SCANIO *(cont.)* SCANIO *(cont.)*

-v	Turn verbosity on.
node	Node number of the node the console is connected to. The default is the boot node.

Description

The **scanio** command runs on the diagnostic station and is intended for use by the system administrator only.

The **scanio** command establishes serial communications with any node in the Paragon system using the scan lines rather than the mesh.

When you invoke the **scanio** command, the command examines the *bootmagic* file to determine the size and configuration of the Paragon system. The following variables are extracted from the *bootmagic* file:

BOOT_CONSOLE	This must be set to **s**. This tells the kernel that the console interface will use the scan lines.
BOOT_NODE_LIST	This variable determines which nodes the system uses. When you attempt to connect to a node, the node number you enter is validated against this list.
MESH_X	The number of cabinets is calculated as *MESH_X*/4.
MESH_Y	The number of backplanes is calculated as *MESH_Y*/4.
BOOT_FIRST_NODE	This variable determines the default node to connect to if the user does not specify a node.

NOTE

The speed of the **scanio** command is somewhere between 1200 and 2400 baud. Therefore, scanio is very slow.

SCANIO *(cont.)* SCANIO *(cont.)*

Node Numbering

Node numbers can be specified in CBS (Cabinet, Backplane, Slot) format (see the **cbs** manual page), or root node numbers. Root node numbers are the numbers used in the *bootmagic* file.

Node numbers are validated against the list of node numbers specified in the *bootmagic* file. You are not allowed to connect to any node other than the nodes defined by *BOOT_NODE_LIST*.

Internal Commands

The **scanio** command sends everything you type to the current node, and displays all text from the node to your standard output. You can enter special commands by typing a tilde (~) immediately after hitting the return key, then typing a special command character. The following command characters are currently supported:

~!	Allows you to invoke a system command. After the system command completes, control returns to the node. You can invoke a shell if you want to execute more than one command.
~#	Allows you to switch nodes. The **scanio** command displays a list of valid nodes along with a prompt. You can use root node numbers or CBS numbers. This list of valid nodes is displayed in root node numbering format.

Bugs And Problems

The **scanio** command runs between 1200 and 2400 baud. This happens because every read across the scan line requires a write. For more information, refer to IEEE 1149.1 1990 Standard.

The **scanio** command never sleeps. It spins between looking at the scan lines and looking at the input device (usually the keyboard). This happens because the node being used cannot interrupt the operating system on the diagnostic station when data is ready.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

async, cbs, fscan

SEC_CLOSE SEC_CLOSE

Prevents a user from logging in.

Syntax

sec_close *username_or_id* **...**

Arguments

username_or_id The username or user ID of the user whose account you want to close.

Description

Use the **sec_close** command to close a user's account. When a user's account is closed, that user cannot login. This command requires *root* privilege.

Return Values

Upon successful completion, the shell status variable returns 0. When an error occurs, the shell status variable returns 1.

Examples

1. To prevent user *ted* from logging in, enter the following:

```
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/16/95     1   12      0         4   open
# sec_close ted
'ted' closed
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/16/95     1   12      0         4 closed
```

SEC_CLOSE *(cont.)* SEC_CLOSE *(cont.)*

Errors

'*user*' not found

You tried to close an account that does not exist.

```
# sec_close mike
'mike' not found
```

Files

/etc/passwd.AGE	Binary file that contains aging information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **sec_create**, **sec_force**, **sec_max_logfails**, **sec_min_max**, **sec_nopasswd**, **sec_open**, **sec_reset**, **sec_show**

SEC_CREATE SEC_CREATE

Enables password security. Takes a */etc/passwd* file and creates */etc/passwd.AGE.*

Syntax

sec_create *initial_date min_age max_age max_logfails*

Arguments

initial_date	The last date when the passwords have been changed. The format is MM/DD/YY where MM, DD, and YY are integers representing the month, day and year. MM is one or two digits; DD is one or two digits; YY must be the last two digits of the year. *inital_date* cannot be a future date.
min_age	Integer specifying the minimum time in days after a password change, during which the password cannot be changed again. Must be greater than or equal to 1 and less than or equal to *max_age.*
max_age	Integer specifying the time in days when the password expires. Must be greater than or equal to 1 and less than or equal to 365.
max_logfails	The maximum number of consecutive login failures. When you login, you are requested for your password. If you enter an incorrect password, you once again receive a login prompt. You can enter *max_logfails* incorrect passwords before you are prevented from logging in. If you exceed *max_logfails*, you cannot login again until the system administrator issues a **sec_reset**. The recorded number of consecutive login failures is set to zero once you successfully login. The user *root* is not affected by *max_logfails. root* can always login, although the number of login failures is still recorded in */etc/passwd.AGE.*

Description

Use the **sec_create** command to set up password security. This command creates the file */etc/password.AGE.* This file is owned by *root* and has permissions 644. It is a binary file. To disable password security, either rename or remove */etc/password.AGE.* This command requires *root* privilege.

If *passwd.AGE* exists and you add a user with **adduser** or **vipw**, that user is assigned default password security characteristics. These are *min_age* = 7, *max_age* = 91, and *max_logfails* = 4. The new user is forced to change his or her password on the first login asttempt.

SEC_CREATE *(cont.)* SEC_CREATE *(cont.)*

Return Values

Upon successful completion, the shell status variable returns 0. When an error occurs, the shell status variable returns 1.

Examples

1. To enable password security, mark all current passwords with an initial date of 8/16/95, set *min_age* to 1, *max_age* to 10, and *max_logfails* to 12, enter the following. The example assumes 17 password entries. The number (in this case 17) includes entries that are nologin entries.

   ```
   # sec_create 8/16/95 1 10 12
   17 entries successfully created
   ```

2. To enable password security, mark all current passwords with an initial date of 4/27/95, set *min_age* to 2, *max_age* to 30, and *max_logfails* to 10, when a current */etc/passwd.AGE* exists, enter the following: Answer **y** to overwrite an existing */etc/passwd.AGE.*

   ```
   # sec_create 4/27/95 2 30 10
   '/etc/passwd.AGE' exists, overwrite (y/n) ? [n] y
   17 entries successfully created
   ```

Errors

`Today is` *date*`, but you specified '`*initial_date*`'`

You specified an *initial_date* that is in the future.

`Invalid month` *MM*`; need 1 <= month <= 12`

`Invalid year` *YY*`; need 70 <= year <= 95`

`Invalid min_age '`*min_age*`'; need 1 <= min_age <= max_age (`*max_age*`)`

`Invalid max_age '`*max_age*`'; need 1 <= max_age <= 365`

`Invalid max_logfails '`*max_logfails*`'; need 1 <= max_logfails <= 10`

SEC_CREATE *(cont.)* SEC_CREATE *(cont.)*

Files

/etc/passwd.AGE	Binary file that contains aging information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **sec_close**, **sec_force**, **sec_max_logfails**, **sec_min_max**, **sec_nopasswd**, **sec_open**, **sec_reset**, **sec_show**

SEC_FORCE SEC_FORCE

Forces a user to change his or her password on next login attempt.

Syntax

sec_force *username_or_id* **...**

Arguments

username_or_id	The username or user ID of the user whose password security characteristics you want to changed

Description

Use the **sec_force** command to force a user to change his or her password at the next login attempt. This command requires *root* privilege.

Return Values

Upon successful completion, the shell status variable returns 0. When an error occurs, the shell status variable returns 1.

Examples

1. To force the user *ted* to change his password at his next login, enter the following:

```
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/16/95     1   12      0         4  open
# sec_force ted
'ted' forced
# sec_show ted
name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259  ** FORCED *    3  200      0         2  open
```

SEC_FORCE *(cont.)* SEC_FORCE *(cont.)*

Errors

```
**NOT FOUND
```

You requested the password security characteristics of a user that does not exist.

```
# sec_show mike

name          uid  last change  min  max  fails  maxfails  status
====          ===  ===========  ===  ===  =====  ========  ======
mike          **NOT FOUND
```

Files

/etc/passwd.AGE Binary file that contains aging information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **sec_close**, **sec_create**, **sec_max_logfails**, **sec_min_max**, **sec_nopasswd**, **sec_open**, **sec_reset**, **sec_show**

SEC_MAX_LOGFAILS SEC_MAX_LOGFAILS

Sets the maximum number of login failures before an account is closed.

Syntax

sec_max_logfails *max username_or_id* **...**

Arguments

max	The maximum number of consecutive login failures. When you login, you are requested for your password. If you enter an incorrect password, you once again receive a login prompt. You can enter *max_logfails* incorrect passwords before you are prevented from logging in. If you exceed *max_logfails*, you cannot login again until the system administrator issues a **sec_reset**. The recorded number of consecutive login failures is set to zero once you successfully login. The user *root* is not affected by *max_logfails. root* can always login, although the number of login failures is still recorded in */etc/passwd.AGE.*
username_or_id	The username or user ID of the user whose password security characteristics you want to change.

Description

Use the command **sec_max_logfails** to set the maximum number of login failures a user can have before being locked out. This command requires *root* privilege.

Return Values

Upon successful completion, the shell status variable returns 0. When an error occurs, the shell status variable returns 1.

SEC_MAX_LOGFAILS *(cont.)*

Examples

1. To prevent a user from changing his or her password, enter the following:

```
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted           259    08/21/95     1   12      0         4  open
# sec_max_logfails 2 ted
'ted' changed
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted           259    08/21/95     1   12      1         2  open
```

Errors

`Illegal argument max: 'max'; need 1 <= max <= 10`

`'user' not found`

You tried to set the maximum number of login failures for a non-existent user.

```
# sec_max_logfails 2 mike
'mike' not found
```

Files

/etc/passwd.AGE	Binary file that contains aging information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

SEC_MAX_LOGFAILS *(cont.)* SEC_MAX_LOGFAILS *(cont.)*

See Also

commands: **sec_create**, **sec_close**, **sec_force**, **sec_min_max**, **sec_nopasswd**, **sec_open**, **sec_reset**, **sec_show**

SEC_MIN_MAX

Sets new values for *min_age* and *max_age*.

Syntax

sec_min_max_ *min_age max_age username_or_id* **...**

Arguments

min_age	Integer specifying the minimum time in days after a password change, during which the password cannot be changed again. Must be greater than or equal to 1 and less than or equal to *max_age*.
max_age	Integer specifying the time in days when the password expires. Must be greater than or equal to 1 and less than or equal to 365.
username_or_id	The username or user ID of the user whose password security characteristics you want to change.

Description

Use the command **sec_min_max** to set password security characteristics *min_age* and *max_age* for a particular user.

Return Values

Upon successful completion, the shell status variable returns 0. When an error occurs, the shell status variable returns 1.

SEC_MIN_MAX *(cont.)* SEC_MIN_MAX *(cont.)*

Examples

1. To change *min_age* and *max_age* for the user *ted*, enter the following:

```
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/21/95    1   12      0         2  open
# sec_min_max 2 13 ted
'ted' changed
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/21/95    2   13      0         2  open
```

Errors

`Invalid min_age '`*min_age*`'; need 1 <= min_age <= max_age (`*max_age*`)`

`Invalid max_age '`*max_age*`'; need 1 <= max_age <= 365`

`'`*user*`' not found`

You tried to set the maximum number of login failures for a non-existent user.

```
# sec_min_max 2 20 mike
'mike' not found
```

Files

/etc/passwd.AGE	Binary file that contains aging information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

SEC_MIN_MAX *(cont.)* SEC_MIN_MAX *(cont.)*

See Also

commands: **sec_create**, **sec_close**, **sec_force**, **sec_max_logfialas**, **sec_nopasswd**, **sec_open**, **sec_reset**, **sec_show**

SEC_NOPASSWD SEC_NOPASSWD

Prevents a user from changing his or her password.

Syntax

sec_nopasswd *username_or_id* **...**

Arguments

username_or_id The username user ID of the user whose password you want to freeze.

Description

Use the command **sec_nopasswd** to prevent a user from changing his or her passwd. The user *root* can always change another user's password, even if **sec_nopasswd** has been executed for that user. This command requires *root* privilege.

Return Values

Upon successful completion, the shell status variable returns 0. When an error occurs, the shell status variable returns 1.

Examples

1. To prevent the user *ted* from changing his password, enter the following:

```
# sec_nopasswd ted
'ted' changed
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted           259  *** ROOT **    1   12     0          4  open
```

Now if the suer ted tries change his password, he sees the following message:

```
% passwd
Changing password for ted.
You are not allowed to change your password. Please contact
your SysAdm !
```

SEC_NOPASSWD *(cont.)* SEC_NOPASSWD *(cont.)*

Errors

`'`*user*`' not found`

You tried to freeze the password of a non-existent user.

```
# sec_nopasswd mike
'mike' not found
```

Files

/etc/passwd.AGE	Binary file that contains aging information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **sec_create**, **sec_force**, **sec_max_logfails**, **sec_min_max**, **sec_open**, **sec_reset**, **sec_show**

SEC_OPEN SEC_OPEN

Allows a user to login after a **sec_close**.

Syntax

sec_close *username_or_id* **...**

Arguments

username_or_id The username or user ID of the user whose account you want to close.

Description

Use the **sec_open** command to open a user's account. When a user's account is closed, that user cannot login. **sec_open** restores the ability to login. This command requires *root* privilege.

Return Values

Upon successful completion, the shell status variable returns 0. When an error occurs, the shell status variable returns 1.

Examples

1. To allow the user *ted* to login (after having been locked out with **sec_close**), enter the following:

```
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/16/95    1   12      0         4 closed
# sec_open ted
'ted' opened
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/16/95    1   12      0         4   open
```

SEC_OPEN *(cont.)* SEC_OPEN *(cont.)*

Errors

`'`*`user`*`' not found`

You tried to open an account that does not exist.

```
# sec_open mike
'mike' not found
```

Files

/etc/passwd.AGE	Binary file that contains aging information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **sec_close**, **sec_create**, **sec_force**, **sec_max_logfails**, **sec_min_max**, **sec_nopasswd**, **sec_reset**, **sec_show**

SEC_RESET   SEC_RESET

Clears the number of consecutive login failures as recorded in */etc/passwd.AGE*.

Syntax

sec_reset *username_or_id* **...**

Arguments

username_or_id	The username or user ID of the user whose recorded number of login failures you want set to 0.

Description

Use the **sec_reset** command to set the recorded number of login failures of a particular user to 0. If you have exceeded the number of allowed login failures, you cannot login until the system administrator runs **sec_reset**. If you try, you get the following message:

```
% rlogin paragon

Too many login-failures for this account !

Connection closed.
```

This command requires *root* privilege.

Return Values

Upon successful completion, the shell status variable returns 0. When an error occurs, the shell status variable returns 1.

Examples

1. To clear the number of consecutive login failures for the user *ted*, enter the following:

```
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/16/95     1   12      4         4  open

# sec_reset ted
Logfails reset for 'ted'
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/16/95     1   12      0         4  open
```

Errors

`'`*`user`*`' not found`

You tried to reset an account that does not exist.

```
# sec_reset mike
'mike' not found
```

Files

/etc/passwd.AGE	Binary file that contains aging information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **sec_close**, **sec_create**, **sec_force**, **sec_max_logfails**, **sec_min_max**, **sec_nopasswd**, **sec_open**, **sec_show**

SEC_SHOW SEC_SHOW

Displays password security characteristics.

Syntax

sec_show [*username_or_id* **...**]

Arguments

username_or_id	The username or user ID of the user whose password security characteristics you want displayed. If this argument is omitted, the characteristics of all users are displayed.

Description

Use the **sec_show** command to display password security characteristics for all users or for specified users. This command requires *root* privilege.

Return Values

Upon successful completion, the shell status variable returns 0. When an error occurs, the shell status variable returns 1.

Examples

1. To display the password security characteristics of the user *ted*, enter the following:

```
# sec_show ted

name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/16/95    1   12    0          4   open
```

SEC_SHOW*(cont.)* SEC_SHOW *(cont.)*

2. To display the password security characteristics of the user *ted* and the user whose user ID is 5039, enter the following:

```
# sec_show ted 5039
name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
ted          259    08/16/95    1   12      0         4   open
t5          5039    04/27/95    2   30      0        10   open
```

3. To display the password security characteristics of all users, enter the following:

```
# sec_show
name        uid  last change  min  max  fails  maxfails  status
====        ===  ===========  ===  ===  =====  ========  ======
root           0    04/27/95    2   30      0        10   open
nobody        -2    04/27/95    2   30      0        10   open
daemon         1    04/27/95    2   30      0        10   open
sys            2    04/27/95    2   30      0        10   open
bin            3    04/27/95    2   30      0        10   open
who            4    04/27/95    2   30      0        10   open
tcb            9    04/27/95    2   30      0        10   open
adm           10    04/27/95    2   30      0        10   open
shane       2051    04/27/95    2   30      0        10   open
ghartog     1729    04/27/95    2   30      0        10   open
ellend       107    04/27/95    2   30      0        10   open
shala       2052    04/27/95    2   30      0        10   open
shane1      2053    04/27/95    2   30      0        10   open
shane2      2054    04/27/95    2   30      0        10   open
shane3      2055    04/27/95    2   30      0        10   open
sdh         5032    04/27/95    2   30      0        10   open
t5          5039    04/27/95    2   30      0        10   open
ted          259    08/16/95    1   12      0         4   open
```

SEC_SHOW *(cont.)* SEC_SHOW *(cont.)*

Errors

```
**NOT FOUND
```

You requested the password security characteristics of a user that does not exist.

```
# sec_show mike

name          uid  last change  min  max  fails  maxfails  status
====          ===  ===========  ===  ===  =====  ========  ======
mike          **NOT FOUND
```

Files

/etc/passwd.AGE	Binary file that contains aging information.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **sec_close**, **sec_create**, **sec_force**, **sec_max_logfails**, **sec_min_max**, **sec_nopasswd**, **sec_open**, **sec_reset**

SHOWFS SHOWFS

Displays disk space statistics and file system stripe attributes.

Syntax

showfs [-k] [**-q**] [**-t** *type*] [*filesystem* | *directory*]

Arguments

-k	Causes numbers to be reported in kilobytes. By default, all numbers are reported in 512-byte blocks.
-q	Quick listing; suppresses the listing of PFS stripe-directory pathnames. (See example below.)
-t *type*	Displays statistics for the specified file system type only. The available file system types include the following:

pfs	Parallel File System (PFS)
nfs	Network File System (NFS)
ufs	UNIX File System (UFS) (Berkeley fast file system)

If the **-t** switch is specified and a filesystem or directory is specified, the **-t** specification is ignored.

file_system	Pathname of a mounted file system.
directory	Pathname of a directory.

Description

The **showfs** command displays statistics on the amount of free disk space in a file system and the stripe attributes of a Parallel File System (PFS). If neither a file system or a directory is specified, statistics for all mounted file systems are displayed. The **showfs** command displays the following information about the specified file systems:

Mounted on File system's *mount point*: the directory on which the file system is mounted. To obtain the file system's device name, use the OSF/1 **mount** or **df** command.

SHOWFS *(cont.)*

512-blks	Total capacity of the file system, in 512-byte disk blocks, by default. The **-k** option can be used to display this in 1K byte blocks.
avail	Number of disk blocks currently available, in 512-byte disk blocks by default. The **-k** option can be used to display this in 1K-byte blocks.
capacity	Approximate percentage of the file system's capacity currently in use.

The **showfs** command displays the following additional information for each PFS file system:

sunit	File system's stripe unit size, in bytes. The unit of data interleaving in a PFS file.
sfactor	File system's stripe factor. The `sfactor` multiplied by the `sunit` equals the size of one PFS file stripe.
sdirs	Description of the file system's stripe group. The stripe group is a list of directories in UFS (typically UFS mount points) or NFS file systems that define the storage locations for the PFS file system.

The stripe attributes of individual PFS files can be displayed by using the **ls -P** command. See the **ls(1)** manual page for more information.

Examples

The following example shows file system information for a PFS file system with four stripe directories, a stripe unit size of 64K bytes, and a stripe factor of four.

```
% showfs -t pfs
Mounted on   512-blks     avail  capacity  sunit sfactor
/pfs          4128988   3689124     1%     65536       4
    sdirs: /home/.sdirs/vol0
           /home/.sdirs/vol1
           /home/.sdirs/vol2
           /home/.sdirs/vol3
```

SHOWFS *(cont.)*

The following example shows default PFS file system information:

```
# showfs /pfs
Mounted on   512-blks     avail  capacity   sunit sfactor
/pfs          3525912   2896292      9%    131072      32
    sdirs: /home/.sdirs/vol0/a
           /home/.sdirs/vol0/b
           /home/.sdirs/vol0/c
           /home/.sdirs/vol0/d
           /home/.sdirs/vol0/e
           /home/.sdirs/vol0/f
           /home/.sdirs/vol0/g
           /home/.sdirs/vol0/h
           /home/.sdirs/vol1/a
           /home/.sdirs/vol1/b
           /home/.sdirs/vol1/c
           /home/.sdirs/vol1/d
           /home/.sdirs/vol1/e
           /home/.sdirs/vol1/f
           /home/.sdirs/vol1/g
           /home/.sdirs/vol1/h
           /home/.sdirs/vol2/a
           /home/.sdirs/vol2/b
           /home/.sdirs/vol2/c
           /home/.sdirs/vol2/d
           /home/.sdirs/vol2/e
           /home/.sdirs/vol2/f
           /home/.sdirs/vol2/g
           /home/.sdirs/vol2/h
           /home/.sdirs/vol3/a
           /home/.sdirs/vol3/b
           /home/.sdirs/vol3/c
           /home/.sdirs/vol3/d
           /home/.sdirs/vol3/e
           /home/.sdirs/vol3/f
           /home/.sdirs/vol3/g
           /home/.sdirs/vol3/h
```

The following example shows the effect of the **-q** (quick listing) switch:

```
# showfs -q /pfs
Mounted on   512-blks     avail  capacity   sunit sfactor
/pfs          3525912   2896292      9%    131072      32
```

SHOWFS *(cont.)* SHOWFS *(cont.)*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

df, fstab(), getpfsinfo(), pfstab, statpfs()

OSF/1 Command Reference: **df(1), ls(1)**

OSF/1 Programmer's Reference: **fcntl(2), getmntinfo(3), mount(2), statfs(2)**

OSF/1 System and Network Administrator's Reference: **mount(8), quot(8)**

SHOWPART

Displays a partition's characteristics.

Syntax

showpart [**-f**] [**-w**] [**-p** | **-l**] [**-nt** *nodetype*] [*partition*]

Arguments

-f Displays the free nodes in a partition. For information on interpreting **-f** output, see the Description section of this manual page.

-l Displays the attributes of the nodes in each subpartition. In the ATTR column, the command displays the attributes common to every node in the subpartition. Node numbers are relative to the specified partition. These attributes appear in column format after the graphic display of the system.

-p Displays the attributes of the nodes in each subpartition. In the ATTR column, the command displays the attributes common to every node in the subpartition. Node numbers are relative to the *root* partition. These attributes appear in column format after the graphic display of the system.

-w Displays vertical and horizontal lines that show cabinet configuration. At the bottom of the display each cabinet number is shown. This argument does not affect the **-f**, **-l**, **-p**, and **-nt** switches.

-nt *nodetype* Displays the position of nodes of the specified type. The *nodetype* argument is one of the following:

attribute	Selects nodes having the specified attribute. The standard node attributes are shown in "Node Attributes" section. For example, the string **mp** selects only MP nodes.
!attribute	Selects nodes *not* having the specified attribute. No white space may appear between the **!** and the *attribute*. For example, the string **!io** selects only nodes that are *not* I/O nodes.

SHOWPART *(cont.)* SHOWPART *(cont.)*

[relop][value]attribute

Selects nodes having a specified value or range of values for the attribute:

The *relop* can be =, >, >=, <, <=, !=, or ! (!= and ! mean the same thing). If the *relop* is omitted, it defaults to =.

The *value* can be any nonnegative integer. If the *value* is omitted, it defaults to 1.

The *attribute* can be any attribute shown in the "Node Attributes" section, but is usually either **proc** or **mb**. (Other attributes have the value 1 if present or 0 if absent.)

No white space may appear between the *relop*, *value*, and *attribute*.

For example, the string **>=16mb** selects nodes with 16M bytes or more of RAM; **32mb** selects nodes with exactly 32M bytes of RAM (uses the default *relop*); **>proc** selects nodes with more than one processor.

ntype[,ntype]...

Selects nodes having *all* the attributes specified by the list of *ntype*s, where each *ntype* is a node type specifier of the form *attribute*, *!attribute*, or *[relop][value]attribute*. You can use white space (space, tab, or newline) on either side of each comma if you quote the entire string of *ntypes*. However, you cannot use white space within an *ntype*.

For example, the string **mp,32mb** or **"mp, 32mb"** selects MP nodes with exactly 32M bytes of RAM. The string **io,gp,>16mb** selects GP-based I/O nodes with more than 16M bytes of RAM. And, the string **io,!enet** selects I/O nodes that are *not* Ethernet nodes.

partition

Absolute or relative pathname of a partition. If you do not specify a partition argument, the command uses the value specified in the *NX_DFLT_PART* environment variable. If the *NX_DFLT_PART* environment variable is not set, the default is the *.compute* partition. You must have read permission on the specified partition.

SHOWPART *(cont.)* SHOWPART *(cont.)*

Description

The **showpart** command displays information about a specified partition. The displayed output consists of general and possibly attribute-specific information about the partition information displayed in a column format. This information is followed by a graphical display of the root partition.

The columns at the top of the **showpart** output provide the following information:

`USER`	The owner of the partition.
`GROUP`	The group of the partition.
`ACCESS`	The access permissions, expressed as an octal number. For example, 777 represents the permissions `rwxrwxrwx`.
`SIZE`	The number of nodes in the partition.
`FREE`	The number of unallocated, usable nodes in the subpartition. A node is *free* when no application is running on it.
`RQ`	The rollin quantum or scheduling type of the partition, as follows:

-	The partition uses standard scheduling.
`SPS`	The partition uses space sharing.
time	The partition uses gang scheduling with a rollin quantum of *time*. The *time* is expressed as a number followed by an optional letter: no letter for milliseconds, **s** for seconds, **m** for minutes, or **h** for hours.

`EPL`	The effective priority limit of the partition. A dash (`-`) indicates a standard-scheduled partition.
`ATTR`	The node attributes common to all the nodes in the partition. This column appears only with the **-l** and **-p** switches.

SHOWPART *(cont.)* SHOWPART *(cont.)*

After these columns of information, **showpart** displays a rectangle that represents the root partition. The numbers to the left of the rectangle show the root-partition node numbers for the nodes in the first column of each row of the root partition. Special characters show the specified partition's size, shape, and position in the root partition. The following special characters indicate information about the partition:

Dot (`.`)	Node that is not in the partition.
Asterisk (`*`)	Without **-f** or **-nt**: any node belonging to a partition. With **-f**: node that is not free. With **-nt**: node matching specified attributes.
`F`	With **-f** but not **-nt**: free node. With **-f** and **-nt**: free node matching specified attributes.
`f`	With **-f** and **-nt**: free node not matching specified attributes.
`o`	With **-nt**: node that belongs to the partition, but does not have the specified attributes.
Dash (`-`)	Without **-w**: an empty slot (unusable node). With **-w** and in line form: a logical break between backplanes.
`X`	Node that failed to boot (unusable node).
|	With **-w** and in line form: a logical break between cabinets.

Examples

To show the characteristics of the partition called *mypart*, whose parent partition is the *.compute* partition, you can use the following command:

```
% showpart mypart

USER     GROUP     ACCESS  SIZE         FREE  RQ   EPL
smith    eng          777     9            0  1h     5

     +---------+
    0| . . . . |
    4| . * * * |
    8| . * * * |
   12| . * * * |
     +---------+
```

SHOWPART *(cont.)* SHOWPART *(cont.)*

The *mypart* partition belongs to the user *smith* in group *eng*. It has permissions 777 (rwxrwxrwx), a size of 9 nodes, no free nodes, a rollin quantum of 1 hour, and an effective priority limit of 5. The rectangle at the bottom of the **showpart** output shows that the root partition is 4 nodes high by 4 nodes wide. The *mypart* subpartition is within the *root* partition and consists of nodes 5–7, 9–11, and 13–15 of the root partition.

The following command has the same output as the first example, but uses an absolute partition pathname:

```
% showpart .compute.mypart
```

To show the characteristics and the free nodes of the partition called *part1*, whose parent partition is the compute partition, you can use the following command:

```
% showpart -f part1

USER      GROUP      ACCESS  SIZE          FREE  RQ    EPL
smith     eng           777    15             6  SPS     5

     +---------+
    0| . . . . |
    4| . * * * |
    8| . * * * |
   12| . * * * |
   16| . F F F |
   20| . F F F |
     +---------+
```

The root partition is 6 nodes high by 4 nodes wide. The *part1* partition has a size of 15 nodes, 6 free nodes, and a rollin quantum set to space-sharing (SPS). The F characters show the location of the 6 free nodes.

The following example shows the positions of the 32M-byte nodes in the partition called *mypart*:

```
% showpart -nt 32mb mypart
 USER      GROUP     ACCESS  SIZE        FREE  RQ    EPL
 smith     eng          777     9           5  15m     5
    +---------+
   0| . . . . |
   4| . * * * |
   8| . o * * |
  12| . o o o |
    +---------+
```

Nodes in the partition having the attributes specified in the *nodetype* string are shown with an asterisk (*); other nodes within the partition are shown with a lowercase letter O (o).

SHOWPART *(cont.)* SHOWPART *(cont.)*

The following example shows the node attributes of the partition called *mypart*. Remember that node numbers are relative to the specified partition when using the **-l** switch:

```
% showpart -l mypart
 USER     GROUP    ACCESS  SIZE       FREE  RQ   EPL  ATTR
 smith    eng         777     9          5  15m    5  1proc,GP

0..2,4,5 1proc,32mb,GP
3,6..8 1proc,16mb,GP

    +---------+
   0| . . . . |
   4| . * * * |
   8| . * * * |
  12| . * * * |
    +---------+
```

If you use **-nt** together with **-f**, free nodes that match the *nodetype* string are shown with a capital `F`, while free nodes that do not match are shown with a lowercase `f`. For example:

```
% showpart -f -nt 32mb mypart
 USER     GROUP    ACCESS  SIZE       FREE  RQ   EPL
 smith    eng         777     9          5  15m    5
    +---------+
   0| . . . . |
   4| . * * * |
   8| . o F F |
  12| . f f f |
    +---------+
```

In the above example:

- Nodes 0, 1, and 2 of the partition (shown as asterisks) are 32M-byte nodes that are not free.
- Node 3 of the partition (shown as a lowercase letter `o`) is not a 32M-byte node and is not free.
- Nodes 4 and 5 of the partition (shown as capital `F`s) are 32M-byte nodes and are free.
- Nodes 6, 7, and 8 of the partition (shown as lowercase `f`s) are not 32M-byte nodes and are not free.

SHOWPART *(cont.)*   SHOWPART *(cont.)*

If you use **-w** on a multi-cabinet system, **showpart** separates the cabinets with vertical lines, separates the backplanes with horizontal lines, displays an offset at the top of each cabinet, and also displays the cabinet numbers at the bottom of the partition. These display enhancements are in addition to the information that other switches show. The following example displays a four cabinet, four backplane system:

```
% showpart -w
 USER      GROUP     ACCESS  SIZE       FREE  RQ    EPL
 smith     eng          777   512          5  15m     5

     |+0        |+4        |+8        |+12       |
     +----------------------------------------+
    0| * * * * | * * * * | * * * * | * * * * |
   16| * * * * | * * * * | * * * * | * * * * |
   32| * * * * | * * * * | * * * * | * * * * |
   48| * * * * | * * * * | * * * * | * * * * |
     +----------------------------------------+
   64| * * * * | * * * * | * * * * | * * * * |
   80| * * * * | * * * * | * * * * | * * * * |
   96| * * * * | * * * * | * * * * | * * * * |
  112| * * * * | * * * * | * * * * | * * * * |
     +----------------------------------------+
  128| * * * * | * * * * | * * * * | * * * * |
  144| * * * * | * * * * | * * * * | * * * * |
  160| * * * * | * * * * | * * * * | * * * * |
  176| * * * * | * * * * | * * * * | * * * * |
     +----------------------------------------+
  192| * * * * | * * * * | * * * * | * * * * |
  208| * * * * | * * * * | * * * * | * * * * |
  224| * * * * | * * * * | * * * * | * * * * |
  240| * * * * | * * * * | * * * * | * * * * |
     +----------------------------------------+
     | Cab  3  | Cab  2  | Cab  1  | Cab  0  |
```

Errors

`showpart: Partition not found`

You specified a partition that does not exist.

`showpart: Partition permission denied`

You do not have read permission for the specified partition.

SHOWPART *(cont.)* SHOWPART *(cont.)*

Files

/usr/bin/showpart
Specifies the command path.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

chpart, **lspart**, **mkpart**, **pspart**, **rmpart**

SNAMES SNAMES

Starts the *snames* server.

Syntax

snames [**-d**] [*command*]

Arguments

-d	Turns on debugging. The **snames** command displays messages on standard error when clients register and lookup ports.
command	Shell command that invokes the snames server as a secondary name service.

Description

CAUTION

Do not attempt to run this command. It is invoked once only, during system startup.

The **snames** command starts the *snames* server. The *snames* server is a Mach message server that does not know about TCP/IP networks and is an implementation of the *netname* name service.

By default, the *snames* server tries to become the primary name service for the system. In this mode, it takes no arguments. Otherwise, the *snames server* becomes a secondary name service. In this mode, it takes a command (often a shell) to run under the secondary name service.

Files

/mach_servers/snames
The snames daemon.

/usr/include/servers/netname.h
The header file that allows applications to register and allocate server names with the mach kernel.

SNAMES *(cont.)* SNAMES *(cont.)*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

machid

SPV SPV

Invoke the System Performance Visualization tool (SPV).

Syntax

spv [{**-f** | **-file** | **-fileName**} *file_name*]
[{**-h** | **-host** | **-hostName**} *host_name*]
X Toolkit parameters]

Arguments

{**-f** | **-file** | **-fileName**} *file_name*

The *file_name* parameter is the name of a file that contains previously-saved SPV data. The **-fileName** flag is the same name as the X resource used to specify a file name. By specifying a file, you can playback previously saved SPV data. Along with the main SPV display window, SPV also displays the Playback dialog box (see the **File Menu Open Playback** command).

{**-h** | **-host** | **-hostName**} *host_name*

The *host_name* parameter is the Internet machine or system name of the Paragon system. The **-hostName** flag is the same name as the X resource used to specify a host name. This parameter is used to specify the Paragon system displayed when you invoke SPV. If you invoke SPV on a Paragon system, the host name need not be specified.

X Toolkit parameters

The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*).

Description

SPV allows you to graphically display the performance of your Paragon system. You can display CPU, mesh, and memory bus utilization values. There are four displays available: CPU, mesh, values, and node. You can zoom between displays

CPU display	Shows CPU utilization values.
Mesh display	Provides a visualization of the Paragon front panel, showing both CPU and mesh utilization.

SPV *(cont.)* SPV *(cont.)*

Values display	Shows numerical values for CPU and mesh utilization.
Node display	Shows CPU, mesh, and memory bus utilization for an individual node.

The mouse buttons allow you to zoom back and forth between displays. The default settings for the buttons are as follows:

Left button	select
Middle button	return to previous display
Right button	display the utilization color mappings dialog

When you click on a node in the display area with the select button, SPV zooms to the node display for that individual node. You can examine the node, and then use the return button to zoom back to the previous display. This allows you to easily examine individual nodes from the CPU or mesh display.

The SPV main window contains the following features:

Title Bar	Lists the title of the tool, and the name of the Paragon system or the name of the SPV data file being replayed.
Menu Bar	Provides a set of pulldown menus to access all the features of SPV.
Status Line	Displays the current SPV status.
Display Area	Contains the visualization displays.

Resources

You can configure SPV by using a resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *Spv* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *Spv* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following application resources are provided to configure SPV:

Spv*fileName

The file opened by SPV to playback previously saved system data.

Spv*hostName

The Paragon system to which SPV connects and for which SPV displays system performance information.

SPV *(cont.)* SPV *(cont.)*

Spv*showUtilizationAs
Controls whether utilization values displayed are current, average, or maximum.

Spv*includeMsgPinCPUs
A boolean (True / False) value that controls whether or not the message-passing CPU is included in utilization calculations.

Spv*computeCPUutilizationAs
Controls whether utilization values displayed are average, minimum, or maximum.

Spv*updateTime
An integer value that controls how often SPV displays are updated.

Spv*zoomScale
A floating-point value that specifies the scale factor for zooming.

Spv*utilizationColors
A set of color names that controls the mapping of colors to utilization values.

Spv*utilizationStipples
A set of stipple names that controls the mapping of stipple patterns to utilization values.

Spv*partitionColors
A set of color names that controls the mapping of colors to partitions.

Spv*partitionStipples
A set of stipple names that controls the mapping of stipple patterns to partitions.

Spv*font1 through **Spv*font6**
A set of font names and sizes that specifies the size of fonts for the levels of zooming.

Spv*foreground
A color name that defines the foreground color for displays.

Spv*background
A color name that defines the background color for displays.

Spv*textForeground
A color name that defines the foreground color for text.

SPV *(cont.)* SPV *(cont.)*

Spv*textBackground
A color name that defines the background color for text.

Spv*alert_dialog.foreground
A color name that defines the foreground color for alert dialogs.

Spv*alert_dialog.background
A color name that defines the background color for alert dialogs.

Spv*selectColor
A color name that defines the color of selected objects.

Files

SPV uses the following files:

Spv	Application defaults file for SPV. This file defines resources that control the appearance and configuration parameters of SPV.
Spv.hlp	Online help text for SPV.

See Also

paraide, **xprof**, **xgprof**, **xipd**, **paragraph**

Paragon™ System Performance Visualization Tool User's Guide

TAR TAR

Manipulates tape archives.

Syntax

tar [-]*required_flag*[**bBEfFhilLmpPsSvw***n*] [**n** | **o**]
[*flag_argument* ...] [**-e** *exception*] ... [**-C** *directory*] ...
[*file* ...]

Arguments

The function performed by **tar** is specified by one of the following required flags:

c	Creates a new archive; writing begins on the beginning of the tape instead of after the last file. This command implies **r**.
r	Writes the named files on the end of the tape. The **c** function implies this.
t	The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.
u	Adds the named files to the tape, if the files are not already there or if they were modified since last copied to the tape.
x	Extracts the named files from the tape. If the named file matches a directory whose contents were written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier ones.

The following flags can be used with the required flag:

b	The **tar** command uses the next argument as the blocking factor for tape records. The default is 20 (larger values can be specified at the risk of creating a tape archive that some systems' tape drives might not be able to restore). Use this flag only with raw magnetic tape archives. The block size is determined automatically when reading tapes (key letters **x** and **t**).
B	Forces input and output blocking to the blocking factor (see the **b** flag). This flag exists so that **tar** can work across a communications channel where the blocking cannot be maintained.

TAR (cont.) TAR (cont.)

E	Uses a file format that supports extended files for the tar archives. Extended files are PFS files larger than 2G -1 bytes.
f	The **tar** command uses the next argument as the name of the archive instead of */dev/rmt[n]*. If the name of the file is **-** (dash), **tar** writes to standard output or reads from standard input, whichever is appropriate. Thus, **tar** can be used as the head or tail of a filter chain. **tar** can also be used to move hierarchies with the command: `cd fromdir; tar cf - . | (cd todir; tar xf -)`
F	Checks certain filenames before archiving. Source Code Control System (SCCS), Revision Control System (RCS), files named **core**, **errs**, **a.out**, and files ending in **.o** are not archived.
h	Forces **tar** to follow symbolic links as if they were normal files or directories. Normally, **tar** does not follow symbolic links, but instead saves the link text in the archive.
i	Ignores checksum errors. The **tar** command writes a file header containing a checksum for each file in the archive. When this flag is not specified, the system verifies the contents of the header blocks by recomputing the checksum and stops with a directory checksum error when a mismatch occurs. When this flag is specified, **tar** logs the error and then scans forward until it finds a valid header block. This permits restoring files from later volumes of a multivolume archive without reading earlier volumes.
l	Tells **tar** to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.
L	Tries to create a symbolic link if **tar** is unsuccessful in its attempt to link (hard link) two files.
m	Tells **tar** not to restore the modification times. The modification time will be the time of extraction. This is always the case with symbolic links.
n	Allows **tar** headers to be created with filenames that cannot be null-terminated if they are exactly the maximum length (as specified in POSIX). This flag is mutually exclusive with the **o** flag (that is, **n**ew vs. **o**ld). When specified, each of these flags turns off the other; neither flag is turned on by default, however.

TAR (cont.) TAR (cont.)

o

The **o** flag is provided for backward compatibility. Specify this flag if the archive will be restored on a system with an older version of **tar**. On output, **tar** normally places information specifying owner and modes of directories in the archive. Former versions of **tar**, when encountering this information will give an error message of the form

```
name/: cannot create
```

This flag suppresses the directory information. It also prevents archiving special files and FIFOs that earlier versions of **tar** would not be able to extract properly. (Note that although anyone can archive special files, only a user who has superuser authority can extract them from the archives.)

When **o** is used for reading, it causes the extracted file to take on the User and Group ID (UID and GID) of the user running the program, rather than those of the tape. This is the default for the ordinary user.

This flag is mutually exclusive with the **n** flag (that is, **n**ew vs. **o**ld). When specified, each of these flags turns off the other; neither flag is turned on by default, however.

p

Restores files to their original modes, ignoring the present **umask**. Set-user-ID and sticky information will also be restored if the user is superuser.

P

Specifies the prefix that is to be stripped off of the filenames archived to or extracted from tape. (See also the **s** flag.)

s

Tells **tar** to strip off any leading slashes from pathnames during extraction. This is useful when restoring a tape that was created on a system with a different file system structure. (See also the **P** flag.)

S *blocks***b** | *feet*[@*density*]

Specifies the number of 512-byte blocks per volume (first form), independent of the tape blocking factor. You can also specify the size of the tape in feet, and optionally density, by using the second form. Feet are assumed to be 11 inches long to be conservative. This flag lets you deal more easily with multivolume tape archives, where **tar** must be able to determine how many blocks fit on each volume.

Note that tape drives vary in density capabilities. The *density* argument is used in the amount of data that **tar** can fit on a tape.

TAR (cont.) **TAR** (cont.)

v	Normally **tar** does its work silently. The **v** (verbose) option makes **tar** print the name of each file it treats preceded by the function letter. With the **t** function, the verbose option gives more information about the tape entries than just their names.
w	Causes **tar** to print the action to be taken followed by the name of the file, and then to wait for the user's confirmation. If a word beginning with **y**, or the locale's equivalent of a **y**, is given, the action is performed. If any other input is given, the action is not performed.
n	Selects an alternate drive on which the tape is mounted by specifying a digit. The default is drive 0 (zero) at 1600 bpi, which is normally **/dev/rmt8**.
-e	Adds the following argument to a list of exception strings that prevent files whose names match exactly from being archived.
-C	If a filename is preceded by **-C**, **tar** performs a **chdir()** to that filename. This allows multiple directories not related by a close common parent to be archived using short relative pathnames. For example, to archive files from **/usr/include** and from **/etc**, one might use the following command line:

```
tar c -C . /usr/include -C /etc .
```

Therefore, if you do not specify an absolute filename, the filename is considered relative to the previous **-C** directory. When you specify this flag multiple times on the command line, make sure to specify subsequent **-C** directories relative to the preceding **-C** directories.

If an error occurs while trying to change to the requested directory, subsequent filenames on the command line that are not absolute (that is, have no leading / (slash)) are skipped until the next **-C** flag is specified.

Only the **-e** and **-C** flags must be preceded by a - (dash) and can be specified more than once on a single command line or interspersed within the list of filenames. All other flags must be specified together (with no separating spaces) before **-e**, **-C**, and the file list. For all flags that require arguments, the arguments must follow the string of flags and be ordered in the same way as the specified flags.

Previous restrictions on the **tar** command's ability to properly handle blocked archives have been lifted.

Description

The **tar** command saves and restores multiple files on a single file (usually a magnetic tape, but it can be any file). It saves and restores data from traditional format **tar** archives.

The actions of the **tar** command are controlled by a string containing, at most, one required flag and possibly one or more optional flags. Other arguments to **tar** are file or directory names specifying which files to dump or restore. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The **LC_MESSAGES** variable determines the locale's equivalent of **y** or **n** (for yes/no queries).

The **-E** switch works as follows when creating a tar archive:

- Using the **-E** switch allows files of any size file to be archived.
- Without the **-E** switch, extended files cannot be archived and a warning is displayed if an extended file is in the [*file*...] list.

On a Paragon system, extended files can be extracted from a tar archive with the following considerations:

- The **tar** command recognizes the extended format of the tar archive.
- The **-E** switch is not necessary.
- Files must be restored in file systems with adequate disk space.
- Extended files must be extracted into a Parallel File System (PFS).

On non-Paragon systems, extended files cannot be extracted from a tar archive created on a Paragon system with the **-E** switch. The **tar** command on non-Paragon systems does not support the extended file format. If at least one extended file is in the archive, an error will occur when extracting files from the archive. For example, the following error message may be displayed: "directory checksum error." If no extended files are in the tar archive, extracting files from a tar archive created on a Paragon system with the **-E** switch will succeed.

Examples

To create a **tar** archive to device */dev/rmt12*, enter:

```
tar cvfb /dev/rmt12 20 -e ./foo -C /usr/glenn -e ./bar \
-e ./logs/logfile -C /usr/gaston
```

The preceding command line specifies a blocking factor of 20. The resulting archive contains all files and directories in */usr/glenn* except for file **./foo** and all files and directories in */usr/gaston* except for *./foo* and *./bar* and *./logs/logfile*.

Notes

1. There is no way to ask for the *n*th occurrence of a file.
2. Tape errors are handled ungracefully.
3. The **u** option can be slow.
4. The current limit on filename length is 256 bytes. The current limit on file links (hard or soft) is 100 bytes.
5. There is no way selectively to follow symbolic links.
6. When extracting tapes created with the **r** or **u** options, directory modification times might not be set correctly.

Errors

```
tar: filename: skipping extended file (-E switch required)
```

The -E switch was not specified when creating a tar archive containing extended files. The extended file is not included in the archive. The *filename* is the name of the extended file.

Files

/dev/rmt*n*	Device name used with the *n* flag.
/tmp/tar*	Temporary file used with the **u** function.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

Functions: **chdir(2), umask(2)**

Files: **tar(4)**

TOP TOP

Displays and updates information about the top CPU processes.

Syntax

top [-s*time*] [*number*]

Arguments

-s*time*	Specifies the delay between screen updates to *time* seconds. The default delay between updates is 15 seconds.
number	Specifies the top *number* processes to display instead of the default amount of 10.

Description

The **top** command displays information about the processes that are the top users on the system. The command periodically updates this information. It uses raw CPU percentages to determine the top processes. The **curses(3)** package is used to do semi-optimal screen updating.

The first lines of the display show general information about the state of the system, including the last process ID assigned to a process, the three load averages, the current time, the number of existing processes, the number of processes in each state (sleeping, abandoned, running, starting, zombies, and stopped), and a percentage of time spent in each of the processor states (user, nice, system, and idle).

The remainder of the screen displays information about individual processes. This display is similar to **ps** command. The PID is the process ID, USERNAME is the name of the process's owner, PRI is the current priority of the process, NICE is the nice amount (in the range -20 to 20), SIZE is the total size of the process (text, data, and stack), RES is the current amount of resident memory (both SIZE and RES are given in kilobytes), STATE is the current state (one of "sleep", "WAIT", "run", "idl", "zomb", or "stop"), TIME is the number of system and user CPU seconds that the process has used, %WCPU is the weighted CPU percentage (this is the same value the **ps** command displays as CPU), %CPU is the raw percentage and is the field that is sorted to determine the order of the processes, and COMMAND is the name of the command that the process is currently running (if the process is swapped out, this column is marked **swapped**).

Files

/usr/bin/top	The **top** command path.

TOP *(cont.)* TOP *(cont.)*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

ps(1)

TRACEROUTE TRACEROUTE

Prints the route packets take to the network host.

Syntax

traceroute [**-m** *max_ttl*] [**-n**] [**-p** *port*]
[**-q** *nqueries*] [**-r**] [**-s** *src_addr*] [**-t** *tos*] [**-v**]
[**-w** *waittime*] *host* [*packetsize*]

Arguments

host	The destination hostname or IP number.
-m *max_ttl*	Maximum number of hops (time-to-live) in outgoing probe packets. The default number of hops is 30 (the same used for TCP connections).
-n	Prints hop addresses numerically rather than both symbolically and numerically. This method of printing saves a nameserver address-to-name lookup for each gateway found on the path.
-p *port*	The base UDP port number used during probes. The default base UDP port number is 33434. **Traceroute** uses a port range that starts with the default base UDP port number and ends with that number plus the number of hops minus one (*base* + (*hops* - 1)). In order for **traceroute** to succeed nothing can be listening on the UDP port range at the host. If something is listening on this range you can alter it by defining a new base UDP port number.
-q *nqueries*	The number of probes per time-to-live. The default number of probes is three.
-r	Bypasses the normal routing tables and causes **traceroute** to send directly to a host on an attached network. If the host is not on a directly-attached network, an error occurs. You can use this option to 'ping' a local host through an interface that has no route through it (e.g., after the interface was dropped by **routed (8C)**).
-s *src_addr*	The source IP address in outgoing probe packets. You must use an IP number (not a host's name) for *src_addr*. On hosts with more than one IP address, you can use this option to force the source address to be something other than the IP address of the interface on which the probe packet is sent. If the IP address is not one of this machine's interface addresses, an error is returned and nothing is sent.

TRACEROUTE *(cont.)*

-t *tos*	The *type-of-service* value in outgoing probe packets. You must supply a decimal integer in the range 0 to 255. The default value for *tos* is zero. You can use this option to see if different types-of-service result in different paths. Not all values of *tos* are legal or meaningful. For definitions, see the IP specification. Some useful values are 16 (low delay) and 8 (high throughput). Note that if you are not running 4.4bsd, setting the type-of-service value may have no effect since network services like **telnet** and **ftp** don't allow control of the type-of-service value.
-v	Specifies verbose output. With verbose output **traceroute** lists received ICMP packets other than TIME_EXCEEDED and UNREACHABLE.
-w *waittime*	The time in seconds to wait for a response to a probe. The default value for *waittime* is three seconds.
packetsize	The outgoing probe datagram length. The default size is 38 bytes.

Description

The Internet is a large and complex aggregation of network hardware, connected together by gateways. Tracing packet routes or tracking down the gateway that is discarding a packet can be difficult. **Traceroute** attempts to trace the route an IP packet follows to some internet host by launching UDP probe packets with a small *time-to-live* (ttl) and then listening for an ICMP "time exceeded" reply from a gateway. Probes begin with a ttl of one and increase by ones until an ICMP "port unreachable" response is received. This response indicates either the packet reached the host or a maximum number of hops was reached during the trace.

The program sends three probes for each ttl setting. It then prints the ttl setting, the address of the gateway and the round trip time for each probe. If the probe answers come from different gateways, the address of each responding system is printed. If there is no response within three seconds (or the time set with the **-w** switch) **traceroute** displays an "*" for that probe. If a host, network, or protocol is unreachable, **traceroute** displays a "!H", "!N", or "!P" after the time. If the source route fails a "!S" is printed. If a fragmentation is needed a "!F" is printed. Finally, if all the probes are resulting in some kind of unreachable condition, **traceroute** gives up and exits.

The only mandatory parameter is the destination host name or IP number. The default probe datagram length is 38 bytes, but this may be increased by specifying a packet size (in bytes) after the destination host name.

TRACEROUTE *(cont.)* TRACEROUTE *(cont.)*

For successful operation the destination host must not process the UDP probe packets. Thus, the default base UDP port number defaults to 33434. If by coincidence the destination host's port is set to this unlikely value, you can change it with the **-p** switch.

NOTE

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use **traceroute** during normal operations or from automated scripts.

Examples

```
% traceroute nis.nsf.net
traceroute to nis.nsf.net (35.1.1.48), 30 hops max, 56 byte packet
1 helios.ee.lbl.gov (128.3.112.1) 19 ms 19 ms 0 ms
2 lilac-dmc.Berkeley.EDU (128.32.216.1) 39 ms 39 ms 19 ms
3 lilac-dmc.Berkeley.EDU (128.32.216.1) 39 ms 39 ms 19 ms
4 ccngw-ner-cc.Berkeley.EDU (128.32.136.23) 39 ms 40 ms 39 ms
5 ccn-nerif22.Berkeley.EDU (128.32.168.22) 39 ms 39 ms 39 ms
6 128.32.197.4 (128.32.197.4) 40 ms 59 ms 59 ms
7 131.119.2.5 (131.119.2.5) 59 ms 59 ms 59 ms
8 129.140.70.13 (129.140.70.13) 99 ms 99 ms 80 ms
9 129.140.71.6 (129.140.71.6) 139 ms 239 ms 319 ms
10 129.140.81.7 (129.140.81.7) 220 ms 199 ms 199 ms
11 nic.merit.edu (35.1.1.48) 239 ms 239 ms 239 ms
```

In this first example the second and third lines are identical. This is due to a buggy kernel on the 2nd hop system (lbl-csam.arpa) that forwards packets with a zero ttl (a bug in the distributed version of 4.3BSD). Note also that you must guess what path the packets are taking cross-country since the NSFNet (129.140) does not supply address-to-name translations for its NSSes.

```
% traceroute allspice.lcs.mit.edu.
traceroute to allspice.lcs.mit.edu (18.26.0.115), 30 hops max
1 helios.ee.lbl.gov (128.3.112.1) 0 ms 0 ms 0 ms
2 lilac-dmc.Berkeley.EDU (128.32.216.1) 19 ms 19 ms 19 ms
3 lilac-dmc.Berkeley.EDU (128.32.216.1) 39 ms 19 ms 19 ms
4 ccngw-ner-cc.Berkeley.EDU (128.32.136.23) 19 ms 39 ms 39 ms
5 ccn-nerif22.Berkeley.EDU (128.32.168.22) 20 ms 39 ms 39 ms
6 128.32.197.4 (128.32.197.4) 59 ms 119 ms 39 ms
7 131.119.2.5 (131.119.2.5) 59 ms 59 ms 39 ms
8 129.140.70.13 (129.140.70.13) 80 ms 79 ms 99 ms
```

```
9 129.140.71.6 (129.140.71.6) 139 ms 139 ms 159 ms
10 129.140.81.7 (129.140.81.7) 199 ms 180 ms 300 ms
11 129.140.72.17 (129.140.72.17) 300 ms 239 ms 239 ms
12 * * *
13 128.121.54.72 (128.121.54.72) 259 ms 499 ms 279 ms
14 * * *
15 * * *
16 * * *
17 * * *
18 ALLSPICE.LCS.MIT.EDU (18.26.0.115) 339 ms 279 ms 279 ms
```

In this second example the gateways that are 12, 14, 15, 16 & 17 hops away either don't send ICMP "time exceeded" messages back or sends them with a ttl too small to reach the sending host. In this particular example the gateways that are 14 - 17 hops away are running the MIT C Gateway code that doesn't send "time exceeded" messages. s. The silent gateway 12 hops away may be the result of a bug in the 4.[23]BSD network code (and its derivatives): 4.x (x <= 3) sends an unreachable message using whatever ttl remains in the original datagram. Since, for gateways, the remaining ttl is zero, the ICMP "time exceeded" is guaranteed to not make it back. The behavior of this bug is slightly more interesting when it appears on the destination system as follows:

```
1 helios.ee.lbl.gov (128.3.112.1) 0 ms 0 ms 0 ms
2 lilac-dmc.Berkeley.EDU (128.32.216.1) 39 ms 19 ms 39 ms
3 lilac-dmc.Berkeley.EDU (128.32.216.1) 19 ms 39 ms 19 ms
4 ccngw-ner-cc.Berkeley.EDU (128.32.136.23) 39 ms 40 ms 19 ms
5 ccn-nerif35.Berkeley.EDU (128.32.168.35) 39 ms 39 ms 39 ms
6 csgw.Berkeley.EDU (128.32.133.254) 39 ms 59 ms 39 ms
7 * * *
8 * * *
9 * * *
10 * * *
11 * * *
12 * * *
13 rip.Berkeley.EDU (128.32.131.22) 59 ms ! 39 ms ! 39 ms !
```

Notice that there are 12 "gateways" (13 is the final destination) and exactly the last half of them are "missing". What's really happening is that rip (a Sun-3 running Sun OS3.5) is using the ttl from the arriving datagram as the ttl in its ICMP reply. So, the reply will time out on the return path (with no notice sent to anyone since ICMP's aren't sent for ICMP's) until a probe with a ttl that's at least twice the path length is sent. In this case, rip is really only seven hops away. A reply that returns with a ttl of 1 is a clue this problem exists. **Traceroute** prints an "!" after the time if the ttl is <= 1. Since vendors ship a lot of obsolete (DEC's Ultrix, Sun 3.x) or non-standard (HPUX) software, expect to see this problem frequently and/or take care when picking the target host of your outgoing probes.

See Also

netstat(1), ping(8)

VM_STAT VM_STAT

Displays statistics about the virtual memory (VM) for the Mach kernel.

Syntax

vm_stat [**-n** *node*] [*interval*]

Arguments

-n *node*	Specifies the node for which to display the Mach kernel. The *node* parameter must be the root-partition node number for an active node. The default is the node you are logged in to.
interval	Specifies the interval in seconds to display the statistics. If this argument is specified, the statistics are updated and displayed every *interval* seconds.

Description

By default with no arguments, the **vm_stat** command displays all accumulated statistics for the node including the following: the page size, object cache performance, and the size of the default pager's backing store. Each output line displays the *change* in each statistic.

The **vm_stat** command displays the following values when the *interval* argument is used:

free	Total number of free pages in the system
active	Total number of pages on the active list. Pages on the active list are currently in use and can be paged.
inactive	Total number of pages on the inactive list. Active pages are currently in use and pageable, but are the first to page out.
wired	Total number of pages wired down. Wired pages cannot be paged out.
faults	Number of faults.
copy	Number of faults that caused a page to be copied (caused by copy-on-write faults).
zeroed	Number of faults that caused a page to be zeroed (caused by zero-fill faults).
reactive	Number of pages that have been reclaimed from the inactive list.
pageins	Number of requests for pages from a pager.
pageouts	Number of pages that have been paged out

VM_STAT *(cont.)*   VM_STAT *(cont.)*

Examples

Enter the following to display virtual memory statistics for the system:

```
% vm_stat -n 16
Mach Virtual Memory Statistics @ node [16]: VM page size 8 Kb
Pages free:                   1175.
Pages active:                  702.
Pages inactive:               1619.
Pages wired down:              600.
"Translation faults":      2644280.
Pages copy-on-write:        252765.
Pages zero filled:          481097.
Pages reactivated:            3222.
Pageins:                     28348.
Pageouts:                     8351.
Object cache: 79951 hits of 84149 lookups (95% hit rate)
```

Enter the following to display virtual memory statistics for the system every 5 seconds:

```
% vm_stat 5
Mach Virtual Memory Statistics @ node [2]: (VM page size 8 Kb, cache hits 94%)
  free active inac wire   faults     copy zerofill reactive  pageins  pageout
  1138    683 1680  595  2657508   253401   484115     3222    29078     8393
  1393    637 1473  592      278       21       60        0       14       15
  1612    578 1327  579      518        5       93        0      191        0
  1612    578 1327  579       22        0        6        0        0        0
```

Every 5 seconds the **vm_stat** command displays updated statistics. The first two lines of output displays the banner. The first line of statistics displays the system-wide statistics. The lines that follow display the changes for each statistics.

Files

/usr/bin/vm_stat
Specifies the command path.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

VM_STAT *(cont.)* VM_STAT *(cont.)*

See Also

ms

WH WH

Searches for files on a search path.

Syntax

wh [-CELMPqRX] [**-p** *path*] [*--options*] **-f** name | *name* ...

Arguments

-C	Searches the path specified by the *CPATH* environment variable.
-E	Searches the path specified by the *EPATH* environment variable.
-f *name*	Searches the current path for files with the *name* parameter. The *file* argument specifies the name of the file to search for. Finds the *file* if the *name* parameter begins with a '-' character.
-L	Searches the path specified by the *LPATH* environment variable.
-M	Searches the path specified by the *MPATH* environment variable.
-p *path*	Searches the path specified by the *path* argument.
-P	Searches the path specified by the *PATH* environment variable. This is the default.
-q	Stops searching after it finds the first file matching the *name* parameter.
-R	Searches recursively through the directory trees rooted along the search path.
-X	Shows the file names in **emacs** error messages.
--options	Specifies command options to be passed to the **ls** command. This is for switches such as **-C** that are interpreted by the **wh** command.
name	Name of the file to search for.

WH *(cont.)* WH *(cont.)*

Description

The **wh** command searches directory paths for files whose names match specified patterns. By default, the **wh** command searches the path specified by the *PATH* environment variable. The command uses the **expand** command to do wildcard expansion in each directory on the path and invokes the **ls** command to display the names of the files found. The **wh** command is capable of searching alternate paths, stopping the search when a file name on the search path is matched, and displaying the file names found in a format recognized as error messages by the **emacs** editor.

The command line for the **wh** command consists of switches and file name patterns. The **wh** command processes its command line from left to right, so a switch applies to file names to the right of the switch only.

Environment Variables

You can set the following environmental variables to use as alternate search paths:

CPATH	Use this path to search for C include files.
EPATH	Use this path to search for **emacs** mocklisp files.
LPATH	Use this path to search for loader libraries.
MPATH	Use this path to search for UNIX online manual pages.
PATH	Use this path to search for programs to execute.

Examples

To search *PATH* for instances of **ls**, enter the following:

```
% wh ls
/usr/bin/ls
```

The command displays the pathname for the ls command file.

WH *(cont.)*	**WH** *(cont.)*

Assume that the *CPATH* environment variable is set as follows:

```
% setenv CPATH /usr/include
```

To search *CPATH* for the two files indicated, stopping at the first instance of each file found, enter the following:

```
% wh -qC stdio.h sys/types.h
/usr/include/stdio.h
/usr/include/sys/types.h
```

The command displays the pathname for the specified include files.

To use *CPATH* for all files with the extension *.h* and display a long listing for each file found, enter the following:

```
% wh -lC time.h
-rw-r--r--  1 root  system  4423 Jan 12 21:52 /usr/include/time.h
```

The command displays the long listing for the file name.

To search *CPATH* recursively for *types.h* (programs must `#include <sys/types.h>`), enter the following:

```
% wh -RC types.h
/usr/include/CC/cc/rpc/types.h
/usr/include/CC/cc/sys/types.h
/usr/include/rpc/types.h
/usr/include/sys/types.h
```

The command displays the recursive listing for the file name.

The **wh** command passes through formatting switches to **ls** using the **--** switch. To search *PATH* for all files whose names is "sh" and display a long, multi-column listing for each file found, enter the following:

```
% wh --lC sh
```

Caveats

The usual caveats about getting wild cards through the shell apply. You may need to quote file name patterns containing wild cards to prevent the shell from expanding them before passing them to the **wh** command.

The **wh** command searches only for file names, it ignores file modes (that is, file protection bits). The **wh** command may therefore return the name of a file that cannot be used as intended. For instance, searching the path specified by the *PATH* variable may find a file that is not executable or searching the path specified by the *CPATH* variable may find a file that is not readable. It is currently advisable to check the file mode manually by using the **ls** command with the **-l**.

Some of the **ls** command's formatting options (**-qfC**) collide with switches recognized by the **wh** command. These switches are accessible only via the escape mechanism (**--qfC**).

Searching the path specified by the *MPATH* variable generally requires the **-R** option because most entries are in subdirectories of the roots listed in *MPATH.* The **key** or **man** command are better for finding entries in the UNIX online documentation.

Using the **-q** switch with recursive searches using the **-R** switch is unpredictable because of the order of traversing the directory tree is not predictable. The depth of recursive searches is bounded by the number of directory files that can be opened simultaneously.

Files

/usr/mach/bin/wh
Specifies the command path.

Limitations and Workarounds

Wild-card expansion is disabled.

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

emacs(1), expand(3), key(1), ls(1), man(1), path(3), sh(1)

XGPROF XGPROF

Invoke XGprof, the graphical front end for **gprof**.

Syntax

xgprof [**-a**] [**-b**] [**-s**] [**-z**] [**-e** routine | **-f** routine] [**-E** routine | **-F** routine] [**-m** *gprofdir*] [**-rows** *numrows*] *X Toolkit parameters*]

Arguments

-a	Suppress printing statically-declared functions.
-b	Provide brief output.
-s	Produce a *gmon.sum* file.
-z	Display routines that have zero usage (time / number of calls).
-e *routine*	Suppress printing the graph profile entry for *routine* and all of its descendants. More than one **-e** option may be given.
-f *routine*	Print only the graph profile entry for *routine* and its descendants. More than one **-f** option may be given. **-f** overrides **-e** if both are included on the command line.
-E *routine*	Suppress printing the graph profile entry for *routine* and exclude the time spent in the routine (and its descendants) from the total. More than one **-E** option may be given.
-F *routine*	Print only the graph profile entry for *routine* and its descendants and also use only the times of the routines in total computations. More than one **-F** option may be given. **-F** overrides **-E** if both are included on the command line.
-m *gprofdir*	Specify *gprofdir* as the path name of the profile output directory to be initially read. This path name can either be an absolute path or a path relative to the current directory. *gprofdir* defaults to *gmon.out.*
-rows *numrows*	Display *numrows* rows in XGprof's profile list.

X Toolkit parameters

The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*). You can specify these parameters on the **xprof** command line.

XGPROF *(cont.)* XGPROF *(cont.)*

Description

XGprof provides a graphical front end to the **gprof** command. XGprof displays the contents of a profile output directory and allows you to select one or more profile files. XGprof displays a **gprof** report for the files in a scrolling text region. The XGprof main window provides the following features:

Title Bar	Lists the title of the tool.
Menu Bar	Provides a set of pulldown menus to access all the features of XGprof.
File List	Lists the contents of the profile output directory.

Resources

You can configure XGprof by using an X resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *XGprof* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *XGprof* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following XGprof application resources are provided to configure XGprof:

XGprof.listRows
An integer that controls the number of entries displayed at one time within the scrollable profile selection list.

XGprof.infoPathName
The default name of the profile directory that XGprof searches for when it begins execution.

XGprof.briefOutput
A boolean (True / False) value that chooses the *Brief output* display option in the settings dialog.

XGprof.zeroUsage
A boolean (True / False) value that chooses the *Display routines that have zero usage* display option in the settings dialog.

XGprof.staticRoutines
A boolean (True / False) value that chooses the *Display information about static routines* display option in the settings dialog.

XGprof.saveSummary
A boolean (True / False) value that chooses the *Save summary if file gmon.sum* display option in the settings dialog.

XGPROF *(cont.)* XGPROF *(cont.)*

Files

XGprof uses the following files:

XGprof	Application defaults file for XGprof. This file defines resources that control the appearance and configuration parameters of XGprof.
xgprof.hlp	Online help text for XGprof.

See Also

paraide, **xprof**, **xipd**, **paragraph**, **spv**

Paragon™ System Application Tools User's Guide

XIPD XIPD

Invoke XIPD, the graphical front end for the Interactive Parallel Debugger (IPD).

Syntax

xipd [*session_name*] [**-user** *account_name*]
[**-host** *paragon_name*] [**-**[**no**]**login**] [**-**[**no**]**debug**]
[**-**[**no**]**analysis**] [**-**[**no**]**symbols**]
[**-core**] [**-coreDir** *directory*]
[**-delay** *seconds*] [*X Toolkit parameters*]

Arguments

session_name The name of a previously-saved XIPD session. If the session file exists, XIPD uses the contents to pre-initialize certain aspects of XIPD. If the file does not exist, *session_name* is assumed to be the name of the Paragon system that XIPD should be used with.

-user *account_name*
The login account name to use for starting a new session on a Paragon system. Use this option when you are not restoring a session and you want to specify a login account name.

-host *paragon_name*
The name of the Paragon system on which XIPD is being used. Use this option when you are not restoring a session.

-[**no**]**login** [Do not] display the login panel. The default is **-login**.

-[**no**]**debug** [Do not] create interface elements specific for debugging applications (like setting breakpoints or stepping execution). The default is **-debug**.

-[**no**]**analysis** [Do not] create interface elements specific for instrumenting programs for performance analysis. The default is **-analysis**.

-[**no**]**symbols** [Do not] use symbol shapes to represent node status. The default is **-symbols**.

-core Causes XIPD to go directly to the Core Analysis dialog instead of the Load Application dialog.

-coreDir *directory*
Causes XIPD to go directly to the Core Analysis dialog and load the contents of the specified core directory.

-delay *seconds*	Establishes the time-out period for XIPD to wait for a response from IPD. This is rarely needed and typically only used when debugging applications that can deadlock when the **step** command is issued. After the given whole number of seconds (the default is 60), XIPD interrupts IPD if IPD has not responded to the XIPD command.
X Toolkit parameters	The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*).

Description

XIPD provides a graphical front end to the Interactive Parallel Debugger (IPD). XIPD graphically displays the status of parallel programs. XIPD also supports performance monitoring by assisting in selecting code to be instrumented for analysis by prof, gprof, and ParaGraph.The XIPD main window provides the following features:

Title Bar	Lists the title of the tool.
Menu Bar	Provides a set of pulldown menus to access all the features of XIPD.
Work Area	Provides an area for the display of dialogs, node status, routine lists, and program execution controls.
Button Panel	Contains push buttons that control the functions of XIPD.

Resources

You can configure XIPD by using an X resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *XIpd* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *XIpd* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following XIPD application resources are provided to configure XIPD:

XIpd.appForeground

A color name that defines the default foreground color for interface elements. This resource and *XIpd*foreground* should be the same.

XIpd.appBackground

A color name that defines the default background color for interface elements. This resource and *XIpd*background* should be the same.

XIpd*foreground
A color name that defines the default foreground color for all XIPD interface elements.

XIpd*background
A color name that defines the default background color for all XIPD interface elements.

XIpd.unloadedStatusForeground
A color name that defines the foreground color for unloaded nodes in the mesh.

XIpd.initializedStatusForeground
A color name that defines the foreground color for loaded but not yet executing nodes in the mesh.

XIpd.activeStatusForeground
A color name that defines the foreground color for nodes selected for load or for executing nodes.

XIpd.breakpointStatusForeground
A color name that defines the foreground color for nodes in the mesh at a break point.

XIpd.stoppedStatusForeground
A color name that defines the foreground color for nodes in the mesh that are stopped.

XIpd.blockedStatusForeground
A color name that defines the foreground color for nodes in the mesh that are blocked waiting for a message.

XIpd.terminatedStatusForeground
A color name that defines the foreground color for nodes in the mesh that have terminated execution.

XIpd.unloadedStatusStipple
A stipple name that defines the stippling pattern used when drawing an unloaded status node.

XIpd.initializedStatusStipple
A stipple name that defines the stippling pattern used when drawing an initialized status node.

XIpd.activeStatusStipple
A stipple name that defines the stippling pattern used when drawing an active status node.

XIpd.breakpointStatusStipple
A stipple name that defines the stippling pattern used when drawing a break point status node.

XIpd.stoppedStatusStipple
A stipple name that defines the stippling pattern used when drawing a stopped status node.

XIpd.blockedStatusStipple
A stipple name that defines the stippling pattern used when drawing a blocked status node.

XIpd.terminatedStatusStipple
A stipple name that defines the stippling pattern used when drawing a terminated status node.

XIpd.brightColor
A color name that defines the color for the bright part of the mesh lines drawn behind the nodes. The mesh is drawn with *XIpd.normColor* and *XIpd.darkColor* as well.

XIpd.normColor
A color name that defines the normal color for the mesh lines drawn behind the nodes. The mesh is drawn with *XIpd.brightColor* and *XIpd.darkColor* as well.

XIpd.darkColor
A color name that defines the color for the dark (shadow) part of the mesh drawn lines behind the nodes. The mesh is drawn with *XIpd.brightColor* and *XIpd.normColor* as well.

XIpd.sourceForeground
A color name that defines the color used for rendering the program text in the source code window.

XIpd.sourceBackground
A color name that defines the color used for the background of the program text in the source code window.

XIpd.breakpointForeground
A color name that defines the color used for the break point icons. The *Preferences* dialog sets this the same color as break point status.

XIpd.monitorpointForeground
A color name that defines the color used for the monitor point icons. The *Preferences* dialog sets this the same color as unloaded status.

XIpd.activeForeground
A color name that defines the color used for active code icons. The *Preferences* dialog sets this the same color as active status.

XIpd.generalFontName
A font name that defines the font used to render most of XIPD's text.

XIpd.codeWindowFontName
A font name that defines the font used to render the text in a source code window. *This should be a fixed width font.*

XIpd.outputFontName
A font name that defines the font used to render the text in windows that display output either IPD (like the pending send queue) or within the console input/output panel. *This should be a fixed width font.*

XIpd.monospaceFontName
A font name that defines the font for displaying highlighted routine names in the main panel's routine list.

XIpd.monospaceDimFontName
A font name that defines the font for displaying non-highlighted (dimmed) routine names in the main window's routine list.

XIpd.showSymbols
A boolean (True / False) value. If this resource is *True*, XIPD uses symbols as well as color to display the status of nodes in the viewpoint panel. If *False*, XIPD uses filled circles.

XIpd.nodeHeight
An integer that defines the height, in pixels, that XIPD draws nodes (this and *XIpd.nodeWidth* should be set to the same value).

XIpd.nodeWidth
An integer that defines the width, in pixels, that XIPD draws nodes (this and *XIpd.nodeHeight* should be set to the same value).

XIpd.geometry
A geometry string that controls the initial size and placement of XIPD.

XIPD *(cont.)* XIPD *(cont.)*

Environment Variables

NX_DFLT_PART	Name of the default partition that XIPD selects automatically within the mesh configuration dialog. If not defined (or invalid), XIPD selects the *.compute* partition.
XAPPLRESDIR	XIPD looks in this directory for the resource file *XIpd* when you modify XIPD resource settings through the *Preferences* dialog. If not defined, XIPD looks in your home directory.
XIPDHOME	Alternative home directory for XIPD session files. If not defined, XIPD stores the session files in your home directory.

Files

XIPD uses the following files:

XIpd	Application defaults file for XIPD. This file defines resources that control the appearance and configuration parameters of XIPD.
xipd.hlp	Online help text for XIPD.

See Also

paraide, xprof, xgprof, paragraph, spv

Paragon™ System Application Tools User's Guide

XPROF XPROF

Invoke XProf, the graphical front end for **prof**.

Syntax

xprof [**-t** | **-c** | **-a** | **-n**] [**-o** | **-x**] [**-l**] [**-z**]
[**-h**] [**-s**] [**-m** *profdir*] [**-rows** *numrows*]
X Toolkit parameters]

Arguments

-t	Sort report entries by decreasing percentage of total time. This is the default.
-c	Sort report entries by decreasing call count.
-a	Sort report entries by increasing symbol address.
-n	Sort report entries alphabetically by symbol name.
-o	Display symbol addresses in octal base.
-x	Display symbol addresses in hexadecimal base. This is the default.
-l	Include local (static-declared) symbols.
-z	Include all symbols, even those associated with zero number of calls and zero amount of time.
-h	Suppress the report header.
-s	Display a summary.
-rows *numrows*	Display the profile output directory's files within *numrows* rows. Ten rows is the default.
-m *profdir*	Specify *profdir* as the path name of the profile output directory to be initially read. This path name can either be an absolute path or a path relative to the current directory. *profdir* defaults to *mon.out*.

X Toolkit parameters

The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*). You can specify these parameters on the **xprof** command line.

XPROF *(cont.)* XPROF *(cont.)*

Description

XProf provides a graphical front end to the **prof** command. XProf displays the contents of a profile output directory and allows you to select a profile file. XProf displays a **prof** report for the file in a scrolling text region. The XProf main window provides the following features:

Title Bar	Lists the title of the tool.
Menu Bar	Provides a set of pulldown menus to access all the features of XProf.
File List	Lists the contents of the profile output directory.

Resources

You can configure XProf by using an X resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *XProf* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *XProf* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following XProf application resources are provided to configure XProf:

XProf.listRows
An integer that controls the number of entries displayed at one time within the scrollable profile selection list.

XProf.infoPathName
The default name of the profile directory that XProf searches for when it begins execution.

XProf.includeNonGlobals
A boolean (True / False) value that chooses the *Include non-globals* display option in the settings dialog.

XProf.includeAllSymbols
A boolean (True / False) value that chooses the *Include all symbols* display option in the settings dialog.

XProf.noReportHeader
A boolean (True / False) value that chooses the *No report header* display option in the settings dialog.

XProf.includeSummary
A boolean (True / False) value that chooses the *Include summary* display option in the settings dialog.

XProf.sortByTime
A boolean (True / False) value that chooses the *Total time* sort option in the settings dialog.

XProf.sortByCall
A boolean (True / False) value that chooses the *Number of calls* sort option in the settings dialog.

XProf.sortByAddress
A boolean (True / False) value that chooses the *Symbol address* sort option in the settings dialog.

XProf.sortByName
A boolean (True / False) value that chooses the *Symbol name* sort option in the settings dialog.

XProf.addressOctal
A boolean (True / False) value that chooses the *Octal* address option in the settings dialog.

XProf.addressHex
A boolean (True / False) value that chooses the *Hexadecimal* address option in the settings dialog.

Files

XProf uses the following files:

XProf	Application defaults file for XProf. This file defines resources that control the appearance and configuration parameters of XProf.
xprof.hlp	Online help text for XProf.

See Also

paraide, xgprof, xipd, paragraph, spv

Paragon™ System Application Tools User's Guide

ZPRINT ZPRINT

Displays data about the Mach kernel's named memory zones.

Syntax

zprint [-w] [-s] [-C] [-H] [*name*]

Arguments

–w	Displays how much space is allocated but not currently in use for each memory zone. This is the space wasted by the zone. Wasted space information appears for each zone in the right-most column of the output. Additionally, a summary of wasted space follows the listing of memory zone information.
–s	Produces a sorted output. With this switch **zprint** displays the memory zones in an order that begins with the zone that wastes the most memory and ends with the zone that wastes the least memory.
–C	Overrides the default columnar output. With this switch **zprint** produces a row-based output that contains current size, maximum size, element size, number of elements, and allocation size only.
–H	Overrides the default columnar headings. With this switch **zprint** does not label the columnar output. You may find this switch useful when sorting output by column.
name	A substring of one or more memory zone names. With this switch **zprint** displays information about memory zones whose names include *name* only.

Description

By default, **zprint** displays information about all memory zones. The information appears in a columnar output. You can use command-line switches to alter the amount of information reported and the way in which it appears.

The default columnar output includes element size, current size, maximum size, current number of elements, maximum number of elements, current elements in use, allocation size, and an allocation count. You can also use the **-w** switch to include a "wasted memory" column. **zprint** truncates long zone names with"$", and replaces spaces with "." characters. These modifications allow for sorting by column. Pageable and collectible zones appear with "P" and "C" on the far right. Zones with extremely large maximum sizes use "----" in the "max size" and "max num elts" fields.

ZPRINT *(cont.)* ZPRINT *(cont.)*

You can override the default columnar output by using the **-C** and **-H** switches. The **-C** switch produces an abbreviated list that is row-based. The **-H** output applies only when displaying information in the columnar format. It suppresses creation of column headings.

Unless directed otherwise, **zprint** prints out information about all memory zones. Specifying the *name* argument instructs **zprint** to look for and display zones whose names match *name* only.

Examples

This example displays information about every memory zone of the Mach kernel. Note that no wasted memory information is provided and the output appears in the default columnar form:

```
%zprint
                              elem    cur    max    cur    max   cur alloc alloc
zone name                     size   size   size  #elts  #elts inuse  size count
------------------------------------------------------------------------------
zones                           52     2K     8K     52    157    52    8K   157
objects                         80    17K   512K    227   6553   226   16K   204
   .
   .
   .
xmm.copy                        36     1K   512K     50  14563     8    8K   227
xmm.export                      24     1K   512K     50  21845     8    8K   341
```

This next example shows the same information along with wasted memory information. Note that the "zone name" column has been clipped in order to show the "wasted" column. In actual output, all columns are visible.

```
%zprint -w
             elem    cur    max    cur    max   cur alloc alloc
             size   size   size  #elts  #elts inuse  size count   wasted
------------------------------------------------------------------------------
               52     2K     8K     52    157    52    8K   157        0
               80    17K   512K    227   6553   226   16K   204      160
   .
   .
   .
               36     1K   512K     50  14563     8    8K   227     1512
               24     1K   512K     50  21845     8    8K   341     1008
TOTAL SIZE   = 613736
TOTAL USED   = 366988
TOTAL WASTED = 246748
```

ZPRINT *(cont.)* ZPRINT *(cont.)*

This next example overrides the default columnar output and also displays information on memory zones whose names contain the string "pager" only.

```
%zprint -C pager
device pager structures zone:
      cur_size:    0K bytes (2 elements)
      max_size:    40K bytes (1024 elements)
      elem_size:   40 bytes
      # of elems:  2
      alloc_size:  8K bytes (204 elements)
dev_pager port hash zone:
      cur_size:    0K bytes (4 elements)
      max_size:    16K bytes (1024 elements)
      elem_size:   16 bytes
      # of elems:  4
      alloc_size:  8K bytes (512 elements)
```

Files

/usr/mach/bin/zprint	**zprint** command.

See Also

machid, hostinfo

Files Manual Pages A

This appendix contains the manual pages for files on the Paragon system and the diagnostic station. All the manual pages in this appendix are online in the */usr/share/man/man4* directory on the Paragon system. The manual pages for diagnostic station files are in the */usr/man/cat.LOCAL* directory on the diagnostic station.

ALLOCATOR.CONFIG ALLOCATOR.CONFIG

Allocates nodes and controls access to partitions on a Paragon system.

Synopsis

/etc/nx/allocator.config

Description

The allocator reads the configuration file */etc/nx/allocator.config* for configuration information. This file contains configuration strings that you can set to specific values. For each string, you can set an unlimited value or accept the default (for strings which unlimited values do not apply) by doing one of the following:

- Omitting the line that would contain the configuration string.
- Commenting out the line that contains the configuration string with a pound sign (#).
- Failing to provide an allocator.config file by either removing or renaming it.

The *allocator.config* file contains the following strings:

SPACE_SHARE=boolean

Specifies whether space sharing is enforced. The *boolean* value specifies the following:

0	Gang-scheduling set for the system; gang-scheduling is allowed.
1	Space sharing set for the system; gang-scheduling is not allowed.

The factory default for the *SPACE_SHARE* parameter is 0 (zero).

This parameter is equivalent to the allocator switch **-tile** used in previous releases. You should use this parameter rather than the **-tile** switch.

See the **mkpart** and **chpart** manual pages for information about specifying scheduling for partitions.

ALLOCATOR.CONFIG *(cont.)* ALLOCATOR.CONFIG *(cont.)*

NUM_GANG_PARTS=integer

Specifies the maximum total number of gang-scheduled partitions allowed anywhere in the system. The value of *integer* must be an integer value greater than 0 (zero).

The factory default for the *NUM_GANG_PARTS* parameter is 1. Note that you cannot explicitly specify an unlimited number of gang-scheduled partitions through *integer*. To specify an unlimited value you must either omit this line, comment it out, or fail to provide the *allocator.config* file.

DEGREE_OF_OVERLAP=num

Specifies the maximum depth to which subpartitions or active entities (active applications or subpartitions) can overlap in a gang-scheduled partition. For example, if **integer** is 2, a single node can be allocated to at most two subpartitions and two active entities at the same time. The value of *num* must be an integer greater than 0.

The factory default for the *DEGREE_OF_OVERLAP* parameter is 2. Note that you cannot explicitly specify an unlimited degree of overlap through *num*. To specify an unlimited value you must either omit this line, comment it out, or fail to provide the *allocator.config* file.

MIN_RQ_ALLOWED=time

Specifies the minimum permissible rollin quantum. Enter *time* as an integer greater than 0 (zero). To operate without a minimum allowed rollin quantum, exclude the *MIN_RQ_ALLOWED* parameter. The argument *time* can be entered as follows:

n	*n* milliseconds. If *n* is not a multiple of 100, it is silently rounded up to the next multiple of 100. The value 0 (zero) cannot be used to specify "infinite" rollin quantum.
*n***s**	*n* seconds. For example, 90s specifies ninety seconds.
*n***m**	*n* minutes. For example, 20m specifies twenty minutes.
*n***h**	*n* hours. For example, 3h specifies three hours.

The *time* value must be less than or equal to 24 hours.

ALLOCATOR.CONFIG *(cont.)* ALLOCATOR.CONFIG *(cont.)*

The factory default for the *MIN_RQ_ALLOWED* parameter is **1h**. If you omit this string, comment it out, or fail to provide an *allocator.config* file, the default value is 100 milliseconds. Note that you cannot explicitly specify an unlimited rollin quantum through *time*. To specify an unlimited value you must either omit this line, comment it out, or fail to provide the *allocator.config* file.

REJECT_PLK=boolean

Specifies whether the application switch **-plk** can be used in a gang-scheduled partition. The **-plk** switch locks parts of each process into physical memory, which reduces paging and improves message-passing latency that can cause problems when the application is rolled out. The *boolean* value specifies the following:

0	The **-plk** switch is allowed in gang-scheduled partitions.
1	The **-plk** switch is not allowed in gang-scheduled partitions. If users request to run an application with the **-plk** switch in a gang-scheduled partition (or a partition with an ancestor that is gang-scheduled), the request is rejected.

The factory default for the *REJECT_PLK* parameter is 0.

USE_MACS=boolean

Specifies whether the allocator must validate user accounts with the Paragon Multi-User Accounting and Control System (MACS). This allows MACS to verify that users belong to valid MACS accounts. The *boolean* value specifies the following:

0	The allocator does not validate user account IDs with MACS.
1	The allocator must validate user account IDs with MACS.

The factory default for the *USE_MACS* parameter is 0 (this line is omitted from the *allocator.config* file by default).

This parameter is equivalent to the allocator switch **-MACS** used in previous releases. You should use this parameter rather than the **-MACS** switch.

ALLOCATOR.CONFIG *(cont.)*

ALLOCATOR.CONFIG *(cont.)*

Examples

The following example shows the default allocation configuration file:

```
SPACE_SHARE=0
NUM_GANG_PARTS=1
DEGREE_OF_OVERLAP=2
MIN_RQ_ALLOWED=1h
REJECT_PLK=0
```

The values in this allocator configuration file specify that gang-scheduling is allowed for the whole system, one gang-scheduled partition is allowed (probably the compute partition), a maximum overlap of two partitions or applications is allowed, a minimum rollin quantum of one hour is allowed, and no restriction on process locking exists.

Files

/etc/nx/allocator.config	Allocator configuration file.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

allocator, **initpart**

BADNODES.TXT BADNODES.TXT

Diagnostic station: Specifies nodes that are nonfunctional on a Paragon system.

Synopsis

/usr/paragon/boot/BADNODES.TXT

Description

The *BADNODES.TXT* file is on the diagnostic station and contains the list of nodes on a Paragon system that have failed or are nonfunctional. The system watchdog updates this file. For information about the system watchdog, see the **fscan** manual page.

The *BADNODES.TXT* file consists of lines that are either bad-node entries or comment lines. A bad node entry has the following format:

```
CBS-location <watchdog> description
```

The *CBS-location* is the node address in CBS notation and the *description* describes how the node failed. Information in a bad-node entry is separated by blank spaces. The following example shows a bad-node entry:

```
00A02 <watchdog> Node failed 3 times
```

This entry specifies that the bad node is at the CBS location 00A02 and the node failed three times.

A comment begins with a pound sign (#). All characters that follow the pound sign on a line are ignored.

The **bootpp** command reads the *BADNODES.TXT* file to find which nodes are bad nodes, and removes any entries for bad nodes from the bootmagic strings in the *bootmagic* file. The processor port on a bad node's corresponding MRC is disabled.

Files

/usr/paragon/boot/BADNODES.TXT
Specifies the nodes that are nonfunctional on a Paragon system. This file resides on the diagnostic station.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

bootpp, **cbs**, **fscan**

## BOOTMAGIC	BOOTMAGIC

Diagnostic station: Specifies bootmagic strings that control how a Paragon system boots.

Synopsis

/usr/paragon/boot/bootmagic

Description

The bootmagic file is on the diagnostic station and can be created or changed by the system administrator only.

The bootmagic file specifies how a Paragon system boots when it is reset. When a Paragon system is reset, the **bootpp** command reads the *BADNODES.TXT*, *MAGIC.MASTER*, and *SYSCONFIG.TXT* files and creates the *bootmagic* file.

The bootmagic file contains bootmagic strings that specify information about booting the system. A bootmagic string has the following format:

```
name=value
```

The *name* part of the bootmagic string indicates a booting or system configuration characteristic. The *value* part of the string specifies the value of the bootmagic string. The *value* can be a string, a node number, or a list of node numbers. For example, the following bootmagic string specifies the boot node for booting a Paragon system:

```
BOOT_FIRST_NODE=0
```

This specifies the system to use node 0 as the boot node.

You do not edit the bootmagic file. You change the value of a bootmagic string in the *bootmagic* file by including the bootmagic string in the MAGIC.MASTER file or in an alternate magic file. Changes are made to the bootmagic file when the **bootpp** command is executed during system reset. For example, changing the following bootmagic string in the *MAGIC.MASTER* file changes the bootmagic string in the bootmagic file when the system is reset:

```
BOOT_FIRST_NODE=7
```

This string changes the boot node to node 7 in the bootmagic file when the system is reset.

Bootmagic Strings

The following lists and describes the bootmagic strings:

ALLOCATOR_NODE=node
Specifies the node where the allocator runs. The default is the boot node.

BOOT_ALT_KERNEL_NAME=kernel
Specifies the pathname of the kernel that is broadcast to the nodes listed in the *BOOT_ALT_NODE_LIST* bootmagic string. The default is */mach_servers/sunmos*.

BOOT_ALT_NODE_LIST=node_list
Specifies a list of nodes on which to run the alternate operating system.

BOOT_COMPUTE_KERNEL_NAME=kernel
Specifies the pathname of the kernel file that is sent to the compute nodes. The default is */mach_servers/mach_kernel.compute*. If defined, this pathname is used instead of the value of *BOOT_KERNEL_NAME* bootmagic string.

BOOT_COMPUTE_NODE_LIST=node_list
Specifies the list of nodes that receive the "light" server and kernel. Compute nodes without any devices attached to them are defined in this list. The list is computed when the system is booted.

BOOT_COMPUTE_STARTUP_NAME=server
Specifies the pathname of the server that is sent to the compute nodes. The default is */mach_servers/startup.compute*. If defined, this pathname is used in place of *BOOT_STARTUP_NAME* bootmagic string.

BOOTMAGIC *(cont.)* BOOTMAGIC *(cont.)*

BOOT_CONSOLE=string

Specifies the console that all the system nodes use. The default is **cm**. The *string* argument specifies the kind of console to use. When a node is booted, the *string* is parsed from left to right until a valid console for the node is determined. The following characters are allowed in the *string* argument:

c	Use the serial line as the console if the node is the boot node. If the node is not the boot node, this character is skipped.
C	Use the serial line as the console if the node has a serial line. Otherwise, this character is skipped.
f	Use the fast scan interface (**fscan**) as the console and provide flow control on user applications.
F	Use the fast scan interface (**fscan**) as the console and provide flow control on user applications.
k	Stop the LED display while waiting for input.
l	Use the LED display while waiting for input.
m	Use the mesh as a console. Do not use this unless no other access is available.
s	Use the scan line as a console when looking for input only.
S	Always use the scan line as a console.

For example, the following specifies a boot console:

```
BOOT_CONSOLE=cm
```

The value **c** specifies using a serial line first. If the compute node does not have a serial line, the value **m** specifies using the mesh as the console interface.

BOOT_DISK_NODE_LIST=node_list

Specifies the nodes that have hard disks attached.

BOOT_EMULATOR_NAME=emulator

Specifies the absolute pathname of the emulator that is broadcast to all nodes and is loaded by the boot node. The default is */mach_servers/emulator*.

BOOTMAGIC *(cont.)* BOOTMAGIC *(cont.)*

BOOT_ENET_NODE_LIST=node_list
Specifies the list of nodes that are Ethernet nodes.

BOOT_FIRST_NODE=node
Specifies the first node that boots on the system (the boot node). The default is node number 0 (zero). The boot node receives the bootmagic strings from the scan line and receives the kernel from the network. This is a mandatory bootmagic string that must be in the *MAGIC.MASTER* file and the bootmagic file.

BOOT_GREEN_LED=string
Specifies when the green LED is turned on and off. Intel recommends that machines use the default value for this symbol. Reasonable values are strings from the set **ckhibwa**. Default is **Dciw**. Meanings are:

D	Intended for kernel debugging (e.g., set BOOT_GREEN_LED=D and call **led_green_[on, off]_debug**() so that debug events can be observed on the LEDs without interference. Production code should *not* use GREEN_DEBUG.
c	GREEN ON at start of **i860_init**() and GREEN OFF at end of **machine_startup**(). GREEN shows KDB input focus. i.e. OFF on entry into KDB, ON if cpu acquires the kdb lock, and OFF on exit from KDB.
h | i | b	(GREEN_HIT) | (GREEN_INTERVAL) | (GREEN_BAR) **pretty_lights**() (called by **master_cpu** every clock tick) has a mechanism to use all five LED's to measure the magnitude of something with a scheme such that it doesn't change faster than the eye can see. These three **pretty_lights**() options use different twists to measure user activity on a node.
w	If none of the **pretty_lights**() options above are enabled this will cause a "heart-beat" on the middle GREEN.
a	One GREEN LED for each CPU ON if the CPU is not in the IDLE loop upon clock tick, OFF otherwise. Thus, it is ON for both user and kernel activity.

BOOTMAGIC *(cont.)* BOOTMAGIC *(cont.)*

BOOT_HOWTO=hex_value
Specifies how the system is to be booted. The default is 0x0. The *hex_value* has the form **0x***hex_digits*, where *hex_digits* is a hexadecimal number. This value is reserved for debugging only. Do not modify this string.

BOOT_IO_NODE_LIST=node_list
Specifies the nodes that are to receive the "heavy" server and kernel. All service nodes and nodes that have attached devices (for example, a hard disk or HIPPI device) are specified. The list is computed when the system is booted.

BOOT_KERNEL_NAME=kernel
Specifies the pathname of the kernel that is loaded on the boot node and broadcast to the nodes on the mesh. The default is */mach_servers/mach_kernel.*

*BOOT_LOAD_SYMBOLS=***0** | **1**
Specifies whether to load the server symbols. A **1** (default) specifies to load the server symbols, while a **0** specifies not to load the server symbols.

BOOT_MESH_X=x_dimension
Specifies the number of slots in the x-dimension (left to right) for the mesh. This value is always a multiple of 4. The **bootpp** command computes this value when it is not specified.

BOOT_MESH_Y=y_dimension
Specifies the number of slots in the y-dimension (top to bottom) for the mesh. This value is always a multiple of 4. The **bootpp** command computes this value when it is not specified.

BOOT_MY_NODE=node
Specifies a node's node number. You cannot change this number. The **bootmesh** command updates this value prior to broadcasting the bootmagic file to each node. Each node has a unique number.

BOOT_NODE_LIST=node_list
Specifies every node in the system. The **bootpp** command computes this value.

BOOT_NUM_NODES=num_nodes
Specifies how many nodes (not slots) are in the system. The **bootpp** command computes this value.

BOOTMAGIC *(cont.)* BOOTMAGIC *(cont.)*

BOOT_RED_LED=string

Specifies when the red LED is turned on and off. Intel recommends that machines use the default value for this symbol. Reasonable values are strings from the set **DcsiglXRme**. Default is **Dcew**. Meanings are:

D	Intended for kernel debugging (e.g., set BOOT_RED_LED=D and call **led_red_[on, off]_debug**() so that debug events can be observed on the LEDs w/o interference. No production code should use RED_DEBUG.
c	If set, green on at start of **i860_init**() and green off at end of **machine_startup**().
s	Used for scan driver debugging.
i	ON during Interrupt service, off otherwise.
g	ON if KDB is doing input. redundant uses in the various console drivers should be deleted.
l	Called only by **roll_led_out**(), which is called only by the mesh "multicomputer" console driver.
X	Called by **mcmsg_nic_check**() (called by **master_cpu** every clock tick) if NIC transmit-in-progress flag is set.
R	Called by **mcmsg_nic_check**() (called by master_cpu every clock tick) if NIC receive-in-progress flag is set.
m	Various uses in the MCP code.
e	ON upon ECC Correctable Errors. Also can be used for other handled error conditions in conjunction with additional bootmagic.

BOOT_REMOTE_KERNEL=address

Specifies the system address and pathname of the remote kernel that is used to boot the boot node from the diagnostic station. The *address* value has the form *IP_ADDRESS:pathname*. This bootmagic string is created automatically by the **reset** command if the string is not specified.

BOOTMAGIC *(cont.)* BOOTMAGIC *(cont.)*

BOOT_ROOT_DEV=device
: Specifies the name of the device containing the *root* file system. The default is *rz0a*. This value must be *md0a* when booting with the RAM disk.

BOOT_STARTUP_NAME=server
: Specifies the absolute pathname of the server that is broadcast to all nodes and loaded by the boot node. The default is */mach_servers/startup*.

BOOT_TIME=time
: Specifies the current time. This time is used to set the system clock. The value is retrieved using the **time(8)** system call on the diagnostic station.

EXPORT_PAGING=node_list
: Specifies the nodes that will provide paging for other nodes. The default is the boot node, which means only the boot node will provide paging services.

NETSERVER=node
: Specifies the node that provides the network services. The default is the boot node.

PAGER_NODE=pager_list
: Specifies which nodes use which pager. The default is that all nodes use the boot node as their pager. The *pager_list* argument has the format:

```
<node_list>pager[:<node_list>pager] ...
```

*RB_MULTIUSER=***0** | **1**
: Specifies booting the system in either single-user or multiuser mode. A **0** (default) indicates booting the system in single-user mode, while a **1** indicates booting the system in multiuser mode.

ROOT_DEVICE_NODE=node
: Specifies the node that contains the *root* file system. The default is the boot node.

ROOT_FS_NODE=node
: Specifies the node that provides the services for accessing the *root* file system. The default is the boot node.

BOOTMAGIC *(cont.)* BOOTMAGIC *(cont.)*

TCP_SPACE_SIZE=buffer_size

Specifies the default socket buffer space available to TCP connections on the Paragon. Recommended values for *buffer_size* are:

65536	ethernet only systems
262144	systems that have a HIPPI/FDDI
524288	systems that have a HIPPI

Use of this bootmagic string also enables the Paragon system to use window scaling with other systems employing window scaling. Note that the effect of *TCP_SPACE_SIZE* is system wide as it optimizes TCP/IP performance over the HIPPI channel.

BOOTMAGIC *(cont.)* BOOTMAGIC *(cont.)*

Examples

The following shows an example *bootmagic* file.

```
BOOT_FIRST_NODE=3
BOOT_CONSOLE=f
BOOT_STARTUP_NAME=/mach_servers/startup
BOOT_EMULATOR_NAME=/mach_servers/emulator
BOOT_KERNEL_NAME=/mach_servers/mach_kernelb
BOOT_ROOT_DEV=rz0a
BOOT_HOWTO=0x0
BOOT_COMPUTE_STARTUP_NAME=/mach_servers/startup.compute
BOOT_COMPUTE_KERNEL_NAME=/mach_servers/mach_kernelb
BOOT_ALT_KERNEL_NAME=/mach_servers/sunmos
BOOT_ALT1_KERNEL_NAME=/puma/sys/puma
BOOT_ARCH=paragon
BOOT_MY_NODE=3
ROOT_FS_NODE=3
ROOT_DEVICE_NODE=3
NETSERVER=3
ALLOCATOR_NODE=3
BOOT_NUM_NODES=26
BOOT_MESH_X=4
BOOT_MESH_Y=12
BOOT_CAB_ROWS=1
BOOT_IO_NODE_LIST=0..4
BOOT_COMPUTE_NODE_LIST=6..7,11,15,19..35
BOOT_NODE_LIST=0..4,6..7,11,15,19..35
EXPORT_PAGING=3
PAGER_NODE=<0..2,4..35>3
BOOT_TIME=826138339
BOOT_REMOTE_KERNEL=137.46.105.100:/usr/paragon/boot/mach_kernelb
BOOT_DISK_NODE_LIST=0..4
BOOT_ENET_NODE_LIST=3
```

BOOTMAGIC *(cont.)*　　　　BOOTMAGIC *(cont.)*

Files

/usr/paragon/boot/BADNODES.TXT
: Specifies the nodes that are nonfunctional on a Paragon system. This file resides on the diagnostic station.

/usr/paragon/boot/bootmagic
: Specifies information for booting a Paragon system. This file resides on the diagnostic station.

/usr/paragon/boot/DEVCONF.TXT
: Specifies the device configuration for a Paragon system. This file resides on the diagnostic station.

/usr/paragon/boot/MAGIC.MASTER
: Specifies the master version of the bootmagic strings for booting a Paragon system. This file resides on the diagnostic station.

/usr/paragon/boot/SYSCONFIG.TXT
: Specifies the hardware configuration for a Paragon system. This file resides on the diagnostic station.

See Also

calls: **time(8)**

commands: **bootpp**, **reset, core**

files: **BADNODES.TXT**, **DEVCONF.TXT**, **MAGIC.MASTER**, **SYSCONFIG.TXT**

CORE CORE

File or directory produced by faulting process(es).

Synopsis

include <*sys/core.h*>

Description

The operating system writes a core file when a process in an application terminates with an error. A process that terminates with an error is called a faulting process. The following support is provided for core files:

- Core files are created for parallel and non-parallel applications.
- Environment variables control where and how core files are generated.
- The **coreinfo** command displays information about core files.

The most common errors that generate core files for applications are memory violations, illegal instructions, bus errors, and user-generated quit signals. The following signals may result in core files being written: SIGQUIT, SIGILL, SIGTRAP, SIGABRT, SIGFPE, SIGBUS, SIGSEGV, SIGSYS, SIGIOT, and SIGEMT. See the **signal(4)** manual page for more information.

If a faulting process is in a regular non-parallel application, the operating system writes a core file named *core*. If a faulting process is in a parallel application, the following happens by default:

- The entire application is terminated.
- A directory named *core* is created.
- In the *core* directory, the operating system creates a core file for the first faulting process. The core file gets a name with the following form: ***core.pid***, where *pid* is the process ID (PID) for the process.
- In the *core* directory, the operating system writes the file named *allocinfo* containing global information about the parallel application.

If a core file or directory exists at the time of a fault, it is silently removed (if permissions allow) and a new one created. A message is displayed only if there is insufficient file space to write the core file or if the *core* directory was created but cannot be written into due to access permission.

If the core action environment variables are set such that more than one faulting process dumps core, the bootmagic string *PARACORE_MAX_PROCESSES* limits how many processes dump core.

Specifying the Location of Core Files

By default, the core file or directory is created in the current working directory for the application. The environment variable *CORE_PATH* can be used to change this default location.

CORE_PATH	Directory pathname where the *core* file or directory is created. The default is the working directory in which you execute an application.

Since it is possible to change directories or to change an environment variable from within an application, the following rules apply as to their effect on the placement of core:

- For non-parallel applications, any change of directory or environment variable (for example, **chdir**(), or **putenv**()) that occurs prior to the fault can effect the location of core.

- For parallel applications (compiled with **-nx**), any change of directory or environment variable within the application does not effect the location of core.

- For host/node applications (compiled with **-lnx**), any change of directory or environment variable that is performed by the host prior to a fault in any process can effect the location of core.

The following example specifies that a directory */usr/develop/corefiles* is where core files or directories should be placed:

```
% setenv CORE_PATH /usr/develop/corefiles
```

NOTE

Writing core files to a PFS directory is not allowed. If *CORE_PATH* specifies a directory in a PFS file system, a core file or core-file directory will NOT be created when an application faults.

Defining Core File Types, Frequencies, and Actions

More than one process in a parallel application may terminate with an error. This happens because there is a period of time between when the first faulting process terminates and the other processes in the application terminate. You can use the following environment variables to specify how and when core files are created:

CORE *(cont.)*

CORE_ACTION_FIRST
Specifies how the operating system handles the first faulting process in a parallel or non-parallel application.

CORE_ACTION_FAULT
Specifies how the operating system handles faulting processes other than the first faulting process in a parallel application.

CORE_ACTION_OTHER
Specifies how the operating system handles non-faulting processes in a parallel application.

You can specify the following values for the environment variables *CORE_ACTION_FIRST*, *CORE_ACTION_FAULT*, and *CORE_ACTION_OTHER*:

FULL	Creates a full core file that includes the entire data region. Default action for first faulting process (both parallel and non-parallel applications).
TRACE	Creates a partial core that includes the register and stack information only.
KILL	Stops the application without creating core files. Default action for faulting (except the first faulting process) and non-faulting processes.

You can also specify the following value for the environment variable *CORE_ACTION_OTHER*:

CONT	Continue executing. Do not stop or create core files.

The following example specifies how core files or core-file directories are created:

```
% setenv CORE_ACTION_FIRST FULL
% setenv CORE_ACTION_FAULT TRACE
% setenv CORE_ACTION_OTHER CONT
```

This example specifies creating a full core file for the first faulting process in an application, a core file with trace information only for other faulting files, and no core files for non-faulting processes.

The default values for the environment variables maximize the debug information available and minimize the file space needed for core files. The environment variables allow you to adjust the amount of core dumped based on your knowledge of the application size (number of processes used) and the file space available.

CORE *(cont.)* CORE *(cont.)*

Use the following guidelines when setting the environment variables for core files or directories:

- The **CONT** action can be specified only for *CORE_ACTION_OTHER*. If **CONT** is specified for either *CORE_ACTION_FIRST* or *CORE_ACTION_FAULT*, the error will not be caught until an application faults. When the application faults, a warning is displayed on the console and the default action for each core action environment variable is assumed.

- If *CORE_ACTION_FIRST*, *CORE_ACTION_FAULT*, and *CORE_ACTION_OTHER* are all set to not create core files (that is **KILL** or **CONT**), then no core file or core directory is created and an existing one is untouched.

- For parallel applications, combinations where *CORE_ACTION_FIRST* is set to **KILL** but *CORE_ACTION_FAULT* and *CORE_ACTION_OTHER* are not set to **KILL** may result in a core directory being created with an *allocinfo* file and nothing else. For example, if *CORE_ACTION_FIRST* and *CORE_ACTION_OTHER* are set to **KILL** and *CORE_ACTION_FAULT* is set to **FULL** when a single process in an application faults, a core directory is created with nothing but an *allocinfo* file.

- For parallel applications executing on a system that has the *PARACORE_MAX_PROCESSES* bootmagic string set to 1 (the default), only *CORE_ACTION_FIRST* is effective. The system ignores *CORE_ACTION_FAULT* and *CORE_ACTION_OTHER*. Under these conditions, the first faulting process is the only process that can dump core.

- For parallel applications executing on a system that has the *PARACORE_MAX_PROCESSES* bootmagic string set to a value greater than one, the system applies *CORE_ACTION_FAULT* and *CORE_ACTION_OTHER* only when the application size is less than or equal to the bootmagic string's setting.

- For parallel applications executing on a system that has the *PARACORE_MAX_PROCESSES* bootmagic string set to -1 (unlimited), the system applies *CORE_ACTION_FIRST*, *CORE_ACTION_FAULT* and *CORE_ACTION_OTHER* as described in the first four bulleted text items.

Maximum Size for Core Files

The maximum size of each individual core file is limited by the **setrlimit()** function. Core files that exceed the limit are not created.

Examining Core Files

You can use the **coreinfo** command and the **IPD** tool to examine core files. The **coreinfo** command displays summary information about processes that have dumped core. The **IPD** tool displays stack tracebacks and data stored in core files.

See the **coreinfo(1)** and **ipd** manual pages for more information about this command and tool.

Core File Format

The format and a brief description of the content of a core file is described below. Refer to the indicated include files for structure details.

Core header Identifies the core file, the process that generated the file (relative to the partition), and the offset from the start of the core file to the beginning of each of the other sections in the core file. The following structure is defined in the include file *core.h*.

```
struct core_hdr {
     int        c_magic;          /* core file magic number */
     unsigned short c_swap;       /* byte swap field */
     short      c_version;        /* core file version */
     int        c_type;           /* core file type */
     struct timeval c_timdat;     /* time & date core file created */
     int        c_signo;          /* signal that killed process */
     int        c_sigcode;        /* signal subcode */
     long       c_numnodes;       /* number of nodes in the partition */
     long       c_node;        /* logical node number of the dumping process */
     long       c_ptype;          /* ptype of the dumping process */
     off_t      c_procinfo;       /* offset of process info */
     off_t      c_applinfo;       /* offset of application info (APPLINFO_T) */
     long       c_nregions;       /* number of region descriptors */
     off_t      c_firstreg;       /* offset of first region desc in core file */
     long       c_nthreads;       /* number of active threads */
     off_t      c_firstthread;    /* offset of first thread_info in core file */
     long       c_activethread;   /* index of last active (faulting) thread */
};
```

Process information

Identifies the process (relative to other processes), the executable, and its arguments. The program name is stored as it is given to the **exec** system call, thus, the root and current directory information is needed so that a full path name can be constructed. Offsets are relative to the start of the core file. The following structure is defined in the include file *core.h*.

CORE *(cont.)* CORE *(cont.)*

```
struct core_proc_info {
     pid_t     c_pid;       /* process id */
     pid_t     c_ppid;      /* parent process id */
     pid_t     c_pgid;      /* process group leader id */
     long      c_prglen;    /* length of program path */
     off_t     c_prgname;   /* offset of relative path name of program */
     long      c_rootlen;   /* length of exec root directory path */
     off_t     c_rootname;  /* offset of exec root directory path */
     long      c_cwdlen;    /* length of exec current directory path */
     off_t     c_cwdname;   /* offset of exec current directory path */
};
```

Application information

Identifies the execution control characteristics. All other application information and partition information is contained in a separate file called *allocinfo* written in the *core* directory. The following structure is defined in the include file *mcmsg/mcmsg_appl.h.*

```
typedef struct applinfo {
     unsigned long     app;              /* application id */
     unsigned long     process_lock;     /* lock process data in memory */
     unsigned long     pkt_size;         /* message packet size */
     unsigned long     memory_buffer;    /* total message buffer to allocate */
     unsigned long     memory_export;    /* total buffer for other nodes */
     unsigned long     memory_each;      /* buffer available for each node */
     unsigned long     send_threshold;   /* send multiple packet threshold */
     unsigned long     send_count;       /* pkts to send when send_threshold */
     unsigned long     give_threshold;   /* send give message threshold */
     unsigned long     noc;              /* number of correspondents */
     unsigned long     rows;             /* number of rows in application */
     unsigned long     columns;          /* number of columns in application */
     unsigned long     unused[12];
} APPLINFO_T;
```

Region descriptors

Describes all regions for a process. The offset value is null if the contents of the region was not written to the region section of the core file. The following structure is defined in the include file *core.h.*

CORE *(cont.)* CORE *(cont.)*

```
struct core_region_desc {
     off_t            r_offset;     /* offset of VM region in core file */
     vm_address_t     r_vaddr;      /* virtual address of region start */
     vm_size_t        r_size;       /* region size (bytes) */
     vm_prot_t        r_prot;       /* VM protection (e.g. VM_PROT_READ) */
};
```

Thread information

Contains state (register) information for each Mach thread in the process. The structure consists of a list of all the registers and is defined in *mach/machine/thread_status.h.*

Region contents

Contains memory image of regions described in the region descriptors section.

Allocinfo File Format

Information for the entire application that is not locally available to a node's server is written to the file *allocinfo* within the *core* directory. The following structures show the format and content of the *allocinfo* file. These structures are defined in the include file *allocinfo.h.*

```
struct allocinfo_hdr {
     int               a_magic;        /* allocinfo file magic number */
     unsigned short    a_swap;         /* byte swap field */
     short             a_version;      /* allocinfo file version */
     nx_part_info_t    a_partition;    /* NX partition info */
     nx_app_info_t     a_application;  /* NX application info */
};
typedef struct {
     uid_t          uid;        /* User Id */
     gid_t          gid;        /* Group Id */
     int            access;     /* Access Permissions */
     int            sched;      /* NX_STD or NX_GANG */
     unsigned long  rq;         /* Rollin Quantum */
     int            epl;        /* Effective priority limit */
     int            nodes;      /* Number of nodes in the partition */
/* NOTES: mesh_x and mesh_y are only set if the mesh is a contiguous */
/* rectangle. Otherwise the are -1.*/
     int            mesh_x;     /* X dimension of partition */
     int            mesh_y;     /* Y dimension of partition */
```

CORE *(cont.)* CORE *(cont.)*

```
/* The enclose_mesh_x and enclose_mesh_y are the
* minimum rectangular dimensions that will enclose
* the partition. These dimensions may contain nodes enclose
* nodes that are not part of the specified partition
*/
     int              enclose_mesh_x;
     int              enclose_mesh_y;
     int              flags_or_size;  /* Internal Use only */
     int              part_id;        /* Internal Use only */
     int              free;           /* Internal Use only */
     int              reserved[7];
} nx_part_info_t;

typedef struct {
     int              size;        /* Number of nodes in application */
     int              nrows;       /* X dimension of application, 1 if
                                    * nodes are not contiguous
                                    */
     int              ncols;       /* Y dimension of application,set to size
                                    * if nodes are not contiguous
                                    */
     int              priority     /* Priority of application */
     unsigned long    rolled_in;   /* Milliseconds this appl rolled in */
     unsigned long    elapsed;     /* Milliseconds this appl rolled in */
     uid_t            uid;         /* UID of user running application */
     gid_t            nx_acctid;   /* NX account id (MACS)
                                    * of application */
     struct tm        start_time;  /* Time stamp of when
                                    * application started
                                    */
} nx_app_info_t;
```

Limitations and Workarounds

There is no means of limiting the size of a *core* directory as a whole.

Once an application faults and core file creation begins, it cannot be interrupted.

See Also

commands: **coreinfo, bootmagic, pspart, nx_pspart**

OSF/1 Programmer's Reference: **setrlimit(2), signal(4)**

## DEVCONF.TXT	DEVCONF.TXT

Diagnostic station: Defines the devices that are attached to each node of a Paragon system.

Synopsis

/usr/paragon/boot/DEVCONF.TXT

Description

The file *DEVCONF.TXT* is on the diagnostic station and can be created or changed by the system administrator only.

This **reset** command and the Paragon diagnostic utilities read the file *DEVCONF.TXT* to determine what devices are attached to the nodes in a Paragon system. This file must exist before executing the **reset** script with the **autocfg** switch.

The file *DEVCONF.TXT* contains a list of strings that specify the following devices for a Paragon system:

- Ethernet devices
- Hard disks
- Hippi devices
- MIO boards
- Paging devices
- RAID devices
- Tape devices

You use the following strings in the file *SYSCONFIG.TXT*:

DEVICES Specifies the beginning of a list of strings that define the system devices.

DISK *cbs* **ID** *id name*
Specifies a node with a stand-alone hard disk device. This specifies the hard disk's cabinet-backplane-slot (CBS) location, SCSI ID, and the manufacturer's name (for example, MAXTOR).

DEVCONF.TXT *(cont.)* DEVCONF.TXT *(cont.)*

END_DEVICES
Specifies the end of a list of strings that define the system devices.

ENET *cbs*
Specifies a node with an Ethernet port at the CBS location specified by the *cbs* argument.

HIPPI *cbs rev*
Specifies a node with an HIPPI board at the CBS location specified by the *cbs* argument. The *rev* argument specifies the revision number of the HIPPI board.

MIO *cbs rev*
Specifies a node with an MIO board at the CBS location specified by the *cbs* argument. The *rev* argument specifies the revision number of the MIO board.

RAID *cbs* **ID** *id* **SW** *rev* **LV** *level* **DC** *count* **SID** *sid name* [**NOPAGER, PAGE_TO** *partition*] [**CTRL***n*]

Specifies a node with a RAID at the CBS location specified by the *cbs* argument. The *id* argument specifies the SCSI ID for the RAID. The *rev* argument specifies the revision number of the RAID software (for example, 3.02). The *level* argument specifies the RAID level. The value of the *level* argument must 0, 1, 3, or 5). The *count* argument is the number of devices in the RAID. The *sid* argument is the SCSI ID, and the *name* argument is the symbolic name of the device that is written to the file *SYSCONFIG.TXT.*

The NOPAGER option specifies that this RAID subsystem is not to be part of a paging tree. The PAGE_TO keyword specifies a particular partition as the one to be used for paging.

If the SCSI controller has two channels, the CTRL keyword indicates the SCSI channel to which the RAID subsystem is attached. *n* indicates the SCSI channel and is either 0 or 1. CTRL0 is the default. If the RAID subsystem is attached to SCSI channel 0, there is no need to include the CTRL keyword.

NOPAGER
Removes the node number from the list of nodes that can be used for paging. Any node that is defined to have a hard disk attached is automatically listed as a potential paging node. This token overrides this. When this token appears on a line beginning with RAID, it indicates that the specified RAID subsystem is not to be used for paging.

PAGE_TO *partition*
By default, all paging uses the partition *rz0b*. When this token appears on a line beginning with RAID, it indicates that the specified RAID subsystem will be paged to *partition* instead of *rz0b*.

The node is added to the list of node numbers in the bootmagic string *PAGE_TO*. The *PAGE_TO* string has no effect on the boot node. The boot node uses the default paging device *rz0b*.

For example, the following bootmagic strings indicate that I/O nodes 2, 3, 7, and 11 are used for paging. Compute nodes 1, 5, and 6 page to I/O node 2; compute nodes 9, 10, and 13 page to I/O node 7; and compute nodes 14 and 15 page to I/O node 11. I/O nodes 2, 7, and 11 page to I/O node 3.

I/O node 2 uses partition *rz0b* on its channel 0 and *rz0c* on its channel 1 for paging. I/O nodes 3 and 11 use *rz0b* on both of its channels. Note that channel 0 (ch0) is not shown because it is the default.

If the paging partition for an I/O node is only the default *rz0b* on channel 0, then that I/O node does not appear in the PAGE_TO list.

```
EXPORT_PAGING=2..3,7,11
PAGER_NODE=<1,5..6>2:<9..10,13>7:<14..15>11:<2,7,11>3
PAGE_TO=<2>rz0b,ch1/rz0c:<3,11>rz0b,ch1/rz0b
```

SIO *cbs rev*
Specifies a node with an SIO board at the CBS location specified by the *cbs* argument. The *rev* argument specifies the revision number of the SIO board. The SIO board is an I/O node with a SCSI interface that has two channels.

TAPE *cbs* **ID** *id name* [**CTRL***n*]
Specifies a node with a tape device at the CBS location specified by the *cbs* argument. The *id* argument specifies the SCSI ID for the tape device. The *name* argument is the symbolic name of the device that is written to the file *SYSCONFIG.TXT*.

If the SCSI controller has two channels, the CTRL keyword indicates the SCSI channel to which the tape drive is attached. *n* indicates the SCSI channel and is either 0 or 1. CTRL0 is the default. If the tape drive is attached to SCSI channel 0, there is no need to include the CTRL keyword.

Examples

The following is an example *DEVCONF.TXT* file that defines the devices on a Paragon system:

```
DEVICES

ENET 00D02
ENET 00D03
RAID 00D03 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3
TAPE 00D03 ID 6 DAT
RAID 01D12 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3
MIO 00D03 H04
MIO 00D02 H04
MIO 01D12 H04

END_DEVICES
```

This example shows three I/O nodes, defined by the lines beginning with MIO. These I/O nodes are at CBS locations 00D03, 00D02, and 01D12.

- The two I/O nodes at the CBS locations 00D02 and 00D03 each have Ethernet ports.
- The SCSI interface of the two I/O nodes at the CBS locations 00D02 and 00D12 have RAID subsystems.
- The SCSI interface of the I/O node at the CBS location 00D03 has a DAT tape drive.

Here is another example that shows two I/O nodes, one of which (called the SIO board) contains a SCSI-16 interface. Both RAID subsystems are attached to this SIO board. One RAID subsystem is attached to SCSI channel 0; the other is attached to SCIS channel 1.

```
DEVICES

ENET 00D02
ENET 00D03
RAID 00D03 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3 CTRL0
RAID 00D03 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3 CTRL1
MIO 00D02 H04
SIO 00D03 H04

END_DEVICES
```

Consider the two RAID lines in more detail. Consider the following three cases:

DEVCONF.TXT *(cont.)* DEVCONF.TXT *(cont.)*

Case 1. Both RAID subsystems are used for paging. Both use the default partition *rz0b*.

```
RAID 00D03 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3 CTRL0
RAID 00D03 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3 CTRL1
```

Case 2. Only the RAID subsystem attached to SCSI channel 0 is used for paging, and it uses the default partition *rz0b*.

```
RAID 00D03 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3 CTRL0
RAID 00D03 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3 NOPAGER CTRL1
```

Case 3. Both RAID subsystems are used for paging. The RAID subsystem attached to SCSI channel 0 uses the default partition *rz0b*. The RAID subsystem attached to SCSI channel 1 uses the partition *rz0c*.

```
RAID 00D03 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3 CTRL0
RAID 00D03 ID 0 SW 3.02 LV 5 DC 5 SID 0 RAID3 PAGE_TO rz0c CTRL1
```

Files

/usr/paragon/boot/BADNODES.TXT
Specifies the nodes that are nonfunctional on a Paragon system. This file resides on the diagnostic station.

/usr/paragon/boot/bootmagic
Specifies information for booting a Paragon system. This file resides on the diagnostic station.

/usr/paragon/boot/DEVCONF.TXT
Specifies the device configuration for a Paragon system. This file resides on the diagnostic station.

/usr/paragon/boot/MAGIC.MASTER
Specifies the master version of the bootmagic strings for booting a Paragon system. This file resides on the diagnostic station.

/usr/paragon/boot/SYSCONFIG.TXT
Specifies the hardware configuration for a Paragon system. This file resides on the diagnostic station.

DEVCONF.TXT *(cont.)*

DEVCONF.TXT *(cont.)*

See Also

commands: **bootpp**

files: **BADNODES.TXT**, **bootmagic**, **MAGIC.MASTER**, **SYSCONFIG.TXT**

FSCAN.CFG FSCAN.CFG

Diagnostic station: Specifies how to start the fscan utility when a Paragon system is booted.

Syntax

/usr/paragon/boot/fscan.cfg

Description

The file *fscan.cfg* is on the diagnostic station and can be created or changed by the system administrator only.

The **fscan** configuration file *fscan.cfg* contains **fscan** commands that specify how to start the **fscan** utility when the system is rebooted. For a list and description of the **fscan** commands, see the **fscan** manual page.

This file is an ASCII file that you can create or change with any standard editor (for example, **vi**). The **reset** command looks for the **fscan** configuration file in your current directory. If the **reset** command does not find the file in the current directory, it automatically creates a file named *fscan.cfg* in your current directory. The default configuration file contains the following commands:

```
set polling on
set autoreboot off
set autoswitch off
set autobucket off
set notify on
define reboot \
   "ksh /usr/paragon/boot/reset skip ignorelock autoreboot"
```

The default configuration file sets polling and automatic rebooting to on; sets automatic node switching, automatic processor disabling, and notification of dead processors to off; and defines the **reboot** command.

The comment character # (pound sign) can be used in column 0 (zero). All characters to the right of the comment character are ignored.

Files

/usr/paragon/boot/fscan.cfg
Contains **fscan** commands that specify the **fscan** configuration. This **fscan** utility looks for this file in the current directory or in the file specified by the **-F** switch.

FSCAN.CFG *(cont.)* FSCAN.CFG *(cont.)*

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

fscan, reset

FSTAB FSTAB

Static file system mounting table.

Syntax

/etc/fstab

#include <fstab.h>

Description

The */etc/fstab* file contains descriptive information about the known file systems. */etc/fstab* is only read by programs and not written; by convention, you create and maintain this file. Each filesystem is described on a separate line; fields on each line are separated by tabs or spaces. The order of records in */etc/fstab* is important because **fsck, mount,** and **umount** sequentially iterate through **/etc/fstab** during their work.

The first field, (*fs_spec*), describes the block special device or remote filesystem to be mounted. For filesystems of type **ufs**, the special file name is the block special file name, and not the character special file name. If a program needs the character special file name, the program must create it by appending the letter **r** after the last **/** (slash) in the special file name.

The second field, (*fs_file*), describes the mount point for the filesystem. For swap partitions, this field should be specified as **none**.

The third field, (*fs_vfstype*), describes the type of the filesystem. The system currently supports four types of filesystems:

ufs	a local UNIX filesystem
pfs	a Parallel File System (PFS)
nfs	a Network File System
swap	a disk partition to be used for swapping

The fourth field, (*fs_mntops*), describes the mount options associated with the filesystem. It is formatted as a comma separated list of options and contains at least the type of mount (see *fs_type* below) plus any additional options appropriate to the filesystem type.

FSTAB *(cont.)* FSTAB *(cont.)*

If the options **userquota** and/or **groupquota** are specified, the filesystem is automatically processed by the **quotacheck** command, and disk quotas are enabled with the **quotaon** command. By default, filesystem quotas are maintained in files named *quota.user* and *quota.group* which are located at the root of the associated filesystem. These defaults may be overridden by putting an equal sign and an alternative absolute pathname following the quota option. Thus, if the user quota file for **/tmp** is stored in **/var/quotas/tmp.user** this location can be specified as:

```
userquota=/var/quotas/tmp.user
```

The type of the mount is extracted from the *fs_mntops* field and stored separately in the *fs_type* field. It is not deleted from the *fs_mntops* field. If *fs_type* is **rw** or **ro** then the filesystem whose name is given in the *fs_file* field is normally mounted read-write or read-only on the specified special file. If *fs_type* is **sw** then the special file is made available as a piece of swap space by the **swapon** command at the end of the system reboot procedure. The fields other than *fs_spec* and *fs_type* are unused. If *fs_type* is specified as **xx** the entry is ignored. This is useful to show disk partitions which are currently unused.

The fourth field supports the following *pfs* options:

stripeunit=*n* Stripe unit size for the parallel file system, where the *n* argument is the size in bytes. The stripe unit size is the size of the unit of data interleaving for regular files. To specify units other than bytes, append the appropriate (upper-case or lower-case) letter to the *n* argument:

k	Kilobytes (1024 bytes)
m	Megabytes (1024K bytes)
g	Gigabytes (1024M bytes)

The default stripe unit size is 64K bytes.

stripedirs=*path1:path2:path3 ...*

Colon-separated list of pathnames that specify the set of directories that define the stripe group for this parallel file system. Each of these directories is used for stripe storage. The default value of **stripedirs** is the set of all available stripe directories as defined in the */etc/pfstab* file. See the **pfstab()** manual page for a description of the */etc/pfstab* file. The **stripegroup** option can be used in place of this option.

stripegroup=*groupname*

Keyword from the */etc/pfstab* file that identifies the stripe group. See the **pfstab** manual page. The default stripe group is *all*, indicating that the set of all available stripe directories will be used for striping in the parallel file system. The **stripedirs** option can be used in place of this option.

FSTAB *(cont.)* FSTAB *(cont.)*

The fifth field, (|*fs_freq*), is used for these filesystems by the **dump** command to determine which filesystems need to be dumped. If the fifth field is not present, a value of zero is returned and **dump** assumes that the filesystem does not need to be dumped.

The sixth field, (|*fs_passno*), is used by the **fsck** program to determine the order in which filesystem checks are done at reboot time. The root filesystem should be specified with a *fs_passno* of 1, and other filesystems should have a *fs_passno* of 2. Filesystems within a drive will be checked sequentially, but filesystems on different drives will be checked at the same time to utilize parallelism available in the hardware. If the sixth field is not present or zero, a value of zero is returned and **fsck** will assume that the filesystem does not need to be checked.

The fields of the **/etc/fstab** file correspond to, but are not in the same order as, the components of the **fstab** structure. The proper way to read records from **/etc/fstab** is to use the routines **getfsent()**, **getfsspec()**, and **getfsfile()**.

```
#define FSTAB_RW "rw" /* read/write device */
#define FSTAB_RO "ro" /* read-only device */
#define FSTAB_SW "sw" /* swap device */
#define FSTAB_XX "xx" /* ignore totally */

struct fstab {
   char *fs_spec;     /* block special device name */
   char *fs_file;     /* file system path prefix */
   char *fs_type;     /* FSTAB_* (for rw, ro, sw, or xx) */
   int   fs_freq;     /* dump frequency, in days */
   int   fs_passno;   /* pass number on parallel dump */
   char *fs_vfstype;  /* file system type, ufs, nfs, etc. */
   char *fs_mntops;   /* comma separated mount options */
};
```

Files

/etc/fstab	Table of file systems mounted at boot
/etc/pfstab	Table of PFS stripe groups

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

FSTAB *(cont.)* FSTAB *(cont.)*

See Also

commands: **showfs, df**(1)

files: **pfstab**

OSF/1 Programmer's Reference: **mount**(2)

OSF/1 System and Network Administrator's Reference: **fsck(8)**, **mount(8)**, **umount(8)**

MAGIC.MASTER MAGIC.MASTER

Diagnostic station: Specifies a master version of the bootmagic strings for booting a Paragon system.

Synopsis

/usr/paragon/boot/MAGIC.MASTER

/usr/paragon/boot/MAGIC.md

Description

The file *MAGIC.MASTER* is a file on the diagnostic station and only the system administrator can create or change this file.

The *MAGIC.MASTER* file contains two kinds of strings that specify how to boot a Paragon system:

- Bootmagic strings specify booting and system configuration information for a Paragon system. See the **bootmagic** manual page for a description and list of the bootmagic strings.
- Reset strings specify reset configuration information. See the **reset** manual page for a description and list of the reset strings.

The **bootpp** command reads the *MAGIC.MASTER* file and uses the bootmagic strings in this file to create the *bootmagic* file. The *bootmagic* file contains the information for booting a Paragon system. The following shows a example bootmagic string:

```
BOOT_FIRST_NODE=7
```

This specifies the boot node to be 7 when the system is booted or reset.

You can add any bootmagic strings to the *MAGIC.MASTER* file. Adding or changing the bootmagic strings in the *MAGIC.MASTER* file changes how the system boots, because the **bootpp** command reads the bootmagic strings in the *MAGIC.MASTER* file each time the system is booted.

The *MAGIC.MASTER* file contains reset strings that are used to set variables in the **reset** command. The reset strings begin with "RST_". See the **reset** manual page for a list of the reset strings. The following example shows a reset string in the *MAGIC.MASTER* file:

```
RST_BOOT_DIR=/usr/paragon/newboot
```

This specifies a non-default boot directory. The **bootpp** command ignores reset strings.

MAGIC.MASTER *(cont.)* MAGIC.MASTER *(cont.)*

If the *MAGIC.MASTER* file is empty or does not contain the following bootmagic strings, the Paragon system cannot boot. No error messages are generated. The *MAGIC.MASTER* file must contain the following string:

```
BOOT_FIRST_NODE=boot-node-number
```

The *BOOT_FIRST_NODE* string specifies the boot node. The *boot-node* value must be the node number of the boot node.

A comment begins with a pound sign (#). All characters that follow the pound sign are ignored. The line continuation character is the backslash (\). If a line is longer than the width of the screen, you can use a backslash to continue the line on the next line.

You can use the **reset** command with the **-f** switch to specify a magic file other than the *MAGIC.MASTER* file from which to get reset information, for example:

```
DS# reset -f mymagic
```

This resets the system and reads the reset information from the *mymagic* file.

The *MAGIC.md* file is the master magic file that is used with the RAM disk. This file must contain the following strings:

```
REAL_FIRST_NODE=boot-node-number
BOOT_FIRST_NODE=0
BOOT_ROOT_DEV=md0a
```

This specifies the node number of the real first node of the system, the boot node as node 0, the root file system device as *md0a.*

MAGIC.MASTER *(cont.)* MAGIC.MASTER *(cont.)*

Examples

The following shows examples of entries in a *MAGIC.MASTER* file.

```
# Bootmagic string sets boot node to 7.
BOOT_FIRST_NODE=7
# Bootmagic string sets the boot console to fscan
BOOT_CONSOLE=f
# Bootmagic strings set how the system LEDs operate.
BOOT_GREEN_LED=Dciw
BOOT_RED_LED=Dcg1XRm
# Bootmagic strings set up the paging tree.
EXPORT_PAGING=0,7
PAGER_NODE=<0>7:<15..126>0
```

Files

The following files are on the diagnostic station.

/usr/paragon/boot/BADNODES.TXT
Specifies the nodes that are nonfunctional on a Paragon system.

/usr/paragon/boot/bootmagic
Specifies information for booting a Paragon system.

/usr/paragon/boot/DEVCONF.TXT
Specifies the device configuration for a Paragon system.

/usr/paragon/boot/MAGIC.MASTER
Specifies the master version of the bootmagic strings for booting a Paragon system.

/usr/paragon/boot/MAGIC.md
Specifies the master version of the bootmagic strings for booting a Paragon system with the RAM disk.

/usr/paragon/boot/SYSCONFIG.TXT
Specifies the hardware configuration for a Paragon system.

MAGIC.MASTER *(cont.)* MAGIC.MASTER *(cont.)*

See Also

commands: **bootpp**, **reset**

files: **BADNODES.TXT**, **bootmagic**, **DEVCONF.TXT**, **SYSCONFIG.TXT**

PARAMETERS PARAMETERS

Defines tunable parameters for the load leveler daemon.

Description

The */etc/load_level/parameters* file defines the parameters for the load leveler. Each parameter is defined on a separate line and may appear in any order in the file. Blank lines and text appearing between pound sign (#) and the end of a line are ignored as comments. An entry starts with the parameter name followed by an equal sign (=) and the value of the parameter. White spaces (spaces and tabs) are ignored. The load leveler has default values for all the parameters. Values different from the defaults must be defined in the *parameters* file.

The parameters are as follows:

fast_node_timeout
: Positive floating-point value that specifies the frequency in seconds after which the load leveler executes the fast node algorithm for static load-leveling. The default is 5.0 seconds.

first_weight_factor
second_weight_factor
third_weight_factor
: Non-negative integer values that specify weight factors for load leveling. Over three different time intervals, the Mach kernel samples the processes that are available to run on the current node and calculates a load average. The three time intervals are 5 seconds, 30 seconds, and 1 minute. A load average is calculated for each time interval. The load leveler uses these three load averages to produce a single load value for the node. This is done by computing a weighted average of the three load averages. The weight factors specify the weight or ratio to use for a time interval. The default values 50, 25, and 25 assign 50% weight to the first load average and 25% weight to the second and third such values. The weights 2:1:1 would produce the same result as 50:25:25. Weights 1:1:1 or 33:33:33 would treat each load average equally. Weights 0:1:0 would make the result the second of the three load averages. The weight factors specify the following:

first_weight_factor
: Weight factor for the 5-second load average. The default is 50.

second_weight_factor
: Weight factor for the 30-second load average. The default is 25.

third_weight_factor
: Weight factor for the 1-minute load average. The default is 25.

PARAMETERS *(cont.)* PARAMETERS *(cont.)*

inclusion_list Integer, either 0 or 1, that specifies whether the commands listed in the */etc/load_level/migrate_commands* file may migrate. The default is 1. The value 1 indicates that the listed commands are the *only* ones that *may* migrate; the value 0 (zero) indicates that the listed commands are the *only* ones that *may not* migrate. This parameter has no meaning for static load-leveling.

minimum_cputime:
Non-negative floating-point value that specifies the minimum amount of processor time in seconds that a process must accumulate to be eligible for migration. The default is 1.0. This parameter has no meaning for static load-leveling.

minimum_overload
Positive floating-point value that specifies the minimum amount a node's load must exceed the system-wide load average for the node to be overloaded. The default is 1.0. When the current node is overloaded, the load leveler daemon tries to migrate local processes to underloaded remote nodes. This parameter is also used by static load-leveling for calculating the lightest-loaded node.

minimum_underload
Positive floating-point value that specifies the minimum amount a node's load must be below the system-wide average for the node to be underloaded. The default is 1.0. This parameter is also used in static load-leveling for calculating the lightest-loaded node.

nodes_to_use=nodelist
Specifies the list of service nodes to use for load leveling. The default is all the service nodes. The *nodelist* argument is comma-separated list of service-node numbers. A range of nodes can be separated by a dash (-).

number_vector_elements
Positive integer that specifies the maximum number of elements in the load vector. The default depends on the number of nodes in the system. Each slot contains a node number and that node's load value. The load leveler daemon operates on a randomly selected subset of nodes and this is the size of that subset.

per_process_avg_load
Positive floating-point value that specifies the average load a single process can generate. The default is 1.0. The load leveler uses this value to calculate the loads of specific nodes after a process has migrated. This parameter has no meaning for static load-leveling.

PARAMETERS *(cont.)* PARAMETERS *(cont.)*

pagein_load

Positive floating-point value that specifies the amount of load that one page-in operation per second causes. The default is 0.05.

pageout_load

Positive floating-point value that specifies the amount of load that one page-out operation per second causes. The default is 0.10.

pgstat_max_interval

Positive floating-point value that specifies in seconds the maximum interval for sampling the paging statistics. The default is 30 seconds. The load-leveler uses paging statistics as well as CPU-utilization to calculate a node's load. The actual interval used for sampling the paging statistics cannot exceed *pgstat_max_interval.*

pgstat_pref_interval

Positive floating-point value that specifies in seconds the preferred interval for sampling the paging statistics. The default is 10 seconds. The load-leveler uses paging statistics as well as CPU-utilization to calculate a node's load. The actual interval used for sampling the paging statistics should be equal to or longer than *pgstat_pref_interval.*

re_dispatch_timeout

Positive floating-point value that specifies in seconds the time period between which the load leveler considers migrating processes. The default is 7.0. This value should be at least as large as *send_timeout*. This parameter has no meaning for static load-leveling.

root_fs_node_source

Boolean that specifies if the node specified by the bootmagic string *ROOT_FS_NODE* should be used as source node for load leveling. The default value is 1. The value 0 specifies not to use the boot node as a source node. The value 1 specifies using the boot node as a source node.

root_fs_node_target

Boolean that specifies if the boot node specified by the bootmagic string *ROOT_FS_NODE* should be used as a target node for load leveling. The default value is 0. The value 0 specifies not to use the boot node as a target node. The value 1 specifies using the boot node as a target node.

PARAMETERS *(cont.)* PARAMETERS *(cont.)*

send_timeout Positive floating-point value that specifies in seconds the time period between exchanges of load information among load leveler daemons on the service nodes. The default is 5.0.

static_min_load_delta
Positive floating-point value that specifies the minimum amount by which the load of the local node must differ from the load of another node in order for the other node to be considered to be underloaded. The default is 1.0.

Examples

The following is an example *parameters* file:

```
send_timeout=            1.0 # frequency in secs for sending load info.
re_dispatch_timeout=     5.0 # frequency in seconds for re-dispatching procs.
first_weight_factor=     70  # 5 second weight factor.
second_weight_factor=    25  # 30 second weight factor.
third_weight_factor=      5  # 1 minute weight factor.
number_vector_elements= 64   # number of load vector elements.
inclusion_list=            0 # 0 = exclusion list, 1 = inclusion list.
minimum_underload=       1.0 # node is underloaded if underload >=parameter.
minimum_overload=        1.0 # node is overloaded if overload >=parameter.
minimum_cputime=         1.0 # minimum runtime of process for migration.
per_process_avg_load=    0.7 # average load generated by a single process.
pgstat_max_interval=    30.0 # maximum interval for sampling paging stats.
pgstat_pref_interval=   10.0 # preferred interval for sampling paging stats.
pagein_load=            0.05 # load for one second's worth of page-in operation.
pageout_load=           0.10 # load for one second's worth of page-out operation.
nodes_to_use=                # defaults to all nodes.
static_min_load_delta=   1.0 # minimum load difference.
fast_node_timeout=       1.0 # frequency in seconds to calculate fast_node.
root_fs_node_target=       0 # use root_fs_node as target for load leveling.
root_fs_node_source=     1   # use root_fs_node as source for load leveling.
```

Files

/etc/load_level/parameters

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

PARAMETERS *(cont.)*

See Also

commands: **load_leveld**, **loadlevel**

files: **bootmagic**

PFSTAB PFSTAB

Static Parallel File System (PFS) stripe group table.

Syntax

/etc/pfstab

#include <pfstab.h>

Description

The */etc/pfstab* file contains entries describing stripe groups that are used by the **mount** command when mounting parallel file systems. A stripe group consists of a set of directories that parallel files are interleaved across.

Each entry consists of a line of the form:

stripedir stripegroup1 stripegroup2...

A *stripedir* is a directory in a UNIX file system (UFS) that is available for stripe storage. There is only one entry in */etc/pfstab* for each *stripedir* in the system. All stripe directories must be specified with full pathnames; relative pathnames are not accepted.

Each *stripegroup* entry is a keyword that identifies a stripe group to which the *stripedir* in the first field belongs. The same *stripegroup* may be specified on multiple lines of the */etc/pfstab* file to indicate that multiple *stripedirs* belong to that *stripegroup.*

Examples

In the following example */etc/pfstab* file, each of the listed directories belongs to the stripe group *all*, the directories */.sdirs/vol0*, */.sdirs/vol1*, and */.sdirs/vol2* belong to the stripe group *left*, and the directories */.sdirs/vol3*, */.sdirs/vol4*, and */.sdirs/vol5* belong to the stripe group *right*. The directory */.sdirs/vol6* has been commented out.

```
/.sdirs/vol0    all   left
/.sdirs/vol1    all   left
/.sdirs/vol2    all   left
/.sdirs/vol3    all   right
/.sdirs/vol4    all   right
/.sdirs/vol5    all   right
#/.sdirs/vol6   all
```

PFSTAB *(cont.)* PFSTAB *(cont.)*

Files

/etc/fstab	Table of file systems mounted at boot.
/etc/pfstab	Table of PFS stripe groups.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

Paragon™ XP/S System Commands Reference Manual: **fstab**, **mount**, **showfs**

OSF/1 Programmer's Reference: **mount(2)**

OSF/1 System and Network Administrator's Reference: **fstab(4)**, **mount(8)**

RPM RPM

Reprogrammable Performance Monitoring (RPM) counters.

Syntax

#include <*i860paragon/rpm.h*>

Description

The */usr/include/i860paragon/rpm.h* file specifies the programming interface to the Reprogrammable Performance Monitoring (RPM) hardware counters and the Mach kernel software (RPM soft) counters. Both the hardware and soft counters are memory mapped into every process. The counters may be read but not written. By defining a pointer to the structures defined in *rpm.h* and then setting the pointer to a specific address, the RPM hardware and soft counters can be accessed.

The RPM hardware counters are specified by the structure *rpm*, which contains the following fields (hardware counters):

rpm_control	32-bit control register used to reset the counters. All counters except *rpm_time* can be reset. The *rpm_time* global clock is reset via the diagnostic station. The process must have root access and perform an authentication process to gain access to a writable page in order to write to the control register.
rpm_time	10 Mhz 56-bit global clock accurate to 100 nanoseconds local to the node and 1 microsecond across all nodes in a system.
rpm_cpu0	Number of 50 Mhz cycles that CPU 0 is bus master.
rpm_cpu1	Number of 50 Mhz cycles that CPU 1 is bus master.
rpm_ltu	Number of 50 Mhz cycles that ltu is bus master.
rpm_exp	Number of 50 Mhz cycles that expansion card is bus master.
rpm_north	One count for every 64 bytes moving North on the mesh.
rpm_south	One count for every 64 bytes moving South on the mesh.
rpm_east	One count for every 64 bytes moving East on the mesh.
rpm_west	One count for every 64 bytes moving West on the mesh.

To access the RPM hardware counters, define a pointer to the structure *rpm*, set the pointer to the address *RPM_BASE_VADDR*, and access the counter via the structure. For example:

```
struct rpm *rpm;
rpm_timer_t global_time;
rpm = (struct rpm *) RPM_BASE_VADDR;
global_time = rpm->rpm_time;
```

The *rpm->rpm_control* register is used to reset the RPM counters to zero. A task must first obtain a writable page to the RPM counters. Normally the *RPM_BASE_VADDR* address specifies a read-only page mapped into every task. In order to obtain a writable page, the task must have root authority (executed by the superuser) and perform an authentication process. The authentication steps are to open the NORMA device **rpm0**, obtain a pager port for the mapped rpm device, and map the pager port into the task's address space. The end result is an address to a writable page that is substituted for the *RPM_BASE_VADDR* address. The RPM counters can then be reset to zero by setting the *rpm->rpm_control* register to 0xFFFF0000.

NOTE

The SPV data collection daemons reset the RPM counters once every second (see *Limitations and Workarounds*).

The *rpm->rpm_time* global clock can not be reset by the *rpm->rpm_control* register. The global clock is reset at the diagnostic station using the diagnostic commands **/u/paragon/diag/gclock** and **/u/paragon/diag/greset**. The RPM global clock is also used by the NX **dclock**() call which returns a double-precision time interval in seconds since the system was booted.

The RPM soft counters are maintained by the Mach kernel on each CPU configured into the kernel. The RPM soft counters are specified by the structure *rpmsoft*, which contains the following fields:

rpms_idle	Number of double precision seconds the CPU has been idle.

Large grain trap handler statistics:

rpms_alltraps	Number of traps.
rpms_it	Number of trap instructions.
rpms_int	Number of interrupts.
rpms_iat	Number of instruction access traps.
rpms_dat	Number of data access traps.
rpms_ft	Number of floating point traps.

RPM *(cont.)* RPM *(cont.)*

Data access trap statistics:

rpms_datld	Number of data access traps on ld.x.
rpms_datst	Number of data access traps on st.x.
rpms_datfldfst	Number of data access traps on fld.x or fst.x.
rpms_datpst	Number of data access traps on pst.
rpms_datpfld	Number of data access traps on pfld.y.
rpms_datauto	Number of data access traps on fld.x++, fst.x++, and pfld.x++.

Data access page fault statistics:

rpms_notdirty	Number of page faults for a store to a clean page.
rpms_notref	Number of page faults for an access to an unreferenced page while locked.
rpms_notwr	Number of page faults for a store to a read-only page.
rpms_pdenotu	Number of page directory entry access violations.
rpms_ptenotu	Number of page table entry access violations.
rpms_pdenotp	Number of page directory entry invalid traps.
rpms_ptenotp	Number of page table entry invalid traps.

Locked sequence related trap statistics:

rpms_lockseq	Number of traps while in a locked sequence.
rpms_lockres	Number of restarted locked sequences.
rpms_lockexp	Number of expired locked sequences.

Floating-point exception statistics:

rpms_fpe_si	Number of floating-point sticky inexact exceptions.
rpms_fpe_se	Number of floating-point source exceptions.
rpms_fpe_mu	Number of floating-point multiplier underflow exceptions.
rpms_fpe_mo	Number of floating-point multiplier overflow exceptions.
rpms_fpe_mi	Number of floating-point multiplier inexact exceptions.
rpms_fpe_ma	Number of floating-point multiplier add-one exceptions.
rpms_fpe_au	Number of floating-point underflow exceptions.
rpms_fpe_ao	Number of floating-point overflow exceptions.
rpms_fpe_ai	Number of floating-point inexact exceptions.
rpms_fpe_aa	Number of floating-point add one exceptions.

For every CPU configured into the Mach kernel there is a corresponding *rpmsoft* structure that contains the statistics for the CPU. At the address *RPMSOFT_BASE_VADDR* is an array of structures, one for each CPU configured into the Mach kernel. The number of CPUs configured into the Mach kernel can be found by using the **host_info()** kernel call. To access the RPM soft data, increment a pointer through the array of the *rpmsoft* structures. For example:

```
struct rpmsoft *rpmsoft;
rpm_timer_t idle_sum_time;
rpmsoft = (struct rpm *) RPMSOFT_BASE_VADDR;
idle_sum_time = 0.0;
for ( i = 0; i < num_cpus; ++i ) {
   idle_sum_time = (rpmsoft + i)->rpms_idle;
}
```

The RPM soft counters can not be written or reset. The counters represent summed statistics since the system was booted.

For a GP node, the first *rpmsoft* structure in the array contains the statistics for the application CPU, while the second *rpmsoft* structure in the array contains the statistics for the message co-processor.

Examples

The following example reads the RPM global clock and converts the 56-bit time to double-precision seconds (the dclock() functionality).

```
#define RPM_CLOCK_FREQ (10000000)
#define _2_to_52d (4503599627370496.0)
#define OR_EXPONENT (0x4330)
#define MASK_EXPONENT (0x000F)
double hz;
struct rpm *rpm;
union {
   unsigned short wordwise[4];
   double value;
} t;
rpm = (struct rpm *) RPM_BASE_VADDR;
t.value = rpm->rpm_time;
hz = 1.0/RPM_CLOCK_FREQ;
t.wordwise[3] = (t.wordwise[3] & MASK_EXPONENT) | OR_EXPONENT;
t.value = hz * (t.value - _2_to_52d);
```

This code converts the 56-bit integer count into a 64-bit double-precision value representing seconds. The code ignores the highest 4 bits of the 56-bit counter (a 52-bit counter, counting at 10Mhz can count for 14.28 years before wraparound occurs).

Consider the representation of a double. Doubles have three fields: a sign field, an exponent field and a fraction field. The sign field is a single bit, 0 for positive and 1 for negative. The exponent field is bits 62 to 52, which can hold integers from 0 to 2047. The actual value of the exponent is the value in the exponent field minus 1023. The fraction field is bits 51 to 0. The actual value of the fraction is 1.*f*, where *f* is the value of the integer in the fraction field. (For information on the hidden 1, refer to IEEE standard 854 for radix-independent floating-point arithmetic.)

To convert the 52-bit integer to floating point, the value of the exponent must be set to 52, and the value $1x2^{52}$ must be subtracted (the hidden 1). To set the value of the exponent to 52, the value of the exponent field is set to 1023 + 52, or 1075 (0x433 hex). To subtract the hidden 1, the value 4503599627370496.0 ($1x2^{52}$) is subtracted. At this point the 52-bit number has been converted to a floating-point representation of the same number. To convert the floating point representation of the 52-bit (10Mhz) counter to seconds, simply multiply by 10M.

RPM *(cont.)* **RPM** *(cont.)*

Limitations and Workarounds

The SPV tool uses the RPM hardware to collect mesh and memory bus utilization information. Once every second the RPM hardware counters are collected and reset on every node in the system. The SPV data collection daemon must be stopped if you want an individual application to collect and interpret the RPM hardware counters. To stop the SPV daemon the *root* user must either select the SPV **File** menu **Data collection** command to temporarily stop the SPV data collection or stop the SPV daemon itself. This is done by invoking **/etc/init.d/spv stop** or modifying the **/etc/init.d/spv** script to not start the daemon during system boot.

The RPM hardware counters wrap around to zero when the max 32-bit count value has been reached.

The original intent of the RPM bus counters (*rpm_cpu0*, *rpm_cpu1*, *rpm_ltu*, and *rpm_exp*) was to report bus utilization information. The original concept was to count bus cycles when a module becomes a bus master. Subsequent performance investigations indicated that becoming a bus master is an expensive operation in terms of bus cycles. Thus, the default bus master is *rpm_cpu0* whether the CPU is using the bus or not. The *rpm_cpu1*, *rpm_ltu*, and *rpm_exp* counters correctly denote the amount of bus utilization for the message coprocessor, the ltu, and when the expansion card is a bus master. However, the *rpm_cpu0* bus counter does not correctly reflect the bus usage of application CPU.

The total utilization of the bus counters (*rpm_cpu0*, *rpm_cpu1*, *rpm_ltu*, and *rpm_exp*) is always a little over 97% because about 2% of the bus is consumed for memory refresh.

See Also

spv, **dclock()**

System Performance Visualization Tool User's Guide

C System Calls Reference Manual

SYSCONFIG.TXT SYSCONFIG.TXT

Diagnostic station: Contains the hardware configuration for booting a Paragon system.

Synopsis

/usr/paragon/boot/SYSCONFIG.TXT

Description

The file *SYSCONFIG.TXT* is on the diagnostic station and can only be created or changed by the system administrator.

NOTE

Intel Scalable Systems Division recommends that you not edit the *SYSCONFIG.TXT* file directly.

The recommended way to add tokens to a node specification in *SYSCONFIG.TXT* is to edit the node specification in *DEVCONF.TXT*, then create a new *SYSCONFIG.TXT* with the command **reset autocfg**. (Exception: if you use an alternative kernel, such as SUNMOS, you must edit *SYSCONFIG.TXT* directly. Any **ALTOS** tokens in *SYSCONFIG.TXT* are lost when you use **reset autocfg**.)

The **bootpp** command and the Paragon diagnostic utilities read the file *SYSCONFIG.TXT* to get hardware configuration information. You can automatically create this file by creating the file */usr/paragon/boot/DEVCONF.TXT* and executing the **reset** script with the **autocfg** switch.

The file *SYSCONFIG.TXT* resides on the diagnostic station. This file contains a list of commands that specify the hardware configuration for a Paragon system including the following:

- The system cabinets.
- The system backplanes.
- The system nodes.
- The devices attached to each node.
- The failed nodes or empty slots.
- The nodes that are paging nodes.
- The nodes that have an ethernet connection.

Commands

You use the following commands in the file *SYSCONFIG.TXT*:

CABINET *number*

Specifies the cabinet that subsequent **BP** and **S** commands refer to. The *number* argument can range from 0 to 31. The following example specifies that the commands that follow are for cabinet 1:

```
CABINET 1
```

BP *backplane rev*

Specifies the backplane that subsequent **S** commands refer to. The *backplane* argument can be **A**, **B**, **C**, or **D**. The *rev* argument is the revision number of the backplane. The *rev* argument is recognized by the Paragon diagnostic utilities, but is ignored by the **bootpp** command. The following example specifies that the commands that follow are for backplane A that has a revision number AC04:

```
BP A AC04
```

S *slot token ...* Specifies the slot and the type board that is in the slot. The *slot* argument is an integer that ranges from 0 (zero) to 15. The token arguments specify the kind of board in the slot and attributes of the board. The following example specifies that slot 0 is a GP node with a revision number AN00, 32M bytes of memory, and an MRC revision number of 04:

```
S 0 GPNODE AN00 32 MRC 04
```

Tokens

The following tokens are recognized by the **S** command:

ALTOS Specifies that an alternative kernel (for example, SUNMOS) will be loaded on the node. The node number is added to the list of node numbers in the bootmagic string *BOOT_ALT_NODE_LIST*. The node number is removed from all other bootmagic strings.

DAT Specifies that a DAT tape drive is attached to the SCSI bus on the node. The node number is added to the list of node numbers in the bootmagic string *BOOT_DAT_NODE_LIST*.

SYSCONFIG.TXT *(cont.)* SYSCONFIG.TXT *(cont.)*

DISK Specifies that a hard disk is attached to the SCSI bus on the node. The kind of disk need not be specified. The node number is added to the list of node numbers in the bootmagic string *BOOT_DISK_NODE_LIST.*

FAILED Specifies the node in the slot is failed. The processor port of the corresponding MRC is disabled.

EMPTY Specifies the slot is empty. The processor port of the corresponding mesh routing chip (MRC) is disabled.

ENET Specifies that an ethernet device is available on the node in the slot. The node is added to the list of nodes in the bootmagic string *BOOT_NET_NODE_LOST.*

ETHER Same as **ENET**.

GPNODE *rev mem*

Specifies that the node in the slot is a GP node. The *rev* argument specifies the revision of the node. The *mem* argument specifies the memory on the node. The *mem* argument can be 16 or 32. The add-on memory daughter card (MRC) memory is not included in the memory size. The *rev* and *mem* arguments are recognized by the Paragon diagnostic utilities, but are ignored by the **bootpp** command.

HIPPI Specifies that a HIPPI board is attached to the SCSI bus on the node. The node number is added to the list of node numbers in the bootmagic string *BOOT_HIPPI_NODE_LIST.*

MAXTOR Same as **DISK**.

MIO *rev* Specifies that an MIO board is attached to the node in the slot. This token is recognized by the Paragon diagnostic utilities, but is ignored by the **bootpp** command.

MPNODE *rev mem*

Specifies that the node in the slot is an MP node. The *rev* argument specifies the revision of the node. The *mem* argument specifies the memory on the node. The *mem* argument can be 16, 32, 64, or 128. The *rev* and *mem* arguments are recognized by the Paragon diagnostic utilities, but are ignored by the **bootpp** command.

MRC rev Specifies the revision number of the MRC. The *rev* argument specifies the revision of the MRC.

SYSCONFIG.TXT *(cont.)* SYSCONFIG.TXT *(cont.)*

NOPAGER Prevents the node number being added to the list of nodes that can be used for paging. By default, any node that is defined as having a hard disk is automatically listed as a potential paging node (in case the automatic paging tree configuration is set). This token overrides this default, so the node is not added to the list of paging nodes.

PAGE_TO *device*

Specifies that the device attached to a node is used for paging. By default, all paging uses the device *rz0b*. The node is added to the list of node numbers in the bootmagic string *PAGE_TO*. The *PAGE_TO* string has no effect on the boot node. The boot node uses the default paging device *rz0b*.

RAID3 Same as **DISK**.

RAID5 Same as **DISK**.

TAPE Same as **DAT**.

SYSCONFIG.TXT *(cont.)* SYSCONFIG.TXT *(cont.)*

Examples

The following are examples of entries in a SYSCONFIG.TXT file. The following command specifies that the commands that follow are for cabinet 1.

```
CABINET 1
```

The following command specifies that the **S** commands that follow are for backplane A.

```
BP A AC04

S 0 GPNODE AN00 32 MRC 04
S 1 GPNODE AN00 32 MRC 04
S 2 GPNODE AN00 32 MRC 04
S 3 GPNODE AN00 32 MRC 04
S 4 GPNODE AN00 32 MRC 04
S 5 GPNODE AN00 32 MRC 04
S 6 GPNODE AN00 32 MRC 04
S 7 GPNODE AN00 32 MRC 04
S 8 GPNODE AN00 32 MRC 04
S 9 GPNODE AN00 32 MRC 04
S 10 GPNODE AN00 32 MRC 04
S 11 GPNODE AN00 32 MRC 04
S 12 EMPTY
```

Slots 0 (zero) through 11 contain 32M-byte GP node boards. Slots 12 through 15 are empty.

The following **S** commands specify that slot 2 has a GP node with ethernet and an MIO board, and slot 3 has a GP node board with ethernet, an MIO board, RAID3, and a DAT tape.

```
S 2 GPNODE AN00 32 MRC 04 ENET MIO H04
S 3 GPNODE AN00 32 MRC 04 ENET MIO H04 RAID3 DAT
```

Slots 16 and 17 contain SIO (SCSI-16) nodes. In Slot 17, the drives connected to both SCSI-16 channels (CTLR0 and CTLR1) are used for paging:

```
S 16 MPNODE AG00 64 MRC 04 NIC B02
S 17 MPNODE AG00 64 MRC 04 NIC B02  SIO H04 CTLR0 DAT DAT RAID3
PAGE_TO rz0c CTLR1 RAID3 PAGE_TO rz0c
```

SYSCONFIG.TXT *(cont.)* SYSCONFIG.TXT *(cont.)*

Files

The following files are on the diagnostic station:

/usr/paragon/boot/BADNODES.TXT
: Specifies the nodes that are nonfunctional on a Paragon system.

/usr/paragon/boot/bootmagic
: Specifies information for booting a Paragon system.

/usr/paragon/boot/DEVCONF.TXT
: Specifies the device configuration for a Paragon system.

/usr/paragon/boot/MAGIC.MASTER
: Specifies the master version of the bootmagic strings for booting a Paragon system.

/usr/paragon/boot/SYSCONFIG.TXT
: Specifies the hardware configuration for a Paragon system.

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes*.

See Also

commands: **bootpp**, **reset**

files: **BADNODES.TXT**, **bootmagic**, **DEVCONF.TXT**, **MAGIC.MASTER**

TAR TAR

Specifies tape archive file format.

Description

The **tar** command dumps several files into one, in a medium suitable for transportation.

A tar tape or tar file is a series of blocks, with each block of size **TBLOCK.** A file on the tape is represented by a header block which describes the file, followed by zero or more blocks which give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an end-of-file indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the **b** key letter on the **tar** command line, with a default of 20 blocks) is written with a single system call. On nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads.

The header block looks like:

```
#define TBLOCK 512
#define NAMSIZ 100
union hblock {
     char dummy[TBLOCK];
     struct header {
          char name[NAMSIZ];
          char mode[8];
          char uid[8];
          char gid[8];
          char size[12];
          char mtime[12];
          char chksum[8];
          char linkflag;
          char linkname[NAMSIZ];
     } dbuf;
};
```

TAR (cont.) **TAR** (cont.)

The *name* field is a null-terminated string. The other fields are zero-filled octal numbers in ASCII format. If the width of each field is given as *w,* each field contains *w -2* digits, a space, and a null, with the exception of the *size* and *mtime* fields, which do not contain the trailing null, and the *chksum* field, which has a null followed by a space. The *name* field is the name of the file, as specified on the **tar** command line. Files dumped because they were in a directory which was named in the command line have the directory name as prefix and */filename* as suffix. The *mode* field is the file mode, with the top bit masked off. The *uid* and *gid* fields are the user and group numbers which own the file. The *size* field is the size of the file in bytes. Links and symbolic links are dumped with this field specified as zero. The *mtime* field is the modification time of the file at the time it was dumped. The *chksum* field is an octal ASCII value which represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. The *linkflag* field is null if the file is a regular or special file, ASCII 1 if it is an hard link, and ASCII 2 if it is a symbolic link. The name that the file is linked to, if any, is in the *linkname* field, with a trailing null. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given i-node number is dumped, it is dumped as a regular file. Subsequently, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved but the file it was linked to is not, an error message is printed and the tape must be manually rescanned to retrieve the file that it is linked to.

The encoding of the header is designed to be portable across machines.

On the Paragon system, *extended files* (files greater than 2G-1 bytes in size) can be created in a parallel system (PFS). The tar format for extended files is identical to the standard tar file format, except for the size field in the header block and the method for calculating the header block checksum. In the standard header, the size field consists of 12 characters containing a zero-filled octal number in ASCII format. Only 11 character are digits; the twelfth character is a terminating space. This limits the size of a file to 8G-1 bytes. The tar format for extended files uses the same header block, but it encodes the file size as a hexadecimal number in ASCII format using all 12 characters. This allows a maximum file size 256T-1 bytes (almost 256 terabytes). This ASCII encoding for extended files preserves the standard format, keeping it portable to other architectures that use a different byte ordering.

The checksum for extended files has been modified to add the value 0x3e9 to the standard checksum. This allows the **tar** command to determine that a header block is in the extended format. If an implementation of the **tar** command does not support the extended file format, this checksum should cause the tar command to fail with a "directory checksum error."

Limitations and Workarounds

For information about limitations and workarounds, see the release notes files in */usr/share/release_notes.*

See Also

OSF/1 Command Reference: **tar(1)**.

Index

Paragon™ System Commands Reference Manual